Robotic Unicycle

Harvey Hughes

Supervisor:Professor Carl Rasmussen

May 24, 2020

Technical Abstract

The application of machine learning to calculate a control policy opposed to using control theory could prove to be highly useful when its not possible to determine system dynamics. However, success outside of simulations has been slow. A robotic unicycle is considered in this project to provide a complex enough problem to fully test the PILCO algorithm.

This project builds upon previous students work dating back to 2005. Throughout the projects history the unicycle being tested has evolved from a large and dangerous model, to a smaller model five years ago. in addition to model change different control algorithms have been used and extensively built up.

The last two years of project work has seen significant improvements and simplifications to the software and electronics. This helped to pinpoint bottle-necks in the learning process but unfortunately the smaller unicycle showed great difficulty at balancing.

Speculation that the unicycle roll limit of 17° impeded learning was thoroughly investigated among additional variables by running simulations. The simulations showed that a mechanical redesign is necessary for learning to occur. This design aimed to upgrade and simplify the physical robot to match the progress made in electrical and software. The new design has a roll limit of 47°, softens fall impact and reliably attaches all sensors.

Numerous problems were found in the old communications methods between phone, computer and Raspberry Pi. These problems introduced large variation in delay between readings, and operated too slowly to provide sufficient learning. For sufficient learning a frequency above 20Hz was

shown to be required. To solve these issues and provide additional simplicity UDP protocols were implemented for all transmissions. Unifying all transmissions this was enables the possibility of removing a computers involvement should it be desired in future.

Upon starting rollouts further problems were located and fixed. The initial orientation was measured to remove offsets created in the dynamics being learnt and therefore improve accuracy. Trajectories end when the unicycle hits the ground, this detection proved to be unreliable and rollouts became invalid. Additional constraint checks involving future predictions were implemented in addition to manual confirmation.

Unfortunately few rollouts were conducted due to problems charging the battery during the Covid-19 lockdown. The rollouts performed demonstrated.....

During simulations and real rollouts policy optimisation often plateaued, or only learnt to wander instead of stay at the origin. The implementation of a quadratic controller and policy exploration techniques is believed to alleviate these issues.

Future work should seek to investigate these issues, taking advantage of the updated and simplified mechanical, electrical, and software systems to identify key requirements.

Contents

I. Introduction

i. Project aims

The aim of this project is to determine a controller that is able to balance a robotic unicycle. The controller starts off with no information about the system, and is therefore required to slowly learn what are good actions to make. This mimics the way in which humans learn new tasks as children by trying new actions and seeing if they're beneficial. This is an ongoing project in the engineering department, dating back as far as 2005. In previous years success has been shown in simulations yet has so far been hard to achieve in real life. A major outcome of this project is to build upon existing work and fix new and outstanding issues in order to bring the success of a real unicycle closer to simulations.

This project aims to demonstrate the effectiveness of implementing the PILCO algorithm [2011. Rasmussen, Deisenroth] on more complex dynamic systems. PILCO

(a) *1m tall 1st model [2011. Rodrigo]* **(b)** *20cm tall 2nd model [2017. Weiser]*

Figure 1: *Previous unicycle iterations*

has previously been shown to tackle simpler problems such as a cart-pole system with high efficiency compared to other algorithms.

A unicycle is a suitable example to test on due to the complex nature of the problem, even for human riders. Balancing on the spot is additionally a less linear and therefore harder problem to solve then simply moving forward. This complexity allows the full extremes of the PILCO algorithm to be explored and therefore demonstrate all flaws and benefits of PILCO.

ii. Previous work

Work has been conducted on robotic unicycles within the engineering Department on and off since 2005 [2005. Mellors, Lamb, Maciejowski]. This unicycle was full sized at 1m, and weighed 35kg shown in figure 1a. Built by Mellors and Lamb the unicycle incorporated a wheel motor and horizontal flywheel motor which mimics the rotation arms can create while riding. Orientating the flywheel in the horizontal plane rater then the vertical plane increases the task complexity as the coupling of both motors is required to correct roll errors.

The problem was originally being solved using control theory approaches. This involved complex analysis of the dynamics in 2D and 3D conducted in 2007-2009 by D'Souza-Mathew [2008. D'Souza-Mathew] and Forster [2009. Forster]. Some success was demonstrated at balancing the unicycle. However, simplifications in the analysis resulted in non-linearities present in real life to not be considered accurately, therefore reducing performance.

In 2010 Mchutchon [2010. McHutchon] applied the RMLC (Reinforced Model Learnt Control) algorithm developed by Rasmussen and Deisenroth to the unicycle, and archived balancing of a unicycle with stabilisers. In 2011 building upon Mchutcons work Queiro [2011. Rodrigo] and Douglass [2011. Douglass] were able to modify the unicycle to balance unrestrained for up to 5 seconds.

The size of the unicycle posed serious safety risks, both in terms of large power usage and the need to limit the unicycle during tests. One way this was done was by attaching to bike racks [2008. D'Souza-Mathew] or suspending the unicycle from the department atrium ceiling [2011. Rodrigo]. These methods

while reducing the risk didn't eliminate it, and caused unnecessary restrictions and disturbances to the unicycle dynamics.

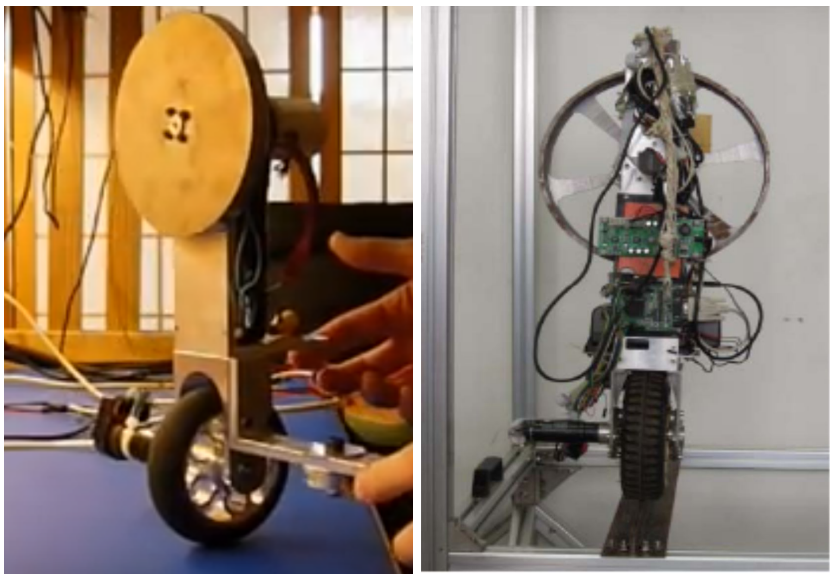
To increase safety a small model unicycle was constructed in 2015 by Tukiainen [**?**] as in figure 1b. PILCO was also implemented on the unicycle [2011. Rasmussen, Deisenroth]. The increased safety of the smaller unicycle allowed unconstrained motion to be tested more easily. Problems balancing the unicycle longer than 2s were encountered at this point, with concerns about identifying the cause for problems. In 2017 an overhaul of the software was done by Wieser [2017. Weiser], and in 2019 an electrical overhaul by Harris [2019. Harris]. This was to simplify the system and allow for more accurate troubleshooting.

iii. Examples of other unicycles

There are numerous examples of other balancing robotic unicycles. Many designs included a large vertical flywheel and use of control theory as seen in figure 2. Orientating the disk this way improves stability and was therefore proved successful at balancing. The improved stability is due to uncoupling the two torques, meaning that to correct a roll error just the flywheel can be used, and to correct a pitch error only the wheel needs rotating. A combination of both actions is required to fix roll errors on a unicycle with horizontal flywheel.

However, examples of unicycles with a horizonal flywheel can still be found. [1990. Vos, Flotow] Vos and Von Flotow analysed a large unicycle with this set-up and were able to balance to some success using control theory. This required multi region control policies to work, and therefore was very complicated.

The robotic unicycles discussed so far have all been developed using traditional control theory. Very few unicycles have been controlled with a machine learning approach. Simulations of a 2D unicycles were successfully balanced [2019. Davis] using standard keras and Tensorflow libraries. However, this approach simplifies the problem back to the much easier inverted pendulum, and additionally didn't deal with the problems associated with real rollouts. The lack of previous success at this particular task highlights its difficulty.



(a) *[2012. mhexrobot]* **(b)** *[2012. Daoxiong, Qi, Guoyu, Xinghui]*

Figure 2: *Other Unicycle designs*

II. Theory

i. Coordinate System

In order to fully define the unicycle state at each time step seven coordinates and their derivatives are required. These coordinates are shown in figure 3 giving the state as

$$\mathbf{x} = [\dot{x}\ \dot{y}\ \dot{\theta}\ \dot{\phi}\ \dot{\psi}_f\ \dot{\psi}_w\ \dot{\psi}_t\ x\ y\ \theta\ \phi\ \psi_f\ \psi_w\ \psi_t]^T$$

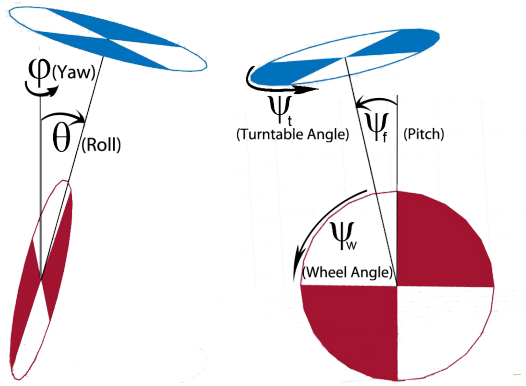


Figure 3: *Angles used to define position*

In order to define rotations in 3D to represent θ, ϕ, ψ_f correctly care must be taken to the order the rotations are applied. This is because rotations are non commutative in 3D unlike in 2D. The convention of Euler angles is used to solve this problem. [2017. Weiser]

There are 12 orders of rotation which can be used. XYZ or 123 convention is used on the unicycle as in equation 1. Where $R_A(\alpha)$ represents a right hand rotation about axis A of α.

$$R_{XYZ}(\psi_f, \theta, \phi)\mathbf{x} = R_X(\psi_f)R_Y(\theta)R_Z(\phi)\mathbf{x} \tag{1}$$

ii. Quaternions

As discussed previously rotations in 3D are difficult to combine due to their non-commutative nature. A handy way of dealing with consecutive rotations is to use quaternions. These are an extension of complex numbers to include three imaginary parts and are written as:

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$$

The following properties ensure quaternions obey the same mathematical rules as 3D rotations:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

A rotation about axis (x,y,z) by α can be defined using the following unit quaternion.

