UNIVERSITY OF BIRMINGHAM

Development of Numerical and Machine Learning Models to Simulate the Use of a Playground Swing

Robotics Group Study

March 2021

Group Members:
Kathryn Holmes (Project Manager),
Benjamin Dawkins (Modelling Subgroup Leader), Emily Faithful, Harry Dexter, Zhengchuan Tang (Editor),
Thomas Thies (ML1 Subgroup Leader), Kieron Dodge, Javid Ahmed, Remi Bahar,
Christopher Rouse (ML2 Subgroup Leader), Hoe-Yin Chai, James Marriott, Xiangyu Gong

Abstract

Through this project, models were developed to represent a swing driven by both a standing and seated swinger. These models were produced analytically in Python and through the implementation of Machine Learning in the Pymunk simulation environment, these models were optimised to simulate humanoid motion by the implementation of penalties based on the effort required for a person to perform the desired motion.

Key Words: Swing, Driving, Seated Pump, Standing Pump, Genetic Algorithm, Deep Q-Learning, Reinforcement Learning, TensorFlow, Model, Physiological Effort.

Contents

1 Introduction

Emily Faithful

1.1 Aims

This project aimed to construct a realistic model of a swinging system which uses a human-like swinger. Simulations of models were to be produced using machine learning and an effort parameter, describing the amount of effort required to perform an action, was to be defined. Unlike in previous years, a physical robot was not used during this project. It is intended, nonetheless, that the models and simulations produced provide insight into how a real robot could be programmed to use a real swing.

1.2 Robotics

Popular culture portrays robots as servants to humans (film franchise *Star War's* C3PO), as emotionless logicians, or, more ominously, as humanity's replacement (à la 1988's *The Terminator* or Isaac Asimov's *iRobot*). In reality, the key uses of robots are not to replace humans, but to do what humans cannot, such as deactivating bombs, handling radioactive materials or carrying out tasks in extreme environments like space [1]. Nonetheless, there is truth fuelling the blockbusters, and it could be rooted in our desire to make robots look and act like us. Ancient Egyptian water clocks used human-like figurines to strike a bell every hour. Two thousand years ago, Petronius Arbiter, the Roman writer, built a doll which could move like a human [2]. In 2018, Ranjit Shrivastav launched the Rashmi Robot, a humanoid robot with emotional interpretation capabilities [3].

This is not necessarily evidence of a God-complex. Making robots imitate us is useful. It means that they are capable of interacting with a world designed for humans. Atlas, a humanoid robot from Boston Dynamics intended to search and rescue, is able to drive, open doors and climb ladders, among other things [4]. To be capable of actions such as these, which are traditionally performed by humans, it seems natural that a robot would be modelled after one. Promised to be incorporated into the workplace and in everyday life, robots are likely to become even more human-like, both to put users at ease and to adapt to our world. Using a swing is one example of a human activity to which a robot's capability may be beneficial.

1.3 Machine Learning

Machine learning is an aspect of artificial intelligence which uses algorithms that improve as they use data. Models are built as algorithms which use 'training data', allowing them to make predictions and decisions without being programmed to do so [5]. In the context of this project, machine learning was used to optimise swinging for a chosen parameter, such as amplitude.

Christopher Rouse

The problem of optimising the motion of a virtual swinger is well-suited to machine learning. While analytical solutions for a basic simple harmonic system are straightforward to calculate, the addition of extra degrees of freedom increases the complexity of the system very rapidly. As a result, for a complex pendulum, descent into chaotic motion is hard to solve for analytically. Therefore, attempting a numerical solution arrived at via machine learning makes sense as this theoretically allows the observation of idealised motion for more complex systems than could practically be calculated. Also, once one machine learning approach is operational, it should be easy to edit the model being used and see how this affects motion without extensive further calculations. Practically, the implementation of such an approach is also realistic in principle

as the range of possible states for the system and actions that could be taken by the program are constrained and well-defined. Finally, it should also be straightforward to define a reward function with swing amplitude as its goal.

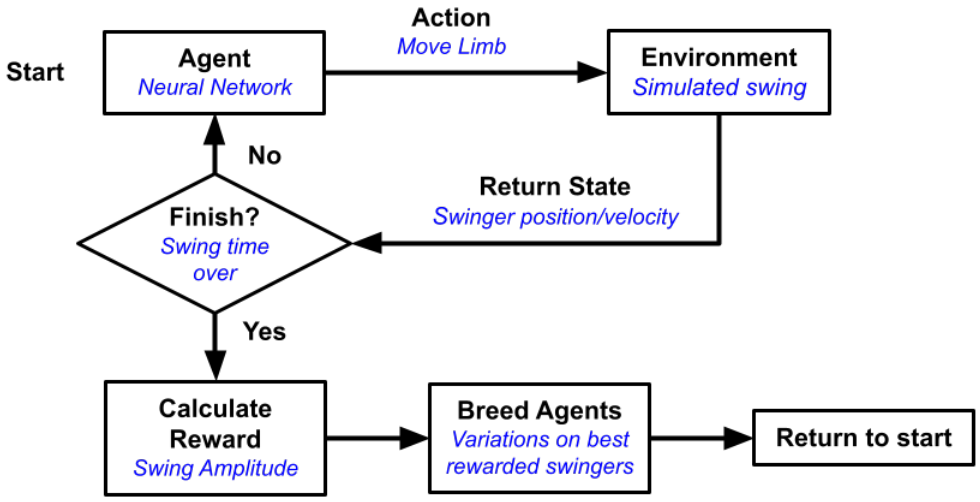


Figure 1: Agent Training Process

The category of machine learning relevant to this problem is reinforcement learning. In brief, reinforcement learning can be thought of in four parts; a state, agent, actions and reward[6]. The state is a description of the relevant characteristics of the system at a point in time e.g. the position and velocity of the limbs of a swinger. The action is then taken by the agent which takes the form of an input to the system based on the current state. The effectiveness of the agent is measured by a reward metric that scores how well the actions it prescribed achieved a defined goal e.g. large swing amplitude. Through many iterations with random agents, those which maximise the reward function most effectively are selected to continue and new random variations are made on those more successful agents. In this way directed trial and error can be used to improve the ability of the computer to complete a task. The agents in this representation are neural networks and the full process for training agents is shown below in Fig 1 and will be repeated until a satisfactory agent is produced.

1.3.1 Standing vs. Sitting Models

Originally, the two groups working on machine learning were split into model and neural network groups. This was subsequently discarded in favour of a group for standing and sitting motion respectively as this allowed for a combination of concurrent work with collaboration that would not cause one group to be slowed by the progress of the other. As relatively self-contained tasks there was a risk of parallel work being done and our efforts not being cumulative. To avoid this issue, proactive steps were taken to ensure collaboration between the groups. These included; using the same simulation environment, working together on similar human models and the sharing of machine learning code between the groups. Though the tasks were ostensibly separate there was consistent collaboration to help achieve them.

After considering various options for a simulation environment Pymunk was selected by the subgroups over 3D alternatives. Pymunk was chosen due to the ample documentation and our familiarity with Python. Due to the deviation this year from using a physical robot that would benefit from simulation in 3D all the physics that was necessary to simulate could be captured in 2D.

1.4 Motivation

Emily Faithful

Using a swing is an intuitive action, commonly performed by children in playgrounds. The act, which may come in useful for a robot as it interacts with the world, is simply understood, yet provides the opportunity to develop sophisticated mathematical models. The simplicity of the action also uncovers the possibility of developing an understanding of machine learning and classical mechanics. Beyond this, the relevance of a project relating to robotics is demonstrated by the continual growth of the robotics sector [7].

1.5 Group Structure

The group was divided into three subgroups, each focusing on a different aspect of the project.

1. Modelling - this subgroup aimed to mathematically model different scenarios and generally provide mathematical background to other subgroups, as well as provide an 'effort parameter'.
2. Machine Learning 1 - this subgroup aimed to create a model for a standing swinger which maximises the oscillation amplitude of a hinged swing whilst minimising the physiological effort - through a machine learning approach.
3. Machine Learning 2 - this subgroup aimed to compare different machine learning algorithms to a common model and compare the motion of hinged swings to unhinged swings. There was a focus on the sitting model.

2 Modelling

2.1 Modelling Introduction

Benjamin Dawkins

This section of the report is focused on the work completed by the Modelling Group. The motivation for the subgroup was to give mathematical background and context to the two Machine Learning (ML) groups, with a view to constraining their models and allowing them to be more realistic. In addition to this, it was thought that making a simplistic version of the models that the ML groups were producing would hopefully help to confirm the validity of their results.

The aims then followed on from these incentives, giving us the goals for the subgroup. The main aim was to develop simplistic, realistic models emulating an individual using a swing - effectively, to produce mathematical models of increasing complexity, starting with a simple pendulum and adding extra detail until a model that is both physically viable and as similar as possible to real life is created. Obviously, the models can never be perfect, but the focus was on trying to make improvements each time. These models were separated into two groups - those mimicking a swinger using the 'standing pump' method, and those of a swinger using the 'sitting pump' method.

When one attempts to envisage a person on a swing, immediately the mind is drawn to images of childhood playgrounds and parks - the smaller child's swing, the scary adult swing, and most importantly the children themselves, pumping their legs back and forth as they swing to try to get higher than their friends. And this is, in effect, the 'sitting pump' method. As the swing descends from its maximum amplitude, if the rider is moving forwards, they should extend their legs out, creating a rotational torque that boosts the amplitude each time by a fixed amount.[8]

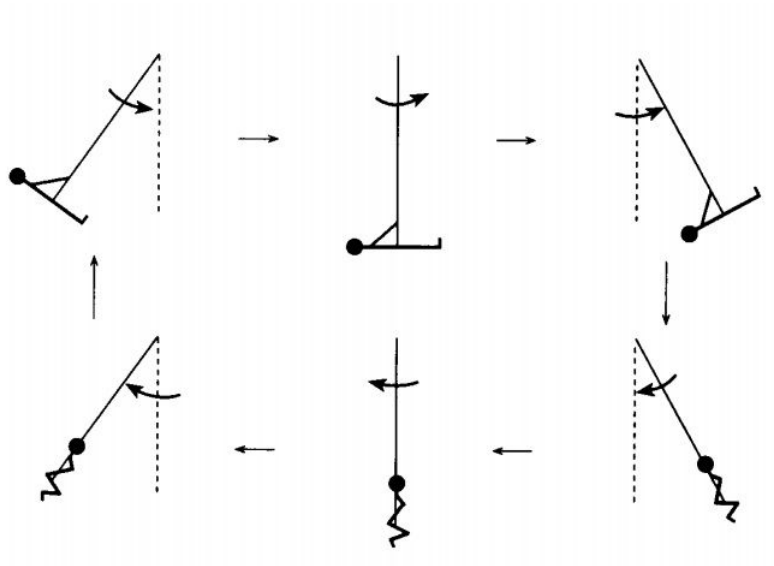


Figure 2: This image shows the method employed by a sitting swinger to maximise their amplitude gain on each swing [8].

This is discussed further in section 2.9. The swinger should then tuck their legs back in as the swing moves backwards, before extending them again as the swing moves forward, in a cycle that continues until a maximum amplitude is reached. This can be seen in figure 2. This maximum occurs when the 'boost' in amplitude is counteracted by the dissipative forces (air resistance, friction, etc.) in the system, leading to no resultant increase in the maximum angle achieved by the swing.

The 'standing pump' method is slightly different. As indicated by the name, the swinger starts crouched at the maximum amplitude of the swing, and stands up when the swing reaches its equilibrium point. Then, they crouch down once more as the swing slows to a stop at its peak amplitude. This can be seen in figure 5. Rather than swinging their legs to create a torque, the rider moves up and down to change the effective length of the pendulum. This changes the moment of inertia, causing a torque - and through conservation of angular momentum, this results in an fractional increase to the maximum angular velocity achieved by the swing. This will be explained in more detail in section 2.3.

The other main aim of the group was to define an 'effort parameter' - an equation designed to quantify the effort that a swinger might put into their motion, in order to maximise the amplitude of the swing. Again, the motivation behind this was to give the ML groups context, and introduce realistic limits on the work done by a person across each half-period of the swing. Initially, the idea was to compare the effort expended by a person to reach the same amplitude using the two different methods, in order to find which one was 'better' - although this was decided against once it was realised that it would be difficult to quantitatively compare the two.

Before exploring the work that was completed by the modelling group, it is important to first understand the mathematical concepts that have been utilised throughout the project. Simple pendulum Physics and mechanics were primarily used, however another method was learnt during this project that was made use of throughout - Lagrangian mechanics. Without prior knowledge, it took a short while to build understanding of this concept to a level at which it was useful, but it was soon realised that it is much easier to use than Newtonian mechanics for more complex systems. There were two equations that were mainly used. Firstly, the Lagrange

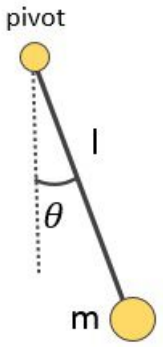


Figure 3: The ubiquitous simple pendulum, the starting point for our models

equation, given by:

$$\mathcal{L} = T - V, \tag{1}$$

which is the difference between the kinetic energy of the system, T, and the potential energy, V. This equation was utilised to find the Lagrangian, $\mathcal{L}$ of the system, which could then be used to find the equations of motion of the system through the Euler-Lagrange equation,

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{r}}\right) = \frac{\partial \mathcal{L}}{\partial r}, \tag{2}$$

where r is any variable used throughout. In more complex cases, this returns the equations of motion in the form of coupled differentials, which can be solved numerically. However, for the simpler models, extracting the needed equations was fairly intuitive.

2.2 Development of Models

Before developing the more complex models of the different swing systems, it was first important to check understanding of the mathematical concepts involved and lay the ground work for later models. This was done by beginning with a very simple version of the models we wanted to end up with; we thought a reasonable starting point would be the simple pendulum, before moving onto the double pendulum and increasing complexity from there. The diagram for a simple pendulum is shown in figure 3, where l is the pendulum length, m the mass and θ is the angle between the pendulum and its equilibrium position (the vertical). The pendulum here also uses a light, rigid rod, so that the centre of mass is at the end. It was also assumed that there was no friction in the system, and both of these assumptions were carried over to the subsequent models (although air resistance was introduced later).

Clearly this is not the most accurate model of a person on a swing, so the next step in the development was to add a hinge. This created a double pendulum when an identical mass was also added to the point where the two rods connect. The double pendulum is a markedly more intricate system, and is a very famous example of chaotic motion in a seemingly simple system [9]. This was not an issue for the utilised model however, as the pendulum we used was being driven and not left to undergo free motion. It was at this point that the development forked into two distinct models - the 'sitting pump' and the 'standing pump'. These more complex models will be explored more in detail in separate sections.

2.3 Standing Model: Boost

A swinger pumping a swing in a standing position can be modelled as a simple pendulum with changing length as shown in figure 4.

Figure 3 consist of mass, m, hanging at the end of the pendulum, which can be driven to move up and down the rod, changing the length of the pendulum to imitate a swinger standing and squatting to pump the swing system.

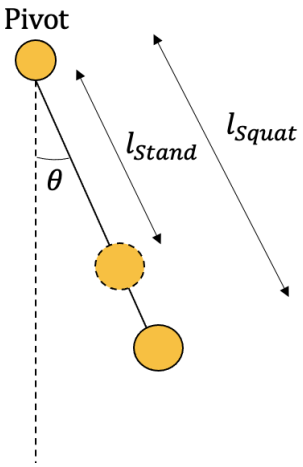


Figure 4: Pendulum with changing length.

In order to efficiently achieve and sustain swinging motion in a standing position on a swing, the swinger needs to stand up and squat down at certain positions during each half period of each swing, such that it supplies the most energy into the swing system to maximise the boost in angular velocity.

$$E \propto F = \frac{mv^2}{l(\theta)}, \tag{3}$$

where E is the energy supplied to the swing system, F is the tension in the swing pendulum, and $l(\theta)$ is the length of the pendulum which varies depending on θ, the angle of the pendulum to the vertical.

From equation 3, the energy supplied to the system is proportional to the linear velocity squared of the swing, and inversely proportional to the length of the pendulum. Therefore to maximise the boost to the angular velocity of the swing, the swinger should stand up at the equilibrium position of the swing where the linear velocity is at it's maximum, and squat down at maximum amplitude of each swing where the linear velocity is at it's minimum. Squatting down at maximum amplitude has no effect on the energy of the swing system because it's temporarily stationary, hence not in circular motion. On the other hand, this also indicates the fact that the system has to be in circular motion in order for the swinger to pump energy into the swing system. How a standing swinger should pump a swing is shown schematically in figure 5.

The boost in angular velocity each time the swinger stands up from squatting position can be derived from conservation of angular momentum,

$$\frac{dL}{dt} = \tau = -mglsin(\theta), \tag{4}$$

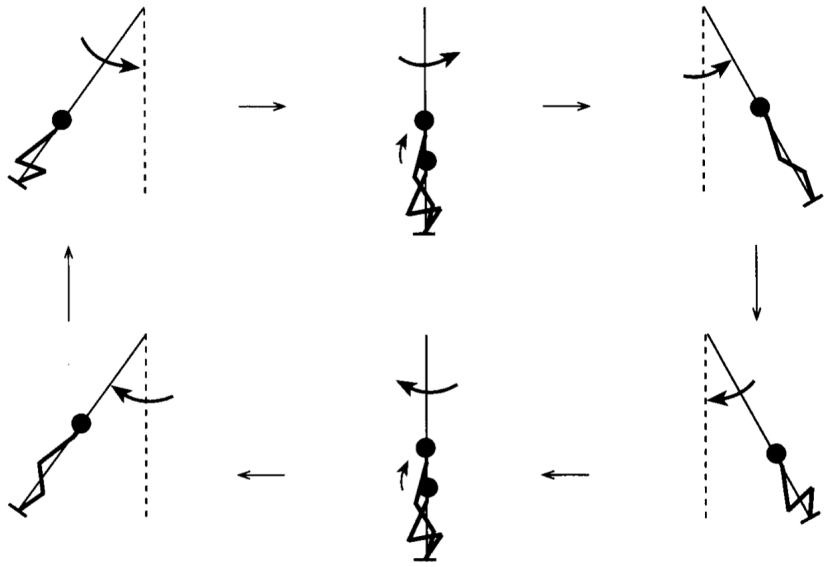


Figure 5: Pumping energy to the swing in a standing position [8].

where L is the angular momentum of the swinging body about the pivot point, and τ is the net torque acting on the body about the pivot point due to gravity, assuming the only force acting is gravity. Substituting L and cancelling m on both sides, equation 4 becomes

$$\frac{d}{dt}(l^2\dot{\theta}) = -glsin(\theta), \tag{5}$$

and

$$\Delta L = l_{St}^2\dot{\theta}_{St} - l_{Sq}^2\dot{\theta}_{Sq} = -\int_t^{t+\delta t} glsin(\theta)dt, \tag{6}$$

where δt is the time taken for the swinger to stand up from squatting position and ΔL is the change in angular momentum. Assuming an instantaneous change such that $\delta t \to 0$, and $\Delta L \to 0$ then rearranging equation 6 obtains

$$\dot{\theta}_{St} = \left(\frac{l_{Sq}}{l_{St}}\right)^2 \dot{\theta}_{Sq}. \tag{7}$$

This gives the boost in angular velocity each time the swinger stands up from squatting position as a ratio of the length of the pendulum in the squatting position and standing position squared. The expression indicates the larger the squatting pendulum length is compared to standing pendulum length, the bigger the boost in angular velocity.

2.4 Standing Model: Lagrangian

The Lagrangian of the standing model shown in figure 4 can be determined using the Lagrange equation given by equation 1 in section 2.1, with the kinetic energy T, and potential energy V, of the system given by

$$T = \frac{m}{2}(\dot{x}^2 + \dot{y}^2), \tag{8}$$

and

$$V = mgy, \tag{9}$$

where $x = l(\theta)sin(\theta)$ and $y = -l(\theta)cos(\theta)$, which represents the position of the swinging body as a function of θ, the angle to the vertical of pendulum.

Differentiating x and y with respect to time will give the velocity of the swinging body in the x and y direction:

$$\dot{x} = (lcos(\theta) + \frac{dl}{d\theta}sin(\theta))\dot{\theta}, \tag{10}$$

and

$$\dot{y} = (lsin(\theta) - \frac{dl}{d\theta}cos(\theta))\dot{\theta}. \tag{11}$$

Substituting $\dot{x}$, $\dot{y}$ and y into equation 8 and 9 then putting the resulting energy expressions into equation 1 will yield

$$\mathcal{L} = \frac{1}{2}m\left[l(\theta)^2\dot{\theta}^2 + \left(\frac{dl}{d\theta}\right)^2\dot{\theta}^2\right] + mgl(\theta)cos(\theta). \tag{12}$$

This is the Lagrangian for the standing model, with l being a function of θ instead of time, t. The reason being trying to make our model as realistically as possible, because a realistic swinger would change their centre of mass depending on the position of the swing, instead of moving as a function of time. For simplicity the Lagrangian using l as a function of time given by

$$\mathcal{L} = \frac{1}{2}m\left[\dot{l}(t)^2\dot{\theta}^2 + l(t)^2\dot{\theta}^2\right] + mgl(t)cos(\theta), \tag{13}$$

was used to derive the equation of motion using the Euler-Lagrange equation, given by equation 2 in section 2.1, with the result being

$$\dot{\theta} = \sqrt{\frac{\ddot{l}(t) - gcos(\theta)}{2l(t)}}. \tag{14}$$

This result was used to produce numerical models of a swing system pumped by a standing swinger.

2.5 Hinged Standing Model

Kathryn Holmes

By adding a massive hinge at the midpoint of the rod, and allowing only the length of the lower section to change length as a function of time, this model is developed to more closely resemble a swing made from ropes/chains whilst still existing within the simplification parameters of this project (namely that rigid light rods are used throughout). A diagram of this model is shown in figure 6.

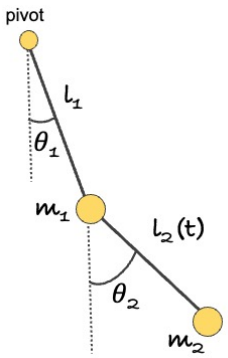


Figure 6: Diagram of Hinged Model for a Standing-driven Swing

The Lagrangian of this model was computed to be

$$\begin{aligned}\mathcal{L} =& gl_1m_1cos(\theta_1) + gm_2(l_1cos(\theta_1) + l_2cos(\theta_2)) + \frac{1}{2}l_1^2m_1\dot{\theta}_1^2 \\ &+ \frac{1}{2}m_2\left(l_1^2\dot{\theta}_1^2 + 2l_1l_2\dot{\theta}_1\dot{\theta}_2cos(\theta_1-\theta_2) - 2l_1\dot{\theta}_1\dot{l}_2sin(\theta_1-\theta_2) + l_2^2\dot{\theta}_2^2 + \dot{l}_2^2\right),\end{aligned} \tag{15}$$

which was then substituted into the Euler-Lagrange Equation shown in equation 2, to give the equations of motion as

$$\ddot{\theta}_1 = -\frac{g}{l_1}sin(\theta_1), \tag{16}$$

$$\ddot{\theta}_2 = \frac{1}{l_2}\left(gsin(\theta_1)cos(\theta_1-\theta_2) - gsin(\theta_2) + l_1\dot{\theta}_1^2sin(\theta_1-\theta_2) - 2\dot{\theta}_2\dot{l}_2\right), \tag{17}$$

where l_2 and $\dot{l}_2$ are defined by the choice of driving parameters. These cannot be solved analytically, and so were later passed to a computer program, discussed in section 2.6, to be solved using numerical methods.

The main difficulty with this model is that a double pendulum naturally becomes chaotic, even when undriven. To counteract this, large amounts of friction are required within the joints, however, even with this large amount of friction, the swing becomes very difficult to drive without becoming chaotic.

2.6 Development of Numerical Model for Standing-Driven Swing

It was decided due to ease of use and extensive experience with the language that the numerically-solved models for the swing would be produced in Python, making use of both Jupyter Notebooks and the Spyder Python environment. A number of models were produced with increasing complexity to ensure at all stages that the algorithms were working as they were expected to.

2.6.1 Damped Simple Pendulum

Initially, the equations of motion derived from the Lagrangian of a simple damped pendulum were solved using the *scipy.integrate.solve-ivp* Runge-Kutta (RK45) integration method, producing the position- and velocity-time graph shown in figure 7. This gave the expected result of sinusoidal position and velocity, $\frac{\pi}{2}$radians out of phase with each other, with smoothly and

equally decreasing magnitude, signifying that the integration method was indeed working as expected.

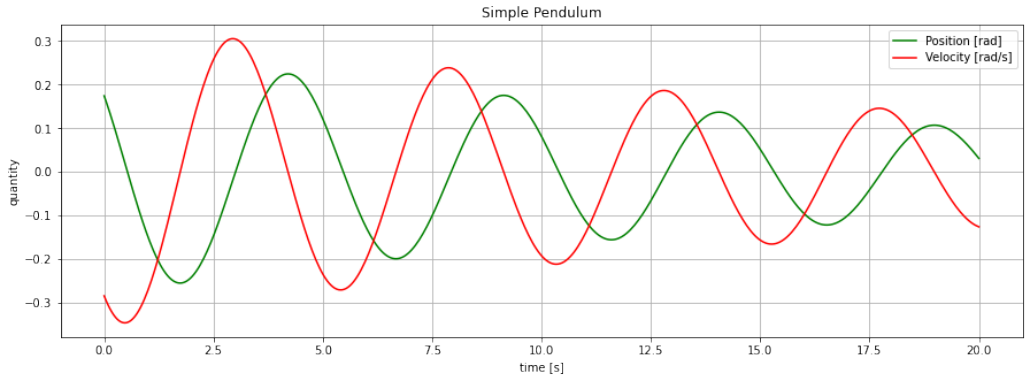


Figure 7: Position- and Velocity-Time graph for a damped simple pendulum solved from Lagrangian Mechanics by Scipy RK45 Numerical Integrator.

2.6.2 Sinusoidally-Changing-Length Driven Simple Pendulum

To emulate a person standing and squatting to drive the swing, the length of the pendulum was changed as a function of time, such that when the rider stands the length decreases (as the centre of mass is moved closer to the pivot) and when the rider squats the length increases. Initially, the length was changed sinusoidally, as this seemed to be the simplest method to get reasonably sharp movements which were still physically possible. By applying a sinusoidal length function with period equal to half the period of the undriven pendulum (as the rider needs to squat/stand twice in every period), the swing is driven as shown in figure 8.

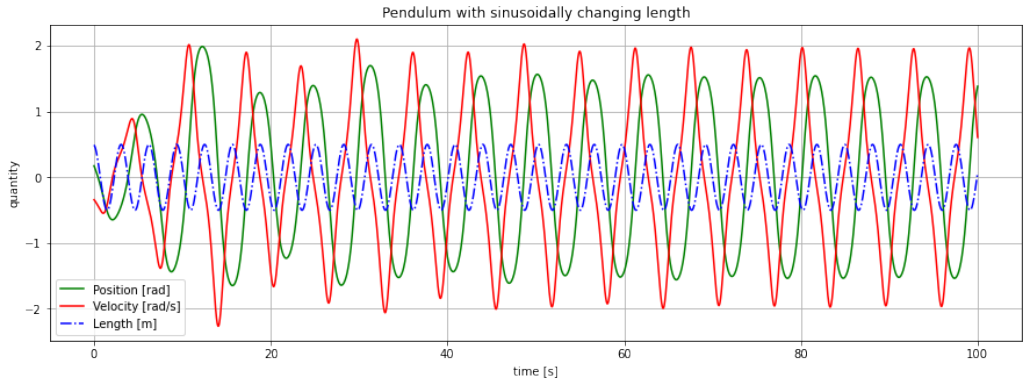


Figure 8: Position- and Velocity-Time graph for a damped simple pendulum driven by sinusoidally-changing length.

As seen in this plot, the amplitude of oscillations is increased initially, reaching a maximum within two periods. After this point, the changing length falls out of phase with the motion (as large amplitude oscillations do not have a fixed period) and briefly hinders it, reducing the amplitude before falling back into phase and maintaining a steady amplitude where the energy provided by the changing length is sufficient to overcome the energy lost due to damping. The change in energy over time is shown in figure 9. This clearly shows the sharp energy increase as the pendulum is driven and the steady-state reached following a series of fluctuations with decreasing magnitude.

