Machine Learning for Control

by

Rodrigo Queiro (DOW)

in collaboration with Alan Douglass

Fourth-year undergraduate project in
Group F, 2010/2011

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work.

Signed: _____ Date: _____

Contents

Technical Abstract

In 2004 and 2005, a robotic unicycle was built by Mellors [1] and Lamb [2] as a Master's project, using a powered wheel and horizontal flywheel designed to emulate the actuation used by a human rider, but they lacked the time to design a controller. Several years later, D'Souza-Mathew [3] and then Forster [4] attempted to stabilise it by adding training wheels, and using the powered wheel to balance in pitch, but were thwarted by problems with the hardware.

The system of the unicycle with a horizontal flywheel actuator had previously been studied by Vos [5] and Naveh [6]. Vos built a physical unicycle and designed a controller for it, but his results were reportedly unreliable. Furthermore, his controller required constant forward motion of the unicycle to correct for roll errors. Naveh built upon Vos's design, coming up with a more capable (and complicated) controller, but did not test it on a physical unicycle.

These projects all sought to build a manageable but accurate analytical model of the dynamics of the unicycle. They then applied non-linear control techniques to try to stabilise the unicycle. This is particularly difficult for unicycle with a horizontal flywheel, since when it rolls to the side it must both rotate in the direction of the fall and then travel forwards to catch itself. This brings gyroscopic effects into play, which are hard to stabilise with analytical control techniques.

An alternative approach is to build a numerical model of the dynamics directly from interaction with the system, and then to computationally design an optimal controller for the numerical model—a reinforcement learning problem. An approach to this problem due to Rasmusssen and Deisenroth [7], Reinforced Model Learnt Control (RMLC), has been successfully used to learn to swing up and balance an inverted pendulum, with prior knowledge only of the states and control inputs of the system. It has also been used to stabilise a computer simulation of the 3D unicycle. One of the most important aspects of RMLC is that it uses Gaussian Processes to learn the dynamics of the system, allowing it to make predictions while keeping track of the certainty of its knowledge about the system.

Previously, McHutchon had successfully applied RMLC to the unicycle with training wheels—this had required several changes to the hardware of the unicycle [8]. His work gave a clear idea of necessary changes to the hardware and software of the unicycle in order to enable it to balance without training wheels. The first aim of the project was to install a new MEMS gyroscope and an Arduino microcontroller, and then to repeat McHutchon's stabilisation experiments. These changes solved a number of problems with the original system, including noisy angle measurements and voltage supply problems. We were also able to take advantage of the flexibility of the new Arduino-based architecture and improve the safety and ease of use of the system.

After making these changes, the RMLC system was able to successfully balance the unicycle (with training wheels). Additionally, we found we could significantly improve the performance of the RMLC-trained controllers by reducing the timestep at which the system is discretised, and by modelling the discretised system as a 2^{nd} order Markov chain. This led to confident and accurate predictions about the performance of the unicycle, and these techniques would be critical in modelling the unicycle without training wheels.

The next step was to prepare the hardware and software for balancing the unicycle. This required the addition of an accelerometer for sensing the initial angle of the unicycle and a new motor controller and speed sensor for the flywheel. We also required new algorithms for keeping track of the unicycle's orientation and position in 3D space, for sensing the flywheel speed more accurately from an extremely unreliable sensor and for applying a quadratic controller to the system. Additionally, we designed and built a wooden skirt to protect the unicycle from falling too far when testing incapable controllers. When this turned out to have too high a moment of inertia, we redesigned it to reduce the weight, and finally settled on testing with a loose rope tied to a balcony above the unicycle.

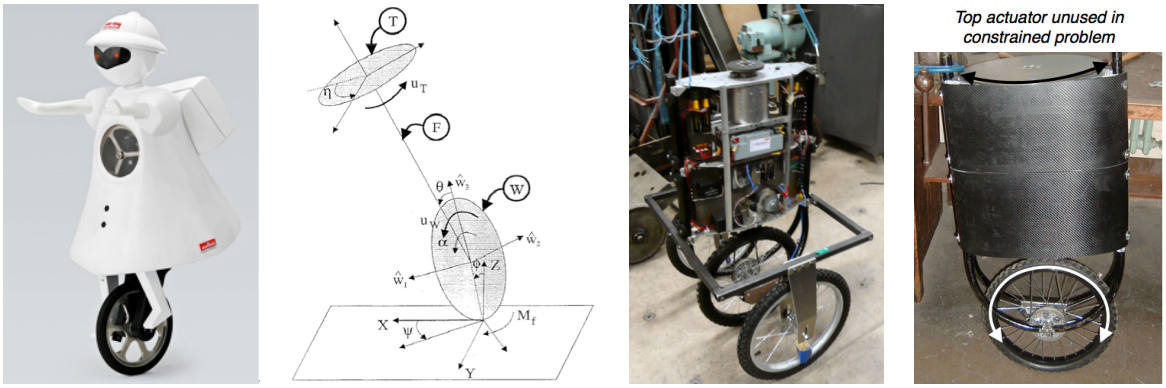
At this point, we started training the unicycle to balance. During the first process, the system became capable of designing controllers that could reliably keep the unicycle vertical until it reached the edge of its range and was pulled over by the rope. Although the learned dynamics model could accurately predict the motion of the unicycle, the system was unable to produce a controller that could stay in one place. This turned out to be the result of a bug in the position tracking—the controller did not have sufficient information to control its position.

