Robotic Unicycle

Final Report

Harvey Hughes

Supervisor:Professor Carl Rasmussen

May 17, 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 with tests into roll limit, sampling time, phone connection and initial orientation being tested first in order to brooch the gap between simulation and real success.

a

Contents

I. Introduction

i. Project aims

The aim of this project is to create 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 from previous years where success has been shown in simulations yet has so far been hard to achieve in real life. A major outcome of this project is the build upon previous years work and fix existing issues, in addition to solving new ones 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 [1] on more complex dynamic systems. PILCO has previously been shown to tackle simpler problems such as a cart-pole system with less data and high speeds 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

(a) *1m tall 1st model [6]* **(b)** *20cm tall 2nd model [8]*

Figure 1: *Previous unicycle iterations*

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 in PILCO.

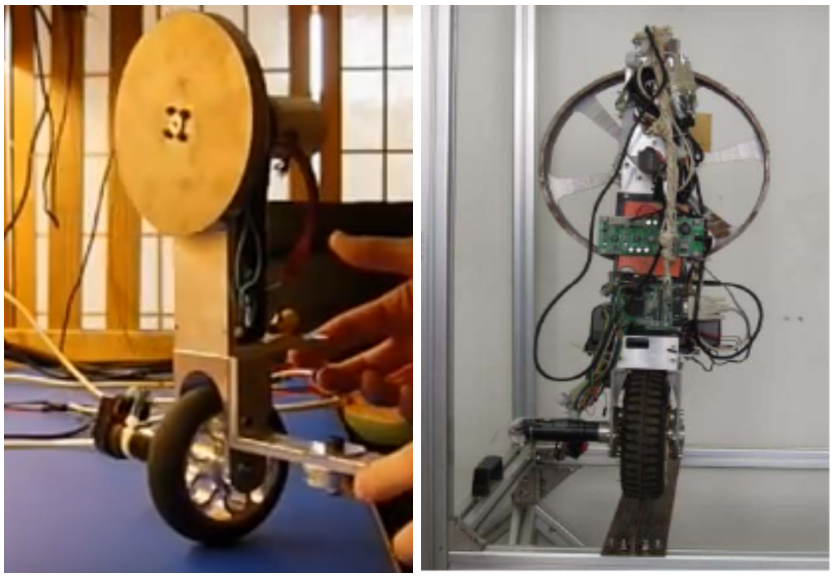
ii. Previous work

Work has been conducted on robotic unicycles within the engineering Department on and off since 2005 [2]. This unicycle was full sized at 1m, and weighed 35kg shown in figure 1a. Built by Mellors and Lamb the unicycle involved a wheel motor and horizontal flywheel motor to mimic the rotation arms can create. Having the flywheel in the horizontal plane increases the complexity over other designs with in in the vertical plane, as it couples the two motors together when a roll angle is corrected.

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 [3] and Forster [4]. 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. In 2010 Mchutchon [5] applied the RMLC (Reinforced Model Learnt Control) algorithm developed by Rasmussen and Deisenroth to the unicycle, and acived balancing of a unicycle with stabilisers. In 2011 building upon Mchutcons work Queiro [6] and Douglass [7] 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 replacing large components and the need to limit the unicycle during tests. One way this was done was by attaching to bike racks [3] or suspending the unicycle from the department atrium [6]. These methods while reducing the risk didn't eliminate it, and caused unnecessary simplifications or disturbances to the unicycle dynamics.

To increase safety a small model unicycle was constructed as in figure 1b. PILCO was also implemented on the unicycle [1]. The increased safety of the smaller unicycle allowed unconstrained motion to be tested more easily. Work began on the miniature unicycle in 2015 by Tukiainen [**?**]. Problems in balancing the unicycle were encountered at this point, with concerns about identify-



(a) *[11]* **(b)** *[12]*

Figure 2: *Other Unicycle designs*

ing the cause for problems. In 2017 an overhaul of the software was done by Wieser [8], and in 2019 an electrical overhaul by Harris [9]. This was to simplify the system and allow for more accurate troubleshooting.

iii. Examples of other unicycles

There are numerous examples of other work conducted on balancing robotic unicycles. Many of these involved a large vertical flywheel and use of control theory as seen in figure 2. Orientating the disk this way improves stability and was therefore quite 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 the wheel only needs rotating. A combination of both actions is required to fix roll errors on a unicycle with horizontal flywheel.

There has been work done on unicycles set-up with a horizontal flywheel. [13]. Vos and Von Flotow analysed a large unicycle and were able to balance to some success. 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 [14] using standard keras and Tensorflow libraries. However, this approach simplifies the problem back to a much easier inverted pendulum, and didn't deal with the problems associated with real models. The lack of previous success at this particular task highlights its difficulty.