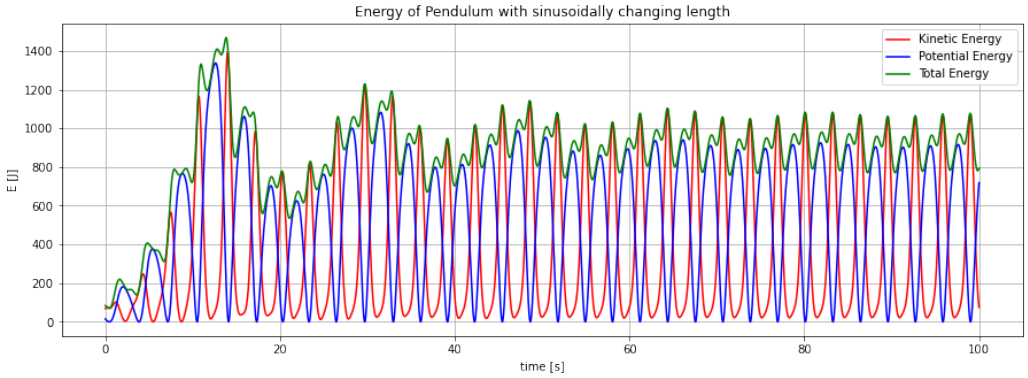


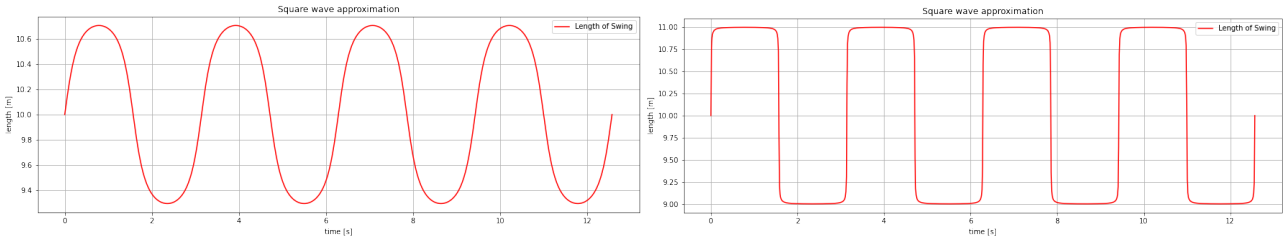
Figure 9: Energy-Time graph for a damped simple pendulum driven by sinusoidally-changing length.

2.6.3 Square-wave Approximation

To simulate the approximately instantaneous length change desired by the models produced earlier, it was decided that the most appropriate way to replicate this numerically would be a square wave. As an ideal square wave would be unphysical due to the instantaneous change producing an infinite acceleration, a less-sharp approximation to the square wave was required. It was found that a smooth, differentiable square wave can be approximated by the inverse tangent of a sine wave, such that the length of the pendulum,

$$L = A * tan^{-1}\left(\frac{sin(\omega t)}{\delta}\right) + L_0[10], \tag{18}$$

where the choice of A controls the magnitude of the fluctuation, L_0 is the median length, and the choice of δ changes the sharpness of the square wave, as shown in figure 10, with smaller values of δ producing a sharper square wave.



(a) Square wave approximation with $\delta = 0.5$

(b) Square wave approximation with $\delta = 0.01$

Figure 10: Examples of approximated square-wave length functions, from Equation 18, for different δ.

2.6.4 Square-wave-Changing-Length Driven Simple Pendulum

By applying the square wave approximation discussed in section 2.6.3 to the model developed for the sinusoidally changing length, and initially setting $\delta = 0.1$, the swing was driven to produce the plot shown in figure 11. Similarly to the sinusoidal model, the amplitude of the swing quickly increases, although this increase is more stable for the square wave driving. It reaches a maximum before falling slightly out of phase with the swing and no longer adding the maximum possible amount of energy, instead simply overcoming the friction in the system and maintaining a steady state.

If δ is decreased, i.e. the square wave made sharper (as in figure 10b), the very abrupt changes to the length hinder the motion as the effect of any slight difference in phase is magnified, causing overdamping and quickly decreasing the swing's amplitude, as shown in figure 12.

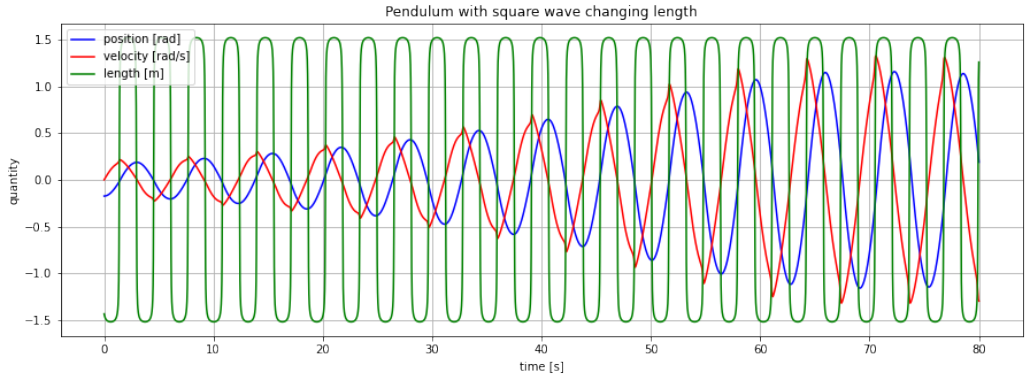


Figure 11: Position- and Velocity-Time graph for a damped simple pendulum driven by a Square-wave-changing length, with $\delta = 0.1$.

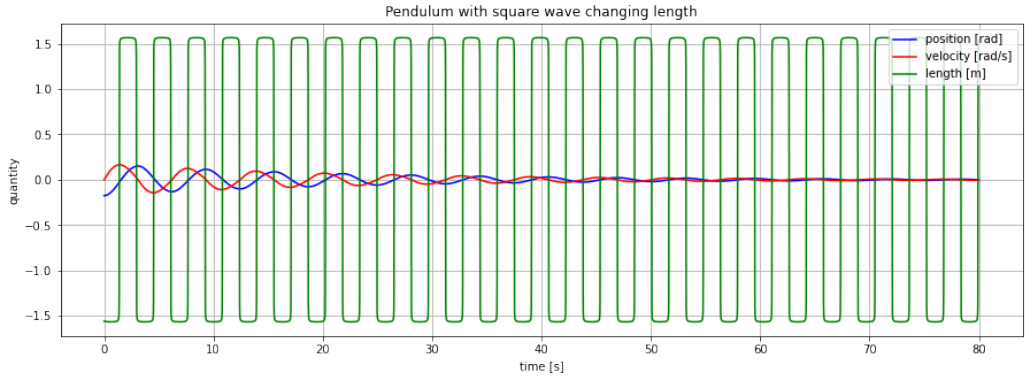


Figure 12: Position- and Velocity-Time graph for a damped simple pendulum driven by a Square-wave-changing length, with $\delta = 0.01$.

In contrast to this, by increasing δ to 0.5 and softening the square wave (as in figure 10a), the swing experiences a far greater acceleration due to the length change, and the amplitude, θ, becomes much greater than π radians, suggesting that the motion is no longer oscillatory and instead the swing is simply spinning around the pivot in one direction. This occurs because the longer period over which the length is changing ensures that the pump continues to occur when the swing is roughly at maximum amplitude or at equilibrium, and so adds energy at every quarter period with no limitation. The position- and velocity-time graph for this is shown in figure 13.

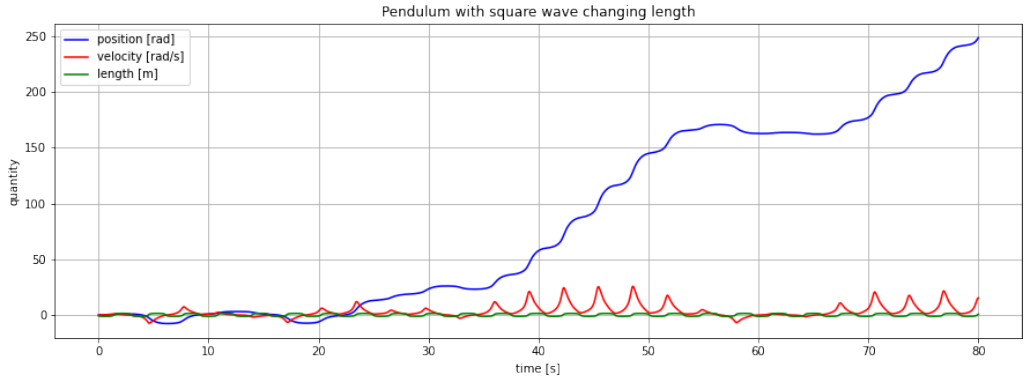


Figure 13: Position- and Velocity-Time graph for a damped simple pendulum driven by a Square-wave-changing length, with $\delta = 0.5$.

For the stable scenario shown in figure 11, the evolution of the energy of the system is shown in figure 14, which demonstrates that the energy increases steadily with time and plateaus to

a constant average value following a brief decrease from the maximum. The energy achieved by the square wave length function after plateau is $\approx 2150J$, which is significantly larger than that achieved by the sinusoidal length function $\approx 1080J$, for the same median length and magnitude of length change, showing that changing the length with a square wave is far more efficient.

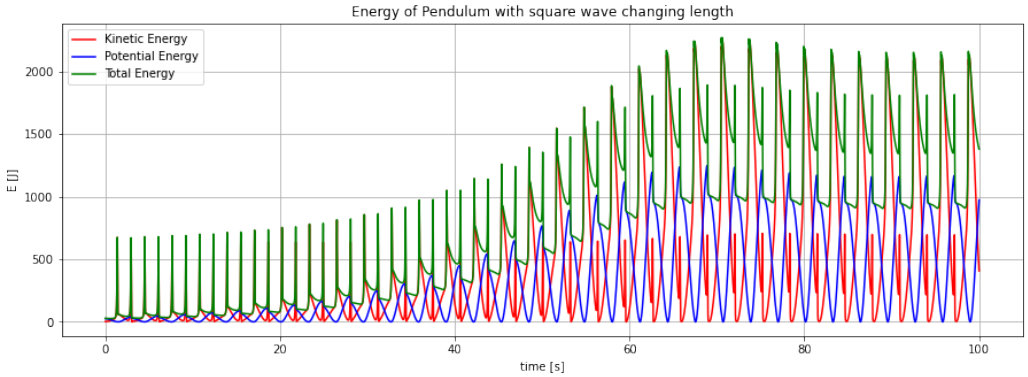


Figure 14: Evolution of the Energy of the square wave driven pendulum with $\delta = 0.1$.

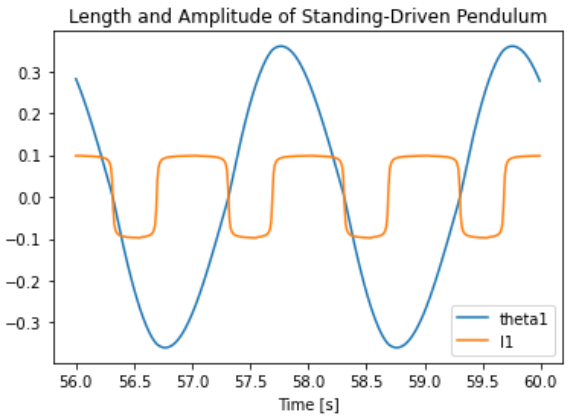
2.6.5 Adaptive Length Change

To overcome the problem of the non-constant oscillation period, a more sophisticated Python program was developed from an existing example [11], in which the optimal driving frequency was found by iterating over a range and finding the driving frequency which resulted in the highest amplitude at the end of one minute of swinging. This frequency was then used as the basis for an adaptive length-change process. To perform this, the motion was integrated using *Scipy.integrate.odeint* to allow for precise control of the time-points at which the function was evaluated. This allowed the frequency of oscillation to be found every quarter-period from the location of turning points and zero-crossings of the amplitude function using *scipy.interpo1ate*. From this, an asymmetrical square wave was produced whose frequency was double that of the pendulum motion at all times, which changed the length of the pendulum just before the maximal and equilibrium points were reached. This quickly reached the steady state of maximum amplitude and there was no dip in energy or amplitude from the driving and swinging becoming out of phase. Plots to show both how the square wave adapted to the changing period of the motion and how the energy varied over time are shown in figure 15.

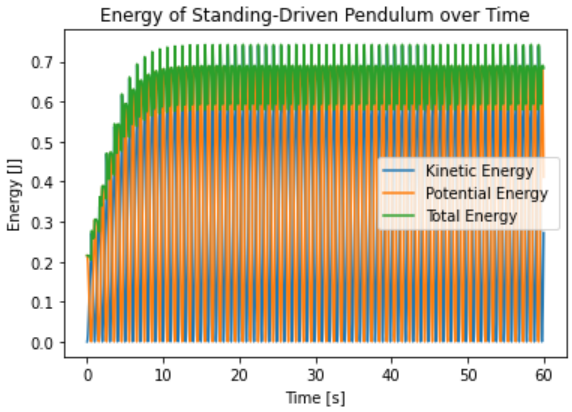
With the adaptive length change, the negative effects of sharpening the square wave discussed in section 2.6.4 do not occur, and instead decreasing delta - which increases the acceleration of the length change - increases the amount of energy added to the system by the pumping.

2.6.6 Square-Wave-Changing-Length Driven Double Pendulum

To construct a more 'realistic' model, a hinge was added as described in section 2.5, with the adaptive-length-change Python program being reused and altered to process this more complicated system. This system suffered from chaotic motion from the start, as can be seen in figure 16 from the sharp turning points, particularly those which occurred on the same side of equilibrium as the last, where the period of the square wave was kept constant like in section 2.6.4 for simplicity. Issues with the code itself (which there was insufficient time to solve) prevented the interpolation finding the zero-crossings after only a few periods due to the sharp motions causing a zero-crossing to be missed and thus putting the length change out of phase with the motion. From research into examples of standing-driven swings, it seems that generally those pumped by standing and squatting consist of rigid A-frame supports with no



(a) Square-wave driven swing with adaptive step-length, section shown at the end of the evolution, showing that the length changes are still occurring at optimal times



(b) Energy change over time from square-wave driven swing, showing the quick increase to maximum energy.

Figure 15: Plots showing results from the adaptive length-change algorithm for a simple pendulum driven by changing length

hinges, so it is possible that driving a hinged swing this way is too difficult to control. However, it is still an interesting mathematical problem which could be a good starting point for future groups, furthering the goal of the modelling group to validate the machine-learning results. The principle of this approach can be seen in figure 17a; figure 17b shows how the energy of the system varied over this short period for the adaptive step length.

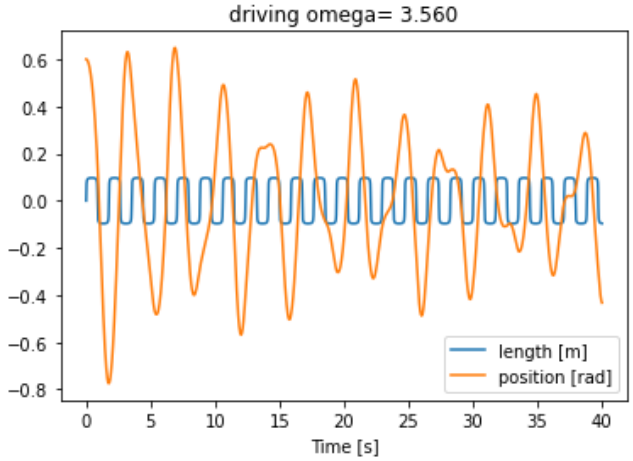
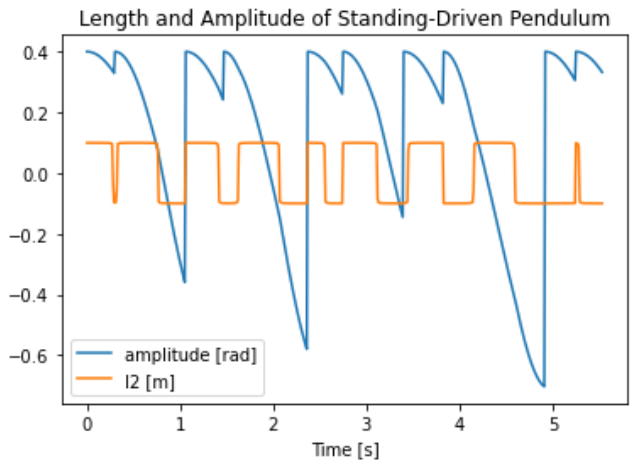
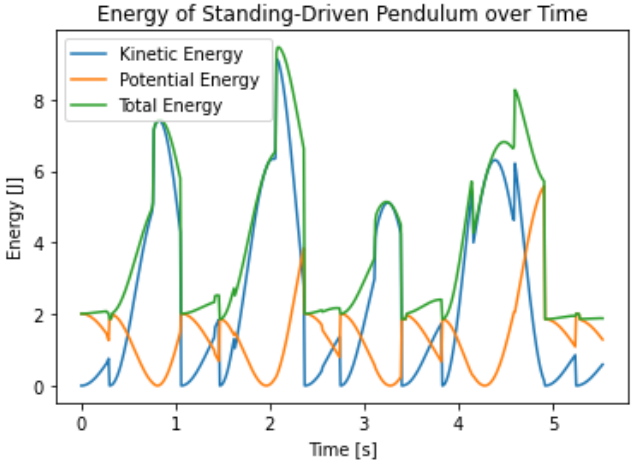


Figure 16: Position-Time plot for hinged pendulum with square-wave changing length, and demonstration of how the length changed over time at the optimal driving frequency found by the program.



(a) Length and amplitude of square-wave driven pendulum with adaptive square-wave step length, which fails at $t \approx 5$seconds



(b) Energy change over time from square-wave driven double pendulum, showing the chaotic nature.

Figure 17: Plots showing results from the adaptive length-change algorithm for a changing-length driven double pendulum

2.7 Standing Model: Effort

Harry Dexter

A system of equations were derived to help develop an understanding for the effort involved when an individual pumps energy into a swing by standing up at the equilibrium position and squatting down at the points where $\dot{\theta}$ is equal to zero. There are 5 key parts that represent different features of the motion.

Zhengchuan Tang

The overall effort parameter for a standing swinger is given by:

$$\epsilon = \frac{100}{15} m \left[\frac{21\Delta\theta_{th}^2 r_{th}^2}{100\Delta t_{th}^2} + \frac{57\Delta\theta_a^2 r_a^2}{500\Delta t_a^2} + g(3 - 2cos(\theta))\Delta h + (a_s + gcos(\theta))\Delta h + \frac{gh}{2}(1 - cos(\Delta\theta)) \right], \tag{19}$$

where the first two terms corresponds to the effort required for the swinger to rotate their thighs and arms, the third term corresponds the effort required for the swinger to stand up, the fourth term corresponds to the effort required to squat down, and the fifth term corresponds to the effort required to maintain balanced while swinging. The factor of $\frac{100}{15}$ takes into account of the efficiency of the swinger's muscles, and m is the mass of the swinger. How each term within the effort parameter are determined will be discussed in more detail within this section.

2.7.1 Efficiency of Muscles

Harry Dexter

Muscles are never 100 percent efficient. Muscle efficiency is the proportion of the total metabolic cost that goes into mechanical work.

The majority of the energy that goes into motion is wasted as heat or other internal processes. It is widely accepted that muscles are between 15-30% efficient [12]. The minimum value of

15% was taken. The efficiency η is shown in equation 20.

$$E_{total} = \frac{1}{\eta} E_{mech} = \frac{100}{15} E_{mech} \tag{20}$$

E_{mech} = Mechanical work done by swinger

2.7.2 Rotating Limbs

Harry Dexter

When a mass is rotated about an arbitrary point (pivot), energy is exerted (work is done). The amount of energy needed will depend on; the moment of inertia of the body (I) and the angle through which it is rotated. [13]

Equation 21 represents the energy required to rotate a body around an angle θ:

$$W = \int_{\theta_0}^{\theta_1} \tau(\theta) d\theta \tag{21}$$

The torque is represented by equation 22:

$$\tau(\theta) = R \perp F(\theta) = I\alpha(\theta) \tag{22}$$

The moment of inertia of a rod (I) about its centre, which is a reasonable representation of a human limb, is:

$$I_{limb} = \frac{1}{12} m l^2 \tag{23}$$

The energy required to move a limb through an angle of θ is represented in equation 24:

$$E_{rotational} = \frac{ml^2}{12} \int_{\theta_0}^{\theta_1} \mid \alpha(\theta) \mid d\theta \tag{24}$$

Equation 21 shows that the work done to move through an angle would be zero because an equal negative torque would be applied when slowing down the limb. Equation 24 takes this into account, as only the absolute value of α is used when calculating the energy required. This is important because in reality, energy is required not only to accelerate the limb but also the decelerate it back to stationary. However, because α is equal to the rate of change of ω the total area under the integral will be the same for all functions of α with constant angle change.

To constrain this motion, the individual on the swing can reduce the angle that they move their limbs through and move shorter lighter limbs more than longer heavier limbs.

These factors don't take into account the human effort factor. Similar to the standing effort it is clear that a smaller torque would be much more realistic for human motion, so smaller torques applied over a longer time should be rewarded in machine learning models.

2.7.3 Standing Term

Harry Dexter

When someone stands up at the equilibrium position they do work against gravity and the centrifugal acceleration. This is shown in equation 25:

$$W_{stand} = F\Delta h = m(g + a_c)\Delta h \tag{25}$$

Where Δh is the change in distance of swingers COM from pivot (change in length of swing), m is the mass of swinger and a_c is the centrifugal acceleration.

To calculate the centrifugal acceleration, conservation of energy, and the equations representing circular motion were used:

$$a_c = \frac{v_{max}^2}{r} = \frac{2gH}{l} = \frac{2gl(1 - cos(\theta_{max}))}{l} = 2g(1 - cos(\theta_{max})) \tag{26}$$

θ_{max} is the maximum angle of previous swing and H is the distance of swinger above equilibrium position at $\theta = 0$.

Subbing equation 26 into 25 and simplifying produces equation 27.

$$\epsilon_{stand} = mg(3 - 2cos(\theta_{max}))\Delta h \tag{27}$$

Clearly this equation is not a good constraint to improve the motion of the standing swinger as the only variable is Δh. It is a description of the difficulty involved when standing up whilst on a swing.

However, it is known that rapid movements which exert very large forces are not realistic for humans and will be much more costly than slower movements in terms of efficiency. A human is most likely to match the reward of increased amplitude to the cost of rapid motion and find the middle ground. Therefore in terms of constraining the motion it is clear that there should be some reward for slower movements. It should also be noted that the motion will be much more difficult for taller and heavier individuals.

2.7.4 Squatting Term

Zhengchuan Tang

The effort required for a swinger to squat on the swing was derived by resolving all forces acting on the swinger at an angle, θ, to the equilibrium position of the swing, also shown in figure 18, where a_s is the acceleration due to the swinger squatting down.

From figure 18, the total force due to motion towards the balance pivot point perpendicular to the swing seat is given by

$$F = m(a_s + gcos(\theta)). \tag{28}$$

From equation 28, the effort required to squat down can be determined by multiplying the distance which the swinger has squatted down by, label as Δh in figure 18:

$$\epsilon_{squat} = m(a_s + gcos(\theta))\Delta h. \tag{29}$$

This expression for effort required to squat down includes work done by the swinger as well as the physiological effort from the swinger. Similar to all other effort terms within the effort

parameter,it is multiplied by a factor of $\frac{100}{15}$, taking the efficiency of the muscles of the swinger to be around 15% [12].

2.7.5 Balancing Term

The effort required for a swinger to remain balanced on the swing was derived by considering all torques acting on the swinger at an angle θ, to the equilibrium position of the swing, as shown in figure 18.

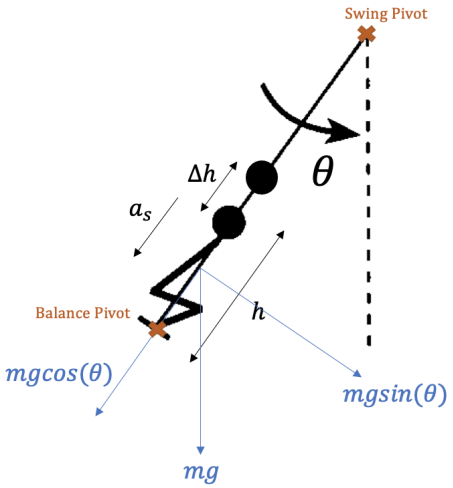


Figure 18: Schematic of a swinger on the swing.

Depending on which side of the equilibrium position the swinger is in, there will be either a clockwise or anti-clockwise torque acting on the swinger due to gravity about the balance pivot point labeled in figure 18, given by equation 30.

$$\vec{\tau} = \vec{r} \times \vec{F} = \frac{h}{2} mgsin(\theta), \tag{30}$$

where h is the height of the swinger when trying to balance, which varies depending on whether the swinger is squatting down or standing up.

To remain balanced on the swing, the swinger needs to generate a counter-clockwise torque with equal magnitude as the torque due to gravity, about the balance pivot point labeled in figure 18. From this, the effort required from the swinger to maintain balanced on the swing can be determined:

$$\epsilon_{balance} = \int_{\theta_1}^{\theta_2} \vec{\tau} d\vec{\theta} = \int_{\theta_1}^{\theta_2} \frac{h}{2} mgsin(\theta) d\theta, \tag{31}$$

integrating this result will give

$$\epsilon_{balance} = \frac{h}{2} mg(1 - cos(\Delta\theta)), \tag{32}$$

where $\Delta\theta$ is the angle which the swing elapses in a given time.

To take into account of the efficiency of the muscles in the body, equation 32 is multiplied by a factor of $\frac{100}{15}$, taking the muscles of the swinger to be around 15% [12].

2.7.6 Position Dependence of the Effort Parameter

The standing effort parameter was plotted against the position of the swing over a half period cycle, where position is measured in units of radians, with the equilibrium position equal to zero radians, as shown in figure 19.

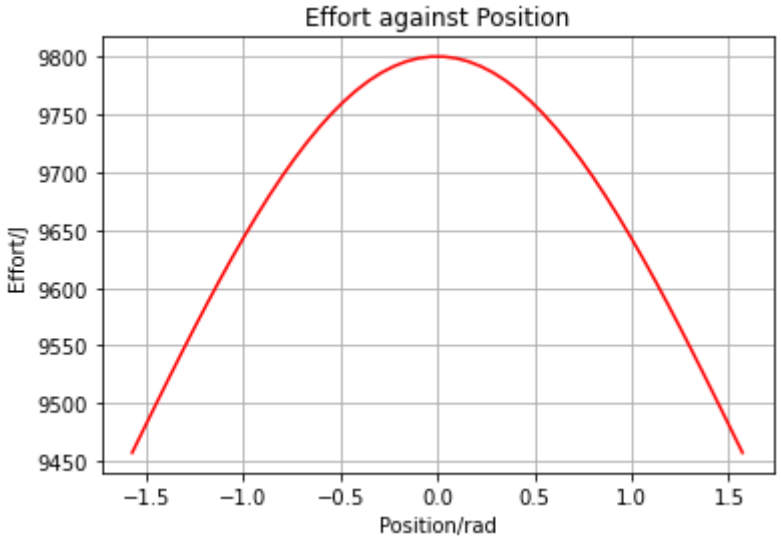


Figure 19: Plot to show how effort varies with position throughout a half period swing.

Figure 19 suggest the maximum effort required from the swinger occurs at the equilibrium position. This is the position where the swinger stands up to pump energy into the swinging system, and it is the dominant term within the effort parameter. The effort then decreases as the swing reaches maximum amplitude on both sides of the equilibrium position, where the swinger then squats down in increase the distance between the center of mass of the body and the swing picot point. This could be due to the fact that the lower the center of mass of the swinger relative to the swing seat, the less effort it is required to maintain in a balanced position on the swing.