$$\mathbf{q}_{rot} = cos(\frac{\alpha}{2}) + sin(\frac{\alpha}{2})(x\mathbf{i} + y\mathbf{j} + z\mathbf{j}) \tag{2}$$

When post multiplied by the current state quaternion, the new updated position is calculated: $\mathbf{q}_{new} = \mathbf{q}_{old} \cdot \mathbf{q}_{rot}$.
The angle rotated during each time step can be easily found using the magnitude and direction of the angular velocity multiplied by timestep: $\theta = |\boldsymbol{\omega}|\delta t$. [2019. Harris]

iii. Gaussian Processes

Update with extra graphs

A Gaussian process is used to model the dynamics and gives a range of possible functions which fit observed data in a non parametric way as to not introduce model bias. A Gaussian process is a generalisation of a multivariate Gaussian to infinitely many variables. As a result a mean and covariance function are used to define it as in equation 3.

$$\begin{gathered} f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}, k(\mathbf{x}, \mathbf{x'})) \\ m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \\ k(\mathbf{x}, \mathbf{x'}) = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x'}) - m(\mathbf{x'}))] \end{gathered} \tag{3}$$

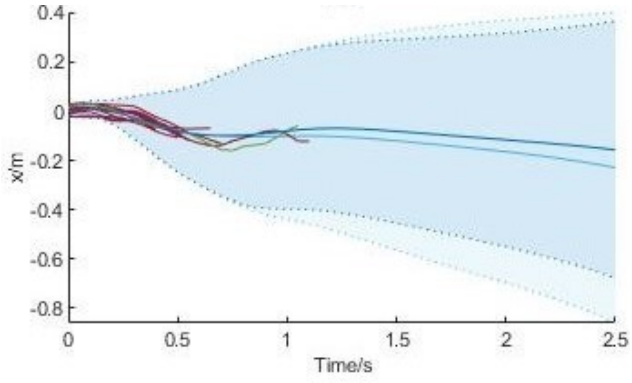
A GP eliminates model bias as a group of functions are fit to the observed data using the mean and covariance function. This ensures that all possible solutions to the dynamics are considered, with the most probably functions taking precedence in predictions.

iv. Optimal Control

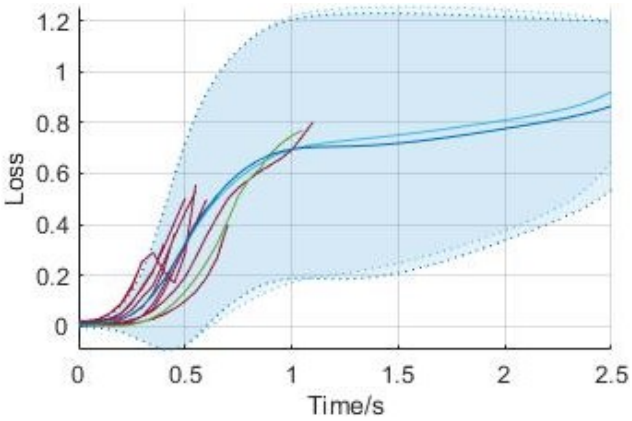
Write this later:

v. PILCO

In policy optimisation an algorithm called PILCO [2011. Rasmussen, Deisenroth] will be used. This method will optimise a policy to decide the best action $\mathbf{u}$ given the unicycles state. To determine optimum actions the expected loss over a horizon is evaluated under the current policy, as in



(a) *x Prediction*



(b) *Loss prediction*

Figure 4: *Predictions from PILCO approach*

equation 4.

$$\pi^*(\mathbf{x}) = argmin \sum_{0}^{t} c(\mathbf{x}^{(i)}) \qquad \mathbf{u}^{(i)} = \pi(\mathbf{x}^{(i)}) \tag{4}$$

A major advantage of PILCO is the ability to retain uncertainty throughout this process. This means an initial state distribution can be propagated forward to generate a state distribution at each time step. These distributions are used to calculate the average instantaneous cost in policy evaluation. The uncertainty grows over time and comes from an initial variation in start position, measurement noise, and process noise.

Figure 4a shows predicted x position, with predictions up to 0.5s showing low uncertainty. After this the model loses the ability to predict the state accurately and the state distribution widens. A similar trend is observed for the predicted loss at each time step in figure 4b.

PILCO uses an iterative approach to optimising policy. Initially a random controller is used to get sufficient data for a dynamic model. Policy optimisation and rollouts under the new policies can then occur until the task is either learnt or terminated.

vi. Cost Function

A cost function is required to evaluate the performance at each timestep. This takes the form of equation 5. Where a, h and θ_{max} represent positive constants, and d(**x**) is the geometric distance of the unicycle tip

from the starting position.

$$c(\mathbf{x}) = 1 - \mathcal{R}(\mathbf{x})$$
$$\mathcal{R}(\mathbf{x}) = exp(-\frac{a}{2h^2}d(\mathbf{x})^2 - \frac{\phi^2}{2(4\pi)^2} - \frac{\theta^2}{\theta_{max}^2}) \tag{5}$$

The cost function is how good actions are justified. For this reason as little information as possible about the dynamic system in question should be imparted vie this function. A purely geometric function using only d(**x**) was originally used. However, to limit excessive spinning in yaw, and reduce the effect of a small roll limit; the latter two terms were added.

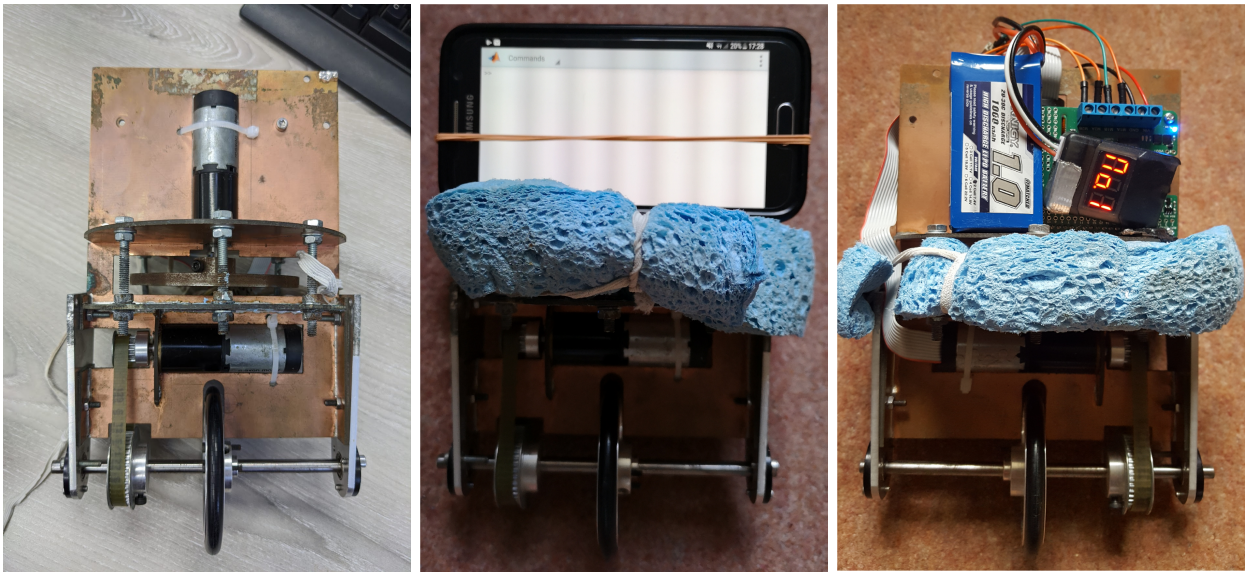
Other forms of cost function may be used such quadratic. The reason the exponential function was chosen was to take advantage of its saturation at +1. This provides a clear maximum cost and ensures that far away points don't dominate. This is especially important as a distribution is used to calculate costs, meaning distant points will have some probability of occurring.

III. Current Unicycle Set-up

i. Mechanical

Figure 5a shows the mechanical design of the unicycle. The unicycle is constructed from PCB boards made of copper coated fibreglass and subsequently welded together. This material is very light and stiff and helped keep the overall weight to about 1.2kg. The unicycle stands at 20cm tall in real life.

Two identical motors are used on the unicycle, these are Maxon 110134 [6]. These motors have maximum continuous torques of 6.31mNm and speeds of 3550rpm. To decrease output speeds, increase torque output and sensitivity; a 14:1 planetary gearhead [7] is attached to each motor. An additional 40:16 gear and belt system is attached to the wheel to further help. Resulting in maximum output torques of 0.205Nm and 0.513Nm for the turntable and wheel respectively. Each motor has an attached encoder which pulses 512 times per revolution, this can be used to accurately track the motor positions but is currently not in use.



(a) *Frame* **(b)** *Front [2019. Harris]* **(c)** *Back [2019. Harris]*

Figure 5: *Current set-up of unicycle*

The flywheel of the unicycle is mounted horizontally and cased within the frame to prevent injury and damage to the motors.
To prevent damage to the unicycle when falling a basic protection of foam was added to the unicycle. The unicycle currently crashed when a roll angle of 17° or pitch of 75° is achieved. The foam didn't stop the initial impact, only when the unicycle later rolled over.

ii. Electrical

The electrical components on the unicycle were simplified by Harris [2019. Harris] to only involve a phone, Raspberry Pi and Raspberry Pi motor board. This simplification was to reduce the number of components brought together for successful rollouts, helping reduce complexity and increase ease of troubleshooting.
The phone is used to take advantage of high quality sensors existing in a device most people own. The on-board gyroscope is used to update the current state, with the possibility of using the other sensors such as accelerometer to calculate initial position.
The Rasberry Pi is used to control the motor torques via the attached motor board. This replaced a normal micro controller and was chosen to allow future adaptability in design as a Raspberry Pi has lots of additional functionality over a micro controller such as wifi.
Attaching the phone and Pi involved rubber bands. This introduced lots of additional noise as the sensor could move between rollouts or during a rollout, therefore effecting the generalisation of learnt dynamics to all rollouts. To minimise this problem the phone was realigned by attaching and reattaching before each trial.

iii. Software

The main body of code is run in Matlab on an attached computer. This code is part of the extensive PILCO toolbox, the following steps are required when programming a scenario in PILCO, with each step using the structures and methods defined in the package.