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

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

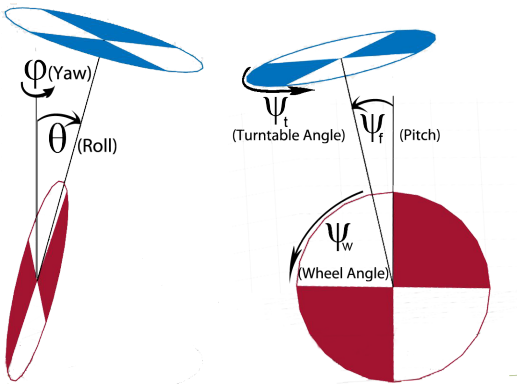


Figure 3: *Angles used to define position*

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. [8] 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 allow quaternions obey the same 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 [**?**]

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

$$\mathbf{q}_{new} = \mathbf{q}_{old} \cdot \mathbf{q}_{rot}$$

iii. Optimal Control

iv. PILCO

In policy optimisation an algorithm called PILCO [1] 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 equation 2.

$$\begin{aligned} \pi^*(\mathbf{x}) &= argmin \sum_{0}^{t} c(\mathbf{x}^{(i)}) \\ \mathbf{u}^{(i)} &= \pi(\mathbf{x}^{(i)}) \end{aligned} \tag{2}$$

v. Gaussian Processes

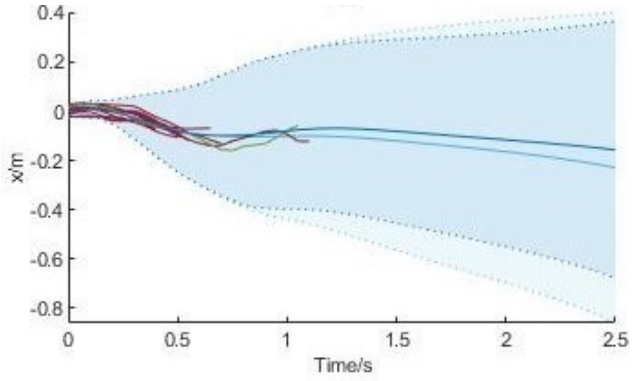
A Gaussian process is used to model the dynamics and gives a range of possible functions which fit observed data in a non parametric way as to not introduce model bias. A Gaussian process is a generalisation of a multivariate Gaussian to infinitely many variables. As a result a mean and covariance function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

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

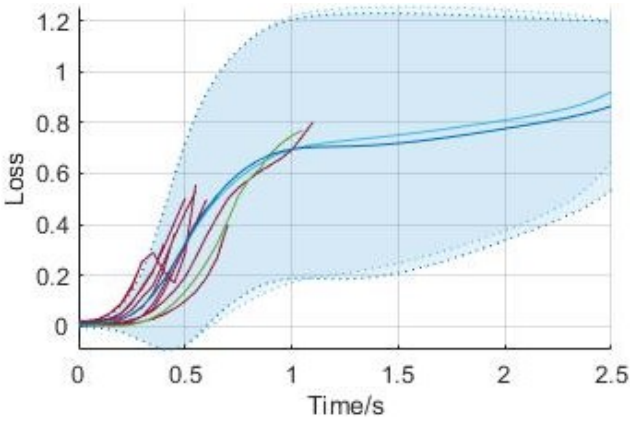
A major advantage of PILCO is the ability to retain uncertainty throughout this process. This means an initial state distribution can be propagated forward to generate a state distribution at each time step. These distributions are used to calculate the average instantaneous cost in policy evaluation. 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.



(a) *x Prediction*



(b) *Loss prediction*

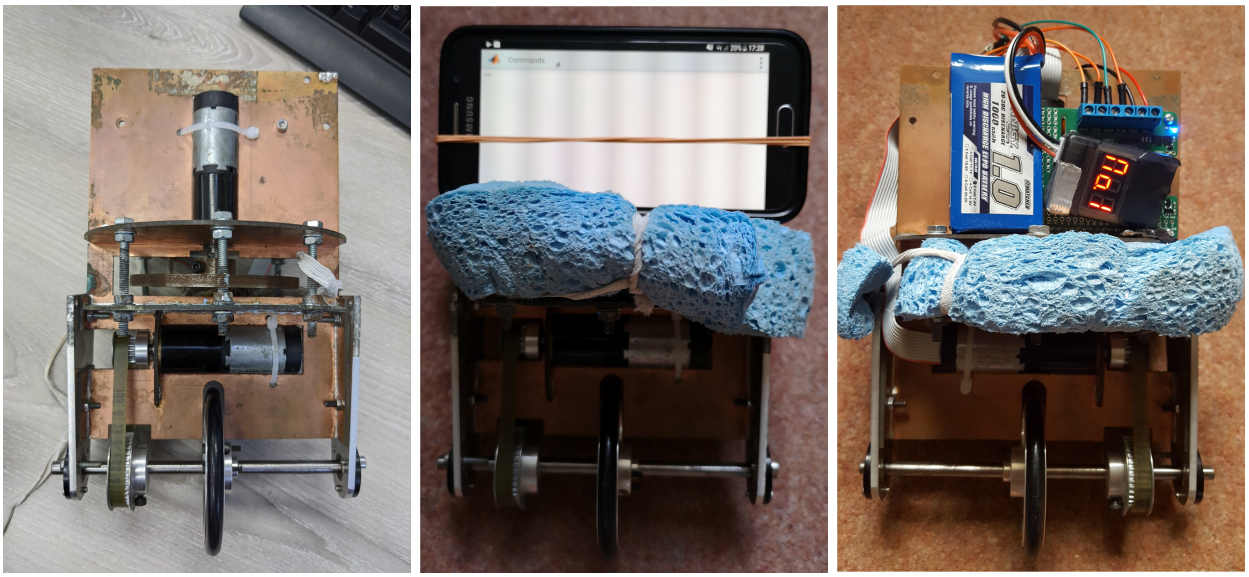
Figure 4: *Predictions from PILCO approach*

from the starting position.