2.8 Seated Model: Lagrangian

Emily Faithful

The kinetic and potential energies of the barbel model shown in figure 20 were found, in order to find the Lagrangian $\mathcal{L} = T - V$. The barbel has three concentrated points of equal mass, m. If the central mass has position (x_1, y_1), the leftmost (x_2, y_2) and the rightmost (x_3, y_3), the positions of each mass are,

- $x_1 = l\sin\theta$,
- $y_1 = l\cos\theta$,
- $x_2 = l\sin\theta - a\sin(\alpha - \theta)$,
- $y_2 = l\cos\theta - a\cos(\alpha - \theta)$,
- $x_3 = l\sin\theta + a\sin(\alpha - \theta)$ and,
- $y_3 = l\cos\theta + a\cos(\alpha - \theta)$.

Summing the potential energies $(-mgy)$ of each point of mass gives,

$$V = -3mgl\cos\theta + mga\cos(\alpha - \theta) - mga\cos(\alpha - \theta) = -3mgl\cos\theta. \tag{33}$$

It can be seen that the elevation of one end-mass requires the falling of the other, with respect to the central mass. Consequently, their heights with respect to the central mass, as described by the second terms of y_2 and y_3, cancel out when summed, such as in the total gravitational potential energy.

Finding the kinetic energy requires differentiation of x and y with respect to time. These component velocities of each point of mass are found to be,

- $\dot{x_1} = l\dot{\theta}\cos\theta$,
- $\dot{y_1} = -l\dot{\theta}\sin\theta$,
- $\dot{x_2} = l\dot{\theta}\cos\theta - a(\dot{\alpha} - \dot{\theta})\cos(\alpha - \theta)+$,
- $\dot{y_2} = -l\dot{\theta}\sin\theta + a(\dot{\alpha} - \dot{\theta})\sin(\alpha - \theta)$,
- $\dot{x_3} = l\dot{\theta}\cos\theta + a(\dot{\alpha} - \dot{\theta})\cos(\alpha - \theta)$ and,
- $\dot{y_3} = -l\dot{\theta}\sin\theta - a(\dot{\alpha} - \dot{\theta})\sin(\alpha - \theta)$.

Summing kinetic energies of each point of mass $(T = \sum_i \frac{1}{2}m_i(\dot{x_i}^2 + \dot{y_i}^2))$ gives,

$$T = \frac{1}{2}m(3l^2\dot{\theta}^2(\cos^2\theta + \sin^2\theta) + 2\alpha^2(\dot{\alpha} - \dot{\theta})^2(\cos^2(\alpha - \theta) + \sin^2(\alpha - \theta))). \tag{34}$$

Simplified, by considering $\cos^2 x + \sin^2 x = 1$, this is reduced to,

$$T = \frac{1}{2}m(3l^2\dot{\theta}^2 + 2\alpha^2(\dot{\alpha} - \dot{\theta})^2). \tag{35}$$

From these two energy equations, the Lagrangian is found to be,

$$\mathcal{L} = T - V = \frac{1}{2}m(3l^2\dot{\theta}^2 + 2\alpha^2(\dot{\alpha} - \dot{\theta})^2) + 3mgl\cos\theta. \tag{36}$$

Using the Euler-Lagrange equation ($\frac{d}{dt}(\frac{\partial \mathcal{L}}{\partial \dot{r}}) = \frac{\partial \mathcal{L}}{\partial r}$), equations of motion can be derived. First, the variable α was considered. The right hand side of the Euler-Lagrange equation is,

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 2m\alpha(\dot{\alpha} - \dot{\theta})^2. \tag{37}$$

Then,

$$\frac{\partial \mathcal{L}}{\partial \dot{\alpha}} = 2m\alpha^2(\dot{\alpha} - \dot{\theta}), \tag{38}$$

and,

$$\frac{d}{dt}(\frac{\partial \mathcal{L}}{\partial \dot{\alpha}}) = 2m\alpha^2(\ddot{\alpha} - \ddot{\theta}) + 4m\dot{\alpha}(\dot{\alpha} - \dot{\theta}). \tag{39}$$

The Euler-Lagrange equation is therefore,

$$\alpha^2(\dot{\alpha} - \dot{\theta}) = \alpha^2(\ddot{\alpha} - \ddot{\theta}) + 2\dot{\alpha}(\dot{\alpha} - \dot{\theta}). \tag{40}$$

Rearranging to make angular accelerations the subjects, two equations of motion are obtained,

$$\ddot{\theta} = \ddot{\alpha} - \frac{(\alpha^2 - 2\dot{\alpha})(\dot{\alpha} - \dot{\theta})}{\alpha^2}, \tag{41}$$

and,

$$\ddot{\alpha} = \ddot{\theta} + \frac{(\alpha^2 - 2\dot{\alpha})(\dot{\alpha} - \dot{\theta})}{\alpha^2}. \tag{42}$$

Similarly, from finding,

$$\frac{\partial \mathcal{L}}{\partial \theta} = -3mgl\sin\theta, \tag{43}$$

with,

$$\frac{\partial \mathcal{L}}{\partial \dot{\theta}} = 3ml^2\dot{\theta} + 2m\alpha^2(\dot{\theta} - \dot{\alpha}), \tag{44}$$

and,

$$\frac{d}{dt}(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}) = 3ml^2\ddot{\theta} + 4m\dot{\alpha}(\dot{\theta} - \dot{\alpha}) + 2m\alpha^2(\ddot{\theta} - \ddot{\alpha}), \tag{45}$$

the Euler-Lagrange equation is obtained as,

$$-3gl\sin\theta = 3l^2\ddot{\theta} + 4\dot{\alpha}(\dot{\theta} - \dot{\alpha}) + 2\alpha^2(\ddot{\theta} - \ddot{\alpha}). \tag{46}$$

From this, two more equations of motion are obtained.

$$\ddot{\theta} = \frac{2\alpha^2\ddot{\alpha} - 4\dot{\alpha}(\dot{\theta} - \dot{\alpha}) - 3gl\sin\theta}{3l^2 + 2\alpha^2}, \tag{47}$$

and,

$$\ddot{\alpha} = \ddot{\theta} - \frac{3l^2\ddot{\theta} + 4\dot{\alpha}(\dot{\theta} - \dot{\alpha}) + 3gl\sin\theta}{2\alpha^2}. \tag{48}$$

Most evident is perhaps the dependence of one angular acceleration on the other. Since torque about a pivot is proportional to the angular acceleration about that pivot ($\tau = I\ddot{\phi}$, where ϕ is some angle), the dependence of $\ddot{\theta}$ on $\ddot{\alpha}$ (and vice versa) demonstrates how the application of torque about the pivot at the seat provides a torque at the support point (pivot at the fixed point to which the rope is attached).

2.9 Seated Model: Boost

Emily Faithful

Pumping the swing from a seated position is perhaps the natural method used to operate a swing. It allows the maximum value of θ reached in a swing cycle, also called the amplitude of the swing, to increase. To pump from a seated position, the user must lean back (perpendicular to the rope) as they move forward, and sit up straight (in line with the rope) as they move backwards, as shown in figure 2 [8]. This was modelled as a rotating barbel, with three points of mass distributed as shown in figure 20 (one in the middle, two at the ends of the barbel).

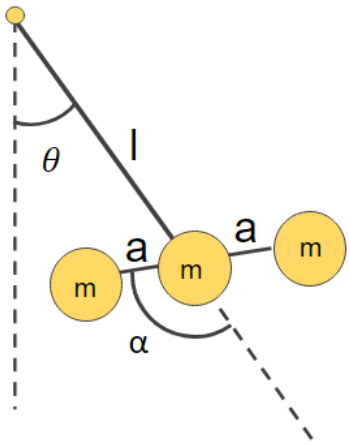


Figure 20: The barbel model. Here α is defined as the angle between the swinger and the rope.

The conservation of angular momentum dictates that,

$$\frac{dL}{dt} = \tau, \tag{49}$$

where L is the angular momentum of the body about the support point (the static point to which the rope is fixed and the point around which θ is measured) and τ is the net torque about that support point. $L = I\omega$, where ω is the angular velocity around a point and I is the moment of inertia. In this case, the angular momentum about the pivot of the entire pendulum system is $m(l^2 + a^2)\dot{\theta} + ma^2\dot{\alpha}$, the sum of angular momenta about the joint between the rope

and the fixed point, and between the rope and the seat, respectively. Using conservation of angular momentum, and dividing through by m, the equation of motion is obtained,

$$\frac{d}{dt}[(l^2 + a^2)\dot{\theta} + a^2\dot{\alpha}] = -gl\sin\theta. \tag{50}$$

The rotation from $\alpha = 0$ to $\alpha = 90^o$ begins at roughly time $t = 0$ and takes time Δt to complete. The equation of motion is integrated over an interval between $t = 0$ and $t \leq \Delta t$ to give,

$$(l^2 + a^2)[\dot{\theta}(t) - \dot{\theta}(0)] + a^2[\dot{\alpha}(t) - \dot{\alpha}(0)] = -gl\int_0^t \sin\theta dT. \tag{51}$$

To obtain the effect on angles, the integrated equation of motion is integrated from 0 to Δt (the whole period of rotation). This gives,

$$(l^2 + a^2)[\theta(\Delta t) - \theta(0) - \dot{\theta}(0)\Delta t] + a^2[\alpha(\Delta t) - \alpha(0) - \dot{\alpha}(0)\Delta t] = -gl\int_0^{\Delta t}\int_0^t \sin\theta dT dt. \tag{52}$$

The right hand side of this integral is on the order of $(\Delta t)^2$. At $t = 0$, the angular velocity of the rotation is 0 ($\dot{\alpha} = 0$), as by definition, the rotation has not yet begun at $t = 0$. In the limit that Δt and approaches zero, the right hand side of the equation vanishes and rearrangement obtains,

$$(l^2 + a^2)\Delta\theta = -a^2\Delta\alpha. \tag{53}$$

The limit means that this is true for fast rotations, where $\Delta t \to 0$. Rearranging the equation for $\Delta\alpha = \frac{\pi}{2}$, gives an overall change in amplitude (change in maximum θ) of,

$$\Delta\theta = \frac{a^2\pi}{2(l^2 + a^2)}. \tag{54}$$

The meaning of this is that a fast, 90^o rotation increases the amplitude by an amount which depends on both the height of the barbel, or swinger, and on the length of the swing. This method of pumping is optimal for low amplitudes, beyond which a standing model is more effective.

2.10 Development of Numerical Model for Seated Swinger

Harry Dexter

For an isolated pendulum that exhibits simple harmonic motion, the total angular momentum of the system is never conserved due to gravity applying an external torque.

$$\frac{d\vec{L}}{dt} = \vec{\tau} = \vec{R} \times \vec{F_g} = mglsin(\theta) \tag{55}$$

However, if an internal torque is applied, for example, by a person on a swing rotating their body, the system will react and apply an equal and opposite torque at the pivot in order to conserve the total angular momentum of the system. Therefore, it is possible for someone to rotate their body at specific points during the swinging motion to increase the amplitude

achieved in each swing. To understand and explore this motion, a simple model was built (shown in figure 20). This model consists of a barbell located at the end of a rigid rod which will rotate to represent the individual rotating their body on the swing. The equations of motion were derived from the Lagrange and Euler-Lagrange equations and solved numerically in Python. [8]

2.10.1 Developing Equations of Motion

The Lagrangian (equation 56) was derived for the model depicted in figure 20:

$$\mathcal{L} = \frac{1}{2}m[3l^2\dot{\theta}^2 + a^2\dot{\alpha}^2 + 2la\dot{\theta}\dot{\alpha}cos(\theta)[sin(\theta) + cos(\alpha)] + 2a^2\dot{\alpha}\dot{\theta}cos(\theta)sin(\alpha) + 2a^2\dot{\alpha}\theta cos(\theta)sin(\alpha) + 2a^2\dot{\alpha}^2sin^2(\alpha)] + 3mglcos(\theta) \tag{56}$$

Simplification of equation 56 leads to equation 57:

$$\mathcal{L} = \frac{1}{2}I_1\dot{\theta}^2 + \frac{1}{2}I_2(\dot{\theta} + \dot{\alpha})^2 + Mlgcos(\theta) \tag{57}$$

$M = m + m + m$
$I_1 = Ml^2$
$I_2 = 2ma^2$

Substituting equation 57 into the Euler Lagrange equation and solving for $\ddot{\theta}$ produces:

$$\ddot{\theta} = -\frac{(I_2\ddot{\alpha} + Mglsin(\theta))}{I_1 + I_2} - \gamma\dot{\theta} \tag{58}$$

Where γ is the coefficient of air resistance/friction.

2.10.2 Numerical Integration

Equation 58 was solved numerically using *scipy.integrate.solve-ivp*.

2.10.3 Sinusoidal Torque

The barbell, shown in figure 20, was driven with a sinusoidal torque shown in equation 59.

$$\ddot{\alpha} = -sin(t\omega_0) \tag{59}$$

The resonant frequency of the pendulum is given by $\omega_0 = \sqrt{\frac{g}{l}}$. Figure 21 shows the resulting motion. *Note:* Initially $\dot{\theta} = 0$, $\theta = 0$ and $\gamma = 0$.

Figure 21 shows that the amplification of the pendulum is linear. Also, a maximum value of θ is reached when the driving frequency and natural frequency of the pendulum are perfectly in phase. The reason the phase is not constant is because at larger amplitudes, the natural period and frequency of the pendulum differ significantly from the small angle approximation of a simple pendulum and therefore the driving frequency of ω_0.

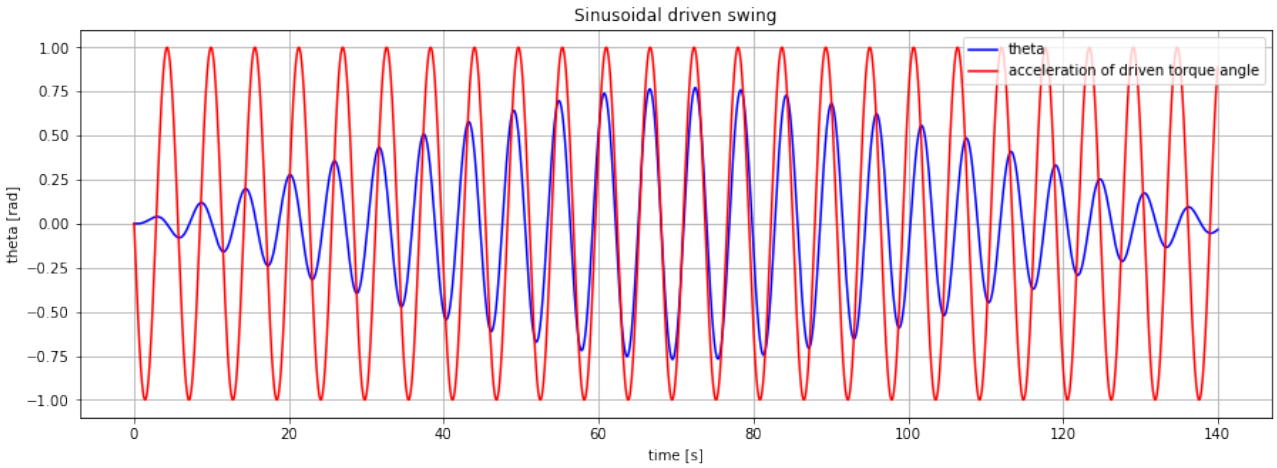


Figure 21: Angle of pendulum against time with overlay of driven torque

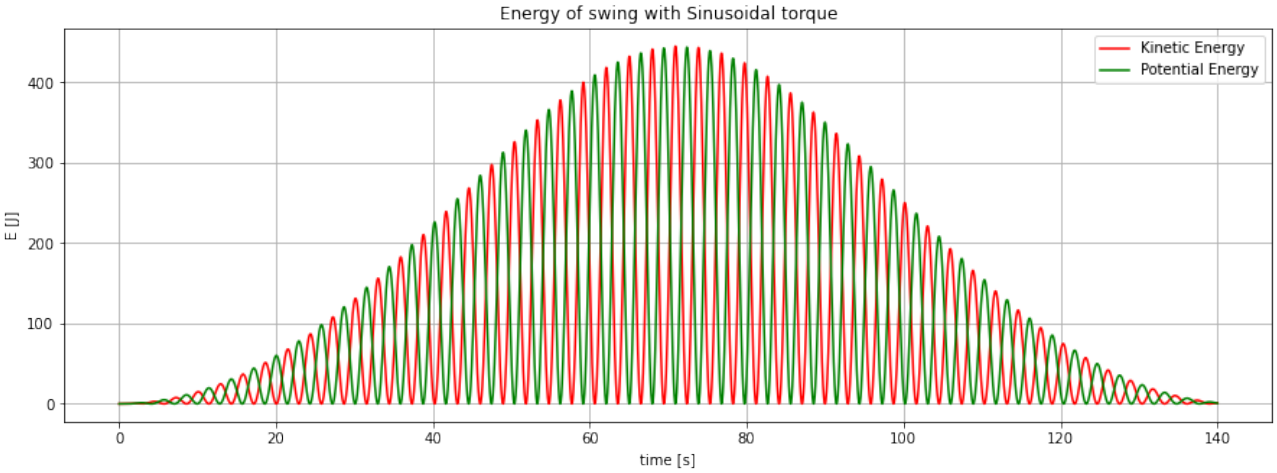


Figure 22: Energy of sine wave driven pendulum

The kinetic and potential energy of the swing was calculated and plotted against time. This is shown in Figure 22. A maximum value for each is reached at the same point that θ reaches its maximum in figure 21. As θ and $\ddot{\alpha}$ move out of phase the energy and amplitude of the system decay back to zero.

2.10.4 Square Wave Torque

Next the barbell was driven with a square wave torque (similar to that used in the standing pump model). This is shown in equation 60 below.

$$\ddot{\alpha} = -Aarctan(sin(t\omega_0)/0.01) \tag{60}$$

In equation 6, A is a normalisation constant, used to constrain the maximum torque to be equal to that of the sine torque. Figure 23 shows the resulting motion. The initial conditions are the same as in figure 21.
Similar conclusions can be made to that of the sine wave torque. The only differences being; a maximum value of θ is reached in a shorter time and a larger maximum amplitude is achieved.

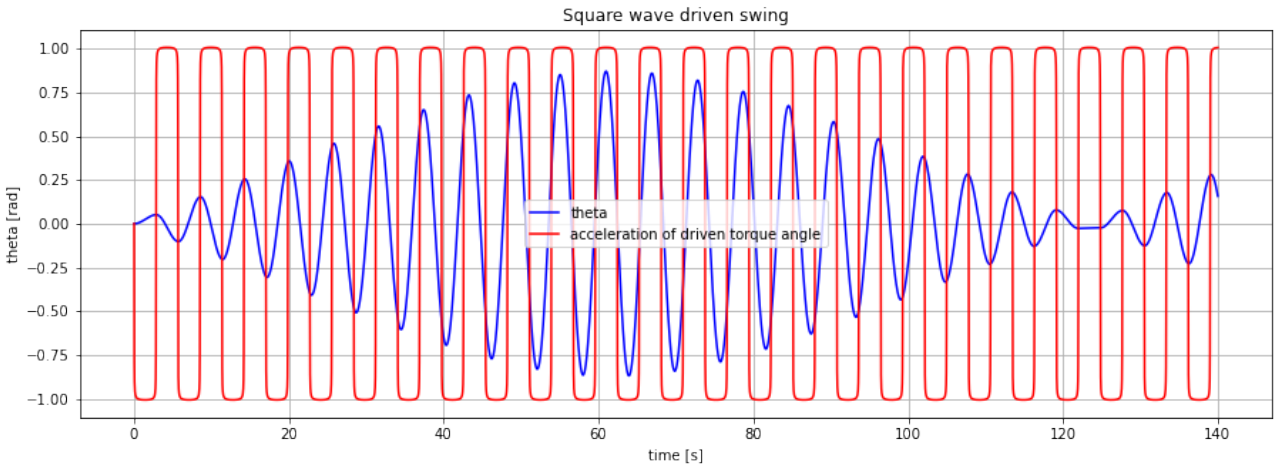


Figure 23: Angle of pendulum against time with overlay of driven torque

2.10.5 Comparison of Driving Torque Functions

Figure 24 below shows how the the total energy of the swing differs when the barbell is driven by a sine torque vs when it is driven by a square wave torque.

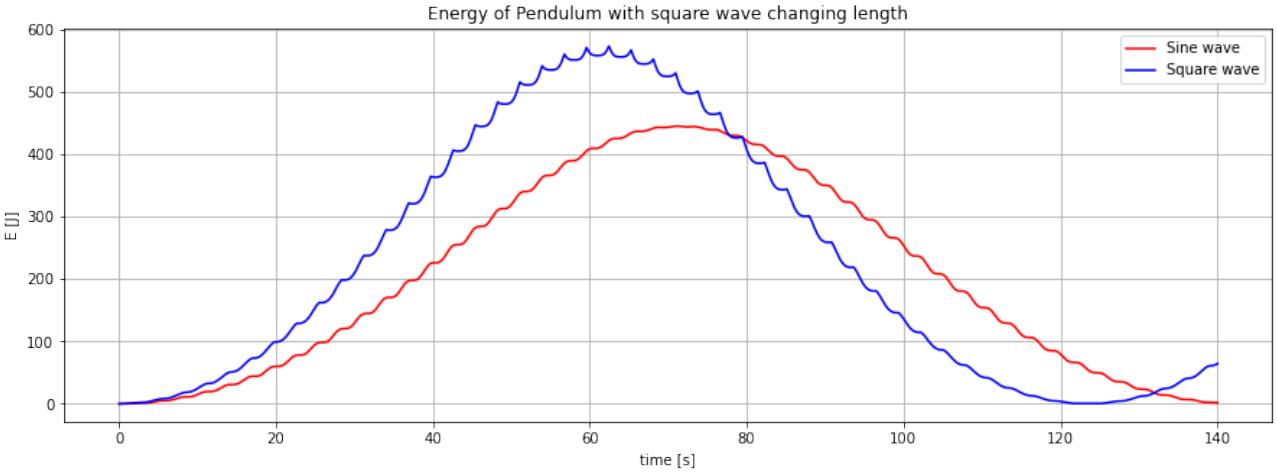


Figure 24: Energy comparison of square wave vs sine wave torque

Figure 24 shows that the square wave torque achieves a larger maximum energy than the sine wave and it achieves this in a shorter time period. However, the sine wave torque increases the energy of the swing more smoothly.

Differentiating the functions in figure 24 w.r.t time, the rate of change of energy of the swing (power) was calculated for each point in time and plotted in figure 25.

Figure 25 shows that the square wave torque has a much larger effect on the rate of energy input into the system, it also demonstrates that the system is very sensitive to the phase difference between θ and $\ddot{\alpha}$, meaning that it is only effective to apply a torque to the system at very specific points in the swing.

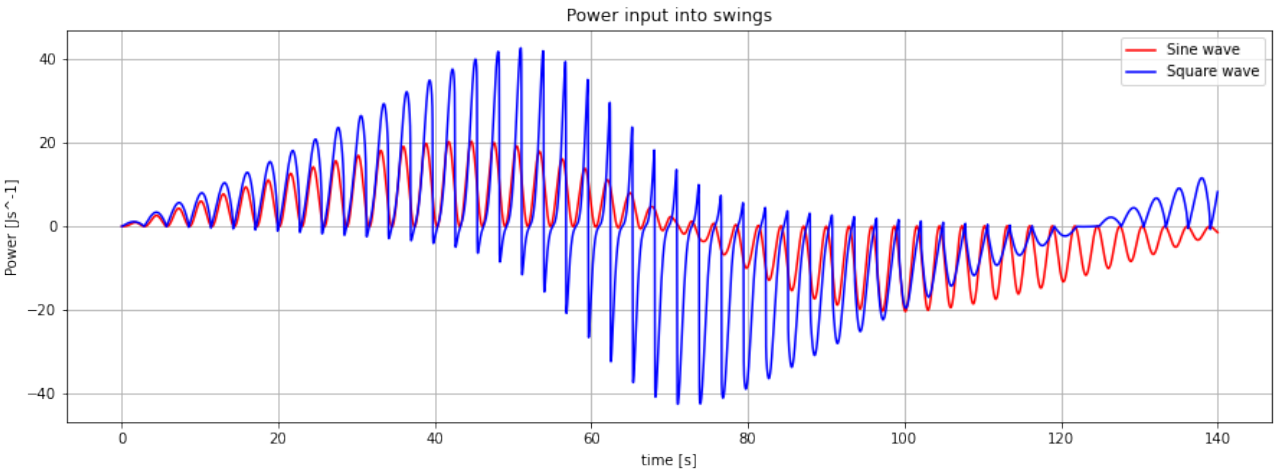


Figure 25: Rate of change of energy of the swing

2.10.6 Conclusions

From the models it is clear that a swing can be driven by a person sitting on it and rotating their body at specific points during the swings motion, and that the point the rotation is made determines the amplitude attained. To maximise the increase in amplitude of the swinging motion, the maximum torque must be applied to the barbell at the point when θ is zero and must be in the opposite direction to the angular momentum of the pendulum at that point. This means that to achieve the maximum impulse from the torque, the driving torque of the barbell must be $\frac{\pi}{2}$ ahead of swing angle θ . This seems to coincide with what is expected, because when you use a swing, you rotate your body with maximum angular velocity at the points of maximum amplitude, therefore the maximum torque is applied at the equilibrium position where you don't rotate, and the torque is in the opposite direction to the previous swing. [14]. However, this is in contrast to the analytical models in this report. They model the rotation as the individual applying an instantaneous torque when $\omega = 0$ and $\vec{L} = 0$, meaning the swing produces an opposite torque to conserve the total $\vec{L}$ of the system. This torque produces a small increase in amplitude. But this numerical model applies the torque at the minimum amplitude ($\vec{L} = \vec{L_{max}}$) acting as a driven harmonic oscillator.

It is interesting that the swing can start with zero initial displacement and velocity and yet someones rotation can still produce motion in the swing. This is contrary to the method of standing pumping which requires some initial $\dot{\theta}$ before you are able pump more energy into the system.

The models suggest that the square wave torque is more effective as it produces a larger maximum amplitude in a shorter period than the sine wave. However, the sine wave motion is much closer the action of a human (so is much more realistic). A human would have a gradual increase in rotation as they reach the maximum points in the swing. They would also be able to change the timing of their rotation, which would keep the constant phase difference between θ and $\ddot{\alpha}$. This is one area where this model is limited and could be improved.

2.11 Seated Model: Effort

Kathryn Holmes

To produce an equation to define the effort required by the swinger to carry out the seated pumping procedure, first the energy required to rotate a limb around a joint was examined. It was decided that for seated pumping, the only contributing motions were the rotation of the

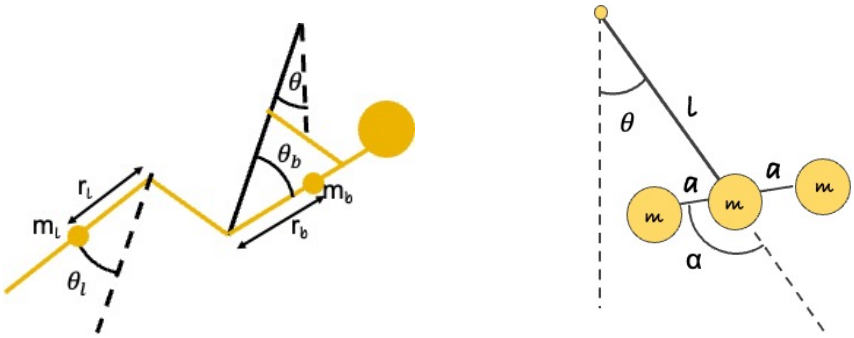
lower leg around the knee and the rotation of the upper body around the hips. From a combination of Newton's Second Law, circular mechanics and the standard equations of constant acceleration, the first term in equation 61 was found. This term appears to be reasonable from a machine learning perspective as the larger the angle through which the limb is rotated and the faster this rotation is made the more effort is exerted and the more harshly this movement is penalised by the machine learning algorithms. Then the work required against gravity was considered. The vertical force required to oppose gravity at any point in the motion was examined and integrated over half a period to give the second term in equation 61. In retrospect, this was not a particularly useful way to define this effort for the machine learning groups as they needed the effort for each movement of the body rather than for each half period, but the 'gravity term' in that case would simply result in an additional term, proportional to the position at a given time, in place of the second term in equation 61. However, the equation found here - despite not being perfect - is not a bad approximation to the effort and provides the necessary constraints for use by the Machine learning group, although to calculate the exact energy that the swinger needs to provide additional factors such as muscle efficiency would need to be included.