- Define all state variables, and their use in policies, ode solvers, and dynamics models
- Number of rollouts to optimise policy over

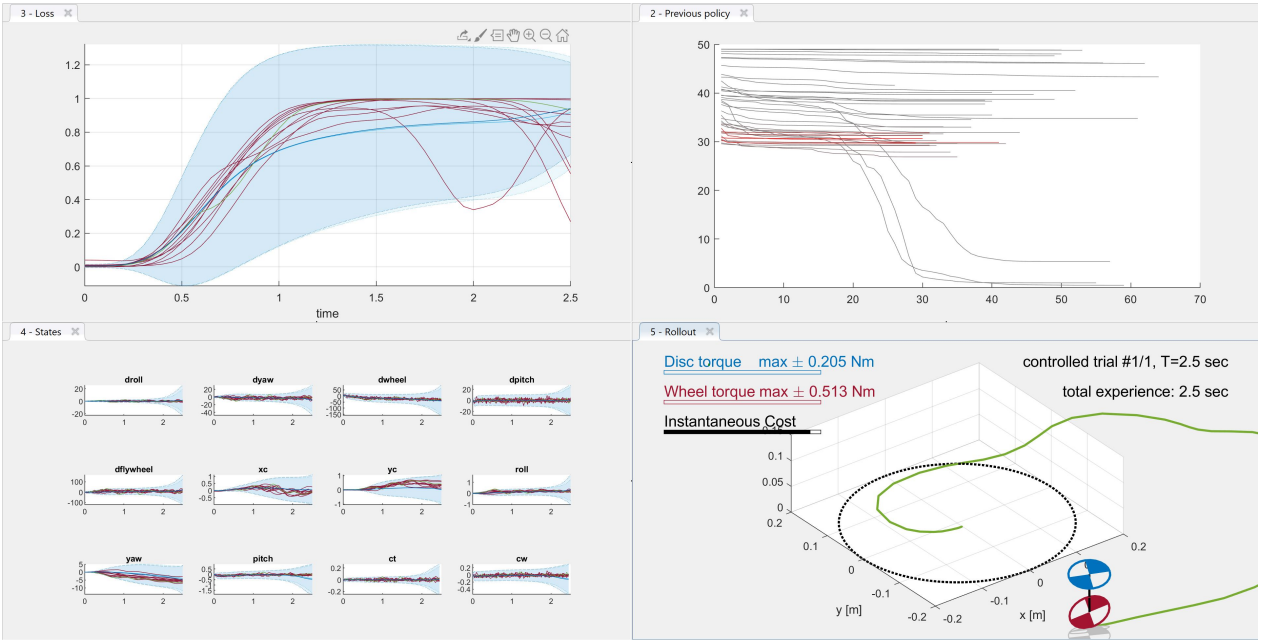


Figure 6: *Information displayed during rollouts*

- Number of random rollouts
- Horizon time
- Initial state distribution
- Policy structure, and optimisation methods
- Plant information including simulation dynamics
- Controller structure
- Dynamics function structure and optimisation
- Cost function
- Optional policy exploration methods
- Iterations of learning dynamics from new rollouts, then re-optimising policy function for future use can then occur

Figure 6 shows the information generated during rollouts. Shown in these plots is states, loss, unicycle animation, and policy improvements between iterations. The state window shows the trajectory of 10 rollouts under the current policy, with the rollout used in training displayed in green. With the predictive state uncertainty and mean displayed in light blue. The dark blue region indicates the updated predictive distribution after re-optimising the dynamics function using the new data. Multiple rollouts are displayed to get a better idea about the behaviours exhibited by the controller, and to check the chosen rollout is typical. The loss window shows the

same 10 rollouts and the cost distribution expected at each timestep. The animation displays the unicycle trajectory and traced path for the green rollout, the current actions and cost is also displayed. Policy learning can be analysed by looking at the policy window. This displays the expected loss, optimized 100 times, for each policy iteration. A horizontal line, or grouping of consecutive policy lines can indicate a reduction in learning rate. Often when little data has been observed the trajectory predictions can be overly optimistic, causing the predicted loss decrease rapidly in some iterations. This prediction is later reverted once more data has been observed.

Additional features such as replaying an animation can be performed by right clicking the desired trajectory line in the GUI, described in detail in by Weiser [2017. Weiser]. Log files are generated of each rollout to allow for easy post processing or reusing old data on modified algorithms.

During rollouts sensor readings from the phone are logged and used within Matlab by using a now discontinued feature of Matlab mobile. This feature allowed the connection of a phone and computer logged into the same Matlab account to share data eg sensor readings over wifi. Position calculations and subsequent policy evaluations were then performed within Matlab to determine motor torques. Motors were controlled using the following commands [2019. Harris] [4]:

- system(mypi, 'sudo python m/enabling_motors.py')
- system(mypi, 'sudo python m/motor_contorl_2.py -240 120')
- system(mypi, 'sudo python m/disabling_motors.py')

These commands opened a command terminal on the Pi through Matlabs integration to Pi package. In this terminal python files could then be executed to turn motors on/off, or change speeds.

IV. Initial Simulation Results

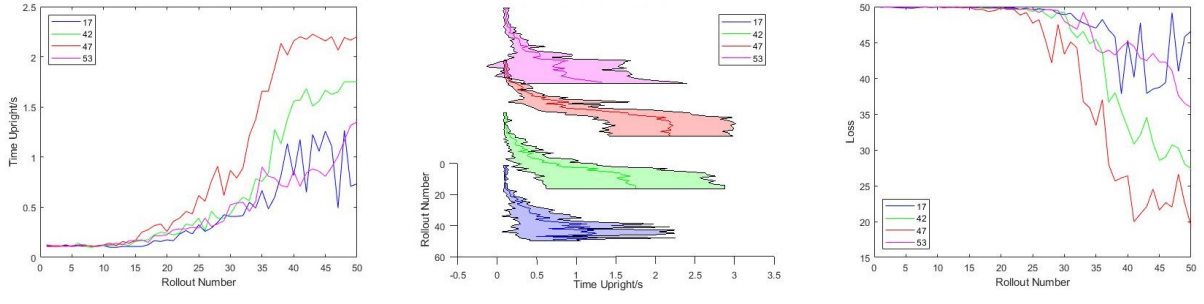
During previous years the Unicycles performance on the real model was relatively poor. A mechanical redesign was considered to fix some of these issues. In order to evaluate the performance of this redesign and additional factors simulations were conducted as they allowed a quick and easy way to alter variables.

i. Test Procedure

Two different measures will be used to determine success. The first being time upright as more rollouts occur. This is to determine any success at balancing the model has. The second is total rollout loss in order to see if success is achieved by staying near the origin instead of staying upright.

ii. Roll Limit

The first factor that would determine the benefit of a large mechanical redesign is the roll limit. In the unicycle model the roll angle cannot exceed 17°. This is thought to inhibit learning as initially the system can't balance and will hit this limit within very few time steps. This could mean the model has insufficient data to successfully learn a policy. Furthermore as the last measurement is removed in real trials due to high accelerations the data available is reduced more. This reduction will have a larger effect on small roll limits due to scaling in the cost function and can cause failure to appear to happen at a low cost. Which could in turn inhibit learning more.

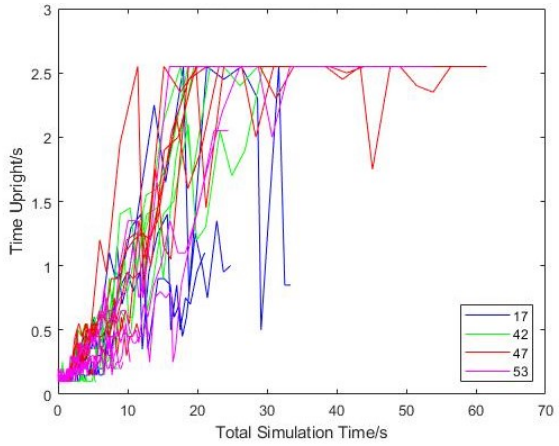


(a) *Average of six tests* **(b)** *With* $\pm\sigma$ **(c)** *Comparing against rollout loss*

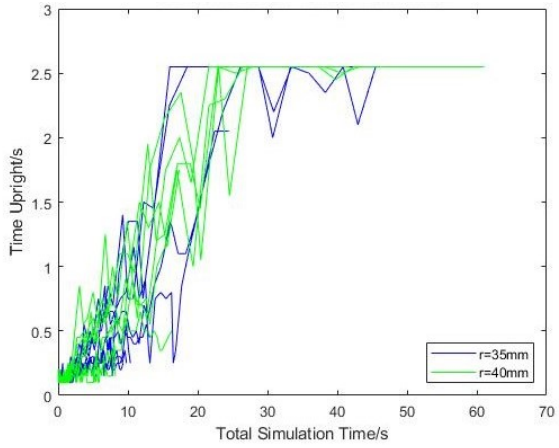
Figure 7: *Simulating the effect of roll limit* θ_{max}

Several designs were sketched to change the roll limit. 42° could be achieved with a new chassis and the same gears, 47° by changing an additional bracket, and 50° by increasing wheel size. Simulations of these can be seen in figure 7.

After 10 random rollouts the roll limit made little difference as in figure 7a. However, when policy optimisation started then a higher roll limit improved success greatly. The improvement between 42 and 47 degrees appears significant, a larger change to 53° caused learning to favour remaining near the origin instead of staying upright and moving away. This different learning method is also good because it shows that a larger limit allows new potentially more successful options for the optimiser. The plateau observed in tests is due to some tests consistently staying upright for the whole horizon time. When looking at rollout total loss in figure 7c the same pattern is observed with a higher roll limit reducing the overall rollout loss. The variation between simulations can be seen in figure 7b and shows greater reliability at higher roll limits, with a standard deviation of spread at 47° close to being higher then the mean at 42°. The spread is very important



(a) *Vary roll limit*



(b) *Vary wheel size*

Figure 8: *Simulated performance compared to experience time*

to create a reliable policy training method. The variation at 53° can once again be explained by some iterations learning to balance at the origin.

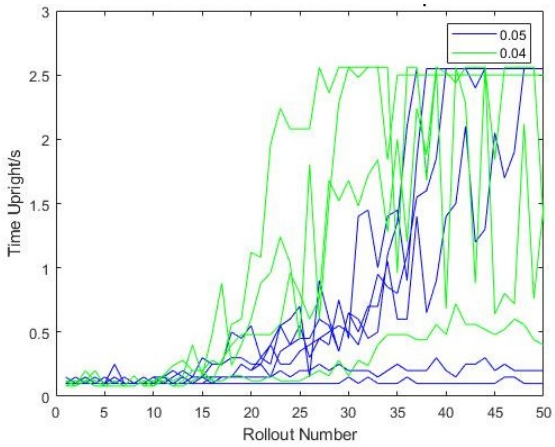
Figure 8a shows the performance against simulation time. This is to see if the greater performance is mainly due to additional data, and therefore better dynamics mod-

elling. All the plots give similar results, with the most successful trials balancing for 2.5s with only 20s of experience. Interestingly there are a few tests at 53° performing as badly as at 17°. This is due to the different balancing technique learnt, which follows a slower learning rate.