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

vi. Cost Function

A cost function is required to evaluate the performance at each timestep. This takes the form of equation 4. Where a, h and θ_{max} represent positive constants, and d(**x**) is the geometric distance of the unicycle tip



(a) *Frame* **(b)** *Front [9]* **(c)** *Back [9]*

Figure 5: *Current set-up of unicycle*

need to label the images and probs make them bigger?

III. Current Unicycle Set-up Review

i. Mechanical

Figure 5a shows the mechanical design of the unicycle. The unicycle was built from PCB boards made of copper coated fibreglass and subsequently welded together. This material was 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 ..... [**?**]. These motors have maximum continuous torques of 6.31mNm and speeds of ....rpm. These motors run fast so to improve sensitivity and increase torque output a 14:1 planetary gearhead [**?**] is attached to each motor. To further increase torque output on the wheel a gearing of 40:16 is implemented via a belt. This results in maximum output torques of the motors as 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 posi-

tions but is currently not in use.

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.

ii. Electrical

The electrical components on the unicycle was simplified by Harris [9] to only involve a phone, Raspberry Pi and Raspberry Pi motor board. This simplification was to reduce the number of components required to be 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 measure 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.

The phone and Pi were unreliability attached using 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 the 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, including policy and dynamics inputs and outputs, variables solved for when simulating and which variables correspond to angles
- Number of rollouts to optimise policy

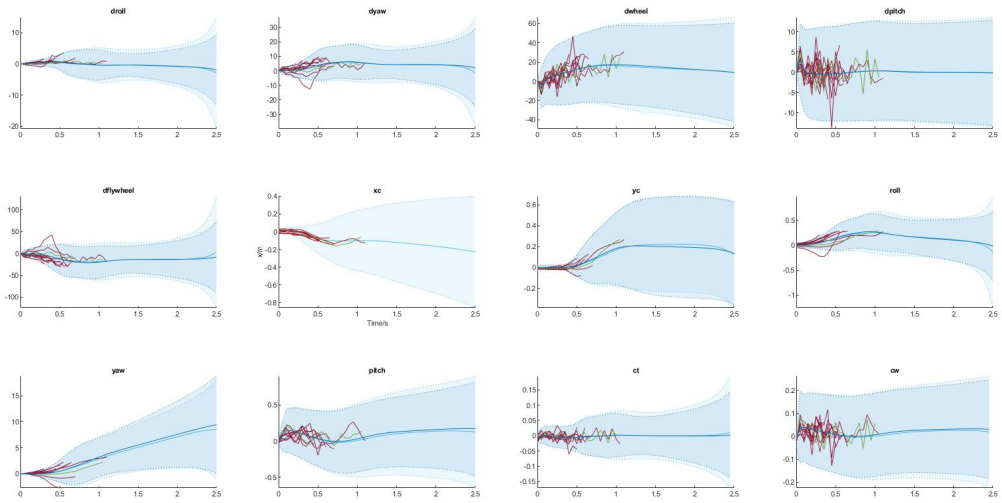


Figure 6: *Information displayed during rollouts*

over

- Number of random rollouts
- horizon time
- Initial state distribution, including state uncertainties
- 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 optimising policy function for use in next rollout can then occur.

Figure 6 shows the information generated during rollouts. Shown in these plots is the states, loss, unicycle animation, and policy improvements across rollouts. The state window shows the trajectory of 10 rollouts under the current policy, with the rollout used in training displayed in green. The predictive state uncertainty and mean is displayed in light blue, with the dark blue indicating an updated prediction 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 [8]. 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 were 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 the motor input torques. These were called using the following commands [9]:

- 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

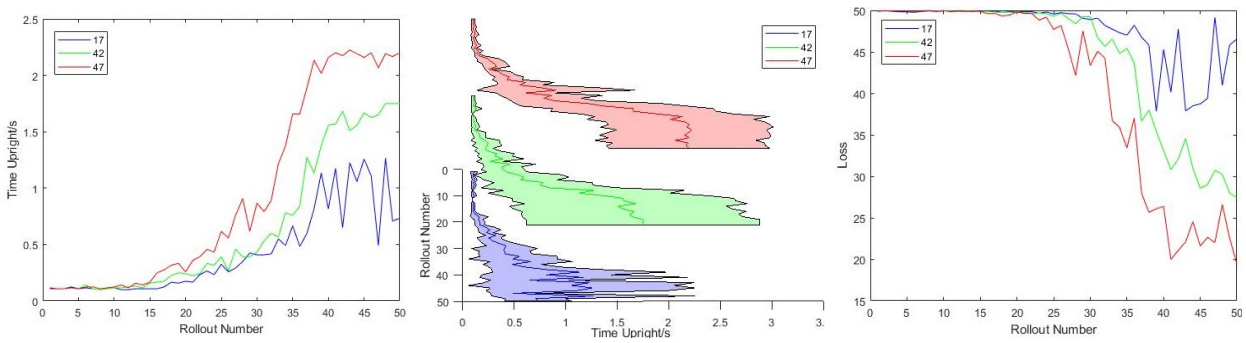
Two different measures will be used to determine success. The first being time upright as more rollouts occur. This is to determine any success at balancing the model has. The second is total rollout loss in order to see if success is achieved by staying near the origin instead of staying upright.