The effort, e, was found to be,

$$\epsilon = \sum_{i=l,b} m_i \left(\frac{r_i^2 \Delta\theta_i^2}{\Delta t_i^2} + g r_i \Delta cos(\theta_i') \right), \tag{61}$$

where l, b denotes the legs and body respectively, $\Delta\theta_i$ is the angle moved through in time Δt by body segment i, and m_i and r_i are the mass and half the length respectively of that body segment. θ_i' is the sum of the limb angle θ_i and the total swing position, θ, such that $\Delta cos(\theta_i')$ is the change in the cosine of θ_i' from the beginning to the end of one half period. These definitions can also be seen from the diagram in figure 26a. Applying this back to the original seated model, as shown in figure 26b, the effort function simplifies to:

$$e = m \left(\frac{2a^2 \Delta\alpha^2}{\Delta t^2} + 2ga\Delta cos(\theta) \right) \tag{62}$$



(a) Diagram to show assignment of effort parameters from equation 61, where θ_l and θ_b are measured with respect to the swing.

(b) Diagram of simplified seated model to show effort parameters from equation 62.

Figure 26: Diagrams used in the derivation of the seated effort function.

2.12 Modelling Conclusions

Ben Dawkins

Consider now the aims that were set out at the start of the project, described in section 2.1. Firstly, to create two distinct sets of realistic models that simulated an individual using a swing via two methods - standing and sitting. The models that were used can be seen in figures 4 and 20 respectively . These models (the changing length and barbell models) were simple, yet were also remarkably close to real life - with only a limited number of reasonable assumptions (e.g. small movement of lower leg while squatting, limbs approximated as rods), they mimicked the actual actions fairly well; it would not be incorrect to say that they were realistic, to a degree. The other main aim, to produce an effort parameter for each model, was also achieved. The parameters for the standing model and sitting model are shown in section 2.7 and section 2.11 respectively. Unfortunately, due to a lack of time, a full set of numerical results were not produced from the effort parameters; this is something that would be rectified - should more time have been available - in order to check the validity of the parameters.

The numerical results that were actually produced were limited to those from the Python models - however the figures gleaned from them are relevant, helping to show how the models have improved over time. Looking back to the section that focuses on the numerical standing model (section 2.6), a marked improvement can be seen when the square-wave approximation for the driving force was used instead of the sinusoidal function. In fact, the square-wave function enabled the maximum energy to almost double, from 1080J to 2150J, showing that a large improvement occurred when this element was changed. A similar result is seen in section 2.10, which discusses the numerical sitting model - again, a large improvement was seen when the sinusoidal approximation was changed for the square-wave.

There were other results obtained during this project too, in the form of derived equations. The Lagrange equations and the equations of motion for a number of different types of pendulum were found - the solutions for the 'standing' pendulum are seen in section 2.4, and for the 'sitting' pendulum in section 2.8.

A future group continuing this research would have a number of different avenues to pursue. One such path is to continue work on the Python models that were produced. The sitting code could be developed further by introducing more complexity into the system. For example, hinges could be included to make the model more realistic (to imitate the point where the hands hold onto the rope of the swing) - also making it more similar to the ML2 models in section 4.2.5. Also, in the model the body is assumed to be a barbell, with equal weights at each point along the rod. This does not reflect real life completely, as different parts of the body weigh different amounts - this could be implemented in the code to improve the model. Development could continue on the standing code, too. Although many good results were found from this model, once the hinges were added, the system becomes chaotic fairly easily, meaning some work could be done into fixing this issue - enabling the model to be worked with in more depth. Both models might further be improved by including friction in the joints of the swing, which would again push them even closer to realism.

Another topic that could be explored for further research is additional development and improvement on the effort parameters. As mentioned in sections 2.7 and 2.11, the parameters that were created involve a few assumptions that mean they do not completely resemble real life. For instance, looking at the sitting model, the barbell could be changed to an asymmetric version which would more closely reflect the distribution of mass in the body. Muscle efficiency could also be included in this equation, as this would provide the ML groups with a more complete picture of the physiological effort, and the actual cost of each action.

The standing effort parameter is not perfect either. Even though muscle efficiency and individual limb weights are included, the equation itself is not as convenient as it could be for the ML groups to utilise, as it is more focused on the effort expended across a period of time, rather than describing the metabolic cost. There are elements of each and it could be useful for a future group to try to adjust this equation, and the one for sitting effort, to make them more relevant to the work that was undertaken by the ML groups.

3 Machine Learning 1: Standing Swinger

3.1 Subgroup Introduction

Thomas Thies

The aim of this subgroup was to learn how to drive a swing to high amplitudes standing up with low effort for a human through a machine learning approach. A simple way of pumping energy into the system if swinging motion is already initiated is to stand when the swing is in the bottom position and squat when that amplitude of the swinging is at its greatest - as analysed by the Modelling subgroup. However, the intricacies of when and how to move each limb such that it requires a low amount of physiological effort to the rider are difficult to capture analytically - more so since the swing system we chose has a hinge, making its motion chaotic. That is to say a small change in initial conditions can lead to drastically different developments in the physical system [9].

A machine learning approach of using a genetic algorithm was therefore taken to find out how a rider should move to maximise the swinging amplitude, under the constraint that limb movements shouldn't be too rapid or jerky. This required creating a model in the Pymunk simulation environment for a human on a swing that captured the essential movements. The interest in this approach was also to see if it could counteract the descent of the system into a chaotic motion. The genetic algorithm was successfully implemented on a variable length pendulum model but unfortunately not on the standing model - although a comprehensive plan is provided of how it would be implemented on a standing model. It was also aimed to be an experiment into how machine learning can be used more widely to gain insight into simple human behaviours.

3.2 Description of the Models

3.2.1 The Variable Length Pendulum Model

Before tackling the problem of attaining a humanlike motion on a model of a standing rider, a simpler system was first used to get to grips with machine learning. The chosen system was that of a simple pendulum of variable length, with a rotating disk, shown in Figure 27 - in accordance with the Modelling subgroup. This model captures two important motions related

to the driving of a swing: the extension of the rod which represents the stand-squat motion - important to pump gravitational potential energy into the system - and the spinning motion which captures the leaning of a swinger to get the angular motion started.

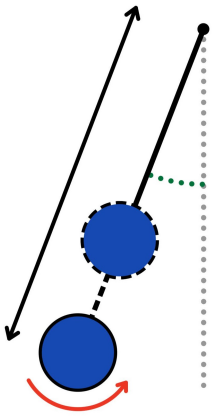


Figure 27: Diagram of the variable length pendulum with rotating disk model. The pivot is at the top, the length of the support rod can be changed, and an actuator controls the circular motion of the disk in blue.

Section 3.4 will go on to explain how this model was created in the Pymunk simulation environment and Section 3.7 how a custom made genetic algorithm taught itself to increase the swinging amplitude of the pendulum.

3.2.2 The Standing Human "Stickman" Model

The aim was to find a model for a standing human on a swing which captures the most important aspects of the motion a human needs to ride a swing. This was condensed to the model shown in Figure 28 with five body parts connected by four actuators (in red) modelling the torque exerted by muscles.

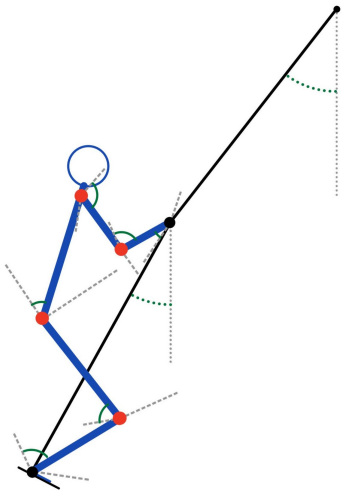


Figure 28: Diagram of the standing human on a hinged swing model. Actuators (in red) are placed at the elbows, shoulders, hips and knees - with the hands and feet free to move. The movement of each of the stickman's joints is constrained to the angular range in the grey lines.

The hands and feet are assumed to be simple joints which can't provide a torque. The angular motion of joints on the body was constrained to realistic limits for each limb shown as grey lines - as discussed in Section 3.5.3, to avoid elbows folding the wrong way for example.

3.2.3 Machine Learning to Avoid Descent Into Chaos

The stickman model has a hinge on the swing making its motion chaotic, in a similar way to the double pendulum system studied by the Modelling subgroup in Section 2.5, and [15] - meaning it is expected to exhibit similar behaviour to the double pendulum. Under small angles, this hinged system has two normal modes of oscillation illustrated in Figure 29 - a symmetric mode where most of the movement comes from the top pivot and an antisymmetric mode where there is much movement in the middle hinge of the swing.

The challenge is to find a way to drive the swing such that only the symmetric modes in Figure 29 are excited so that they contribute to a large swinging amplitude at the pivot, and that the antisymmetric modes are suppressed.

This is where a machine learning approach, as outlined in Section 3.8.2 can play a key role in finding a suitable method for driving the swing by continuously correcting for motions that tend towards a chaotic state or antisymmetric motions in a way that is very taxing to do in closed form.

(a) symmetric mode

(b) antisymmetric mode

Figure 29: Diagram of the modes of oscillation of a double hinged pendulum in small angle approximation to illustrate how (a) symmetric modes are desired since they contribute to a high swinging amplitude, (b) whereas the antisymmetric modes do not so must be avoided.

3.3 Simulation Environment

Javid Ahmed

Pymunk is a 2D physics engine adapted from Chipmunk to work in Python. A space is created in which to work, and different objects such as shapes and bodies are added to this space so they can interact with each other. Different parameters such as the gravity and damping of the space can be defined to change how the bodies behave in the space. Pygame is used to render the Pymunk simulation graphically.

An `App` class was written to run the simulation. The basic parameters of the simulation such as the space and screen size are defined in this class. A method called `run` was written to handle all the events of the simulation. Every time step, this was called and all the bodies and constraints update to behave however they have been defined. Additionally, each time step, the simulation would have to be drawn to the screen in order for the user to see what is going on.

Pymunk has detailed documentation on how to use it along with some worked examples, so this made it easy to learn how to use this engine. Other physics engines such as PyBox2D were considered, but there was not much documentation for these physics engines and thus it would

have been difficult to learn and start using these. 3D physics engines such as PyChrono and PyBullet were also considered, but the motion of our system can be fully described in 2D and there is no additional physical insight to be gained from working in three dimensions, so this would add an unnecessary layer of complexity both in terms of setting up the environment and applying machine learning to it, therefore it was decided to stick to two dimensions.

3.4 Variable Pendulum Simulation

Remi Bahar

This subsection is in reference to the files: `Simulation\variable_pendulum.py` and `Simulation\config.json` in the `Simulation` repository on the group GitHub page [16].

The pivot of the variable pendulum was created using a static body in Pymunk so that it would remain in place without being affected by other forces. The disk of the pendulum was modelled as a circle and the pendulum rod was modelled using the `pymunk.PinJoint` constraint that joined the disk and pivot.

The pivot position, disk position, mass, radius, and gravity were all defined using a *json* configuration file which the Python script read. This meant that it would be easier to change the simulation options when it came to applying machine learning as the *json* file could be changed instead of the code.

Figure 30 shows the initial set-up of the variable pendulum.

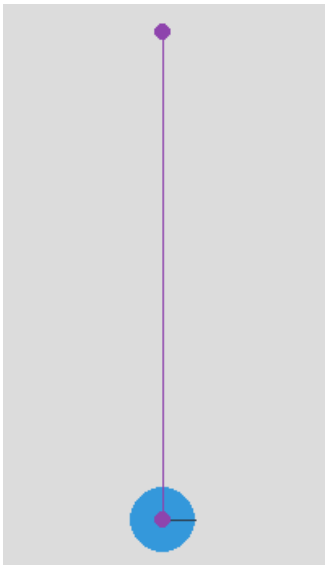


Figure 30: Variable Pendulum Initial Position

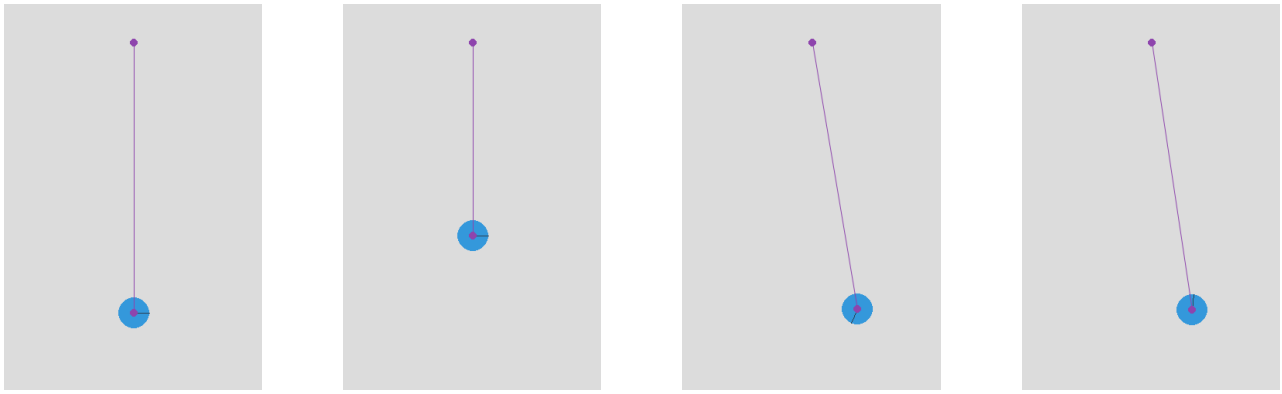
In order to prepare the model for machine learning, controls were created for the user to interact with the simulation using the Pygame module. This allowed the simulation to be tested before applying machine learning and for the machine learning to use the same functions that were used when pressing the arrow keys, for an output.

The down and up keys simulated standing motion by increasing and decreasing the `joint.distance` property to a maximum and minimum value also defined in the configuration file. All the mass of the pendulum was located in the disk so by moving the disk up and down the center of mass was moved as it would for a human swinger during standing pumping albeit in a very simplistic way, this is described in more detail by Modelling in section 2.3.

The right and left keys applied a force to the top of the disk directly above its center of mass acting in the right and left direction respectively using the `body.apply_impulse_at_local_point` method. This exerted a torque and therefore angular acceleration on the disk which modelled swinging pumping which in a simple form consists of applying a torque toward the equilibrium position at the maximum amplitude of the swing.

The spinning disk also allowed self-start of the variable pendulum as in order to start the pendulum in a swinging motion from its equilibrium position a torque had to be applied to it to achieve angular acceleration. Standing pumping provided no torque so could only be used as a pumping method when the pendulum was already in motion, this is described in more detail by Modelling in section 4.6.

Figure 31 shows the standing and sitting pumping of the variable pendulum.



(a) Variable Pendulum after being lengthened by pressing the down arrow

(b) Variable Pendulum after being shortened by pressing the up arrow

(c) Variable pendulum after a clockwise torque was applied using the right arrow

(d) Variable pendulum after a anti-clockwise torque was applied using the left arrow

Figure 31: Variable Pendulum after standing and sitting pumping

The final way of interacting with the variable pendulum was using the space key to stop the spinning motion of the variable pendulum by setting `body.angular_velocity` to 0. This was added because one method the machine learning could use is to not perform any action so as not to incur a cost in effort.

3.5 Standing Stickman Simulation

Javid Ahmed

This subsection is in reference to the files: `Simulation\Standing\stickman.py` and `Simulation\Standing\config_standsquat.json` in the `Simulation` repository on the group GitHub page [16].

In the stickman simulation, a swing was created and a stickman model was added to this swing. The stickman was initialised in a standing position, and has the ability to squat down and stand up again in order to pump the swing.

A configuration *json* file was created where all the information about the stickman and swing was stored, as well as some environment factors such as the strength of gravity.

3.5.1 Creation of the Swing

The swing, shown in figure 32, was composed of two rods, created using `pymunk.Segment` objects. These segments are created by defining a position where its origin should be, and a

direction vector using polar coordinates is defined to determine the length of the segment and the angle at which it is created. The equation for this is shown in equation 63,

$$\vec{d} = \begin{bmatrix} l * \cos\theta_{swing} \\ l * \sin\theta_{swing} \end{bmatrix}, \tag{63}$$

where l is the length of the segment and θ_{swing} is

$$\theta_{swing} = \theta_{system} + 90. \tag{64}$$

If $\theta_{swing} = 0$ then the swing hangs vertically. Increasing this angle rotates the swing clockwise.

The rods swing from a top pivot point, so the origin of the first segment is the position of the top pivot point. The position of the second segment is equal to the position of the first segment plus the direction vector of the first segment so that the origin of the second segment is at the end of the first segment. The swing was defined such that the two segments are inclined at the same angle to begin with.

The segments were connected using `pymunk.PivotJoint` constraints which creates a pivot point at the point of connection, allowing the segments to freely rotate around that point.

In this state, the swing is effectively a double pendulum system. The motion of a double pendulum system is inherently chaotic, so this can make it more difficult for the stickman to learn how to swing. The two segments can be constrained to have the same angle by using a `pymunk.SimpleMotor` constraint (explained in more detail section 3.5.2), which would make the system a single pendulum system i.e. a rod swing. Thus, the only factor contributing to the overall motion of the system would be the stickman standing and squatting, changing the centre of gravity of the system along the rod.

The motion of the swing is damped due to friction in the top hinge. This had to be coded in by applying the force on the top segment. The direction of the force is in the opposite direction of the segment's velocity in order to decelerate the segment. The magnitude of this force is proportional to the segment's angular velocity. This was implemented by first fetching the velocity of the segment using `body.velocity` which fetches the velocity of the segment at its centre of mass i.e. halfway along the segment. Then, this was converted to an angular velocity using

$$\omega = \frac{v}{r}, \tag{65}$$

where v is the velocity of the segment and r is the distance along the segment, so half the length of the segment. The force is then calculated using

$$f = -\omega * f_{coeff}, \tag{66}$$

where f_{coeff} is the coefficient of friction. This takes into account the direction of the force to be applied as well as the magnitude. This force was applied halfway along the segment. The variables used in the Pymunk simulation were not normalised, so for any noticeable effect, f_{coeff} had to be very large. This is explained further in section 3.5.5.

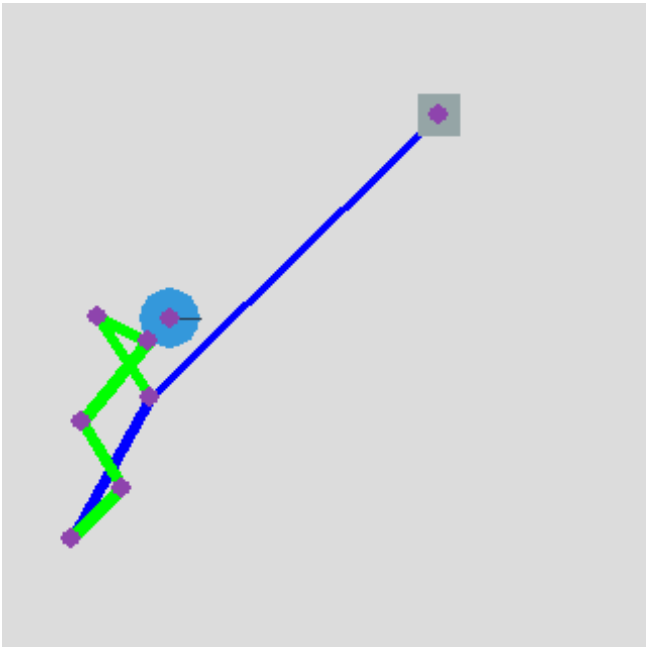


Figure 32: The stickman on the swing.

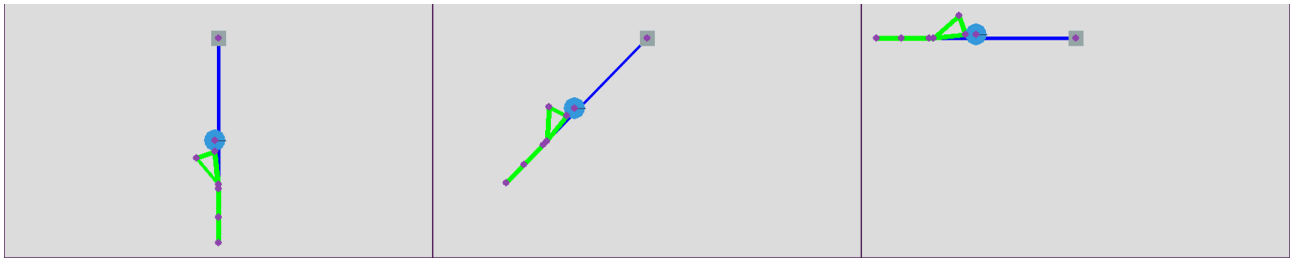


Figure 33: Setting θ_{swing} to 0, 45, and 90 respectively.

3.5.2 Creation of the Stickman

A class called `Stickman` was written to store all the bodies and constraints regarding the stickman and swing, the configuration file is passed through as an argument. A member of this class is passed through the `App` class with the name `stickFigure`. The swing is generated in a method called `Stickman.generateSwing`, and the stickman is generated in a method called `Stickman.generateStickman`.

The stickman was created in a similar way to the swing. Figure 34 shows how the stickman was created. First, a foot position is defined to be the position of the second swing segment plus its direction vector so the foot is at the end of the segment. The lower leg is created using a `pymunk.Segment` object from this foot position in the same way as the first swing segment is created using the top hinge position. Each subsequent limb is created at the end of the previous limb. The direction vector is calculated in the same way as shown in equation 63, except l is now the length of the limb, and θ_{swing} is replaced by θ_{system}. The effect of changing this angle is shown in figure 33. The stickman always shares the same rotation as the swing. All of the limbs are connected using `pymunk.PivotJoint` constraints. The stickman's hands and feet were connected to the end of the first and second swing Segment respectively using `pymunk.PinJoint` constraints to simulate the stickman standing on the swing and holding onto it.

Table 1 shows the masses and lengths for each of the stickman's limbs. The relative lengths were measured by simply measuring my own limbs which was sufficient as an error of a few

Limb	Relative Mass	Relative Length
Lower leg	6.43	44
Upper leg	11.13	48
Torso	46.02	65
Upper arm	3.08	35
Lower arm	2.30	30
Head	8.23	5 (Radius)

Table 1: Masses and lengths of each limb.

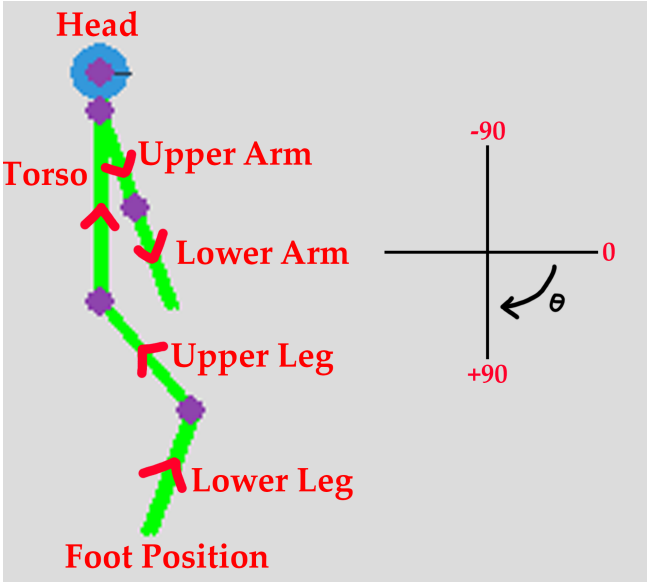


Figure 34: The creation of the stickman using position and direction vectors.

centimetres would not impact the system. The relative masses are the average proportions for a human. [17]

Initially, in order to get some of the stickman's limbs to not go through each other such as the upper and lower leg, the segments utilised collisions. However, these collisions ended up causing problems in the simulation since the limbs would collide at the points at which they are connected, generating unwanted forces so there would be additional motion contributing to the overall system which should not be there. The stickman would also collide with the swing when in reality, the stickman should be in line with the swing, but Pymunk is a 2D engine so it cannot work with three dimensions and thus collisions between the stickman and swing also had to be turned off. This was done by setting the collision group for all of the segments to be the same. The new constraints of the stickman to stop it from having unphysical angles between limbs are outlined in the next section.

Additionally, the stickman initially would fall and there would be no rigidity at all between the limbs. To solve this, at each joint, a `pymunk.SimpleMotor` constraint was added. Setting the motor's rate to 0 would fix the angle between two limbs, thus adding rigidity to the stickman's body by utilising the max_force property. The max_force property corresponds to the maximum force the constraint can apply to the two bodies to which it is connected to keep the angle between them constant if the motor is off. A high maximum force fixes the angle between two limbs, whereas a low maximum force allows for free rotation about the pivot point. This also allowed for the stickman to be able to move each limb independently by setting the rate for the corresponding joint motor to some speed defined in the configuration file. A `Stickman.moveLimb` method was defined which would allow for each motor corresponding to

the desired limb could be activated to move the limb provided. A `Stickman.stayStill` method was also defined which sets the rates of each of the motors to 0 which could be called to turn off all the motors if the stickman should stop moving.

The `pymunk.SimpleMotor` constraints were initially constrained to a static body b_0 which was created for convenience to utilise the constraints more easily. However, this caused the stickman's joints to have a fixed angle relative to the world space instead of relative to the rest of his limbs. This meant that the stickman would have a fixed angle regardless of where the swing was in its rotation i.e. the stickman could have the angle shown in the second picture of figure 33 when the swing was at $\theta = \pm 45$, whereas the stickman should rotate with the swing to be realistic. This also led to the issue that the stickman's centre of gravity would be offset from directly below the pivot point due to the constraints applying forces to keep the angles constant. This was then changed so that the constraints were between two limbs, so now the stickman will rotate with the swing, which also solved the centre of gravity problem. In doing this, the constraints began moving independently which is what the intended mechanic was to begin with, but the movement functions (to stand and squat) must be altered such that every joint involved is moved manually, as opposed to other joints responding automatically.

A dictionary called `Stickman.joints` kept track of all the limbs and their corresponding joints, joint motors, and maximum angles for flexion and extension so that it was easier to work with these different quantities and objects. However, this could be refactored potentially as a class to make it more object-oriented.

3.5.3 Constraining Stickman Angles

Remi Bahar

Building upon the use of motors to control the motion of the stickman, constraints were added so that the joints of the stickman were forced to remain within a certain angular range. The aim of this was to ensure the stickman's motion resembled human-like motion.Table 2 shows the joint angle constraints that were decided upon based on research [18].

Table 2: Joint Angle Constraints, Green - constraints added to simulation, Amber - constraints left to be added, Gray - constraints not added because the stickman did not have these joints

Joint	Flexion Angle (degrees)	Extension Angle (degrees)
Knee	130	15
Lumbar Spine (called pelvis in our simulation)	75	30
Shoulder	180	60
Elbow	150	180
Neck	70-90	55
Wrist	80-90	70
Hip	110-130	30
Ankle	45	20

The joints were constrained using the `Stickman.applyConstraints` method which was called every iteration of the simulation. The idea was to set the rate of the motor to 0 when a constraint was breached, thus locking the motor and preventing it from continuing to move in that direction.

The execution of this proved to be more complicated. The built-in `body.angle` method calculated the angle of a limb relative to its initial position instead of a universal direction so was

not suitable and there was no built-in way to calculate the joint angle. Instead we used the following vector equation to calculate the angle, θ, using the pelvis as an example,

$$cos(\theta) = \frac{v_1 \cdot v_2}{\mid v_1 \mid\mid v_2 \mid} \tag{67}$$

where v_1 is the upper leg vector, v_2 is the torso vector. The limb vectors were calculated by subtracting the position of the limb body from the position of the next limb's body, i.e.

$$v_1 = \texttt{self.torso.body.position} - \texttt{self.upper\_leg.body.position} \tag{68}$$

We defined the joint angle so that it was 0 if the two limbs were straight, negative if the upper limb was rotated anticlockwise in this case if the pelvis was undergoing extension, and positive if the upper limb was rotated clockwise, in this case if the pelvis was undergoing flexion. Figure 35 illustrates the logic used to decide when to lock the pelvis motor, both the constraints for the pelvis joint and target angle (if applicable) for the pelvis joint are used.