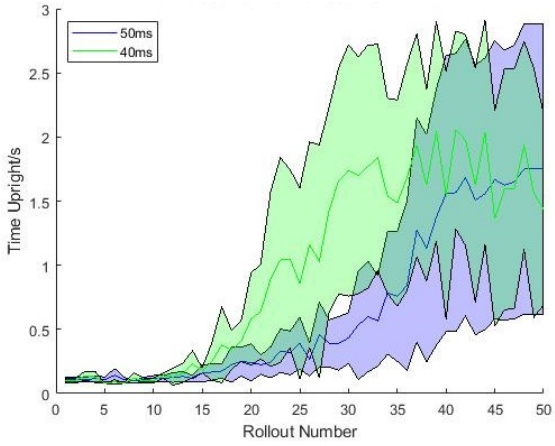
iii. Sampling Time

Sampling time is another important factor to consider when optimising the training situation of the real unicycle. A faster sampling time effects the following:

- Dynamics change slower and more linearly between readings, decreasing modelling complexity and improving predictions.
- A zero order hold controller is used on the motors, with torque calculated at the start of a step. As the unicycle moves this action is non optimum, therefore an error is introduced.
- More data points are collected, this can help build better models but at the expense of extra computational time.
- Every policy call injects noise into the predictions from process and measurement noise. This can cause a more uncertain trajectory prediction over the horizon time.



(a) *All rollouts*



(b) *Distribution ($\pm\sigma$)*

Figure 9: *Effect of sampling time change*

Figure 9 shows the results of changing timestep for a roll limit of 42°. The extra computation time at 40ms resulted in tests lasting upwards of 40 hours compared to 4, as a result fewer simulations were conducted. The lower performing trial at 40ms

showed promise as it learnt to balance in a different way to all other trials at this roll limit, it prioritised staying at the origin over staying upright. This shows that there is a possibility of the timestep greatly effecting learning ability. At the faster sample time the majority of trials learn faster for the initial 30 rollouts, after this performance became more similar between all trials, likely due to a finite horizon time being used. This shows that if enough iterations are performed you can eliminate the effect small changes to sample time have. For this reason a timestep of 50ms is a good compromise between learning rate and computation time.

iv. Wheel Size

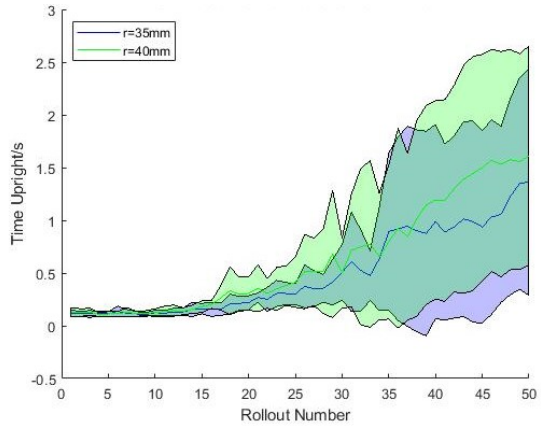
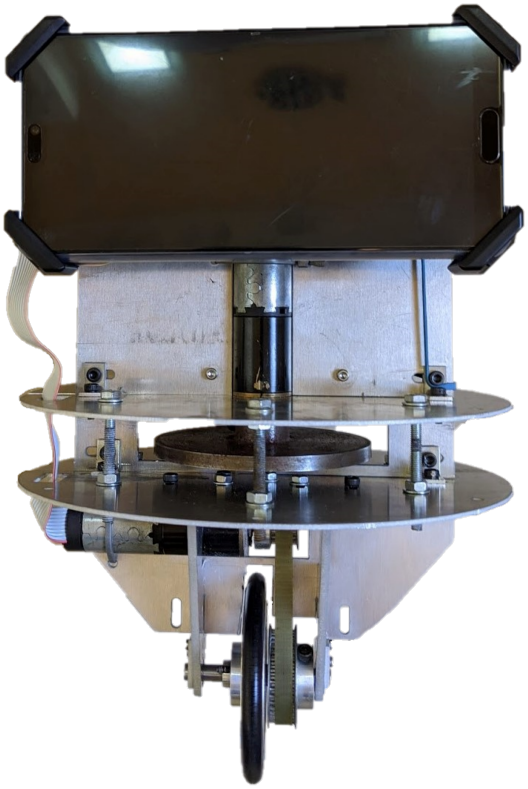
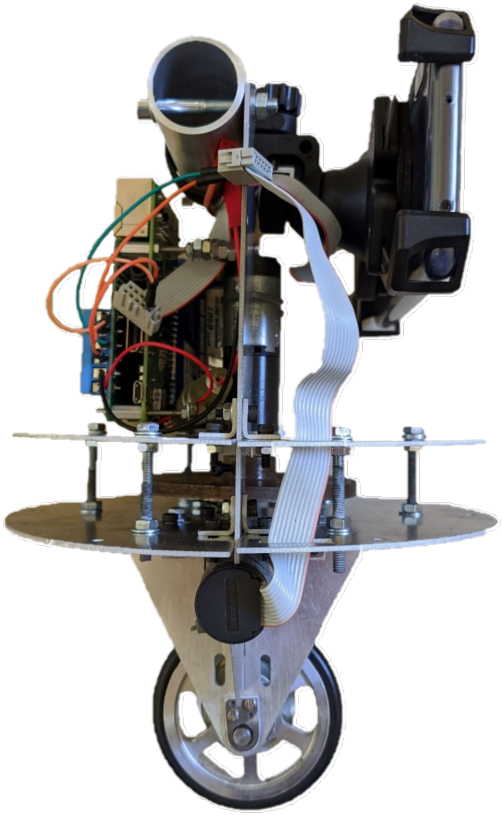


Figure 10: *Effect of wheel size when roll limit is 5=53°*

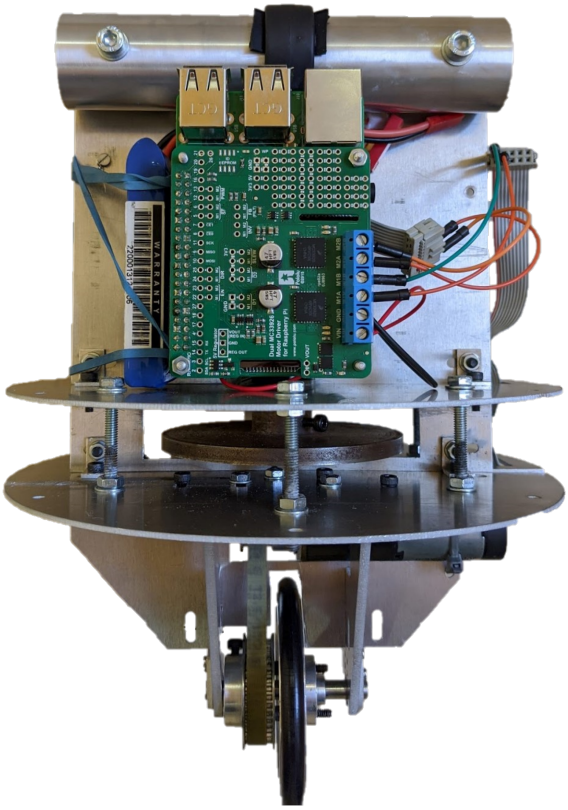
When testing a roll limit of 53° the change in wheel size wasn't considered. A larger wheel may effect performance as it increases torque output, and improves sensitivity. To see these effects a wheel with 35mm and 40mm radius were simulated. Figure 10 shows that small changes to wheel size have no effect on learning rate as their difference is explained by rollout randomness. For this reason the 35mm wheel at 47° roll limit appears the best choice of design.