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 degrees. 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 and can cause failure to appear to happen at a low cost. Which could in turn inhibit learning more.

Several designs were sketched to change the roll limit. 42 degrees could be achieved with a new chassis and the same gears, and 47 degrees by changing an additional bracket. 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 change between 42 and 47 degrees appears significant which shows more progress is likely to be made by increasing roll limit further. The plateau observed at 47 degress 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



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

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

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 degrees close to being higher then the mean at 42 degrees. The spread is very important to create a reliable policy training method.

Higher roll limits could be achieved by changing other parts. 53 degrees would require a 5mm larger wheel. This change would require evaluating the effect wheel size had on performance as well as roll limit.

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 extra computational time.
- Every policy call injects noise into the predictions from process and measurement noise. This causes a more uncertain trajectory prediction over the horizon time.

Not enough tests have been simulated

yet to fully determine the effect. However, new learning behaviours which prioritised distance to origin over just being upright were observed. Therefore showing promise that sampling time can greatly effect learning.

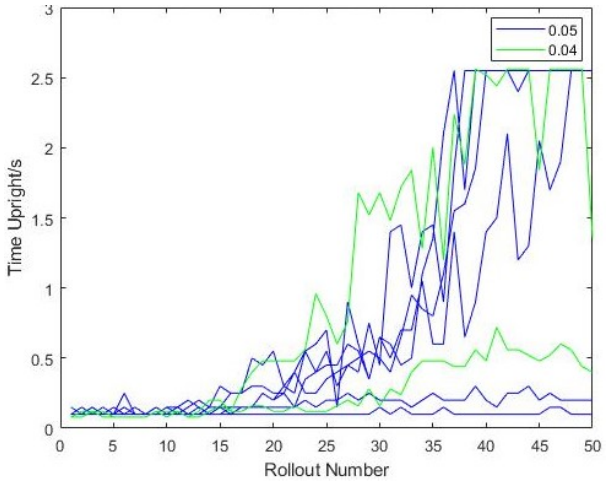


Figure 8: *Effect of sampling time change*

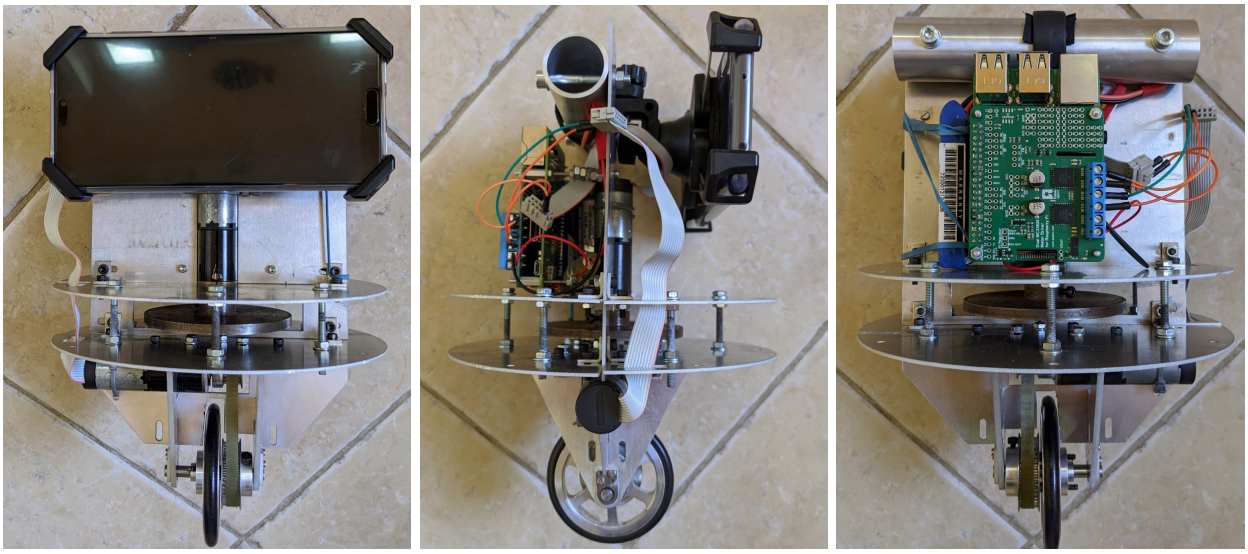
Figure 8 shows the results of changing timestep for a roll limit of 42 degrees. Not enough tests have been conducted yet due to the increased simulation time meaning no conclusions can be drawn yet. The lower performing trial at 0.04s did show promise as it learnt to balance in a different way to all other trials, it prioritised staying at the origin over staying upright. This shows that there is a possibility of the timestep greatly effecting learning and should be pursued further.