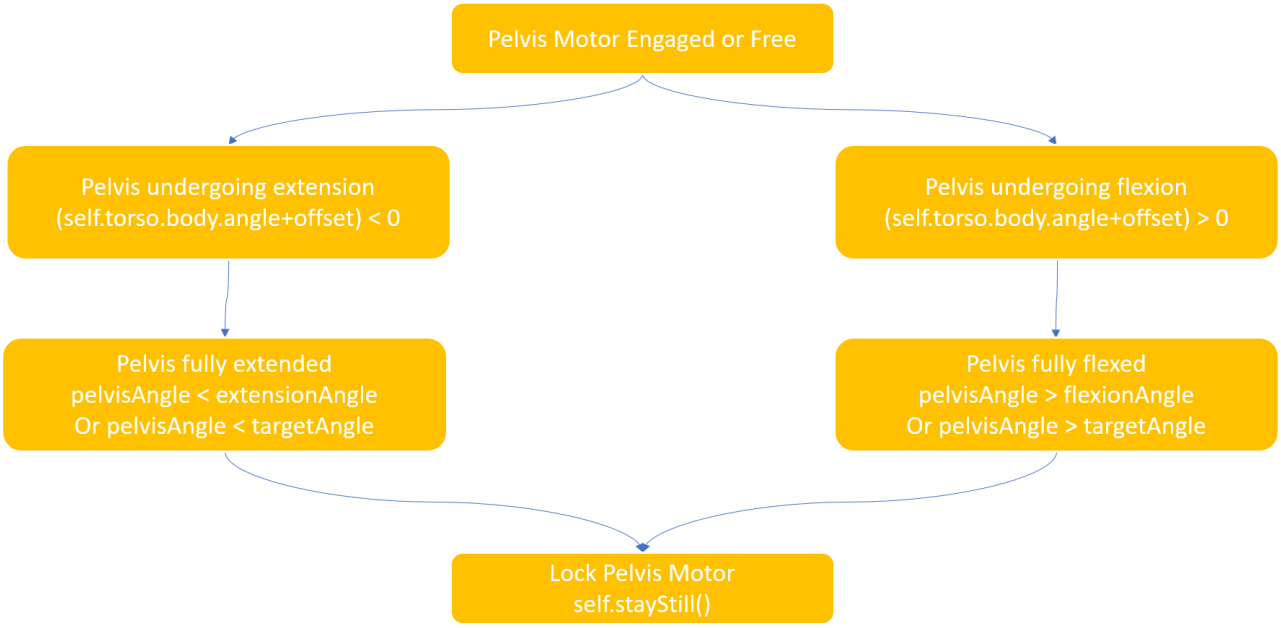


Figure 35: Flowchart for pelvis angular constraints

It was assumed that the front of the stickman was facing to the right in the simulation as shown in figure 34.

3.5.4 Driving the Stickman without Machine Learning

Javid Ahmed

Before applying machine learning to the stickman model to teach the stickman how to swing, a method called `Stickman.makeDecision` was defined to tell the stickman when to stand and when to squat. Currently, the stickman will stand when the swing is completely vertical, and squat when he approaches the maximum amplitude he previously reached.

A list called `Stickman.maxAngles` is used. The initial maximum amplitude, which is the first item in the list, is set to be whatever angle the swing is at when the simulation starts.

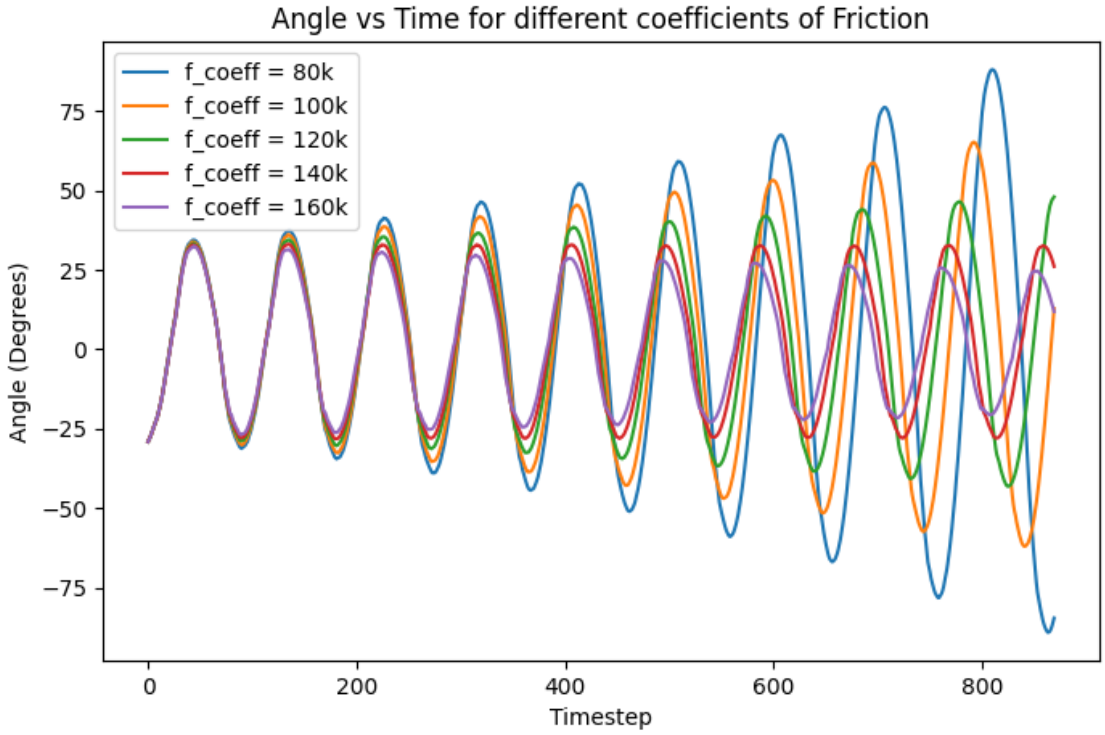


Figure 36: A graph to show the angle of the swing as a function of time, and how this is affected by changing the coefficient of friction.

The stickman squats, and when the swing becomes vertical, the stickman will stand. Then, when he approaches said maximum amplitude, he will squat, and when the swing's amplitude becomes larger than the maximum amplitude previously and its angular velocity approaches 0 (i.e. the maximum amplitude of the swing is being approached before it will start to decrease), the amplitude of the swing will be appended to the list. This process then repeats when the swing travels in the opposite direction, so that each subsequent maximum amplitude is increasing.

If the stickman does not get close enough to the maximum amplitude as he is swinging, he is set to stop his motion and will freely swing without squatting until the swing comes to a stop, because the current algorithm has not been refined to pump the swing at the perfect times, since this is what the machine learning is for. This method allowed for the graphs in the following sections to be obtained so the motion can be analysed.

3.5.5 Swing Angle as a Function of Time

As mentioned in section 3.5.1, friction was applied to the top segment of the swing to dampen the motion. Figure 36 shows how changing f_{coeff} affected the oscillation of the system. As expected, it can be seen that a smaller coefficient of friction allows the swing to reach a larger amplitude. This amplitude does eventually level out to become constant. If the coefficient is large enough, then it can be seen that the swing does not increase in amplitude and the stickman would simply maintain the amplitude of the swing's oscillation without increasing or decreasing which happens at around $f_{coeff} = 160000$. For larger coefficients, the stickman would eventually stop squatting as it would not reach a sufficient amplitude to meet the requirement to squat again.

3.5.6 System Energy as a Function of Time

Remi Bahar

The total energy of the system was calculated using the `Stickman.calculateTotalEnergy` method which was run for each iteration of the simulation. This method looped through a dictionary of swing segments and stickman limbs. For each body, the kinetic energy was calculated simply using Pymunk's built-in `body.kinetic_energy` method. The gravitational potential energy was calculated using equation 9. The mass, m, of the body and the graviational field strength, g, could be simply obtained from `body.mass` and `space.gravity[1]` respectively. The height was taken as the height of the center of gravity of the body above the ground which was defined as the bottom of the screen, so

$$\begin{aligned} \texttt{h} = \texttt{self.config["environmentConfig"]["screenSize"][1]} - \\ (\texttt{self.body.position[1]} + \texttt{self.body.center\_of\_gravity[1]}) \end{aligned} \tag{69}$$

where the second index [1] is used for the vertical component. The following graph was obtained after running the simulation for some time with the automated stickman pumping as described in section 3.5.4. The friction co-efficient was set to 1.2×10^5 and the initial angle of the swing to 20°.

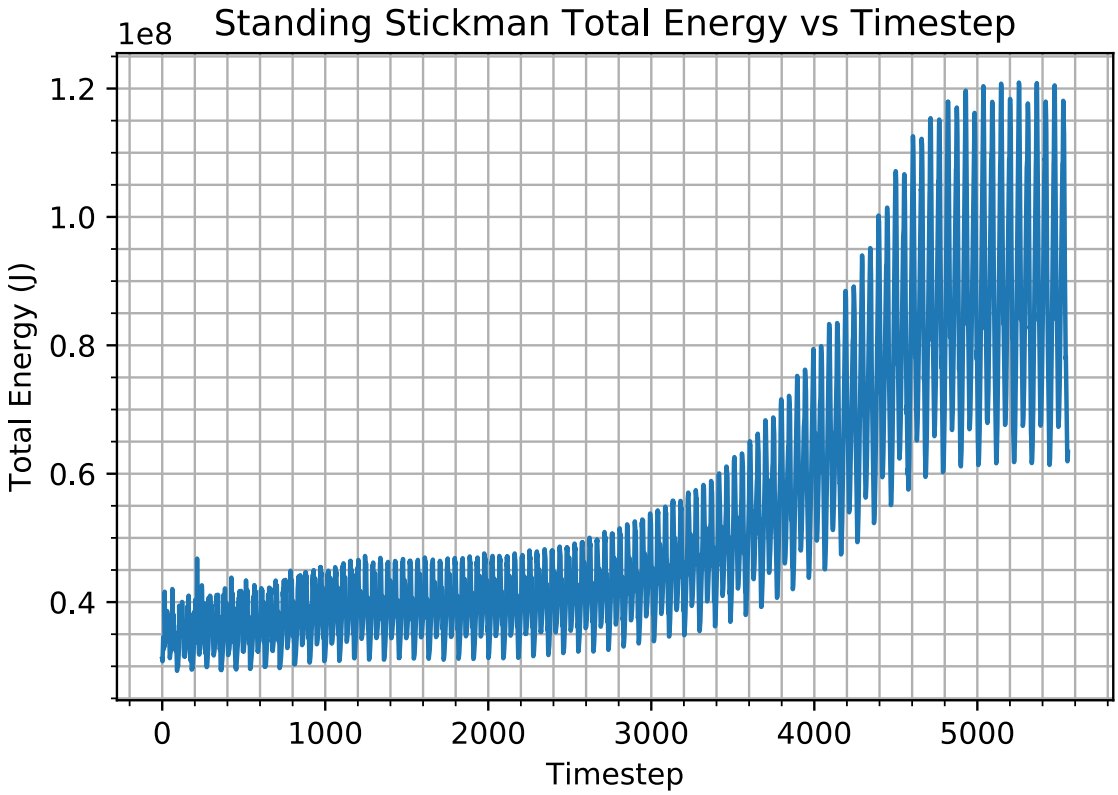


Figure 37: Total Energy against Timestep for a Standing Stickman

As seen by figure 37, the standing motion of the stickman pumped energy into the system as seen by the peaks of the total energy increasing.The total energy increased fairly slowly at first then more quickly before reaching a maximum energy of around 120 MJ with small fluctuations. This fits in with what was observed in general by Modelling in figure 15b.

3.5.7 Getting the Simulation Ready for Machine Learning

Remi Bahar

The final step in getting our simulation ready for machine learning involved implementing some flexibility in the extent to which the simulation would force the stickman to have realistic motion versus the extent to which the machine learning would implement this by including terms in its effort function that discourage unrealistic motion. Giving the machine learning more flexibility would give it more scope to investigate human-like motion and reduce the complexity needed for the simulation but it would make the task of the machine learning more difficult.

Since it was not clear where in this spectrum to choose, we gave the user the option to set their own balance between these contrasting approaches by adding the following three variables to the configuration file.

1. `ragdollEnabled`
    - When set to `true` the motors in the stickman would initially be disengaged allowing him to fall similarly to a ragdoll and hang freely under the swing. This would give the machine learning scope to investigate how to make the stickman stand up as well as pump the swing
    - When set to `false` the motors in the stickman would be locked causing the stickman to stand-up and remain in his initial position
2. `constrainJointAngles`
    - When set to `true` the joints in the stickman would be constrained to not allow the machine learning to bend any limbs into unnatural positions
    - When set to `false` the joints in the stickman would be free to move to any angle, giving the machine learning scope to encourage realistic limb positions, perhaps by punishing unrealistic positions using the effort function
3. `motorUsingTargetAngle`
    - When set to `true` the motors would take an angle as an output from the machine learning and drive the joint to that angle during pumping
    - When set to `false`, the motors would just take a motor rate as an output from the machine learning to drive pumping

3.6 Genetic Algorithm

Kieron Dodge

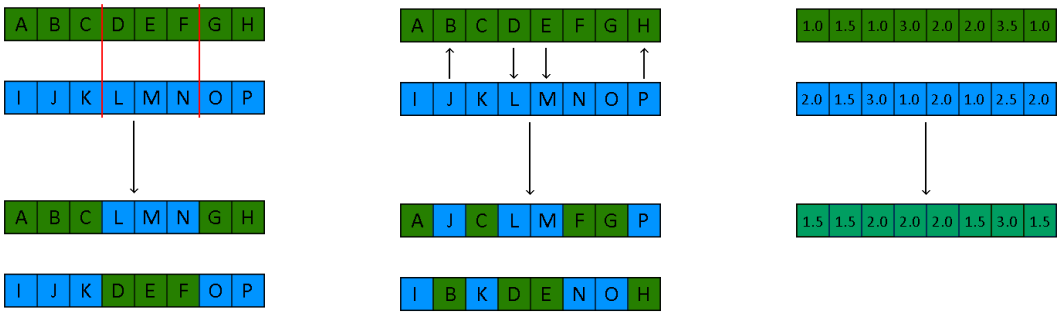
This subsection is in reference to the `Simulation\genetic` repository on the group GitHub page [16].

3.6.1 Theory

Genetic algorithms are inspired by the concept of natural selection and evolution and involve evolving a population of solutions ('individuals') to find progressively better solutions to an optimisation problem. Individuals possess chromosomes consisting of genes, which are passed to new individuals during the evolutionary process and slightly altered in the hope of finding better chromosomes. A simple analogy to a chromosome is that of a byte, where the byte represents the whole chromosome and the 8 bits represent the genes. Any reordering or flipping of the bits represents a unique chromosome.

Implementation can be achieved by initialising a population of individuals with, typically random, chromosomes. The best chromosomes are found by evaluating a 'fitness function' which is unique to every problem. Some chromosomes are selected to 'reproduce' with the probability of selection proportional to a chromosome's fitness. The proportion of the population selected for reproduction is problem-dependent. However, one must take care to ensure the pool of selected individuals is small enough such that a large number of poor individuals does not skew the probability of selecting the best chromosomes but large enough that there is some genetic variation. Genetic variation is essential to prevent early convergence to a less optimal solution (Section 3.6.2.

Reproduction is simulated using a crossover operator, which swaps genes between two parent chromosomes to produce one or more new child chromosomes. Examples of crossover operators are k-point crossover, uniform crossover and average crossover. K-point crossover defines k crossover points at some genes, after which genes are swapped to create two new chromosomes. Uniform crossover loops through the chromosome and has a predefined probability of a gene being swapped. Average crossover can be applied to genes represented as floats and calculates the mean value from two parent genes.[19] These methods are described diagrammatically in Figure 38.



(a) K-point Crossover (k=2) (b) Uniform Crossover (c) Average Crossover

Figure 38: Diagrammatic representation of crossover operators.

Following crossover, chromosomes may not show significant amounts of genetic variation. Mutations may be applied to genes to discourage this.[20] An example of mutation is random mutation, where a gene may be changed to a random value in some predefined range. One may also choose to perturb a gene by some percentage of its current value.[21]

The algorithm terminates once some criteria are reached. Commonly, the algorithm terminates after a set number of time/generations or once a minimum fitness has been achieved.

The expectation is that every generation of individuals should perform better than the last and eventually converge on a solution. Convergence to a more optimal solution can be encouraged through the use of elitist selection. This process demands that the best individual in a generation is inserted into the following generation unaltered, preventing the best solution from getting worse with time.[22]

Some algorithms introduce more complex changes to what has already been described. Stanley & Miikkulainen's NEAT algorithm describes a 'genome', a list of 'node genes' (specifying possible input, output and hidden node connections) connected by 'connection genes' (specifying the connected nodes, the weight and whether or not the connection is active).

NEAT allows mutations to both the weight on a connection gene and the network structure, allowing connections between nodes to be randomly created/destroyed and further allowing the creation of new nodes (Figure 39). The latter mutation extends the genome. Ordering of the new, extended genome is aided by what Stanley & Miikkulainen describe as an 'innovation number', describing 'a chronology of the appearance of every gene in the system'. This process removes the need to predetermine a neural network structure as the algorithm will seek the best topology. All neural networks are initialised with all inputs completely connected to the outputs with no hidden layers.

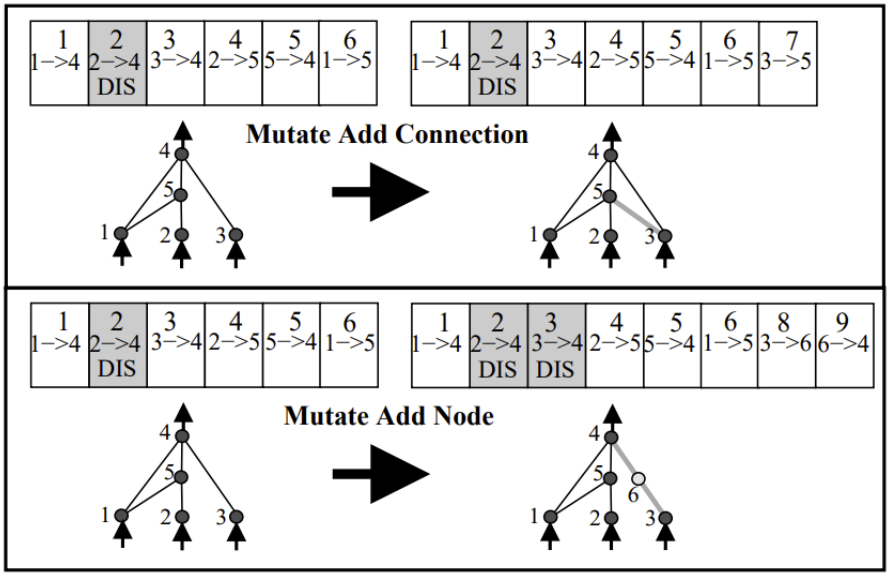


Figure 39: Diagrammatic description of NEAT mutations, lifted directly from Figure 3 of Stanley & Miikulainen's paper. Genes are structured to show, from top to bottom, the innovation number, the node connection and whether the gene is active or not (DIS representing disabled). When a mutation adding a new node that interrupts the link between two existing nodes occurs, the genome is extended and the gene previously linking two nodes is disabled.

Stanley & Miikkulainen note that it is faster to optimise a smaller network than a larger one, and structural mutations usually temporarily reduce the fitness of offspring, typically eliminating offspring with distinct structures. This problem is resolved by introducing speciation, where individuals with similar structures compete against each other and not the whole population.

NEAT was applied by Stanley & Miikulainen to the double pole balancing problem[23] with velocity inputs as a benchmark and compared to other algorithms. The closest compared algorithm to that described in Section 3.6.1 is that of 'Evolutionary Programming' (EP).[24] NEAT demonstrated a clear performance increase, obtaining a solution to the double pole balancing problem in 24 generations, compared to 150 generations when one applies the EP algorithm.[25] Implementation of NEAT is, therefore, a key goal for this project.

3.6.2 Theoretical Limitations

Defining a fitness function can be a time-consuming process. Care must be taken as a poorly defined fitness function can be detrimental to the evolution of solutions, preventing convergence, and may take some time to show in the data. In the extreme case, a poorly defined fitness function may even lead to an incorrect solution. Fitness functions can become much more complicated than anything detailed in this project, in which case it can become computationally expensive to evaluate.

Individuals in a genetic algorithm compete with each other, meaning any reduction in fitness is seen as detrimental by the algorithm, even if a mutation would be better long term. As a result, genetic algorithms tend to converge at some fitness level. This behaviour is more common in smaller problems and is observed in Section 3.7. Mitigations exist and primarily involve maintaining genetic diversity. However, there is no general solution to this problem[26], so one must take further care in defining a fitness function.[27]

3.6.3 Implementation

We have implemented a genetic algorithm in this project by initialising a population of neural networks (the chromosomes) with random weights connecting the nodes (the genes). The neural network and genetic algorithm have been written exclusively in Python from scratch with inspiration from Le Fourner's YouTube series and GitHub repository.[28] This approach was taken to allow complete control over how the algorithm is implemented.

Genes are encoded by storing weights in a list of NumPy[29] arrays, where the length of the list is the number of layers. The n-th element represents a list of length equal to the number of nodes in the n-th layer. This list's m-th element is a NumPy array of length equal to the number of nodes in the (n+1)-th layer. The elements of the NumPy array represent the weight between nodes. For example, recalling that Python begins counting from 0 and not 1, the element [0][2][0] represents the weight on the connection between the 3rd node on the 1st layer and the 1st node on the 2nd layer.

The neural networks in this program have a predefined structure consisting of an input layer, an output layer and a user-specified number of fully-connected hidden layers. Output is requested once per time step. After a fixed number of time steps, the current generation of individuals terminates and reproduction occurs. Reproduction is performed through a user-defined crossover, followed by random perturbations of neural network weights. Termination of the algorithm is left to the user but occurs, by default, upon completion of the 30th generation.

We explored multiple approaches to reproduction and introducing genetic variation. These approaches were implemented on the variable pendulum model (Section 3.4) with results shown in Section 3.7. This investigation was carried out parallel to creating a realistic model to determine the most suitable parameters for the algorithm. We explored the following:

- Insertion of random, non-inheriting individuals (up to 25% of the population)
- Elitist selection
- Average/Uniform Crossover

We allowed the configuration file to have an option for modifying machine learning parameters. This allowed multiple distinct simulations to be run in parallel and was useful for data collection. Parameters in the configuration file are described below.

1. `populationSize` (int)
    - the number of individuals per generation
2. `hiddenLayers` (list)
    - the structure of hidden layers; [number of layers, number of nodes per layer]
3. `randomEnabled` (bool)
    - flag to enable/disable inserting random individuals into a generation

4. `numSteps` (int)
    - the number of time steps per generation
5. `mutationRate` $\in$ `[0, 1]`
    - the probability of perturbing a weight
6. `selectionPercentile` $\in$ `(0, 1]`
    - the proportion of individuals selected for reproduction
7. `crossoverMethod` $\in$ `[uniform, average, none]`
    - the crossover method used
8. `elitistEnabled (bool)`
    - flag to enable/disable elitist selection

3.6.4 Limitations of Implementation

Implementing a genetic algorithm in this project is limited by the lack of more advanced features, specifically structural mutations. Between the two machine learning subgroups, there were significant difficulties trying to set up the NEAT-Python package.[30] In the interest of time, implementation of the NEAT algorithm was deprioritised as an extension activity in favour of obtaining results from the described implementation. Given more time, the prescription of Stanley & Miikkulainen would have been followed and added to the custom code.

There was no method of terminating a generation early when appropriate (e.g. if fitness has not changed for some period of time). Consequently, simulations can run for much longer than is necessary, which meant we were unable to collect as much data as initially hoped.

3.7 Application of Machine Learning to the Variable Pendulum Model

The described genetic algorithm was applied to the variable pendulum model with no rotating disc. To determine a fitness function, we referred back to the subgroup goal - to achieve efficient swinging motion in a way that is not effortful for a human. One may formulate this as an optimisation problem, suitable for a genetic algorithm, by demanding that the swinging angle is maximised and that the effort is minimised.

The neural network associated with a pendulum accepts the angular position, angular velocity, angular acceleration and maximum height as inputs and a command to extend up or down as outputs. Maximum height is given as an input to give the system more information about where the equilibrium position is located. There are two hidden layers with 25 nodes. The structure of the neural network was arbitrary and it may be beneficial for future work to develop this fixed structure.

The variable pendulum with no rotating disk cannot "self-start" and must be initialised at some amplitude, θ_{init}, measured in degrees. The angle is only measured in degrees in the determination of the fitness function as this amplifies the probability of selection and is measured in radians for all other purposes. A proxy for the effort was defined as the number of times the extension of the swing changes (the number of 'pumps'), n. The maximum amplitude achieved by the model is labelled θ_{max} and is only rewarded up to angles of 90 degrees. These metrics

provide enough information to define a simple fitness function,

$$F = \begin{cases} \dfrac{\theta_{max} - \theta_{init}}{n}, & n > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{70}$$

We observed premature convergence through use of this function and some simulations failed to attain $\theta_{max} > 90$. Through trial and error, we determined a new fitness function which placed different weightings on the amplitude and the effort,

$$F = \begin{cases} \dfrac{(\theta_{max} - \theta_{init})^2}{\sqrt{n}}, & n > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{71}$$

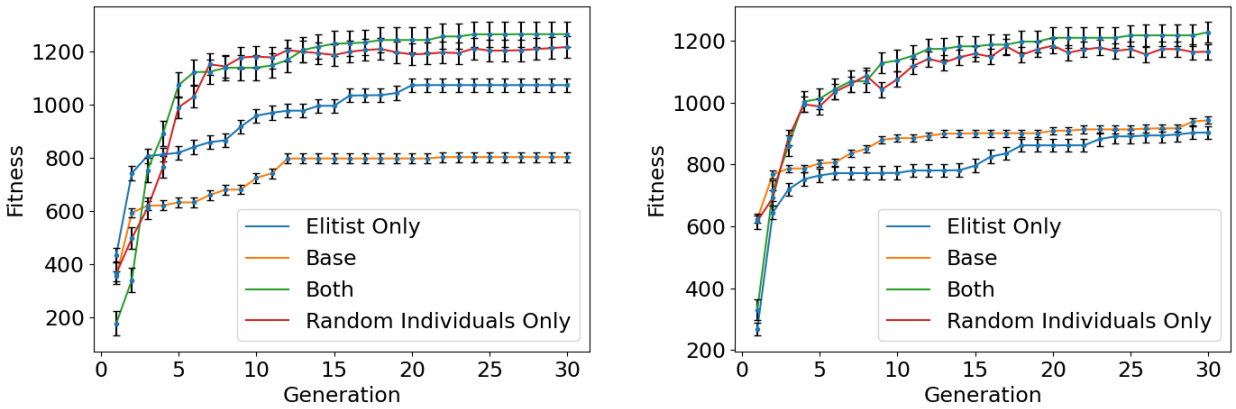
3.7.1 Determination of Machine Learning Parameters

As touched upon in Section 3.6.3, we used the variable pendulum model to explore different genetic algorithm parameters. The configuration file parameters associated with this investigation are detailed in Table 3.

Parameter	**Value**
randomEnabled*	true, false
elitistEnabled*	true, false
crossoverMethod*	uniform, average
populationSize	200
hiddenLayers	[2, 25]
numSteps	600
mutationRate	0.005
selectionPercentile	0.05

Table 3: Configuration parameter values. Starred parameters represent those changed during the investigation.

We performed nine simulations (the maximum number of simultaneous simulations) over 30 generations for every permutation of Table 3 and retrieved the fitness of the best individual every generation. The average fitness of the population was ignored as we are only interested in the best swinging motion. The results of this investigation are shown in Figure 40 and Table 4.



(a) Average Crossover

(b) Uniform Crossover

Figure 40: Results of an investigation into elitist selection, genetic variation and crossover methods. The fitness values graphed are averaged over nine simulations. 'Base' refers to the case where elitist selection is disabled and random individuals are not inserted into the population. Error bars represent the standard deviation divided by the square root of the number of simulations.