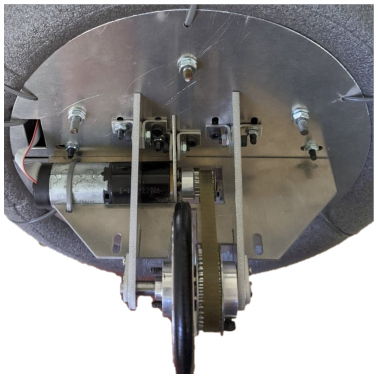
(a) *Front*



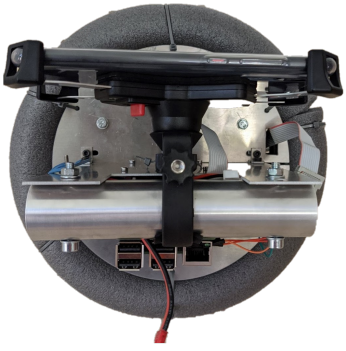
(b) *Side*



(c) *Back*

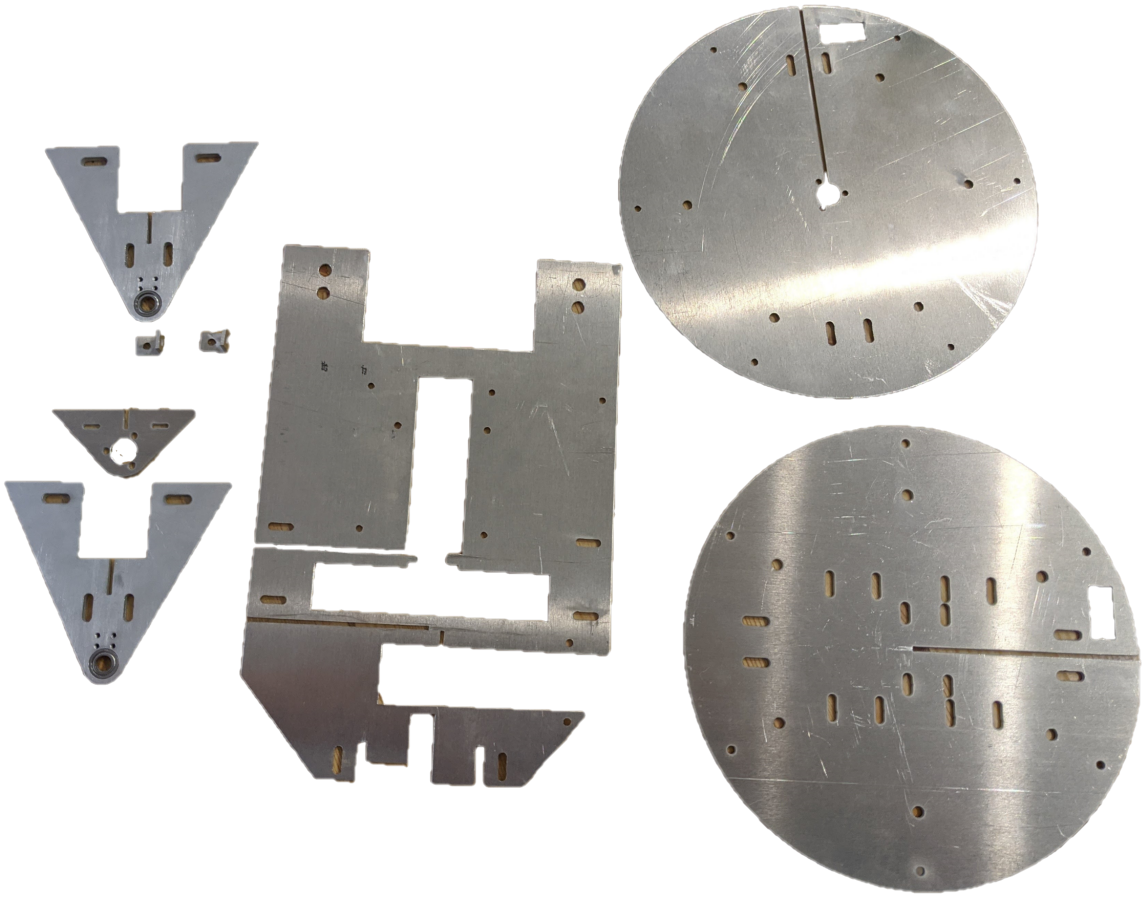


(d) *Bottom*

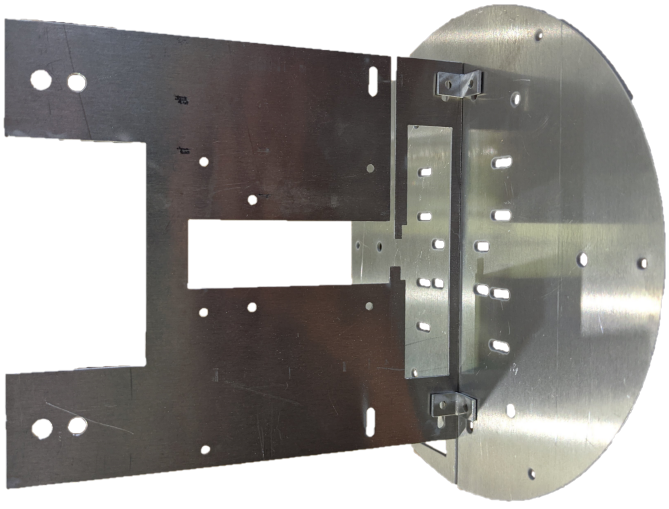


(e) *Top*

Figure 11: *New set-up of unicycle*



(a) *Parts*



(b) *Construction*

Figure 12: *Drawings of new unicycle design*

V. Mechanical Redesign

Speculations from previous years and simulations results conducted indicated that the unicycle was unable to learn to balance with a roll limit of 17 degrees. This prompted the need for a new unicycle design pictured in figure 11 and 12. In order to allow for a good design the following principles were adopted:

- Replace welding with bolts, to allow for easy part replacement if required in future
- Maintain rigidity when switching to bolts
- Keep parts as simple as possible
- Allow for variation in parts eg phone size

Figure 13: *New design for higher roll limit*

i. Changes

i.1 Roll limit

From simulations it was shown that increasing the roll limit to 47° was required for learning to be possible. Increasing the limit much further was deemed unnecessary by simulations.

Figure 13 shows a comparison between the old 17° design and new 47° design. This was made possible using mostly the same parts from the original design, with the exception of bearings. This was to maintain the gear ratio shown to be effective, and to reduces the need to source new parts that would make very marginal differences.

The first step to increasing roll limit was to bring the gears and motors as central as possible. This required decentralising the motor position and therefore CoM. The unicycle would still be able to learn with an off-centre design. However, this is cor-

Design	Simplicity	Wear	Damping	Replacing	Total
1	5	3	3	5	**16**
2	1	4	4	3	12
3	4	3	4	3	14

Table 1: *Protection system analysis*

rected for with off-centred Pi mounting. Once all the parts had been centralised it was found that the roll limit was 42° with the axle bracket being the limiting factor.
Three steps were implemented to reduce this limit. Reduce size of bearing holder, use smaller 10x6mm bearing, and minimise size of axle mount. These changes allowed the roll limit to become 50° as desired, with the extra 3° to allow for ample deceleration by the new protection system.

i.2 Protection

The next problem to solve was protecting the unicycle from the many falls it would experience. To be effective this would need to decelerate the unicycle at a slow rate, be easy to attach and fix if damaged, be mounted to not decrease the roll limit by too much.
Three designs were proposed, an evaluation of which can be seen in table 1. Design one consisted of a static material eg rubber or foam being attached, design two was a large ring attached to the main chassis by springs, and design three used an inflated inner tube. The design that proved to be the most effective at achieving the initial aims was a static material damper. This was due to its simplicity and ease of replacement, while still maintaining effectiveness. To decide the material various items were tested by hand. These were rubber tubing, foam insulation tubing, hose pipes, and silicon seals. Out of these materials only the foam tubing was the desired stiffness and thickness for the task. The tubing came with the added benefit of being the cheapest and has the option to increase stiffness if required by threading extra material inside it.

$$d = \frac{h - \frac{r}{2}}{tan\theta} - \frac{r}{2} - \sqrt{r^2 - \frac{t^2}{4}} = 86mm \quad (6)$$

The foam was to be mounted to the hori-

zontal metal disks protecting the flywheel. A roll limit of 47° was designed for the point of contact on the tube allowing for 3° of deceleration to occur. This corresponded to a displacement of 8mm in the foam. The required disk diameter was approximated using equation 6. Where t=26mm is the disk separation, r=15mm is the foam radius and h=115mm the height of the midpoint between disks.

i.3 Mountings

The previous method of mounting the phone used elastic bands. This resulted in lengthy repositioning of the phone between trials to minimise noise introduced by sensor position change. A design aim was to decrease this variation by providing a secure mounting that was capable of fixing any phone, in case this changed in the future.

To have reliable and repeatable phone position the attachment must resisting all degrees of freedom and have at least 3 reference planes for repeatable phone alignment. Due to these strict requirements and desire for size variability it was decided to use an existing phone mounting system. This would increase part size, but provide the desired functionality without needing to fabricate complicated parts. Different bike and car mounts were evaluated with the Greyfay PB04AC bike mount eventually chosen [5]. This mounts onto a handlebar like design at the top of the unicycle as in figure 11e.

The Raspberry Pi was attached using long bolts located on the opposite side to the phone. In addition to the battery these were located slightly off centre, see figure 11c. This countered the off centred motor position for the wheel, and successfully brought the CoM to the centre. The Pi was located further away from the central plane of the unicycle then desired, this was due to the inability to find suitably short screws, and spacers because of closures due to Covid-19. By shortening these screws the moment of inertia would reduce and could improve unicycle balance.

i.4 Material

On the old unicycle model a copper coated fibre glass was used, as it had high stiffness but low weight. Due to designing within solidworks and requiring some machined parts the EDG office who manufactured the parts last time were unable to help

Part	Mass/kg	x/m	y/m	z/m	About CoM / x$10^{-5}kgm^2$			About ground / x$10^{-3}kgm^2$		
					I_{xx}	I_{yy}	I_{zz}	I_{xx}	I_{yy}	I_{zz}
Phone	0.2	0.06	0.7	0.2	46	8	38	8.5	8.7	1.0
Back Plate	0.12	0	0	0.12	60	40	20	2.3	2.1	20
Disks	0.1	0	0	0.11	19	19	38	2.8	2.8	0.76
Bot Motor	0.12	0	0	0.09	4.6	0.7	4.6	1.0	1.0	0.05
Top Motor	0.12	0	0	0.15	4.6	4.6	0.7	2.7	2.7	0
Axel support	0.02	0	0.02	0.07	0.46	2.3	1.8	0.2	0.2	0.05
Flywheel	0.2	0	0	0.11	8.1	8.1	16	2.5	2.5	0.16
Pi + battery	0.15	0.04	0	0.16	14	8	6	4.0	4.0	0.3
Handlebar	0.067	0.015	0	0.22	1.3	12	1.3	3.2	3.3	0.3
Phone holder	0.1	0.02	0	0.21	4.2	2.1	2.1	4.5	4.5	0.06
Total								31.7	32.1	2.64

Table 2: *Moment of Inertia estimate*

again. Therefore a change in material was required to 1.6mm plate aluminium, with the manufacturing done in the Dyson centre workshop. This ensured that the frame would be stiff enough to limit unwanted vibrations without adding too much weight.

ii. Inertia

Table 2 shows estimations of the moment of inertia of the new unicycle about the ground contact point. The estimations involved approximating each part to be a simple object such as planes, disks or cylinders. Then using the formula from the mechanics data book the inertia about each parts CoM could be calculated and moved by the parallel axis theorem to the contact point. The estimated inertias are 31.7, 32.1 and 2.64 $10^{-3}kgm^2$ for the x,y and z axis respectively. The inertia about the x and y axis are very similar as desired, meaning the unicycle should fall more uniformly in all directions. An estimation on the old unicycle was conducted for I_{zz} [2019. Harris], Harris

assumed the unicycle was a cylinder and then checked his answer using measured angular accelerations. The value Harris calculated was $2.54x10^{-3}kgm^2$ which is very similar to the new unicycle model.

The inertia about the vertical z axis is 16.4 times that of the flywheel. This means that to conserve angular momentum the flywheel needs to spin this much faster then the maximum rate of change in yaw. The motor has a maximum load speed of 3550rpm, after the 14:1 planetary gear this corresponds to a maximum angular speed of 26.5rads^{-1}. Therefore a correction in yaw of up to 1.6rads^{-1} or a quarter rotation per second is possible. This speed seems high enough to enable learning.

$$\begin{aligned} T = mglsin\theta = I\ddot{\theta} \\ \ddot{\theta} = \frac{mgl}{I}sin\theta \\ dot\theta = \frac{mgl}{I}sin\theta t + c_1 \\ \theta = \frac{mgl}{2I}sin\theta t^2 + c_1 t + c_2 \\ t = \sqrt{\frac{2\theta I}{mglsin\theta}} \end{aligned} \tag{7}$$

The time required for the unicycle to fall without motor input can be calculated using equations 7. At t=0, $\dot{\theta}$ and θ = 0. This means that both constants are 0. By assuming the CoM is at l=0.16m and m=1.5kg the time to fall 45° is calculated to be 1.2 seconds. However, from measuring the unicycle fall this time in reality is nearer to 0.6s. This reduction in time by half is caused by the wheel rolling as the motors were off, therefore the assumption of rotation about the contact point is false, meaning I is reduced. While still present during rollouts this change will be less severe as powering the motors stops them moving freely. This means a falling time between 0.6-1.2s is expected. With a timestep of 50ms this allows 12-24 recordings and commands to be present, even in bad rollouts. When random actions are taken fall time will further reduce as most actions are bad, however not by much since each action is independent from the last so can counter bad actions. Therefore a large improvement on the last unicycle model is expected which initially saw around 3 readings before the 17°limit was reached.

iii. Design Evaluation

While constructing the unicycle it was apparent that the new design was more rigid and adaptable then the last model. Having the flywheel protection disks made out of one piece, instead of two welded to-

gether was the main source of this additional strength. The addition of the bolts and extra room also made assembling the model much easier which is promising if future modifications are needed. Some bolts are currently missing due to the inability to acquire more during the lockdown period of Covid-19. The addition of these bolts and ensuring all bolts and cable ties are attached tightly is very important for minimising vibrational noise in future test runs.