iv. Wheel Size

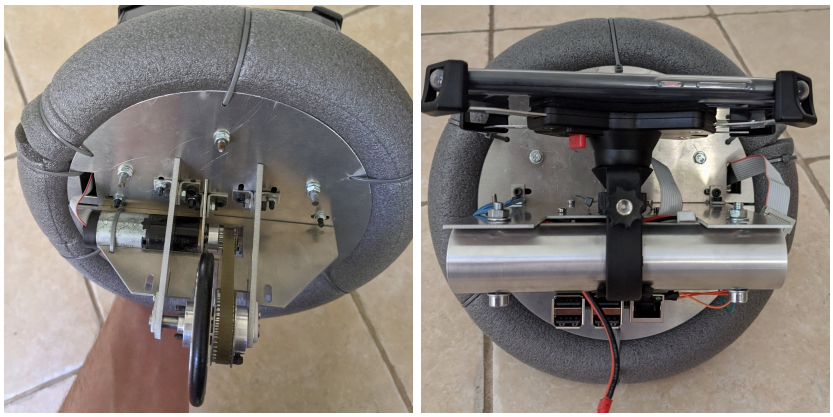
v. Future Simulations

Additional factors are to be simulated to determine the variables that effect learning rate the most. These are the following:

- Higher roll limits with larger wheels.
- More sampling times.
- Strength of the cost function.
- Motor torques.

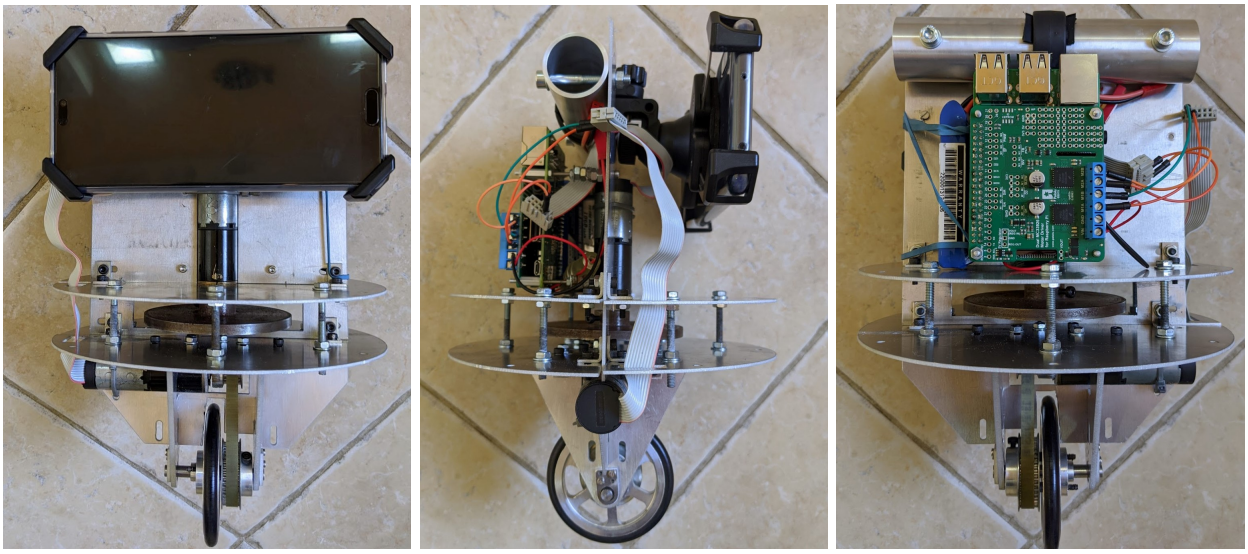


(a) *Front* **(b)** *Side* **(c)** *Back*

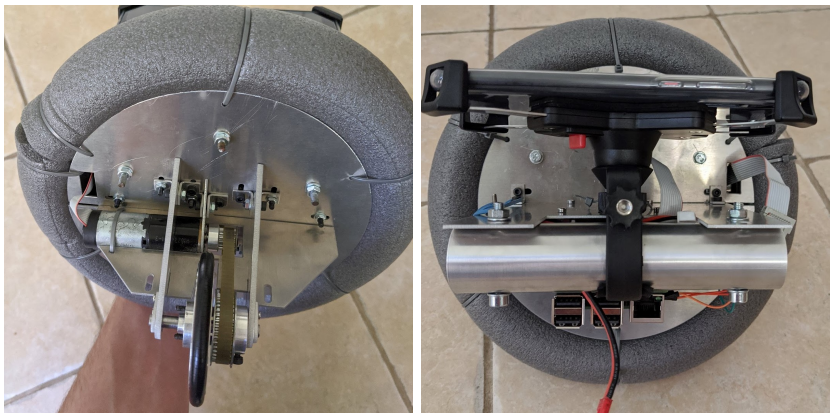


(d) *Bottom* **(e)** *Top*

Figure 9: *New set-up of unicycle*



(a) *Front* **(b)** *Side* **(c)** *Back*



(d) *Bottom* **(e)** *Top*

Figure 10: *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 9 and 10. 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 with the use of bolts
- Keep parts as simple as possible
- Allow for variation in parts eg phone size