Method	Final Fitness (average)	Final Fitness (uniform)
Base	803.6 ± 18.3	943.0 ± 12.0
Elitist Selection	1074.0 ± 25.6	903.5 ± 21.5
Random Individuals	1217.4 ± 40.7	1164.9 ± 24.8
Both	1265.3 ± 46.9	1228.3 ± 33.8

Table 4: Tabular representation of fitness values corresponding to generation 30 in Figure 40.

Figure 40 and Table 4 show elitist selection and the introduction of random individuals benefited the algorithm. Fitness using average crossover, elitist selection and random individuals was 57% greater than the corresponding base method. The same method instead using uniform crossover also saw benefit with a 30% increase in fitness.

The data suggest that elitist selection only sees benefit where there is enough genetic variation. The uniform crossover method is less diverse than the average crossover method as the neural network weights are copied directly from parent networks. In contrast, a completely new weight is determined when reproducing via average crossover. Comparing the 'Elitist Only' data to the respective base models shows that elitist selection by itself is detrimental (-7%) to the algorithm when uniform crossover is applied but significantly beneficial (+34%) when average crossover is applied. When genetic variation is introduced to the population through the insertion of random individuals, elitist selection provides a small benefit to fitness of 4% and 5% for the uniform and average crossover methods. However, there is insufficient evidence to suggest that this is statistically significant. There is no significant difference between crossover methods when elitist selection is applied and random individuals are inserted.

We strongly encourage future years to continue this investigation by adjusting other parameters (e.g. mutation rate, selection percentile), applying different crossover methods and running more simulations. In the interest of time, we ended this investigation at this point. Further use of the genetic algorithm has applied an average crossover method, elitist selection and insertion of random individuals, justified by the data presented in this section.

3.7.2 Results of Machine Learning

We trained the variable pendulum with no spinning disk model over 55 generations to verify that the machine learning agrees with the underlying physics. We collected data on the swinging angle as a function of time and the timing of the 'pumping' action. We expected that the swinging angle should increase as a function of time and that the system will learn to pump up at the equilibrium point, $\theta = 0$.

Results of the investigation into swing angle as a function of time are shown in Figure 41. The general trend shows that angle increases as a function of time, which is a trivial result. More interestingly, one can observe that this graph is not a sinusoidal wave with increasing amplitude, indicated by the skewed peaks and troughs. This result is evidence that there is a net increase in energy in the system. In particular, energy is being added to the system at the equilibrium position and less is removed at either maximum, indicated by a steeper gradient on the approach to a stationary point and a lower gradient on the approach to the equilibrium position. The model reached an amplitude of $\frac{\pi}{2}$ in a total of 17 pumps.

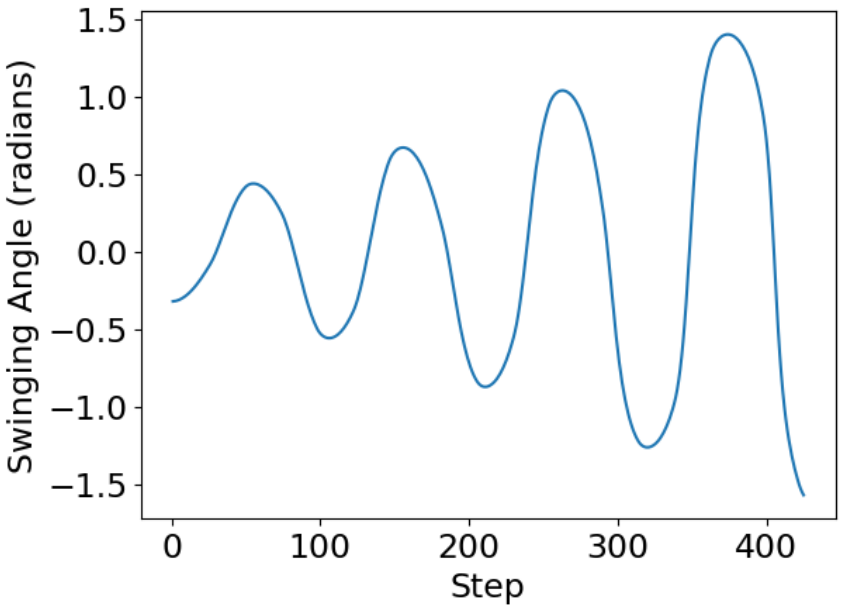


Figure 41: Plot of swinging angle as a function of time step.

Timing of the bottom pump was determined by plotting a histogram of angles where the pump occurs (the 'pumping angle') and fitting a Gaussian of the form $Ae^{-(x-\mu)^2/2\sigma^2}$ to this data, as shown in Figure 42. The mean and standard deviation of the fit were determined as $\mu = 0.2559 \pm 0.0006$ and $\sigma = 0.511 \pm 0.003$. These values suggest that the genetic algorithm successfully determined the best angle for inserting energy into the system is, within 0.06σ, the equilibrium point, in agreement with our expectation. The standard deviation is large, implying the algorithm has not trained the model perfectly. We encourage future years to determine new neural network structures to investigate if the pumping angle can be more consistent.

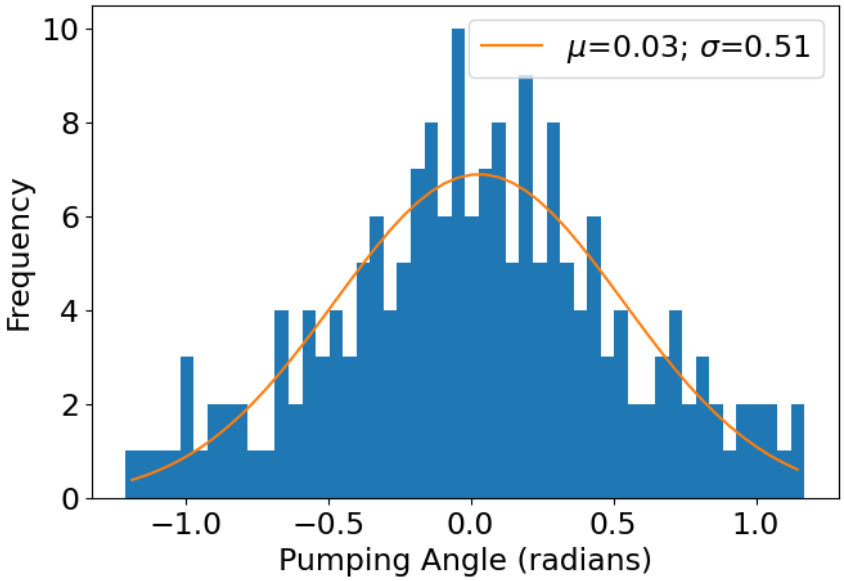


Figure 42: Histogram of pumping angles for the trained variable pendulum model. Overlay is a Gaussian fit with A=6.9, μ=0.03 and σ=0.51.

The angle where the swing would pump down is dependent on the energy in the system, so an investigation of this sort is inappropriate.

3.8 Application of Machine Learning to the Standing Stickman Model

Thomas Thies

The initial aim was to apply the genetic algorithm to the standing model of a human on a swing, although time did not allow for this implementation to be finished. An outline will be given below of what was done so far to achieve this goal, along with the next steps that would have been taken.

3.8.1 Standing Stickman Fitness function

Similarly to the variable pendulum model, the stickman model needs a *fitness function* to evaluate how well a particular stickman is performing its task of maximising swinging amplitude whilst minimising the required effort.

The modelling subgroup went through some detailed analysis in Section 2.7 to determine what the effort required from a standing swinger was for a particular movement. However, a simpler metric for effort was first needed to train the machine learning model on that captured the main aspects of a motion that make it effortful. The effortful aspects were deemed to be the rapidity of a movement represented by the angular velocity of a limb ($\dot{\alpha}_i$), and the "jerkyness" of a movement captured by the angular acceleration of a limb ($\ddot{\alpha}_i$).

If the time during which a movement develops is discretised into timesteps, the effort E_t at a timestep t is therefore given by Eq. (72):

$$E_t = \sum_i \{B_i \dot{\alpha_i}^2 + C_i \ddot{\alpha_i}^2\} \tag{72}$$

where i is the index of a limb, A_i, B_i and C_i are constants to be determined through trial

and error, and the squared terms are simply to strongly discourage the machine learning from picking large values of $\dot{\alpha}_i$ and $\ddot{\alpha}_i$. This isn't supposed to be an accurate metric of the effort but simply of what kind of motions should be avoided.

The maximum attained angle θ_{max} should be rewarded in the fitness function since the goal also to to maximise the swinging amplitude. Therefore the fitness function F that would have been used is:

$$F = A\theta_{max}^2 - \sum_t E_t \tag{73}$$

where we are summing the effort E_t over all timesteps t.

The effort parameter found by the Modelling subgroup would then have been used to work out how much effort the stickman needed for a particular motion, and compared to their square wave up-down pumping approximation.

3.8.2 Plan of Implementation on the Standing Stickman Model

The difficulty in applying machine learning to the stickman is that we are working under the constraint that the hand to foot distance must be fixed. A method was therefore devised to embed that constraint in the behaviour of the stickman, as will be explained.

The same genetic algorithm as was used on the variable pendulum model can be implemented on the stickman model making use of the fitness function described in Eq. (73). A slight change would have been making the network bigger since the behaviour is more complex than for the pendulum.

However, the neural network would have had as an output the new angle to which the top three actuators in Figure 43 must be set - namely the elbow, shoulder and hip.

The weights at nodes of the neural network are initiated randomly, therefore most of the time it would call for positions of the limbs which wouldn't respect this fixed hand-foot distance (if the feet weren't attached). Situations where limbs are applying torques against each other must be avoided - for example we don't want the arms pushing the body downwards and the legs pushing it up since that would provide no net movement and would be rather effortful.

The knee angle would therefore have then been set after the position of the top three actuators was decided by the neural network such that it satisfies the fixed hand-foot distance. Figure 43 shows the fixed hand-foot distance as the large green circle on the outside - with the smaller green circle representing the range of motion available to the knee once the other three motor positions are chosen. Geometrically, the foot position to choose looks like one of the intersections of both green circles, indicated by the orange arrows, and it would have to he the closest one to the previous foot position.

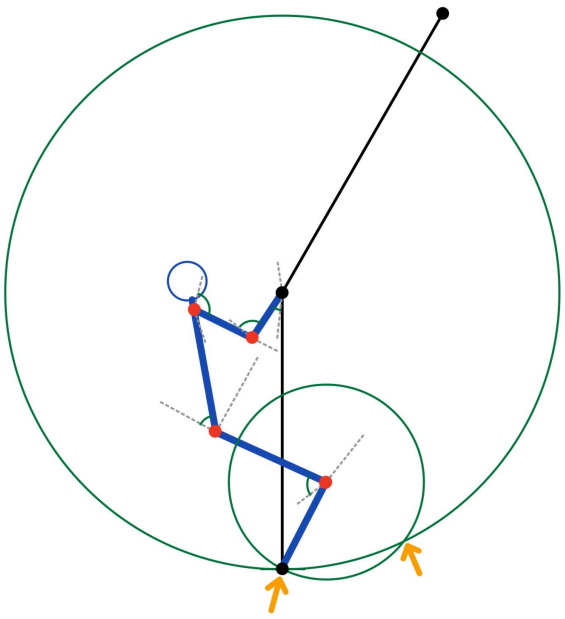


Figure 43: Diagram illustrating how the position of the elbow, shoulder and hip would be chosen first followed by the knee angle such that it respects the fixed hand foot distance. The choice of knee angle should correspond to the intersection between the two green circles, pointed out by the orange arrows.

If there is no solution for the fixed hand-foot distance (so geometrically if the green circles don't intersect), then the closest foot position would be chosen. Then Δd_t, difference between this hand-foot distance and the target, at a particular time t, would be added as a penalising term in the fitness function such that this behaviour is suppressed. The new fitness function adapted from Eq. (73) that should be used is therefore:

$$F = A\theta_{max}^2 - \sum_t \{E_t - \Delta d_t\} \tag{74}$$

The same genetic algorithm as used for the variable pendulum could then be applied on this stickman model with the fitness function in Eq. (74) to study how well it avoids the descent of the system into a chaotic state and to see whether it optimises for some unintuitive but unefortful motions.

3.9 Advice for Future Years on the Standing Simulation

Remi Bahar

We advise future projects to initially use our simulation of a 2D standing stickman to help with the sometimes steep learning curve of Pymunk and machine learning which can take a while to make progress with. This simulation encompasses the key components of a standing model, namely standing and squatting motion, being able to remain still, and joints which are constrained to angles based on human physiology. The extensive use of the configuration file also allows the simulation to be altered significantly without having to directly change the code.

This simulation could then be built-upon by adding more joints with motors and angular constraints as listed in table 2. Additional effects could also be investigated such as air resistance.

Kieron Dodge

We encourage future years to expand our genetic algorithm code to include more complex features such as structural mutations. This will help with training a more complex model, such as our standing swingman, as it removes the time-consuming process of finding a suitable neural network structure. We also encourage further investigation into determining parameters for the algorithm, including changing mutation rate and selection percentile.

3.10 Machine Learning 1 Conclusion

Thomas Thies

A simple model of a standing swinger consisting of a variable pendulum was created in the Pymunk physics engine and a custom made genetic algorithm was successfully trained to pump this pendulum such that the swinging amplitude is maximised for a minimal effort. The pumping time found was in accordance with the Modelling subgroup to within 0.06σ. It was determined that using elitist selection and inserting random individuals in the population is the most effective way to implement the genetic algorithm. A more sophisticated stickman model of a standing swinger was also created and a method was devised to use the same genetic algorithm on this model to maximise the swinging amplitude whilst minimising the physiological effort exerted by the swinger. This method was unfortunately not implemented on the stickman standing model. Although the groundwork was laid down to study how well this method avoids the descent to a chaotic state and to gain insight into new types of human motions from the movements optimised by this model.

4 Machine Learning 2: Sitting Swinger

Christopher Rouse

4.1 Subgroup Goals

This subgroup was focused on optimising swinging action for a sitting human. Energy can be supplied to such a swing either by exerting an internal torque as the human rotates about its axis or by adding gravitational potential energy by raising and lowering the centre of mass at different points in the swing. Given the limited range of vertical mobility, rotational input from the angular momentum of the swinger plays a larger role than for the standing swinger and we were interested in observing if this was used by the machine learning algorithms and whether the inclusion of an effort parameter affected this. Initially, attempts were made to run the code written by previous projects, however, due to various inter-dependencies this was not possible and so it was necessary to construct our approaches entirely independently.

The approach taken was for one member to focus on the generation of successively more complex and realistic models while the other three members produced alternative but comparable machine learning strategies to apply to them. Various algorithms were considered including Q-learning, DeepQ-learning and simple Neuro-evolutionary approaches, some hard-coded from scratch and others relying on machine learning libraries such as TensorFlow. This approach puts machine learning at the centre of the project as a subject of investigation in itself and not simply a means to an end, this is the reason for the inclusion of Goal 2. Finally, an attempt was also made to apply an effort parameter in the reward function for the machine learning approach with this in mind.

Therefore, the three primary goals of the subgroup were as follows:

Goal 1: Develop a sitting human model that displays realistic human motion.

Goal 2: Compare the success of different machine learning approaches in teaching the sitting model to swing.

Goal 3: Determine and code a reward function that includes human effort and observe the effect on motion.

4.2 The Model Developed for Machine Learning by ML2

James Marriott

4.2.1 Goals for the Model

When developing the model, the main focus concerned providing an environment within which the aforementioned goals of the Subgroup 4.1 could be explored. Given the goal of displaying **realistic** human motion, the primary objective was to construct a simple model swinger with correct body proportions, under the influence of a realistic gravitational force. Secondly, in order to compare the success of the different algorithms, the model had to be compatible with each of the methods which were developed without heavy modification.

An initial area of concern was the time constraints within which a satisfactory model was required. In order for the Machine Learning contingent to have enough time to tailor their algorithms to the model, as well as extract some results, a deadline of Week 6 was set for its completion. A similar, less complex, model was provided earlier to allow for some simultaneous progress between the two areas.

4.2.2 The Choice of Environment

Given the time constraints within which an initial model was required, the decision regarding the optimal environment for the model needed to be made quickly. As a result, the extent to which the choices were explored was limited. The decision was quickly narrowed down to a choice between Pymunk, a 2D physics library powered by Python, and Unity, a cross-platform game engine.

The existing work from previous years' Robotics group studies was a key factor behind the selection of these two environments; the thought process being that the predecessors' code combined with their lessons learnt would act as a good guide. In addition, it would provide a point of comparison with which to assess the project's progress.

Pymunk, released in 2007, is a Pythonic 2D physics library built using Chipmunk, a physics library in C. Through implementing the Chipmunk architecture in a Python IDE, it exploits the advantages of the high-level programming environment; namely the relative ease of use, shallow learning curve and the simplicity of installation [31]. Unity, first released in 2005, is a cross-platform game engine, based in $C++$ designed with the ability to run simulations in both 2D and 3D domains [32]. Models are generated using purpose built API with support primarily for the languages: $C\#$, and to a lesser extent, UnityScript.

Operating within a Python IDE was a major contributing factor for the selection of Pymunk over Unity or any similar engines. The existing familiarity of both the members of the ML1 and ML2 subgroups meant that operating with Python was a key requirement; ensuring that more time could be spent understanding the new API rather as opposed to gaining familiarity with a new language. Furthermore smoother integration with popular 'Python-based' machine learning platforms such as TensorFlow was expected. In addition, having both machine learning subgroups working with the same API in the same language presented potential opportunities for comparison between seated and standing swing mechanics as well as their respective ML algorithms.

4.2.3 An Outline of the Model's Structure

The model adopted for use by the machine learning algorithms developed by the subgroup consists of 3 main Python classes. These classes combine to generate the overall system and each contains numerous functions, relevant to their particular area of the model.

Firstly the 'Swing' class is outlined. This contains the function responsible for generating the swing component of the model. The 'generateSwing' function delineates the top (anchor) position of the swing whilst simultaneously providing the capability for joints to be added or removed with custom spacings. This is achieved through the implementation of a 'config' file where these values can be customised. The most recent iteration of this model also allows for the start angle to be customised from within the 'swing' class.

Secondly the class: 'Person', responsible for generating the swinger, is introduced. The first function, 'init' is responsible for initialising the various components of the swinger e.g. the overall position, mass, limbs and motors. Next, 'generatePerson' assigns a relative mass to each component along with its position and shape. Furthermore, the pivot points and respective motors' positions are included here. Crucially this class contains the rotation limits for the motors, restricting the model to realistic ranges of motion for a human; $\frac{3\pi}{2}$ radians total for the knee and $\frac{\pi}{2}$ radians for the hips.

Finally, the 'learning area' class is outlined. The content of the previous two classes: 'The Swing' and 'The Person' are introduced into the overall self.space object. In Pymunk the simulation

takes place in the Space object, a 2D physics environment. In addition, the input commands, available through the Pygame library, are defined here. For example, if the UP arrow key is 'pressed' the knee motor traverses at a fixed rate of $\frac{-4\pi}{3}$ radians per second. Naturally, this rate is mirrored if the DOWN arrow key is pressed. Similarly, the hip motor is controlled via the LEFT and RIGHT arrow key with a rate of π radians per second in both the positive and negative directions. Notably, a KEYUP (key is raised) event type is included in the loop, with the task of turning off the motor when the respective key is no longer depressed.

The proportions of the model were derived from the average relative limb lengths and masses provided by the Plagenhoef study [33]. It provides the body segment data collected in the form of relative percentages of each dimension's total, e.g. the mass of a limb as a percentage of the total mass of the body. This synergises well regarding the mass of each component of the swinger, given that the Pymunk component masses are also defined as a percentage of the mass of the complete body. Table 1 provides the dimensions used for the finalised ML2 model.

Limb	Relative Length (%)	Relative Mass (%)
Trunk and Head	40.75	63.36
Thigh	23.2	10.5
Lower Leg and Foot	24.7	6.18

Table 5: The relative lengths and masses as a percentage of their respective total values for the body for each limb.

4.2.4 The Challenges Faced Developing the Model in Pymunk

Several challenges, some as yet unsolved, arose through the development of the model; limiting the desired complexity and realism obtained given the time-frame. It should be noted that the majority of these were due, in large part, to a lack of familiarity with the Pymunk platform as opposed to the limitations of the library itself.

Firstly a strange interaction with the motors was observed when an adjoined limb reached the limits of rotation designated in the person class. It was quickly realised that this was due to the motor attempting to maintain a constant rate of rotation despite it's connection to a limited limb. This led to a vibrational effect which the subgroup determined would affect the integrity of any future results. In order to correct this the KEYUP function, mentioned earlier, was introduced to set the motor rate to zero when not required.

Having introduced a second pivot point to the swing to mimic a handhold on a real-world swing, a problem involving collisions between this new pivot point and the 'person' object developed. Ideally, the torso should pass through this pivot point freely as a real swinger would lean forward; however, the objects collided leading to an undesired pushing effect on the pivot. This opposes the desired pulling effect on the backwards lean, made possible with arms (not added since no solution was found). A possible explanation for this is that the 'swing' and the 'person' exist in separate classes preventing their respective 'anti-collision' commands from functioning correctly.

Simultaneously problems with 'secondary' limbs, i.e. limbs attached to other mobile limbs, grew with the addition of the head and foot features seen in Figure 44c. These additional limbs tended to migrate away from their expected positions on the swinger. This occurs since Pymunk calculates the positional error of the limb relative to where it should be (attached to the pivot), applying a corrective force in response. As the limbs are moving via the pendulum effect of the swing as well as the movement of their joined limb, the stability of the simulation suffered and led to some peculiar interactions. An example being a constant inward restorative

force on the head towards the neck joint acted as a centripetal force with the head orbiting its pivot. Regarding the model depicted in Figure 44c, no solution was implemented for this and the bulk of the ML run-through used the second stage of the model seen in Figure 44b for this reason. Fortunately, the framework developed by the ML1 subgroup defines the position of theses 'secondary' limbs relative to the angular position of their attached limbs as opposed to the pivot position which seems to somewhat solve this issue.

4.2.5 The Progression of the Model

The above-mentioned challenges both helped and hindered the 'long-run' development of the model, leading to the evolution of several stages shown in Figure 44. Figure 44a is the initial model with a motor at the knee and the thigh and torso forming a fixed 'L shape'. In this form the lower leg was limited to a range of motion of $\frac{3\pi}{4}$ radians; $\frac{\pi}{4}$ clockwise and $\frac{\pi}{2}$ anticlockwise from the starting position shown in Figure 44a.

To improve the realism of the simulation the dimensions of the initial model were changed (Figure 44b) using the relative masses and lengths shown in Table 5. Furthermore, a second motor-driven pivot was added to the hips, allowing for a movable torso with a similar $\frac{3\pi}{4}$ radian range of motion. At this point, the motor 'turn-off' solution mentioned in 4.2.4 was also included to prevent vibration at the limits of the range of motion of both movable limbs.

Moving on to the final iteration of the original model, shown in Figure 44c, a pair of secondary limbs (the head and foot) were added. A shoulder joint as well as jointed arms were also briefly added but, due to the challenges presented by the 'secondary' limbs, were quickly abandoned. Also shown in Figure 44c is an additional joint on the swing, added through the 'config' file. The intention of this was to simulate handholds on a real swing had the arms been successfully added. Due to collisions between the torso/head and this additional joint the range of motion of the torso was altered relative to the second iteration of the model to $\frac{\pi}{2}$ to prevent the torso from crossing the joint. Eventually, due to the complications of the changes out-weighting their potential gains, the decision was made not to implement them and instead use the second stage for collecting results from the ML algorithms.

Currently, a new model, based on the infrastructure developed by the modelling component of ML1 3.5, is being adapted; the progress of which can be seen in Figure 45. As can be seen there remain some errors with some parameters leading to strange lengths and angles of several components. In the long term, more uniformity between the seated and standing models would be ideal since far better comparisons could be drawn between the two styles.

4.2.6 Future Improvements for the Model

As previously mentioned, a working integration with the ML1 model would be a good step forwards with the aim of achieving the goals defined at the beginning of the projects; both for the subgroup and the overall group study. This would allow for a greater, more uniform, comparison between both the seated and standing modes of pumping a swing as well as the algorithms powering the motion.

Needless to say, investing the time in gaining familiarity with a more powerful engine like Unity should be considered, especially with the goal of achieving a more realistic and stable model. Furthermore, the option of exploring the system in three dimensions would certainly be interesting; however due to the motion of the swing being dominated by two axes, potentially unnecessary.

Assuming that in the future there are multiple machine learning subgroups it is recommended

(a) The initial model

(b) The second iteration of the model

(c) The final iteration of the model

Figure 44: The development of the model through the project

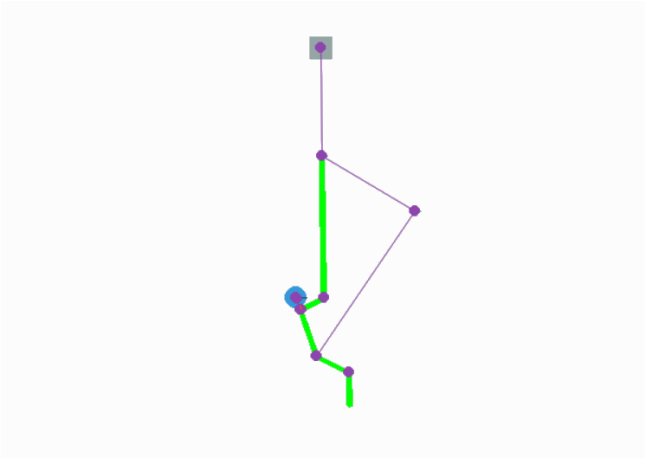


Figure 45

that a degree of similarity is kept regarding the models used; this would help to more efficiently tackle the overall group goals with several teams pulling in the same direction.

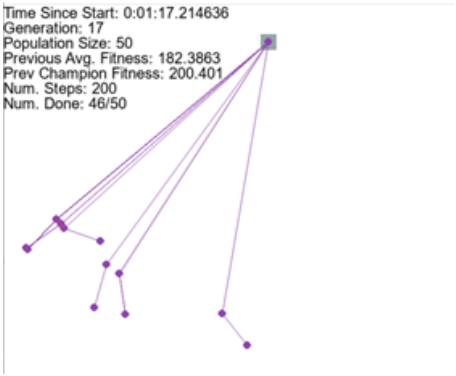
4.3 The Genetic Algorithm

Christopher Rouse

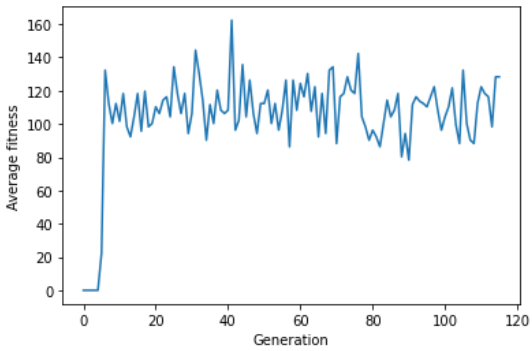
Written from scratch by Kieron Dodge of the Standing Swinger subgroup (section 3.6) this genetic algorithm incorporates evolutionary neural networks with a combination of sexual and a-sexual breeding of superior algorithms in each iteration. The basic architecture for this was shared for application to the sitting swinger but significant adaptation was required to allow this evolution to work well for a more complex swinging model instead of a model which allowed

only the extension and retraction of a pendulum.

Initially, this was attempted with a simple two-part hinged pendulum as shown in Fig 46a. In order to adapt the algorithm, the hinged model had to be produced along with functions to create and destroy the model for each iteration and revised update and fitness calculation functions. These changes presented some issues, primarily the computer found the processing and rendering much more challenging with the extra degrees of freedom and the population size had to be reduced to 25 percent of the original value. However, after these adjustments were made the algorithm was quick to optimise the swinging motion as would be expected given the simplicity of the problem. The fluctuations in average reward seen in fig 46 demonstrate the greediness of the algorithm as it continues to look for non-existent improved solutions.



(a) Visual feedback



(b) Hinged swing reward data

Figure 46: Genetic Algorithm on hinged swing

This approach was then applied to the simple sitting swinger model. Initially, this model incorporated only the motion of the legs of the swinger and no arm or torso motion. The result of this application was a partial success. The program took far longer to improve on itself when applied to the seated swinger model as the correct action to take at a given time was much more dependant on the angle of the swing and so a more nuanced approach was required. The result was a low amplitude improvement in swing angle but this was inconsistent. In search of better results, the approach was attempted with the more complex model including a movable torso and arms but this was not completed.