The attachment of the Pi and phone proved very successful, allowing access to all the ports on both devices. The handle bar was a secure way of attachment allowing for adequate variation in either phone orientation or size. Due to the mounting at the top of the unicycle the moment of inertia about the wheel when falling is increased by around 50%, as seen in table 2. This doesn't appear to be adversely affecting the unicycle performance as falling occurs slow enough to react to. To improve this in future weight could be removed from the inside of the handle bar, or mounting holes could be moved lower down the back plate. This movement would also reduce the vibrations carried up this plate, which add unwanted noise to the sensor readings. Once again this problem doesn't appear to be too large and much easier methods of reduction could be implemented such as stiffeners attached to the plate edges.

The foam protection ring proved to be very effective. Roll angle was only marginally reduced as calculated, and the unicycle is able to safely fall without supervision. This part can easily be replaced by simply buying a new tube [2] and 7 cable ties.

Due to a change in material and slightly increase in size the overall weight of the unicycle was increased from 1.2kg to 1.5kg. Inertia calculations and conducted rollouts show that the increase in weight and inertial still allows for good performance. However, if this proved problematic the easiest change would be to introduce a larger flywheel as there is room in the casing, or reposition the phone mounting. A larger flywheel would help increase yaw and roll adjustments, and moving the phone would reduce the largest contributor to I_{xx}, I_{yy} and therefore decrease topple speed.

VI. Software Changes

A number of problems were detected in the software running a real rollout. These faults were constraint detection and state calculation.

i. State Calculation

Previously state calculations incorrectly assumed wheel and flywheel angular velocities was directly the input torques. A slight correction to include gearing ratio, and no load speed was required to fix this. Additionally x,y position was reset at each measurement and therefore needed correcting.

ii. Constraint Detection

When the unicycle hits the floor the rollout should terminate and trim any readings made after the impact. This is the ensure training occurs on valid dynamic data, and the unicycle can't cause damage while fallen over. It was noted that this detection was unreliable and often failed.

The process of constraint detection followed: Implement policy $\mathbf{u}=\pi(\mathbf{x}) \rightarrow$ Get measurement $\rightarrow$ Calculate next state, if constraint exceeded then terminate. This caused problems because the constraints are checked under the current state, which is calculated using current $\boldsymbol{\omega}$ and last δt. Had the collision occurred between the last two measurements then the current state would be calculated using $\boldsymbol{\omega}$ after an impact, therefore incorrectly calculation angle.

In order to remedy this error a prediction one step into the future was added. This position was then checked against the constraints and noted for next timestep if exceeded. To check if a collision occurred as predicted $\boldsymbol{\omega}$ is checked against the last position. In the event of a collision the impact would cause large accelerations in approximately opposite direction to before impact. These accelerations would be larger then those possible with a change in torque meaning thresholding could detect collisions.

This new method greatly improved detection. However, some errors still occurred meaning a manual check was introduced into rollouts to ensure valid data

was passed into PILCO.

VII. Results

i. Phone

i.1 Connection

Previously the phone connected to the Matlab scrip on the computer using Matlab mobile and wifi. However, this feature has since been removed from Matlab mobile and delays caused by using wifi were believed to occur occasionally. This could introduce extra noise and incorrect state measurements, therefore decreasing learning ability.

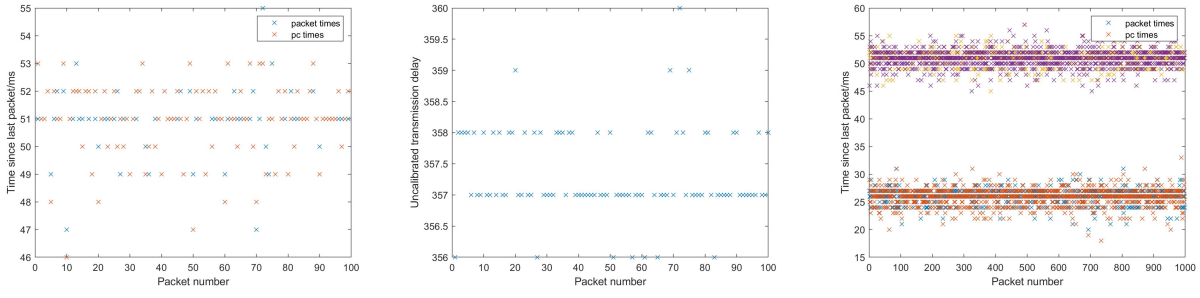
HyperIMU [1] was used to replace Matlab mobile. This allowed connection using wifi and USB. The transmission protocol followed was UDP using the packet layout:

$$[t, a_x, a_y, a_z, \dot{\alpha}_x, \dot{\alpha}_y, \dot{\alpha}_z \#]$$

UDP was chosen over serial communication protocols to take advantage of existing software and allow for wifi to easily be used again in the future if desired.

Tests were conducted to determine the reliability of this method. Figure 14a shows the time between consecutive measurements on both the phone side and computer side. A mean of 50.89ms and standard deviation of 1.27ms was recorded over 1000 measurements. Variation is undesirable but this is small and times are known so won't prove problematic. A similar spread was observed at lower sampling times as shown in figure 14c where a 25ms sampling time resulted in a mean of 25.88ms and standard deviation of 1.38ms. This shows that this software is suitable if faster sampling times are required.

Due to different clocks on the computer



(a) *Time between measurements* **(b)** *Uncalibrated delay* **(c)** *Different sampling times*

Figure 14: *Testing connecting by USB*

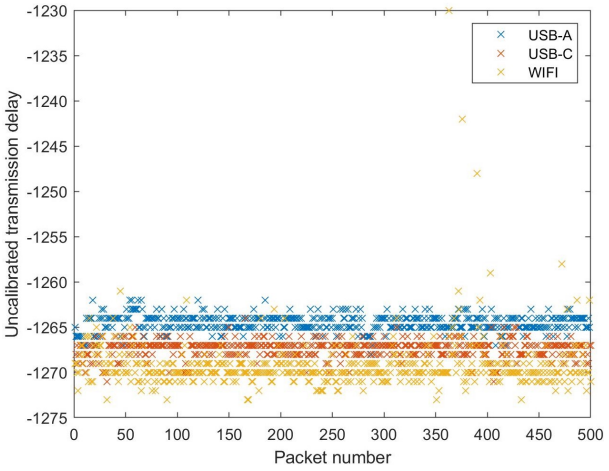


Figure 15: *Effect of changing transmission medium*

and phone true transmission delay wasn't possible to measure. Figure 14b shows this uncalibrated delay. The majority of packets take one of two times to transmit with a few outliers taking ±2ms. A low variation is desirable as it means actions can be applied for the correct length of time, as to not alter the trajectory away from the policy prediction.

Test	USB-A	USB-C	WIFI
1	0.99	1.05	3.02
2	1.24	1.26	1.25
3	1.10	1.02	1.84

Table 3: *Transmission delay σ in ms*

The delay over Three transmission mediums were compared in figure 15. A smaller number on these graphs corresponds to a faster transmission. It was noticed that the difference between mediums was too large, this was explained when conducting the tests in different orders. The two clock times drifted apart after each test causing the exact delays to be incomparable, therefore flipping test order resulted in the opposite order to the figure. However, the variation in delay is what is most important when deciding method as this is what can cause unexpectedly long applied actions. Wifi showed some delays of nearly a whole time step in figure 15. The standard deviation was calculated in Table 3. In test two the variation was almost identical amongst methods, but in the other tests wifi again showed greater variation. For this reason either USB connection is a better choice for use on the unicycle. However, the additional noise introduced by an extra cable effecting dynamics may outweigh the transmission delay advantage and will need investigating.

i.2 Orientation

Imperfect starting positions in real rollouts can cause a constant offset in position throughout tests unless its determined at

the start or learnt. Learning the offset is possible when training; however due to the offline nature of rollouts it can't currently be learn on new runs. Measuring the initial orientation is therefore required, which also allows accurate uncertainty measurements ensuring predictions incorporate as much information as possible.

To calculate the initial orientation the roll and pitch angles between transformed accelerometer reading and vertical, $\mathbf{a}_0 = [0, 0, -1]^T$ needs to be determined. A transformation is required from the raw sensor acceleration to convert between phone and unicycle co-ordinates, this transformation being:

$$\mathbf{a} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \hat{\mathbf{a}}$$

The unicycle has only two angular DoF initially, this means that one angle can be set to 0. For this we choose yaw as it only depends on the starting position and is penalised when higher. The angles are measured for two seconds before initiating the rollout to estimate initial uncertainty.

$$\delta\mathbf{a} = \mathbf{a}_0 - R\hat{\mathbf{a}}$$
$$\begin{bmatrix} \theta \\ \psi_f \end{bmatrix} = 2 \begin{bmatrix} tan^{-1}(\frac{\delta\mathbf{a}_x}{\sqrt{\delta\mathbf{a}_y^2 + \delta\mathbf{a}_z^2}}) \\ -tan^{-1}(\frac{\delta\mathbf{a}_y}{\sqrt{\delta\mathbf{a}_x^2 + \delta\mathbf{a}_z^2}}) \end{bmatrix} \quad (8)$$

Equation 8 calculates the initial euler orientation. This can then be easily converted to quaternion form with two quaternions rotations : $(1 + 0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k})R_Y(\theta)R_X(\psi_f)$.