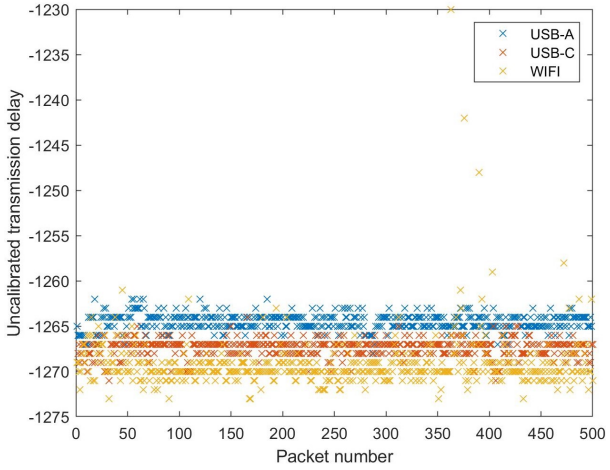


Figure 11: *New design for higher roll limit*

i. Changes

i.1 Roll limit

From simulations it was shown that increasing the roll limit to 47° was required to allow policy learning. Increasing the limit much further simulations deemed unnecessary.

Figure 11 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 should still be able to learn with an off-centre design, however this would still be corrected for with later part placement. Once all the parts had been centralised it was found that the roll limit was 42° with the axle bracket the limiting factor.

Three steps were implemented to reduce this limit. Reduce size of bearing holder,

Design	Summary	USB-C	Total
1		1.05	3.02
2		1.26	1.25
3		1.02	1.84

Table 1: *Protection system analysis*

use smaller 10x6mm bearing, and minimise size of axle mount. These changes allowed the roll limit to become 47° as desired.

i.2 Protection

The next problem to solve was protecting the unicycle was 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 1 consisted of a static material eg rubber or foam being attached, design 2 was a large ring attached to the main chassis by springs, and design 3 was 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, 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.

$$d = \tag{5}$$

The foam was to be mounted to the horizontal metal disks protecting the flywheel. A roll limit of 45° was designed for the point of contact on the tube allowing for 2° of deceleration to occur. This corresponded to a displacement of ..... in the foam. The required disk diameter for mounting was calculated using equation 5. Where t=26mm is the disk separation, and r=15mm is the foam radius.

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 the cross trial effect of sensor position change. This introduces a lot of avoidable noise if done incorrectly. The 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 f 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 need to fabricate complicated parts. Different bike and car mounts were evaluated with the ..... bike mount eventually chosen **[?]**. This mounts onto a handlebar like design at the top of the unicycle as in figure 10e.

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 10c. 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 shorter 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

A copper coated fibre glass was used previously, 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 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. Design Evaluation

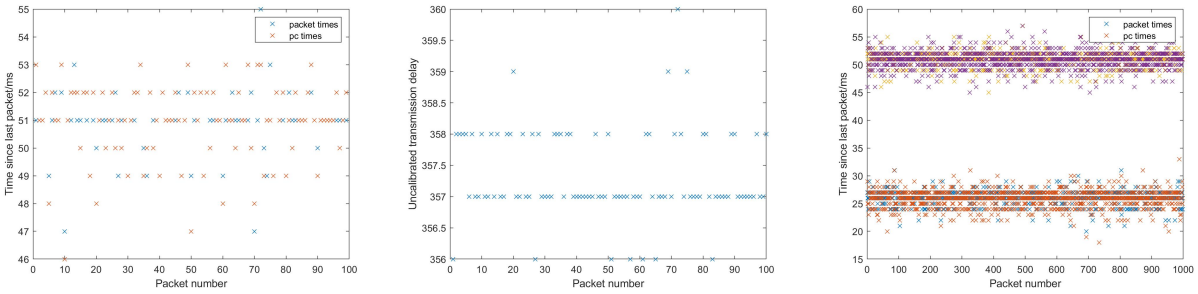
While constructing the unicycle it was apparent that the new design was more rigid and adaptable then the last model. Having the flywheel protection disks made out of one piece, instead of two welded together was the main source of this additional strength. The addition of the bolts also made assembling the model much easier which is promising if future modifications are needed. The most difficult part of assembly was securely attaching the top motor, this was due to the inaccessibility of the screws mounting it. This process was still easier then the last model as ample room had been made to remove the flywheel when the motors attached, allowing more room for tools. 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 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 and allowed for adequate variation in either phone orientation or size. Due to the mounting towards the top of the unicycle the moment of inertia about the wheel when falling is increased. This doesn't appear to be adversely affecting the unicycle performance as falling isn't happening too fast to react to. To improve this in future when the department is back open, weight could be removed from the inside of the handle bar, or mounting holes could possibly 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 [15] 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. This change hasn't been shown to adversely

effect performance yet. However, there is a slight worry that with this increased weight and moment of inertia the current motors and 200g flywheel will be too small to quickly correct the unicycle yaw and roll. If this proved problematic the easiest change would be to introduce a larger flywheel as there is room in the casing.