4.3.1 Effort Incorporation

Equation 62 shows the effort parameter as defined in section 2.11 for a seated swinger. The application of this parameter in code was not straightforward for several reasons. Primarily the code written for the seated swinger relied on a motor with a constant rotation rate at each of the joints. This meant that to incorporate speed of limb movement the Pymunk model had to be rewritten to allow for variable velocity. This alteration was attempted but due to limited time was not completed.

Had this change been successfully implemented the effort between consecutive time-steps would have been summed across the allowed swing time and subtracted (with a scaling factor) from the increase in swing amplitude to produce an improved fitness function. It was hypothesised that an effect of the effort parameter would be to reduce the impact of the internal torque exerted by the agent rotating about its axis. As the energy input from the internal torque exerted is strongly linked to the limb velocity, the penalty exerted by the effort parameter may have led the agent to focus instead on adding energy by slowly raising the centre of mass at the low point of the oscillation and lowering the centre of mass at the high points thus resembling

more closely the approach of a human swinger. This is perhaps something for future projects to observe.

4.4 Deep Q Learning

Hoe-Yin Chai

4.4.1 Introduction

In order to efficiently understand the principles of machine learning and therefore implement it to the sitting model, Deep Q-Learning was chosen as an optimal starting machine learning algorithm to apply. It has been documented to great amounts of detail by past Robotics group studies[34] and there are lots of information online regarding the implementation of Deep Q-Learning. One of the main considerations for applying machine learning to a swinging model, has been to obtain data and results that would support the swinging of a real physical swinger. Deep Q-Learning is not commonly used in many real life applications however, as the algorithm has been shown to exhibit large instabilities[35] while learning and may require vasts amounts of data before demonstrating stable performance. These issues are why the main focus of implementing and using Deep Q-Learning, has been to quantify the effect of different machine learning parameters on the quality of learning, rather than obtaining large sets of optimized swinging solutions.

4.4.2 Theory

Deep Q-Learning is a more advanced form of Q-learning[36], utilizing an Agent-Environment model which uses reinforcement learning (Section 1.3). In Deep Q-Learning, the reward function is continuously passed back to the agent after the simulation state is either initialized or updated, rather than at the end of an episode. The reward function in Deep Q-Learning is known as a Q-value. The form of the Q-value for a certain state s_t, is shown in equation 75[37]. It is more commonly known as the **Bellman Equation**

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma max(Q(s_{t+1}, a_t)). \tag{75}$$

The Q-value is defined for a state,action(s_t, a_t) pair. $r(s_t, a_t)$ is the immediate reward given to the agent after an action a_t has been applied to the current state s_t; this is dependent on the environment's reward function, which is one of the machine learning parameters investigated in section 4.4.4. The second part of the Bellman equation represents the future predicted Q values after an action a_t has been applied. *Q(s',a)* is related to *Q(s",a)* and *Q(s",a)* is related to *Q(s"',a)* and so on, such that the Q-value dependence on future Q-value states is shown by equation (76)[37]

$$Q(s_t, a_t) \rightarrow \gamma(Q(s_{t+1}, a_t)) + \gamma^2(Q(s_{t+2}, a_t)) + \cdots + \gamma^n(Q(s_{t+n}, a_t)). \tag{76}$$

γ, the discount factor, affects the contribution of these future rewards. In the initial stages, the agent does not have enough knowledge to accurately predict future Q-values so makes an arbitrary guess. But as the agent explores new states, it improves on its ability to predict these future Q-values and will begin to converge on an optimal policy for picking actions to maximise Q-value.

To allow the agent to explore many different states and improve its understanding of the environment, a variable ϵ is introduced; ϵ is able to take on any value between zero and one. There is always a probability equal to the value of ϵ, for an action chosen by the agent to be

completely random, which allows for the exploration of new states. ϵ will decay throughout the simulation as an optimal policy is converged upon, but has a minimum value.

Deep Q-Learning uses a neural network to link together and predict sequential Q-values. The neural network uses reinforcement learning through calculation of the loss function, which is used to update the network of Q-values, such that it will eventually converge on an optimal policy. The loss function[37] used in our implementation is the mean squared error of the predicted Q-value with the target Q-value (77). The part within the square brackets is the target Q-value and it is predicted using the Bellman equation (75). Variable α is the learning rate of the neural network and is effectively a weighting applied to the loss function. α must decay throughout the simulation, so that the network can converge on an optimal policy

$$L = \left([r(s_t, a_t) + \gamma max(Q(s_{t+1}, a_t))] - Q(s_t, a_t)\right)^2 . \tag{77}$$

The machine learning variables mentioned in this section and their initial values as used in the implementation are summarised in Table 6 below for reference.

Variable	Definition	Value
α	Learning rate	0.01
α_{decay}	Learning rate decay factor	0.65
ϵ	Epsilon parameter	1
ϵ_{decay}	Epsilon decay factor	0.97
ϵ_{min}	Epsilon minimum	0.1
γ	Discount factor	0.97

Table 6: Deep Q-Learning variables.

4.4.3 TensorFlow

To create the neural network required for Deep Q-Learning within Python, TensorFlow was used. TensorFlow is a machine learning software library with a particular focus on deep neural networks. A Dense layer in TensorFlow is described as *"A densely-connected neural network layer"* on its official API documentation[38]. This Dense layer is applied to the Q-values of the algorithm to form an interconnected neural network where each Q-value, *Q(s,a)*, is now connected to all the possible consequential Q-values based on the different actions which can be taken. The different components which make up the state and action vectors are listed in tables 7 and 8. These are the vectors which describe the swing environment and human model from section 4.2.5.

States					
Swing ang.velocity	Swing angle	Leg angle	Leg ang.velocity	Torso angle	Torso ang.velocity

Table 7: States of the swing environment. (Size:6)

Actions				
Leg rotation		Torso rotation		No rotation
Forwards	Backwards	Forwards	Backwards	Do nothing

Table 8: Action space. (Size:5)

The state vector is firstly passed through a Dense layer with 100 neurons/nodes; meaning that every state is connected to 100 other neurons. This is known as a hidden layer, as the state vector is not the desired output. *Q(s,a)* is the required output from the neural network, so therefore a final output Dense layer with 5 nodes, the size of the action space, is applied to the previous layer which forms our network. The network now consists of 100 different nodes (s), each connected to 5 other nodes, where each of those five output nodes represents a specific unique *Q(s,a)*. Each neural layer within the network requires an activation function. An activation function[39] helps the network understand and learn from patterns in the data by deciding on whether the signal output from a previous node should be given as an input to the next node and in what form the data should be passed. Last year's group investigated into how different activation functions affected the rate of learning for a Deep Q-Learning algorithm[34]. Based on this, the linear activation function was used for the output layer and for the hidden layer, ReLU(rectified linear unit activation function) was chosen. ReLU was chosen due to its consistency across many runs; something which was important to achieve in our results, given the shorter time frame of the project. The complete, annotated code, explaining more in depth the logic behind implementing Deep Q-Learning within Python is found through the path *Simulation/Sitting/sitting_swinger_q_learning.py* on the group's GitHub named **UOB-Robotics-2021**.

4.4.4 Deep Q-Learning Results and Discussion

At first, only a simple humanoid model (section 4.2.5) was used to run the Deep Q-Learning algorithm on. Despite its simplicity, the model was still important for testing whether the coded Deep Q-Learning algorithm was able to take a reward function and maximise it. To understand the effect of changing machine learning parameters, a plot of the reward, average reward and greatest swing angle achieved throughout the simulations was collected. The average reward for an episode is the sum of the current reward and all the rewards previously, divided by the episode number. The machine learning algorithm was run for a set amount of time, before the simulation environment is reset and the learning parameters (Table 6) are updated accordingly. This is known as an episode, with the simulation allowing the user to set how many episodes are to be run and the length of each episode.

4.4.5 The simple humanoid model

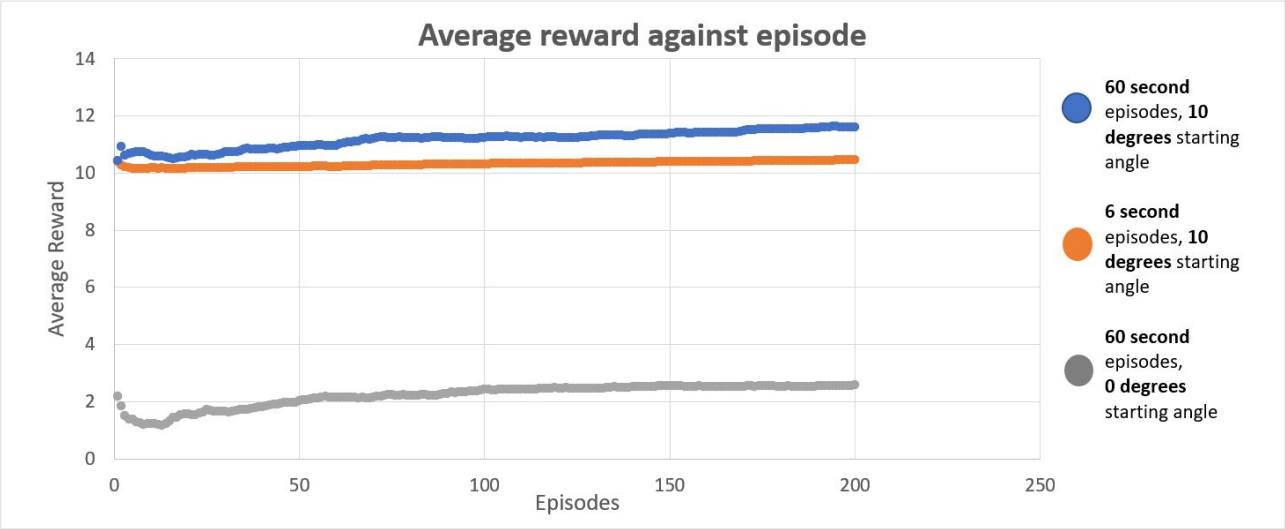


Figure 47: Average reward plotted against episodes for the reward function: best swing angle achieved during an episode.

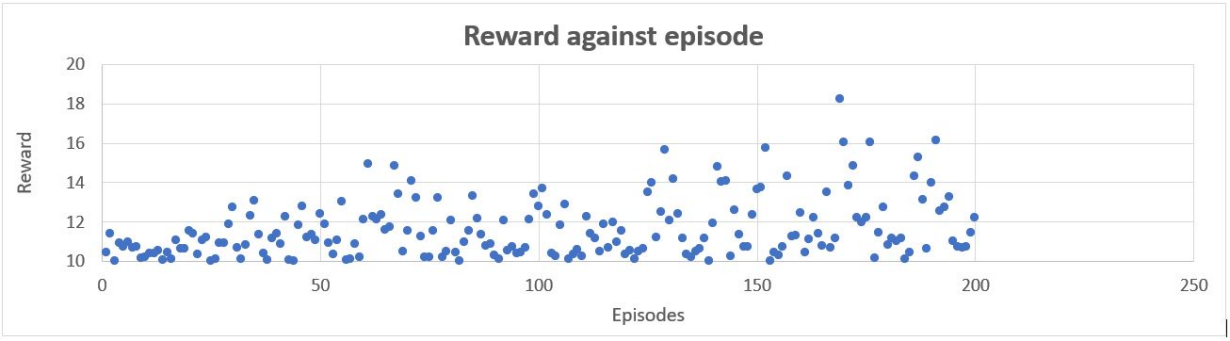


Figure 48: Reward obtained after each episode for a swing starting at 10° and a reward function: best swing angle achieved during an episode.

The general increase in average reward throughout the episodes (Figure 47), indicated that the algorithm was able to learn and maximise its reward function. The scatters on Figure 48 further show that the algorithm is building upon the best rewards achieved as there is a positive correlation between the individual rewards and episodes. It is important to note that the overall increase in the reward/best swing angle across all the simulations is very low. This suggested that a reward function solely focused on the best swing angle achieved during an episode may not be very efficient at reaching an optimized swinging solution. Therefore, different reward functions were investigated, before eventually deciding to build upon last year's reward function which was $\theta + \omega^2$.

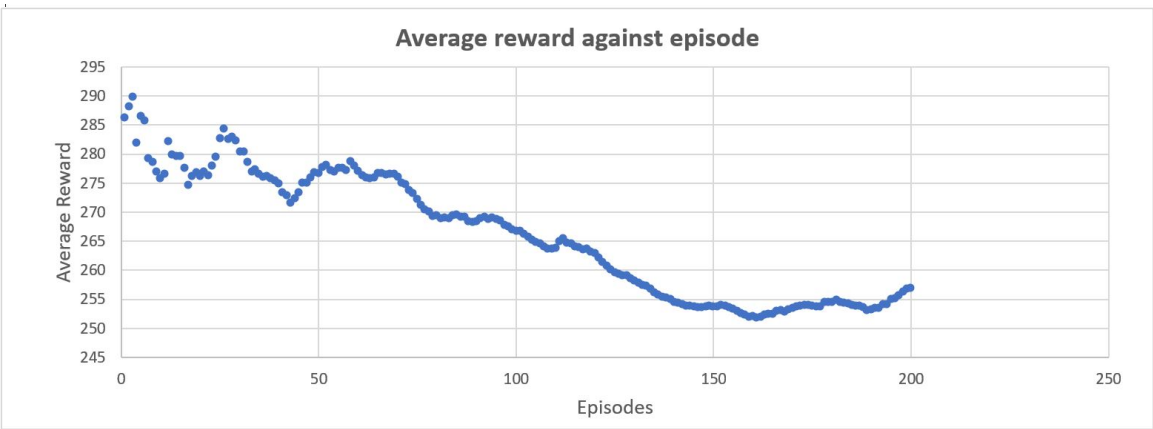


Figure 49: Average reward plotted against episodes for 60 second episodes, a starting swing angle of 10° and the reward function: $\theta + \omega^2$.

This new reward was given at the maxima and minima amplitudes of the swing. It continuously incorporates the swing's angular velocity and displacement, which helps to promote the consistent pumping of energy into the swing. The previous reward differed, in that the reward was only increased when a new best swing angle is achieved. This led to scenarios in which the swinger was not always rewarded for displaying correct, in-phase motion with the swing, as the swing may not have been at a new best swing amplitude. One observed downside to this new reward however is that because the reward is given continuously throughout the episode, depending on when the swinger is pumping energy into the swing, this can cause big variations in the final reward at the end of each episode. You can see this in Figure 49, where an initial random solution which correctly pumps the swing early in the episode has caused the average reward to be abnormally large at the start. The initial pumping of the swing is important for getting the swing moving periodically, but with this simpler swinger model, it bore too much of an effect on the reward. But as this was only a very basic swinger model, the same reward

function was continued to be developed on the final, more realistic humanoid model, where the new features changed the reward function's behaviour.

4.4.6 The realistic swinger model

These next simulations now include a damping factor, an effort parameter (Section 2.11) and the introduction of the finalized swing/swinger model (Section 4.2.5). The reward function now incorporates the effort parameter by subtracting its value from the reward. This is to encourage swing pumping which is in phase with the motion of the swing and penalize when the motion is not beneficial to the swing. Overall these changes helped make the simulations more realistic, aligning with ML2's first goal. Here the aim was to begin changing some of the more underlying machine learning parameters like α, ϵ, γ and quantify its effect on the quality of machine learning. However, the introduction of these new features brought a lot of new complexities and unexpected results to the simulation. In particular, at low swing angles the position of the pivots on the swinger would often clash when the torso was upright and the legs were fully retracted back. This caused the swing system to vibrate and led to an incorrect issuing of the reward because the program was taking both the maxima and minima parts of both the swing and vibrational oscillations. This effect can be visualised in the figures below.

Figure 50: Reward obtained after each episode throughout the simulation where the swinger is vibrating after starting at a low swing angle of 5°.

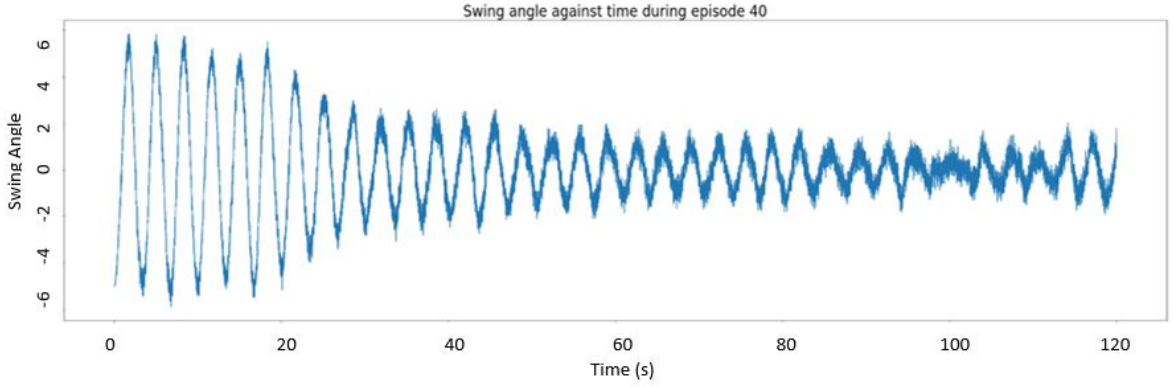


Figure 51: Swing angle against time during the 40th episode of a simulation where the swinger is vibrating.

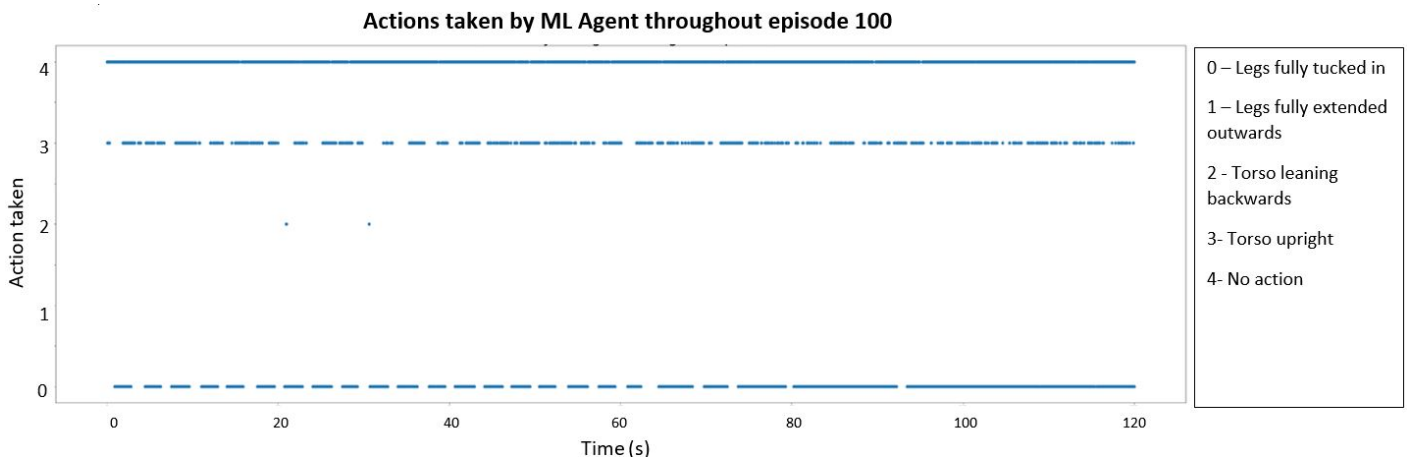


Figure 52: Actions taken by the swinger during the 100th episode in which the swinger has learnt to optimize the reward by causing vibrations.

Figure 51 visualizes the scenario in which the agent is using the glitch in the model as a way of achieving a high reward. You can see that as the swing is damped , the vibrations begin to pick up which cause oscillations along the wave itself. This is essentially tricking the environment into giving out more rewards than it should and is shown by the sharp peak in reward at Episode 40 (Figure 50). The majority of the episodes however, end with a zero reward because this policy, where the agent allows the swinging oscillations to damp and set off the internal vibrations, does not occur glitch 100% of the time. The issue still caused a big problem for being able to quantify the effect of numerous machine learning parameters as the glitch would prevent any usable data to be collected. The actions taken by the swinger from Figure 52 show that only two actions are ever chosen in this glitched selection policy. This if for the hundredth episode, where the agent has began converging on its optimal policy for maximising the reward. You can see that in this scenario, the agent never drives the legs forward or leans back at all. The three different sets of simulation data on Figure 50 represent the simulations in which the α, ϵ and γ parameters were altered. As mentioned, this unexpected behaviour of the swinger slowed down the data collection process and the main focus was therefore shifted towards the fixing of this issue, so that the program was able to come to a valid and optimized swinging solution for the final model.

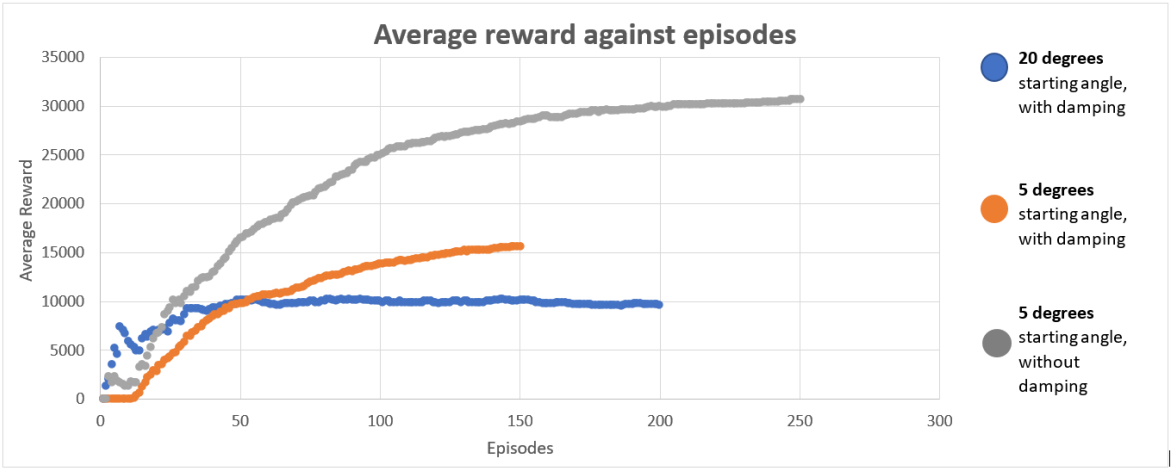


Figure 53: The average reward plotted against episode number for 120 second episodes and a reward function: $(\theta + \omega)^2$.

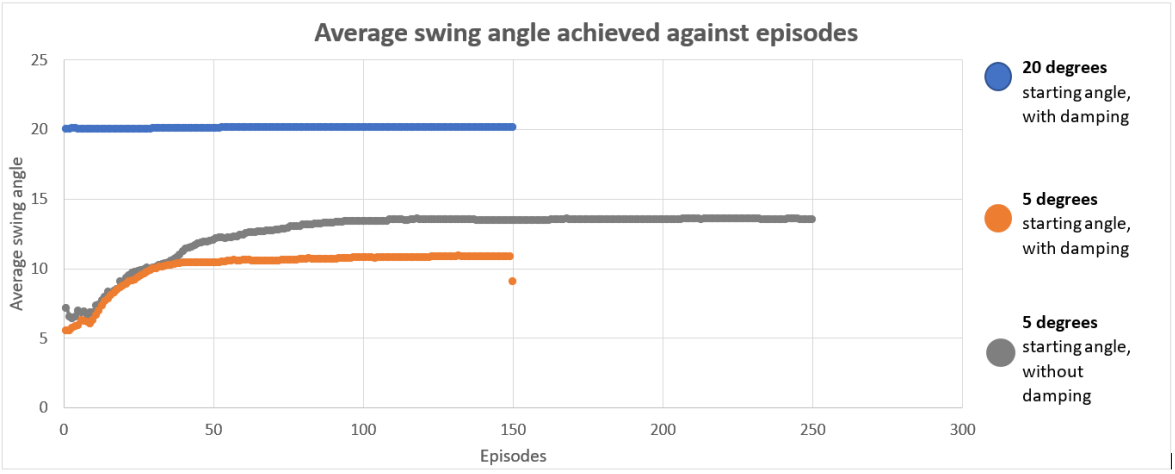


Figure 54: Best swing angle obtained after each episode for 120 second episodes and a reward function: $(\theta+\omega)^2$.

Figure 53 shows 3 simulations which yielded more successful machine learning outcomes. The blue data points show how the introduced effort parameter and damping factor have helped reduced the disparity between the swinger pumping the swing near the start of the episode and the swinger doing so near the end. The effort parameter reduces the reward based on the effort exerted by the swinger when pumping, which helps balance out the gain in the final reward from pumping early in the episode. The damping factor forces the swinger to be consistent in the energy it pumps into the swing throughout the episode, as its angular velocity will quickly fall off. Both of these elements has caused the requirement for a high reward to be dependent on more than just when throughout the episode the swinger chooses to correctly pump the swing; it is now more vital that the motion is carried out throughout the remainder of the episode. Comparing the blue points to that in Figure 49, you can now see the correct and expected learning curve for the swinger , with a convergence towards certain reward. For the same simulation configuration in Figure 54, you can see that although it was able to improve upon its reward, the swing angle never really increased, meaning that the pumping from the swinger was counteracting the loses to the kinetic energy due to the damping term. The figure captions show how the reward function as slightly altered, which was due to the scale of the effort parameter added. Both the angular velocity and angle of the swing were now both squared, so that the reward is positive when the swinger demonstrates correct swinging motion. This was something that was not observed with the reward: $\theta + \omega^2$, when the effort parameter was added.

The orange and grey data points show simulations which started at a lower swing angle, but where the internal oscillation glitch never occurred and so the swinger was able to converge on a valid swinging solution. Figure 54 shows that the simulations which started at 5°, were able to improve upon its best swing angle far more efficiently than any other particular swing setup. This suggests that a reward function which considers the effort parameter is optimal for promoting correct swinging motion. The reason this is not observed by the blue data points however, could be due to the fact that the swinger receives a larger base reward given that it starts at a large initial swinging angle. This is seen in Figure 53 where the blue points start at a reward of around 5000, whereas the simulations starting at 5° begin closer to the origin. The correct in-phase motion of the swinger is the only way for the reward to increase when starting near the equilibrium, compared to when it starts at 20° and the swinger can receive rewards for simply maintaining the already energetic swing. This could lead to non optimal solutions for higher starting swing angles, but a scenario involving a high starting angle would have to be

trialled for a zero damping term, to check that frictional losses is not a limiting factor.

The comparison between an environment with and without damping, for a low starting swing angle, is shown by the orange and grey data points. In Figure 53, you can see that from the very first episode, the non-damped swing was able to achieve rewards greater than zero, meaning that the swing never fell drastically below its initial angle. This was very important given that the vibrational glitching of the swinger only occurs as the swing angle approaches zero. As the non damped swinger is able to explore more viable solutions in the initial stages, compared to the damped swinger which only obtained zero rewards in the first few initial episodes, the non damped swinger had more of a chance of finding an optimized swinging policy compared to the damped swinger. This is indicated by the steeper rise of the average reward for the non damped swing compared with the damped swing. The final swing angle converged upon by the damped swinger was also 4° higher than for the non damped swing. The damping term inside the swing's environment should therefore be zero, when aiming to more reliably find the best swinging solution for a humanoid model. The more optimized solution achieved by the non damped swinger is represented by figure 56, where you can see that the swinger is periodically pumping energy into the swing by extending and retracting its leg, plus leaning backwards when the legs are tucked in. Comparatively, the damped swinger never uses any torso motion, or fully tucks its legs in, which is why the solution for the damped swing was decided as non-optimal. The gradual and smooth increments of the swing's amplitude for the non-damped swing during the last episode of the simulation is shown in Figure 57.

Figure 55: Action map showing the final set of solutions obtained for a damped swing.