ii. Raspberry Pi

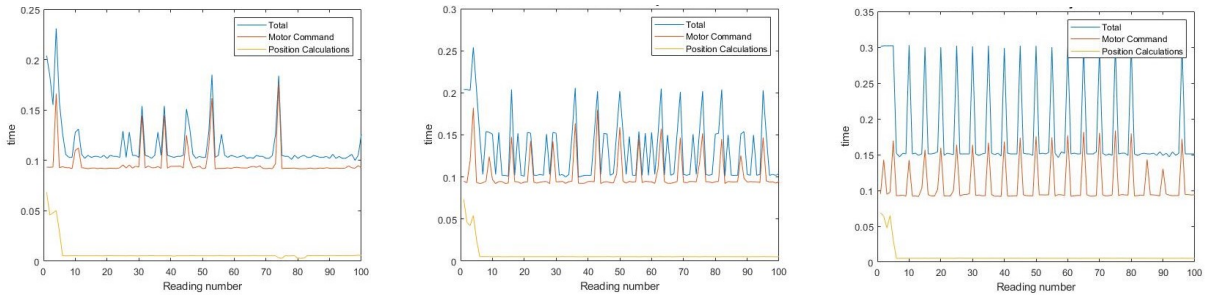
ii.1 Connection

Connection to the Raspberry Pi is done using the Matlab support package for Pi devices [3]. An initial set-up is required for this package. Involving connecting the Pi using a Ethernet cable, and following the package installation guide using the Raspberry Pi 3 Matlab image. Once set up the following command can be run in Matlab:

$$mypi = raspi('169.254.0.2', 'pi', 'raspberry')$$

This assigns the connection to the variable mypi, using the default IP address, username and password. This connection can then be used to send commands with:

$$system(mypi, 'sudo\ python\ m/filename.py')$$



(a) δt = *25ms* **(b)** δt = *50ms* **(c)** δt = *150ms*

Figure 16: *Testing connecting from Matlab to Pi using Matlab system command*

This command opens a terminal on the Pi and runs the python file specified. Three files were used [4] "enabling_motors" , "motor_control_2" and "disabling_motors". These turned on, set motor speed and turned off the motors respectively.

During testing it became apparent that there was a high degree of variation in time required to change motor speed. This is unwanted because it introduces error to the dynamic model which predicts for a constant timestep. If this variation also resulted in larger timesteps then 50ms the learning ability would be drastically reduced as less data is record initially, and the dynamics are less linear therefore harder to model. Increasing the capabilities of the dynamics model to include variable timesteps was considered. However, this would increase the problem dimensionality therefore the data collected may have to increase to effectively reduce uncertainty. Figure 16 shows this variation occurring when controlling the motors, for a number of different timesteps. Initial recordings took longer in each test, the execution time then settled to be at minimum 90ms, with some spikes up to 160ms. This meant that lots of sensor readings weren't being processed.

The cause of this delay was found to be the Matlab system command. This command waits for a completed message to be returned before the Matlab script proceeds. Due to the command opening a new terminal, and python file each time it took at minimum 90ms. Possible solutions were tested to avoid this; adding '&' at the end of a command stops the script waiting for a reply, however this is incompatible when

executed through a Pi. The command was also ran as a batch on a sub process, this allowed the script the proceed without delay but didn't successfully control the motors. The implemented solution looked to minimise the need to open a new python script and terminal for each change in motor speed. For this to be implemented UDP was once again used to send motor commands. The use of UDP allows for the future possibility of removing a computers involvement in rollouts as both the Pi and phone communicate with it. A script is ran for the duration of the rollout on the Pi, this listens to a UDP port and actions any commands sent. To initiate this script and avoid the need to wait for a response an alternative to the system command was used:

$$openShell(mypi);$$

This opens a SSH connection to the Pi and a terminal on the computer. The following command is then run in this terminal to initiate listening on the Pi:

$$sudo\ python\ m/start\_rollout.py$$

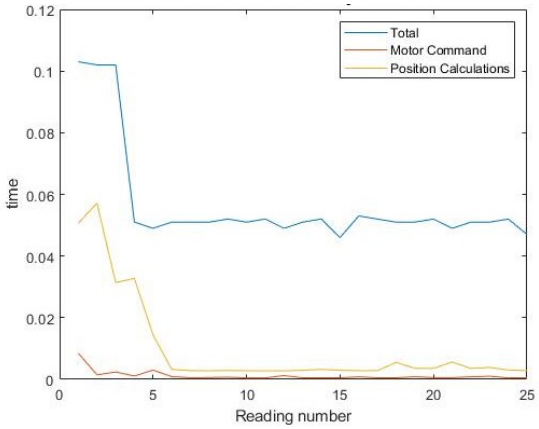


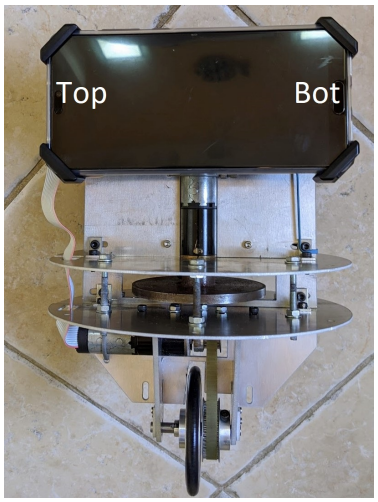
Figure 17: *Testing connection to Pi sending torques by UDP*

The terminal confirms a live connection and displays motor commands sent using the following format:

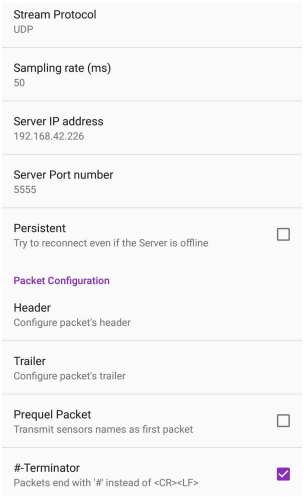
$$[Flywheel\ Speed,\ Wheel\ Speed]$$

The script is terminated by sending # in this command, instead of numbers.

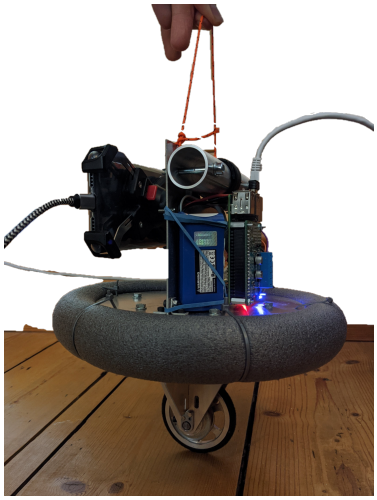
Figure 17 shows the success this method has at bringing the timestep to the desired length, and reducing the variance in time from 0.038 to 0.017. Problems still occur for the initial commands which take twice as long, this was fixed by optimising the rollout loop in Matlab ensuring variables are initiated beforehand, thus decreasing calculation time.



(a) *Phone set-up*



(b) *HyperIMU settings*



(c) *Release position*

Figure 18: *Steps in set-up*

iii. Rollouts

iii.1 Test Procedure

Mechanical Set-Up

- Attach phone as in figure 18a with the top of the phone facing to the left of the unicycle. Ensure the phone attachment is vertical and secure
- Attach the Raspberry Pi using an Ethernet cable to the computer running Matlab
- Connect phone to computer using a USB-A/C cable
- Connect Raspberry Pi to 5V power source, and attach on board battery

Software Set-Up

- Turn on USB tethering in the phones settings (settings/network and internet/hotspot and tethering). This often works more reliably with the phone in aeroplane mode
- Find the ipv4 address of the attached computer. On windows this is found by typing ipconfig into a command window
- Open HyperIMU on the phone and go to settings and enter the IPv4 address, port 5555 and desired sample rate(this is often 1ms too slow) as in figure 18b

- Save these settings and click start logging from HyperIMU

Rollouts

- In an open flat space position the unicycle, and run "doitreal.m" script from the connected computer
- A SSH window from the computer to the Pi will then open. Typing the command "sudo python m/start_rollout.py" will initiate listening mode on the Pi, and display all motor torques and time between subsequent messages
- Ensure the unicycle is held still and vertical, either by holding onto the frame or string as in figure 18c
- The orientation is then measured for a few seconds before the motors start, at this point the unicycle is released
- After the unicycle has fallen the motors should terminate, and confirmation of a successful rollout is required in Matlab
- Policy optimisation and dynamics learning then ensues, the unicycle could be disconnected during this step
- This is repeated until the desired rollouts are completed

iii.2 Tests

Due to limited access to the department caused by lockdown, and inability to find the correct cable for the battery charger minimal tests were conducted. The results of these tests can be seen in figure **??**. Limited success at balancing was achieved, this is likely attributed to low battery charge and insufficient rollouts performed. It was noticed that in one test the original policy learnt set the flywheel torque to maximum, and only after subsequent optimisations was this changed. This could signify the flywheel is currently too small for the unicycle, and a larger one may be required.

Project Changes Timeline	
Michaelmas	• Understand PILCO and its repository • Solve run time error caused by windows • Draft new unicycle designs • Simulate roll limits • Redesign and test communication to phone • Implement original state measurement
Lent	• Simulate wheel size • Simulate timestep • Evaluate unicycle designs and create CAD model • Get parts manufactured • Dismantle old unicycle and assemble new model • Test motors are working
Easter	• Increase motor command execution speed • Correct rollout state calculations • Improve constraint detection system • Research methods to incorporate encoders • Perform real tests • Research exploration techniques and controllers
Key	Mechanical Electrical Software

Table 4

VIII. Conclusions

i. Progress

This project continues on from previous years work. The unicycle was shown to be successful at learning controllers in simulations, but various issues inhibited learning on the model unicycle.

The work conducted on this project has taken a variety of forms as mechanical, electrical and software changes were required. A summary of this is seen in table 4. This can be broken down into simulations evaluating important parameters for learning, mechanical redesign implementing these discoveries and improving old communication systems to reduce noise and variation.

Initially simulations were conducted to establish the effect certain parameters had on learning. It was found that a new model was required for learning to occur due to the inability to learn at roll limits as low as 17°. Simulations different timesteps also provided additional insight. By reducing timestep from 50ms to 40ms learning occurred with fewer policy iterations, even allowing controllers to learn to balance at the origin and not wander off. The reason for this improvement is due to; increased linearity, additional data, and optimum torques active for longer. However, at 40ms the training time could be 10x longer then at 50ms. For this reason additional experiments should be conducted to determine the best timestep between 40-50ms to compromise speed and effectiveness.

As simulations showed inadequate learning on the current unicycle model, a new model was designed. This model would also address problems in fall protecting, sensor stability and repeatable test procedures. The design increased the roll constraint to 47°, increased simplicity while maintaining rigidity. The increased size of unicycle sparked worries about insufficient motor power. However, simple changes could be made to correct this if required.