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

Figure 12: *Testing connecting by USB*

VI. 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 decreasing learning ability. HyperIMU [10] 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 12a 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 12c 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 and phone true transmission delay wasn't

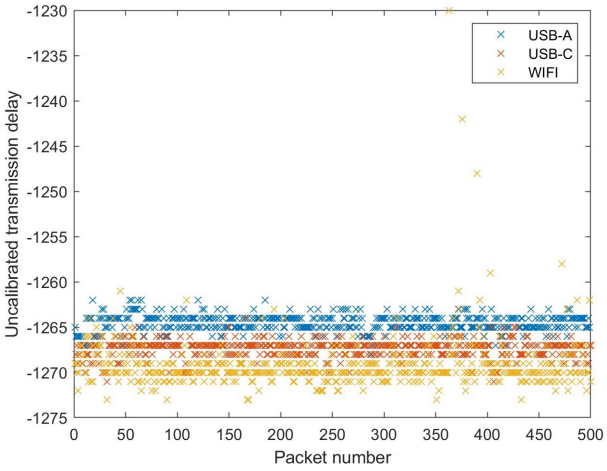


Figure 13: *Effect of changing transmission medium*

possible to measure. Figure 12b shows this uncalibrated delay. The majority of packets take one of two times to transmit with a few outliers taking $\pm$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 2: *Transmission delay σ in ms*

The delay over Three transmission mediums were compared in figure 13. 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 13. The standard deviation was calculated in Table 2. 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. The extra noise introduced by an extra cable effecting dynamics however 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 can't be currently done on new runs. Measuring the initial orientation is therefore required, which also allows accurate uncertainty measurements ensuring predictions incorporate as much information as possible.

$$\delta \mathbf{a} = \mathbf{a}_0 - R\hat{\mathbf{a}}$$

$$\begin{bmatrix} \theta \\ \psi_f \end{bmatrix} = 2 \begin{bmatrix} tan^{-1}(\frac{\delta \mathbf{a}_x}{\sqrt{\delta \mathbf{a}_y^2 + \delta \mathbf{a}_z^2}}) \\ -tan^{-1}(\frac{\delta \mathbf{a}_y}{\sqrt{\delta \mathbf{a}_x^2 + \delta \mathbf{a}_z^2}}) \end{bmatrix} \quad (6)$$

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

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

The unicycle has only two angular DoF initially, this means that one angle can be set to 0. For this we choose yaw as this angle 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.

Equation 6 calculates the initial euler orientation. This can then be easily converted to quaternion form with two quaternions rotations : $(1 + 0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k})R_Y(\theta)R_X(\psi_f)$. Quaternions extend complex numbers to 4D and are required to accuracy keep track of repetitive rotations due to following the same mathematical rules. [9] Quaternion form is then used for all subsequent rotations calculated using gyroscope readings. A rotation about axis (x,y,z) by α can be defined using the following unit quaternion.

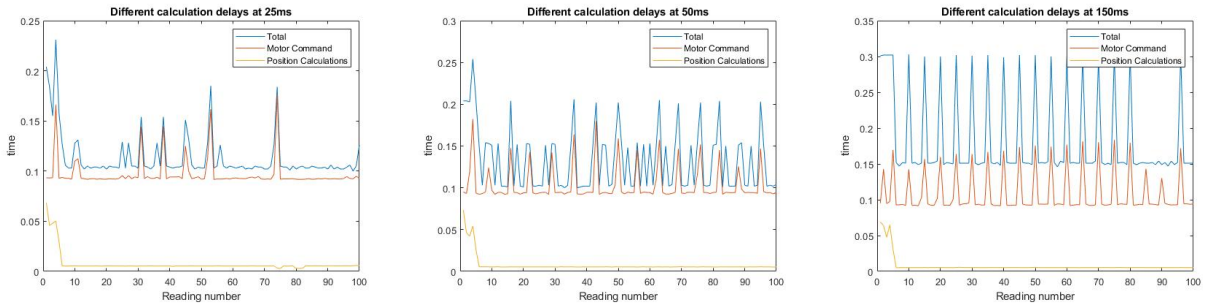
$$\mathbf{q} = cos(\frac{\alpha}{2}) + sin(\frac{\alpha}{2})(x\mathbf{i} + y\mathbf{j} + z\mathbf{j})$$

$$\mathbf{q}_{new} = \mathbf{q}_{old} \cdot \mathbf{q}$$

ii. Raspberry Pi

ii.1 Connection

Connection to the Raspberry Pi is done using the matlab support package for Pi devices [16]. An initial set-up is required for this package. This involves 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:



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

Figure 14: *Testing connecting from Matlab to Pi using Matlab system command*

$$mypi = raspi('169.254.0.2','pi','raspberry') \tag{7}$$

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 to the Pi:

$$system(mypi,'sudo\ python\ m/filename.py') \tag{8}$$

This command opens a terminal on the Rasbian OS on the Pi and runs the python file specified. Three files were used **[?]** "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 changes to 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 more non linear and harder to model. Figure 14 shows this variation occuring when controlling the motors, for a number of different timesteps. The first few recordings took longer in each test. the execution time then settled to be at minium 100ms, 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.