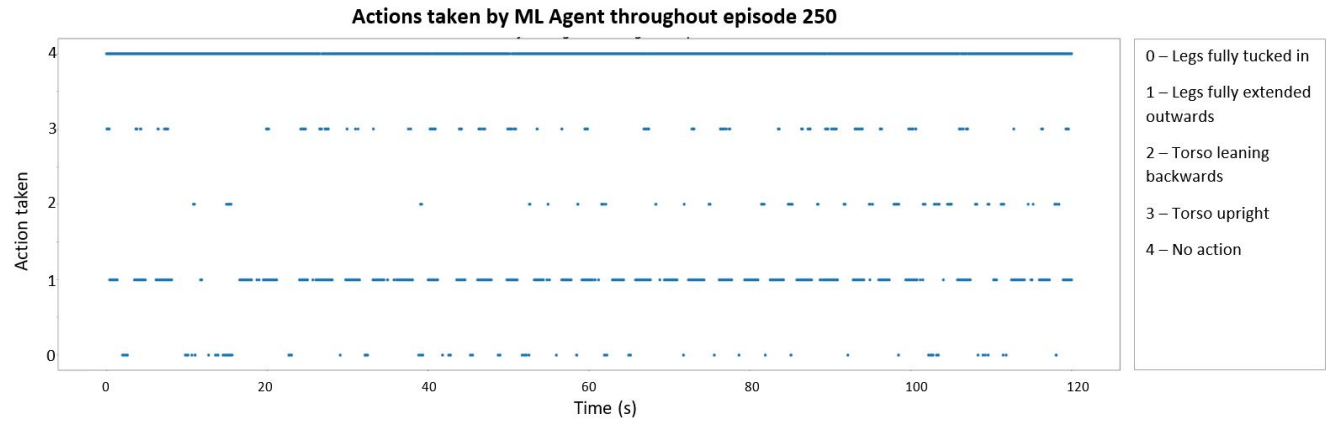


Figure 56: Action map showing the final set of solutions obtained for a non damped swing.

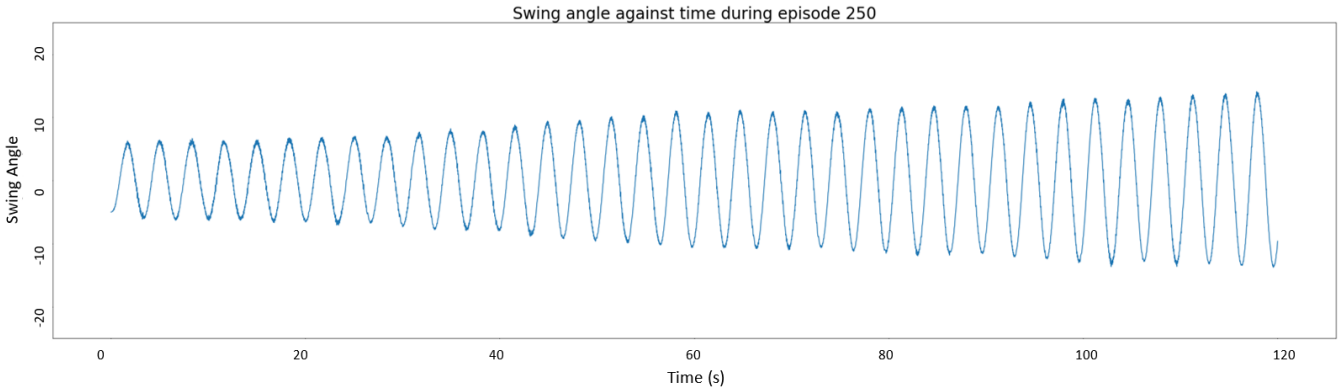


Figure 57: Swing displacement against time for the optimized non-damped swing, which used 120 second episodes, a 5° starting displacement and a reward function: $(\theta + \omega)^2$.

4.4.7 Future Considerations

A lot of the limiting factors involved with the data collection side of machine learning, are attributed to the number and speed to which machine learning simulations were able to be run. This meant that it was difficult to effectively quantify the different machine learning parameters used in Deep Q-Learning, which was one of the primary goals as mentioned in Section 4.4.1. With physical access to more computers, machine learning could and should be run simultaneously to obtain enough data to compare against each other and provide a better chance of being able to quantify the machine learning variables. It would also make troubleshooting the algorithm far easier, which is what took up more of the project time than anticipated, e.g. the vibrational glitching. Another point is that the Deep Q-Learning algorithm is inherently unstable[35], as mentioned in the section's introduction. This led to the algorithm never producing any stable solution to the humanoid swinging problem and is why the average reward was plotted, rather than the individual rewards of each episode which never stabilized. This is not ideal when aiming to apply machine learning to a real physical model. Therefore it is more important to begin with the implementation of other machine learning algorithms such as GNARL and NEAT, which have been shown to have a more stable performance[34]. Although the process of implementing Deep Q-Learning has provided a solid understanding into how reinforcement learning and neural networks are used in machine learning, I believe it would be more beneficial for future groups to only go through the theory of Deep Q-Learning and focus more on implementing the GNARL or NEAT algorithms as mentioned.

4.5 From DQN to DDPG

Gong Xiangyu

In the previous part, we have introduced a simple understanding of Deep Q-learning (DQN). The core idea of the DQN needs people to calculate the value function and select the action according to such a valued function. Hence, people also called such an algorithm as value-based.

However, another method for reinforcement learning (RL) called the policy-based, which does not select the action based on the value function, may yield better results and be easier to understand, such as Policy Gradient.

4.5.1 (Deterministic) Policy Gradient

From the introduction of Deep Q-learning(DQN), people understood that the nature of reinforcement learning using the value function approach requires feedback from rewards to train

the model. The input to the model is the current state, and the output of the model is the long-term feedback, and one makes a choice of action based on the size of the feedback value. It is natural to ask why people would want to use a deep network to obtain feedback and then select actions based on that feedback, rather than using a deep network to select actions directly. This is where the idea of policy gradients comes into play.

Building upon the q value from section: 4.4.2, an approximate value of action can be calculated by a value function:

$$q(s, a, w) \approx q_\pi(s, a), \tag{78}$$

where s is the state of the environment and a is the action, π is the policy guides the function to select a suitable action and the w is a parameter to describe this policy, or in other words, w is the weight parameter for Neural Network in Deep Q. Similarly, in the Policy-based method, the formula below can allow people to describe a policy, π, by a parameter θ in a continuous function:

$$J(\boldsymbol{\theta}) = \pi_\theta = \pi(a|s, \boldsymbol{\theta}) \approx \pi(a|s). \tag{79}$$

A commonly used policy function is the normal distribution, which is represented by:

$$\pi(a|s, \boldsymbol{\theta}) = \frac{1}{\sigma(s, \boldsymbol{\theta})\sqrt{2\pi}} exp\left(-\frac{(a - \mu(s, \boldsymbol{\theta}))^2}{2\sigma(s, \boldsymbol{\theta})^2}\right), \tag{80}$$

$$\mu(s, \boldsymbol{\theta}) = \boldsymbol{\theta}_\mu^T \boldsymbol{x}_\mu(s), \tag{81}$$

$$\sigma(s, \boldsymbol{\theta}) = exp(\boldsymbol{\theta}_\sigma^T \boldsymbol{x}_\sigma(s)), \tag{82}$$

where $\boldsymbol{x}_\mu(s)$ and $\boldsymbol{x}_\sigma(s)$ are the eigenvectors of the state. This policy function is not unique. If required, people can use the softmax or other function to replace the expression of the $\pi(a|s, \boldsymbol{\theta})$.

After representing the policy as a continuous function, people can use optimisation method to find the optimal policy. A common method is the gradient ascent, which can be described by:

$$\nabla_\theta J(\boldsymbol{\theta}) = \sum_s d^\pi(s) \sum_a \boldsymbol{\pi}_\theta(s, a) \nabla_\theta \log \boldsymbol{\pi}_\theta(s, a) Q^\pi(s, a), \tag{83}$$

where the $d^\pi(s)$ is the stationary distribution of states under policy π, and $Q^\pi(s, a)$ is the value of a state-action pair given a policy, the detail of this can be found in the Reference [40]. The input of the Policy Gradient is the state and possible action of the agent, and the output is the distribution of the possible action selecting. The pseudo-code is given in appendix A.1. People can also select a specific action from this distribution instead of output the distribution directly. Then, this idea is called Deterministic Policy Gradient.

4.5.2 Actor-Critic

The Actor-Critic is a well-known structure of the reinforcement algorithm, which combined the value-based and policy-based methods. It allowed people to calculate the value function but did not guide the action choice directly. The action selection is given by the Actor part, which will be a policy-based method. The Critic part will be a value-based method, which is used to guide how the gradient in the policy-based method descent. This structure is shown in Fig.58.

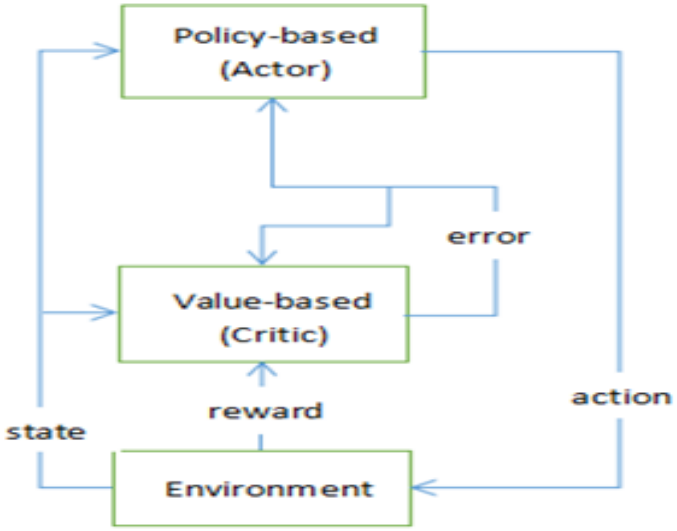


Figure 58: The draft of the Actor Critic structure.

This figure reveals the data flow of this structure. Firstly, the Environment feeds the state and reward data to the Critic part and only feeds the state data to the Actor. Then the Actor will choose an action to update the Environment. Then Environment feeds data to both other parts again. The Critic part will calculate an error from now on and use it to guide self and Actor part update their Neural Network (NN) parameter. After some episodes, the action that the Actor chooses will hence become reasonable.

4.5.3 Deep Deterministic Policy Gradient

The Deep Deterministic Policy Gradient (DDPG) is a robust reinforcement learning algorithm, which absorbs all the advantages of the Deep Q-Learning and the Policy Gradient. In section 1.1, it mentioned the DPG method needs a standard to select a specific action from a continuous possible action distribution. Then, how setting this standard becomes an enormous problem to apply the DPG in practice. Fortunately, from previous, Deep Q-learning is an expert to deal with such problems. Hence, people can use Deep Q- learning to guide how to make the DPG give a reasonable action choice. This methodology is called the Deep Deterministic Policy Gradient.

Like the evolution from Q-learning to Deep Q-learning, people can also use the feature of the target network and the Experience replay to promote the learning process. The target network feature means that both DQN and the DPG will have two Neural Networks, the target network, and evaluate the network. Then, the total number of the NN is four. For the Experience replay, this method allows the algorithm to disorder the relevance of data and learn from historical data. It can make the algorithm convergent much quicker. The flow draft is given in the Fig.59.

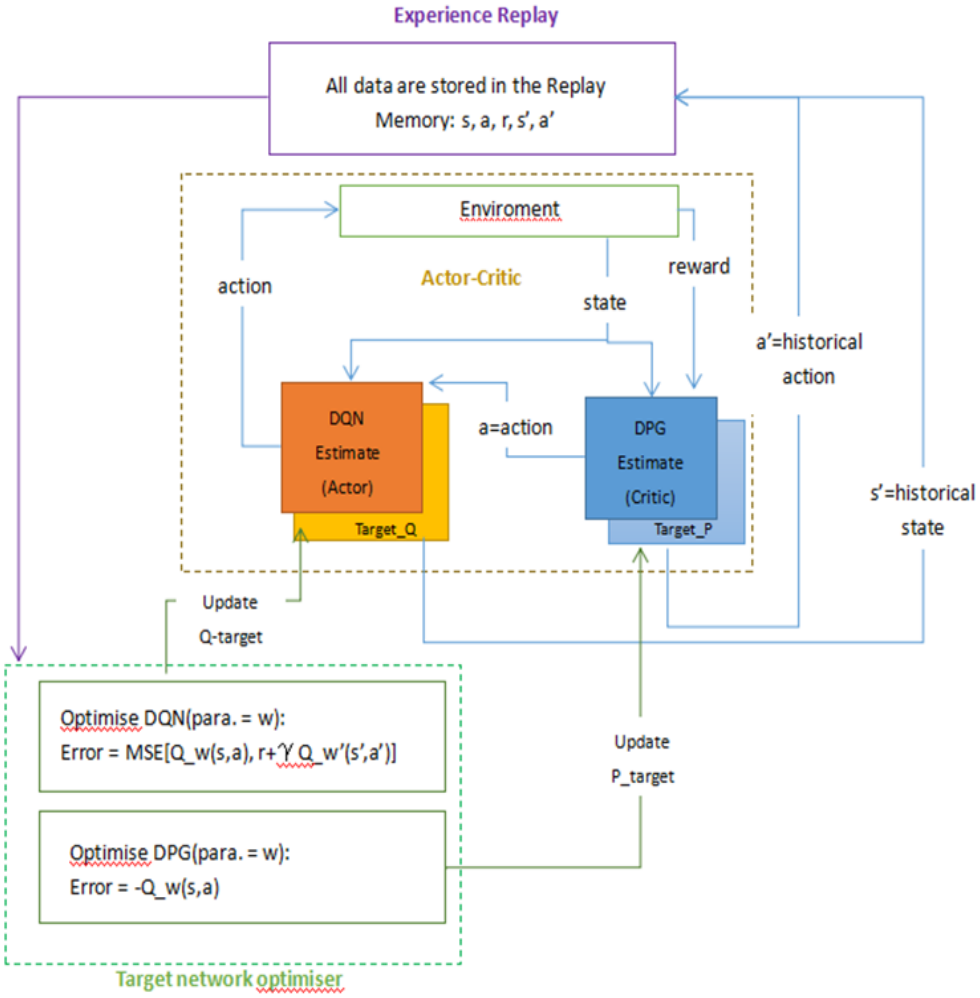


Figure 59: The flow draft for the DDPG algorithm.

This figure describes the relationship between each module in the DDPG algorithm. People can find that there are two boxes in the dashed line. The brown one is the Actor-Critic part, which is the basic same as the introduction in section 4.5.2. There is two difference here. The first one is people can use a double network, estimate network, and the target network, in both Actor and Critic part, where the Q is DQN and P is DPG. Another one is that people can use the Target network optimizers, which shows in the green dash line box, to guide the network update instead of the errors. The gamma here is a constant between 0.1 to 1, called rewards discount. People also can see there are purple boxes at the top of figure 59. That is the Experience Replay function. When writing the code, people can create a list to memory all the information in the system, i.e. state, action, and rewards.

4.5.4 Result of DDPG

Gong Xiangyu

In this project, I use the seat's position, which the swing person is seated on, the swing angle, and its angular velocity as the input state. There are two motors in the position between body and legs, and between body and torso from Jame's model. Then, the action can be the rotation rate of these two motors. Hence, the dimension of the state and the action are 4 and 2, respectively. When each episode begins, I record the swing person's initial position and use the difference between the initial position and the current position while swinging as the reward.

From the theory section, people know the DDPG is a combination of the DPG and the DQN algorithm. Both methods require selecting several parameters, such as the learning rate, alpha, and the reward discount. I select the gamma equals 0.8, and learning rates for both Actor and Critic are equal to 10^{-4}.

As for the NN in the Actor part, this NN consists of one input layer, three hidden layers, and one output layer. Initially, I select the ReLU6 as the activator for all three hidden layers. However, it not works because when the input of the ReLU6 is negative, it will return zero, so that everything behind will be zero. Hence, I change the activator as the SeLU function, which will return some small numbers instead of zero. Because the rotation of the Joint of the swinger should have both positive and negative, I select the Softsign function as the activator of the output layer. The activator selection of the Critic part is the same caused by a similar reason.

When the NN training is finished, the result of the training will be saved in a set path. Then, people can use the trained network to generate the swing of the person without trained again. A diagram of the reward versus the number of steps is given in Fig.60.

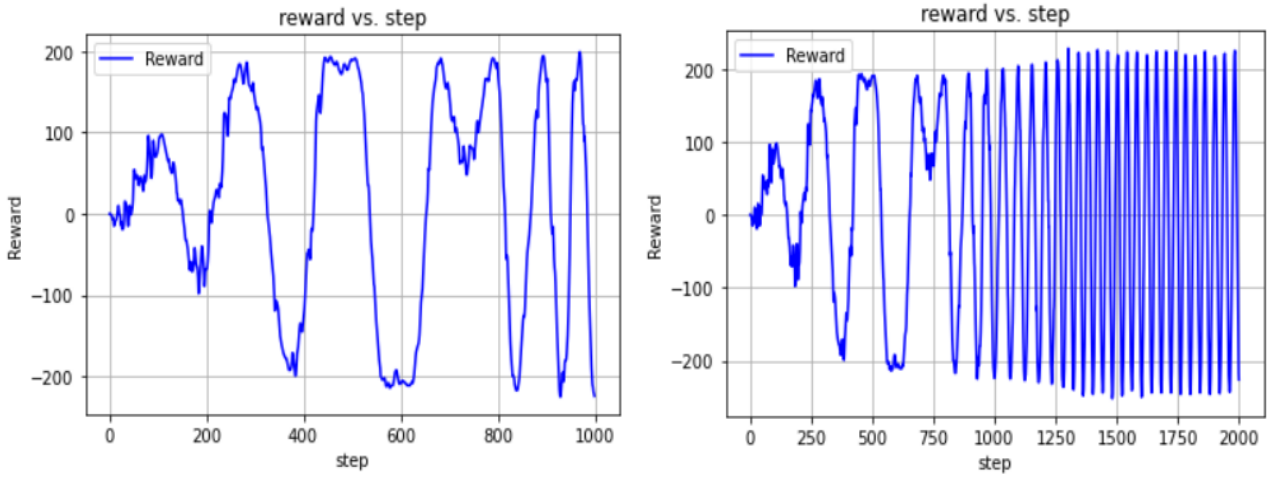


Figure 60: Training result in reward versus time

The figure reveals that the offsets of the position of the swinger will be larger and larger under the automatic control by the DDPG algorithm. However, this is a version without adding any rotation constraint on the legs and torso. Unfortunately, if people add the constraint to make the swinger humanoid, the algorithm will make the torso and legs of the swinger always move in one direction.

4.5.5 Conclusion of DDPG

Gong Xiangyu

In conclusion, the DQN algorithm can only treat the discrete problem. People need to introduce the Policy Gradient to deal with a continuous problem. The Deterministic method should be introduced when the problem needs a specific action rather than a probability distribution of possible actions. However, the standard on the selection of such specific actions is difficult. People can use the Actor-Critic structure to set the standard of DPG by the DQN method. Without the constraints on the joint of the swinger, the DDPG algorithm is well-preformed. In the future, people can try to change a different constant in the algorithm or setting the different input states and rewards to deal with the constraints problem.

4.6 Machine Learning 2 Conclusion

Christopher Rouse

4.6.1 Results

A complex model with hands attached to the swing and multiple limbs was achieved, unfortunately however, machine learning was not successfully implemented on this model due to time constraints. Several functional machine learning approaches were produced in Q-learning, DeepQ-learning and Neuro-evolutionary frameworks and applied to a variety of simplified models. Goal 1 was partially achieved as a sitting swinger was taught to swing, however, as a model that could exclusively move its legs this is far from the realism that was aimed for and the resultant increases in swing angle were not impressive. For Goal 2 it was possible to draw some conclusions about how different parameters in Q-learning affected the rate at which machine learning models improved. Goal 3 was attempted with the coding of the effort parameter developed by the modelling group and was implemented with varying success, speculation as to what might have been observed and the physics behind that was also included. This stabilised the swing amplitude a little but unfortunately provided little insight due to the simplicity of the model to which it was applied.

4.6.2 Advice for future years

This year significant time was dedicated to the construction of Pymunk models and the writing of new machine learning algorithms. Unlike in previous years, a machine learning approach (the genetic algorithm) was written entirely independently of existing frameworks. Usefully, it relies on very few external packages and so future projects should have no difficulty running and using this code. We suggest that next year take the the genetic algorithm produced this year and extend it for their own purposes[16]. While not achievable this year, it is hoped that with more time, the approach begun this year can be generalised to more complex model and swing designs in future.

In addition, due to the complexities of running many concurrent machine learning simulations the code can be slow to run. If future years are able to attempt more complex simulations it may be useful to investigate use of the department servers for supplementary computing power.

5 Conclusion

In conclusion, through the formation and analysis of the Lagrangians of a variety of pendulum-based models, numerical models were produced to simulate a seated and standing rider on a simple rigid swing, and a model of a standing swinger on a hinged swing was attempted. The effort required to drive a swing when seated was defined as the sum of the energy required to rotate a limb around a joint and the energy required to support a limb against gravity, as shown in equation 61; the effort required when standing was defined as the sum of the energy required to rotate a limb to allow the swinger to squat, the work done against gravity through changing the swingers height as they stand/squat, and the energy required to remain balanced on the swing, as shown in equation 19.

Within the Machine Learning environment Pymunk, models of both a seated and standing swinger were produced, with a new genetic algorithm written to precisely cater to the needs of this project. This was a necessity as previous groups' algorithms were not transferable due to packages used. Both genetic and Deep-Q learning were implemented to varying degrees of success, concluding that the non-sinusoidal nature of the motion indicates that for a standing

swinger energy is added to the system at equilibrium and removed at the maximal points. Neither machine learning group managed to implement their machine learning for a hinged swing, however, they laid the groundwork and produced sufficiently sophisticated and robust models that this should provide a good starting point for next years' project, and as informing future work is one of the main goals of this group study, this would be considered successful. The implementation of the effort functions to restrict the machine learning models to more human-like movement was partially successful, but requires more attention - the implementation of this could be a key goal for future groups.

6 Acknowledgements

The group would like to express their gratitude to Dr Miguel Navarro-Cía and Dr Wolfgang Theis for their invaluable support and guidance throughout this project, and to previous years' groups for providing the inspiration for our starting points.

References

[1] Lisa Nocks. *The Robot: The Life Story of a Technology*. Greenwood Publishing Group, 2007.

[2] Weilyn Pa. Robotics: A brief history. 1999. https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/history.html#:~:text=The%20earliest%20robots%20as%20we,industry%2C%20but%20did%20not%20succeed.

[3] Yogesh P, Rohit, Alok, and Kumar. Growing momentum in the various field of robotics-a review. *International Journal of Innovative Science and Research Technology*, 4(8), 2019.

[4] DRC. Darpa robotics challenge finals 2015. 2015. https://archive.darpa.mil/roboticschallenge/finalist/ihmc.html.

[5] Karen Hao. What is machine learning? *MIT Technology Review*, 2018. https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/.

[6] Junling Hu. Reinforcement learning explained, Dec 2016. https://www.oreilly.com/radar/reinforcement-learning-explained/#: :text=There are three basic concepts,state, action, and reward.text=Action is what an agent,actions an agent can take.

[7] Global robotics market revenue 2018-2025. 2021. https://www.statista.com/statistics/760190/worldwide-robotics-market-revenue/.

[8] Stephen Wirkus, Richard Rand, and Andy Ruina. How to pump a swing. *The College Mathematics Journal*, 29(4):266–275, 01/09/ 1998.

[9] Carl W. Akerlof. The chaotic motion of a double pendulum. http://instructor.physics.lsa.umich.edu/adv-labs/Chaotic%20Double%20Pendulum/Pendulum_2012_09_26.pdf.

[10] ybeltukov (Mathematica Forum User). Make a smooth differentiable sawtooth waveform. https://mathematica.stackexchange.com/questions/38293/make-a-differentiable-smooth-sawtooth-waveform.

[11] Wolfgang Theis. Standing swinger example code, 2020.

[12] Chris J.Barclay. Efficiency of muscle. *Efficiency of Skeletal Muscle*, 24/02/ 2021. https://www.sciencedirect.com/science/article/pii/B9780128145937000062.

[13] Torque and rotational motion, 2021. https://www.ch.ic.ac.uk/local/physical/mi_4.html.

[14] Peter L. Tea and Am. J. Harold Flak. The pumping of a swing from the seated position.

[15] William B. Case. The pumping of a swing from the standing position. *American Journal of Physics*, 64:215, 1996.

[16] Robotics github. https://github.com/UOB-Robotics-2021.

[17] Body segment data. https://exrx.net/Kinesiology/Segments.

[18] Joumana Medlej. Joint limitations (with pictures), 06/03/ 2014. https://design.tutsplus.com/articles/human-anatomy-fundamentals-flexibility-and-joint-limitations--vector-25401.

[19] Umbarkar A.J. and Sheth P.D. CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW. *ICTACT Journal on Soft Computing*, 06(01):1083–1092, October 2015.

[20] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2), June 1994. https://doi.org/10.1007/bf00175354.

[21] Siew Mooi Lim, Abu Bakar Md. Sultan, Md. Nasir Sulaiman, Aida Mustapha, and K. Y. Leong. Crossover and mutation operators of genetic algorithms. *International Journal of Machine Learning and Computing*, 7(1):9–12, February 2017. https://doi.org/10.18178/ijmlc.2017.7.1.611.

[22] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. In *Machine Learning Proceedings 1995*, pages 38–46. Elsevier, 1995.

[23] A.P.Wieland. Evolving neural network controllers for unstable systems. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pages 667–673 vol.2, 1991.

[24] N. Saravanan and D. Fogel. Evolving neural control systems. *IEEE Expert*, 10:23–27, 1995.

[25] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, June 2002.

[26] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997. https://doi.org/10.1109/4235.585893.

[27] Auttri Chakravarti, Pritam Mitra, and Yashvant Wadher. Genetic algorithms: The reality, 2006.

[28] Alexandre Le Fourner. Neural networks tutorial. June 2019. https://github.com/alexandrelefourner/neural_networks_tutorial.

[29] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi,

Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. https://doi.org/10.1038/s41586-020-2649-2.

[30] Alan McIntyre, Matt Kallada, Cesar G. Miguel, and Carolina Feher da Silva. neat-python. https://github.com/CodeReclaimers/neat-python.

[31] Victor Blomqvist. Pymunk api reference. Pymunk Documentation.

[32] Unity Technologies. Unity user manual 2020.3 (lts). https://docs.unity3d.com/Manual/index.html.

[33] Plagenhoef Stanley, Evans F Gaynor, and Abdelnour Thomas. Anatomical data for analyzing human motion. *Research quarterly for exercise and sport*, 54(2):169–178, 1983.

[34] James Nunns, Ari Gold, Alex Mclean, Owen Mawer, Shomari Mills-Legerton, Thomas Rocke, Louis Miranda-Smedley, Henry Shaw, Jake Ward, Sam Wheeler, Robert Noakes, Will Petchey, and Jack Sanders. Achieving efficient swinging of a nao robot through effort-restricted machine learning. pages 38–43, Mar 2020.

[35] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Z Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations. *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32(No. 1):page 3223, 2018. https://ojs.aaai.org/index.php/AAAI/article/view/11757.

[36] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, Vol. 120(No. 486-489):pages 5–16, Feb 2020. https://arxiv.org/pdf/1901.00137.pdf.

[37] Ankit Choudary. A hands-on introduction to deep q-learning using openai gym in python. April 2019. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.

[38] Tensorflow dense layer api. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense.

[39] Avinash Sharma V. Understanding activation functions in neural networks, Mar 2017. https://arxiv.org/pdf/1901.00137.pdf.

[40] Richard S. Sutton et al MIT Press. Approximation advances in neural information processing systems. *Policy Gradient Methods for Reinforcement Learning with Function*, 12:1057–1063, 2000.

A ML2 Appendix

Gong Xiangyu

A.1 Pseudo-code of Policy Gradient

Algorithm 1 Policy Gradient

Initialize θ arbitrarily
for each episode $[s_1, a_1, r_2, s_{T-1}, a_{T-1}, r_T] \approx \boldsymbol{\pi}_\theta$ **do**
  for t=1 to T-1 **do**
    $\theta \longleftarrow \theta + \alpha \nabla_\theta \log(\boldsymbol{\pi}_\theta(s_t, a_t) v_t)$
  end for
end for
return θ