Data communication between Matlab, phone and Raspberry Pi was completely redesigned. Delays in wifi communication and sending motor commands were solved by changing all transmissions to use UDP protocols. An additional benefit of this approach enables direct communication be-

tween Pi and phone. This means that in future rollouts could be performed without an attached computer.
Unfortunately few real rollouts were conducted..

ii. Future Work

Future work should first seek to evaluate the performance of the new model, as limited rollouts were possible this year. These investigations should aim to determine weather the increased unicycle inertia is holding learning back. If so changes to phone position and flywheel size should be made to reduce this effect.
The last three years of project work have seen updates to the software, electrical and mechanical set-up. This means the unicycle is now at a point where a variety of factors can be investigated more deeply, to discover what techniques are required for problems of this form when using the PILCO algorithm.

ii.1 Encoder

Currently the wheel and flywheel angular velocities are calculated using the input torques. This approximates the relationship to be linear, which becomes less true the faster and further from upright the unicycle becomes. This approximation causes inaccuracies in the trained dynamics model introducing process noise, and increases learning complexity.
In order to correct this error the motors should be measured directly. This is done using the attached encoders [8]. Pulses are generated 512 times per revolution, which under normal motor speeds produces pulse frequencies up to 50kHz.
Both positive and negative pulses require counting to accurately determine speeds. This could be done using additional hardware or for simplicity interrupt routines on the Pi [2013. Hamlin]. The motor states could then be transmitted to Matlab simply using UDP.

iii. Quadratic Controller

All simulations and rollouts conducted so far have used a linear policy control function to determine the best action from the current state as in equation 9. There is the question of whether a linear policy is sufficient to control a unicycle. A quadratic policy of the form of equation 10 may be

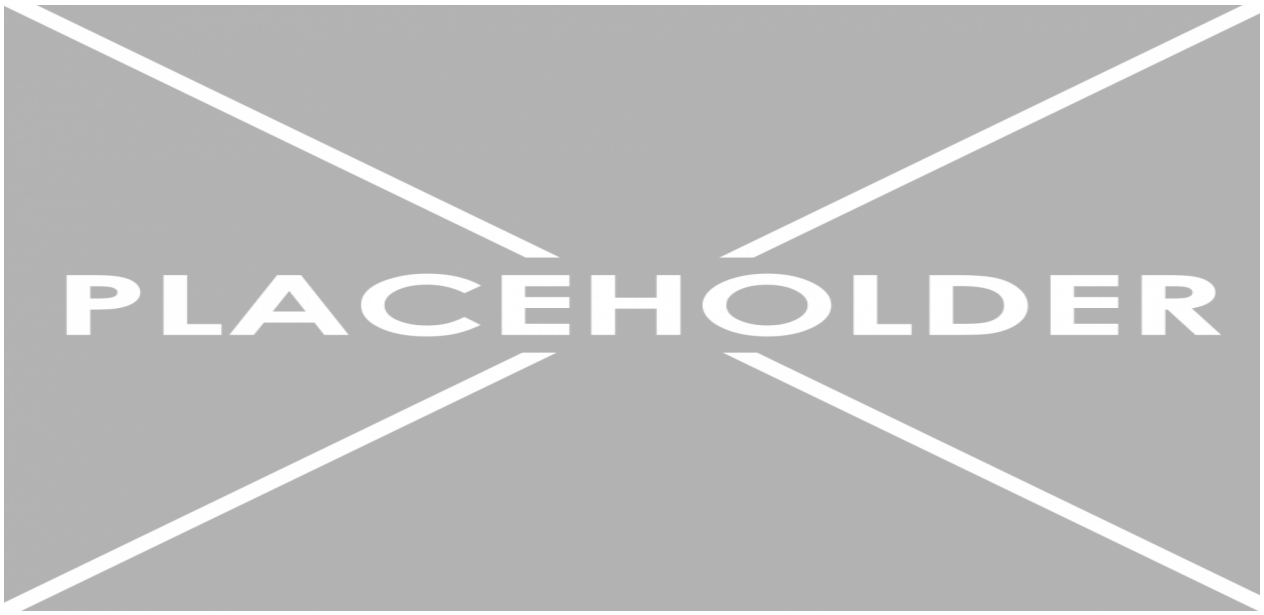


Figure 19: *Optimum torque directions*

more suitable.

$$\pi(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{p} \tag{9}$$

Figure 19 depicts the optimum action to be taken for the state space covered by $[\theta, \psi_f]$. This ideal action is the action which minimises the distance or angle moved in that direction to return the unicycle to the desired position of $[\theta, \psi_f]$ = [0,0].

To correct pitch a linear policy is clearly sufficient. When the unicycle tilts forward a negative torque on the wheel is desired, and vice versa for backwards tilt. However, when looking at correcting both pitch and roll a quadratic policy appears to be required. This can be easily visualised as the flywheel rotating away from the vertical axis within each quadrant. The vertical axis corresponds to a correction in just pitch which is linear. A torque away from this position results in the unicycle moving towards it. This means that the direction of torque required on the flywheel alternates between each quadrant, and is effectively an XOR problem. [2011. Rodrigo]

$$\pi(\mathbf{x}) = \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{W}\mathbf{x} + \mathbf{p} \tag{10}$$

Upon further thought this may not be a problem. The optimal solution may be quadratic but by rotating the other direction the same axis where only pitch is being corrected can be achieved. This can involve a much larger angle of up to 180° to work. When roll is small the two directions be-

come more similar, and therefore shouldn't pose a problem for most states the unicycle will visit. This non optimal route allows a linear policy to work where two quadrants have the optimal torque direction and two have the sub-optimal direction. Weiser believed that with a linear controller the unicycle learnt to either always fall forward or backwards in order to remain in the optimal regions. [2017. Weiser]

Testing using quadratic policies could be performed in future to allow an exact evaluation of using a linear approximation. This approximation is useful because the policy parameters optimised over are greatly reduced in a linear model. Therefore training speed is far quicker.

iii.1 Exploration

During simulated rollouts it became clear that it was likely policy optimisation could plateau. After this only minor improvements to trajectories were made. This may be caused by a linear policy being inadequate and therefore only so much progress is possible, normally resulting in trajectories drifting around the origin.

Another possible fix is to introduce policy exploration. Currently the policy iteratively optimised based off observed data, called exploitative learning. This means that local optimum can stop learning, and unvisited areas of the state space will not be explored. Exploring these areas, or jumping outside of current policy optimisations may be required to effectively learn the problem, even if this requires additional computation time.

Various systems for this exploration exist within the PILCO repository. One particularly interesting method involves fantasising how data collected from new regions could reduce model uncertainty and therefore allow smart exploration to occur. [2016. Mcallister]

Implementing methods such as this could prove vital to success on the real unicycle.

iii.2 Removing Matlab from rollouts

Currently the unicycle is incapable of running independent from a computer. This is slightly impractical and additional wires may become a nuisance if balancing is achieved. For these reasons it may be beneficial to remove the need for a computer and Matlab script during rollouts. This would require writing the policy function onto the Pi, allow trajectory, and policy

transfer onto the Pi, and using a UDP port
between phone and Pi.

References

[1990. Vos, Flotow] D. W. Vos and A. H. Von Flotow, "Dynamics and nonlinear adaptive control of an autonomous unicycle: Theory and experiment," in 29th IEEE Conference on Decision and Control, pp. 182187, IEEE, 1990.

[2005. Mellors, Lamb, Maciejowski] Mark Mellors and Andrew Lamb and J Maciejowski. , 2005 , roboticunicycle.info

[2008. D'Souza-Mathew] Neil D'Souza-Mathew. , "Balancing of a Robotic Unicycle". , MEng Thesis. CUED. , 2008, roboticunicycle.info/documents/MyFinalReport.pdf

[2009. Forster] David Forster. "Robotic Unicycle" MEng Thesis. CUED. 2009

[2010. McHutchon] Andrew McHutchon, "Machine learning for Control". MEng Thesis. CUED. 2010

[2011. Rasmussen, Deisenroth] M. P. Deisenroth and C. E. Rasmussen. , "PILCO: A Model-based and Data-Efficient Approach to Policy Search". , 2011., http://www.icml-2011.org/papers/323_icmlpaper.pdf.

[2011. Rodrigo] Rodrigo Queiro. "Machine Learning for Control". MEng Thesis. CUED. 2011 https://github.com/drigz/IIB-Project/blob/master/iibproject.pdf

[2011. Douglass] Alan Douglass. "Machine Learning for Control". MEng Thesis. CUED. 2011

[2012. mhexrobot] mhexrobot. 2012. https://edgetriggered.wordpress.com/category/unicycle-robot/

[2012. Daoxiong, Qi, Guoyu, Xinghui] G. Daoxiong, P. Qi, Z. Guoyu, and L. Xinghui, "Lqr control for a self-balancing unicycle robot on inclined plane". 2012.

[2013. Hamlin] Jay Hamlin. 2013. https://www.raspberrypi.org/forums/viewtopic.php?f=72&t=33880

[2016. Mcallister] Rowan McAllister. 2016. "Predicting Next-Step Loss Distributions in PILCO"

[2017. Weiser] Eric Weiser. "Robotic Unicycle" . MEng Thesis. CUED. 2017

[2019. Harris] Arsalan Harris. "Robotic Unicycle" . MEng Thesis. CUED. 2019

[2019. Davis] Matthew Davis. 2019 https://www.mdavis.xyz/unicycle/

Software and Hardware

[1] Ianovir: HyperIMU app ianovir.com/works/mobile/hyperimu/

[2] Protective foam. https://www.diy.com/departments/climaflex-polyethylene-foam-pipe-lagging-l-1m-dia-15mm/5413257001143_BQ.prd

[3] Matlab support package for Raspberry Pi. https://uk.mathworks.com/help/supportpkg/raspberrypiio/index.html

[4] Python library for the pololu dual mc33926 motor driver for raspberry pi.. https://github.com/pololu/dual-mc33926-motor-driver-rpi. Accessed:23-05-2019

[5] Phone mount. https://www.amazon.co.uk/dp/B07GQLYPZZ/ref=cm_sw_em_r_mt_dp_U_hfBXEbFJZDY22

[6] Maxom Motors. https://www.maxongroup.com/maxon/view/product/110134

[7] Maxon Planetary Gears 1:14. https://www.maxongroup.com/maxon/view/product/134158

[8] Maxom encoder - 512 pulses/rev. https://www.maxongroup.com/maxon/view/product/sensor/encoder/Magnetische-Encoder/ENCODERMR/ENCODER-MR-TYPM-128-512IMP-2-3KANAL/201937

IX. Appendix

i. Risk Assessment Review

ii. Covid-19