And due to the command requiring opening a new terminal, and python file each time it took at minimum 90ms. Possible soultions were tested to avoid this; adding '&' at the end of a command stops the script waiting for a reply, hwoever this is incompatible when executed through a Pi. The command was also ran as a batch as a sub process, this allowed the script the proceed without delay but didnt succesfuly 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. 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 command and avoid the need to wait for a responce an alternative to the system command was used:

$$commandtoopenasshwindow \tag{9}$$

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

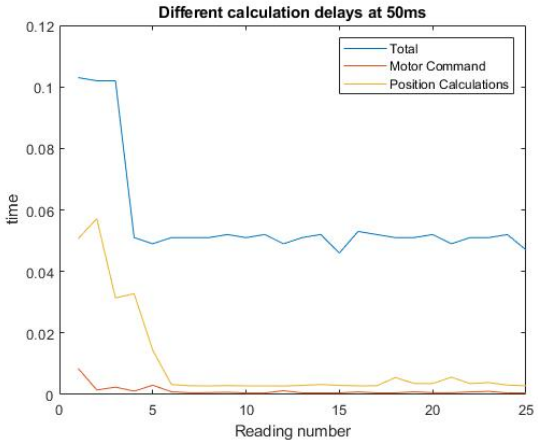


Figure 15: *Testing connection to Pi sending torques by UDP*

$$sudopythonm/..... \tag{10}$$

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

$$fileformat \tag{11}$$

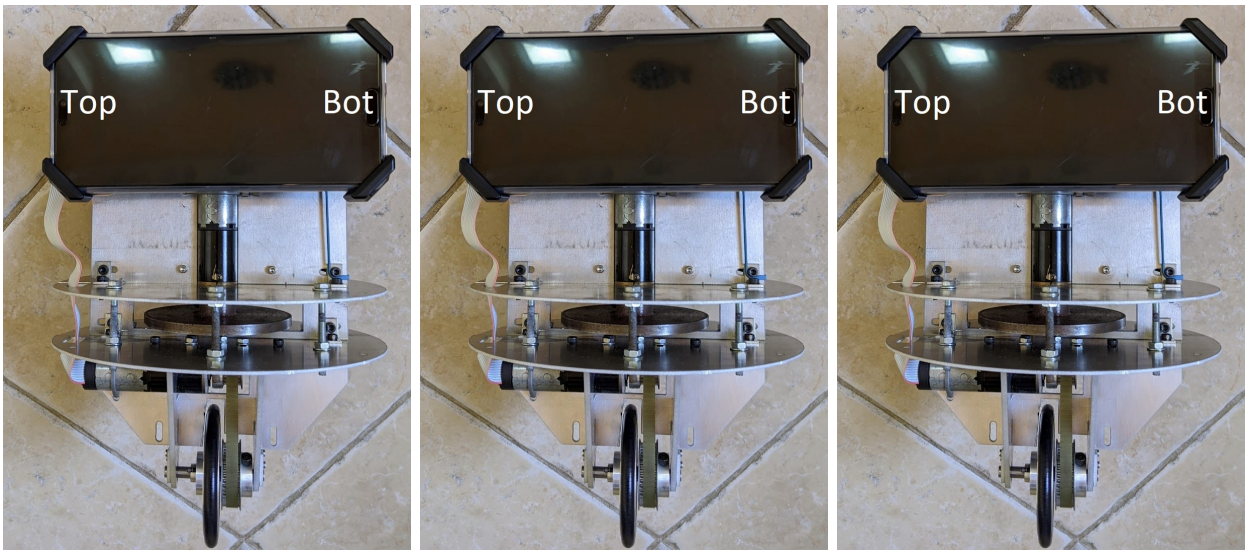
The script is terminated by sending # in this command.

Figure 15 shows the success this method has at bringing the timestep to the desired length. 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 the yellow line.

iii. Motor

maybe something about calculating right-
ing angles? and stiff

iv. Rollouts



(a) *Phone set-up* **(b)** *HyperIMU settings* **(c)** *Release position*

Figure 16: *Steps in set-up*

iv.1 Test Procedure

Mechanical Set-Up

- Attach phone as in figure 16a 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 16b
- Save these settings and click start logging from HyperIMU

Rollouts

- In an open flat space position the unicycle, 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/..... " will initiate listening mode on the Pi, and display all motor torques and time between subsequent messages
- Make sure that the unicycle is being held still and vertical at this point, holding onto the string improves repeatability and release smoothness as in figure 16c
- The orientation is then measured for a few seconds before the motors start, at this point let go
- 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

VII. Conclusions

i. Progress

ii. Future Work

After the further simulations has been performed and evaluated the following changes will be considered in the future:

- Redesign unicycle to allow for higher roll limit, different motors, greater modularity, robustness to falls and secure phone mounting.
- Effect of independence between state variables on real rollouts.
- Adjustments to cost function.

iii. Additional rollouts

iv. Encoder

v. Exploration

vi. Mass adjustment

vii. Removing Matlab from rollouts

References

[1] 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.

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

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

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

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

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

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

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

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

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

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

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

[13] 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.

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

[15]

[16]

[17]

VIII. Appendix

i. Risk Assessment Review