Once this problem had been fixed, a second training process was initiated, but failed due to a sensor problem. Finally, a third training process was undertaken, which followed the same initial process of learning to balance as the first. To prevent it from reaching the limit of its restricted range, the loss function was changed to penalise off-centre positions very harshly, although after this point, the system appeared to sacrifice stability in favour of controllers that would remain near the centre. This produced controllers that could frequently balance for 2–3 seconds, but only rarely longer. The training sequence could not be continued due to the computational cost of controller training, and due to the time limitations of the project.

Despite the failure to produce a controller that could reliably, indefinitely balance the unicycle, the RMLC system produced some promising results, including controllers that could control pitch and roll angles reliably but fell foul of the limited range of positions, and one trial in which the unicycle remained upright for 5 seconds. It is hoped that future work on the system will able to resolve some of the various issues that may have caused problems for the RMLC controllers, and finally achieve the task of balancing the unicycle.

1 Introduction

Balancing a unicycle is a very challenging task for a human rider. Many attempts have been made to achieve this task, using a variety of models for the action of the rider. Some represent the rider as a flywheel or pendulum in the coronal plane, allowing direct compensation of falling to the side [9, 10], as in Figure 1(a). Other use a more realistic (and challenging) model of a flywheel in the horizontal plane [5, 6], as in Figure 1(b), but none of these have reliably balanced a real unicycle.



(a) Murata Girl (b) Naveh's Model (c) 2D Problem (d) 3D Problem

Figure 1: Different balance problems

In 2004/2005 Mellors and Lamb [1, 2] built a robotic unicycle, shown in Figure 1(d), intending to design a controller to balance it. However, they were only able to complete the construction of the unicycle. In 2007/2008, D'Souza-Mathew resumed work, replacing a wheel sensor and attempting to design a controller. He simplified the problem by removing the ability to fall to the side, reducing the dynamics to two dimensions: the inverted pendulum (from now on referred to as the **2D system**). This is shown in Figure 1(c). He was unable to balance the unicycle due to hardware problems.

Next, in 2008/2009 Forster analysed the dynamics of both the 2D problem and the unrestricted 3D unicycle [4]. Again, hardware problems prevented him from balancing the 2D system, and although he designed a controller for the 3D unicycle, it was not even tested in simulation. Given the simplicity of his approach compared to those of Vos and Naveh [5, 6], it appears unlikely to work.

One thing all these approaches have in common is that their first step is a series of simplifying assumptions about the dynamic system. They ignore the non-linearities like motor dead-zones and wheel friction that are present in any real-world system, and attempt to design a controller to stabilise the idealised system. In many cases, this approach is very successful. However, D'Souza-Mathew and Forster found that their model was invalid since the unicycle's motor drive didn't react fast enough. Vos's approach neglected all Coriolis and gyroscopic effects, and failed to balance in trials where these were

important. Naveh's more complex approach required multiple control laws for different regions of state space, and was only tested in simulation.

An alternative "intelligent" approach to control involves learning the dynamics of the system directly, instead of relying on assumptions and mechanical analysis. Various methods for this have been used, but many require prohibitively large amounts of data from the system. One method due to Rasmussen and Deisenroth, known here as Reinforced Model Learnt Control (RMLC), achieves unprecedented data efficiency, and has been successfully used to stabilise a computer simulation of a 3D unicycle [7, 11].

In 2009/2010, McHutchon successfully applied RMLC to the 2D system [8]. However, since he had to make significant changes to the unicycle hardware and software to achieve this, he did not have time to attempt to balance the 3D unicycle.

The principal objective of this project is to apply RMLC to the unrestricted 3D unicycle. This includes the solution of problems identified by McHutchon, as well as other problems identified during the project.

1.1 Division of Labour

This project was conducted as a joint project between the author and Alan Douglass. The vast majority of the work was done together, and as such it is difficult to attribute ideas or components of the system to one or the other. When only one was responsible for a particular component, such as a circuit or module of the code, it is focussed upon in their report, and the other report will simply refer to it.

Although this may raise difficulties for marking the projects, it was essential for the actual work. We would never have been able to make such progress if we had not been able to build on each other's ideas, to spot each others mistakes and to rely upon another point of view when stuck.

2 Reinforced Model Learnt Control

The main technique in this project is Reinforced Model Learnt Control, diagrammed in Figure 2. At its core, it assumes that the system (in this case, the unicycle) can be modelled in discrete time as:

$$\boldsymbol{q}[n+1] = \boldsymbol{f}(\boldsymbol{q}[n], \boldsymbol{u}[n])$$

In this equation, $\boldsymbol{q}[n]$ is the state of the system at time n, and $\boldsymbol{u}[n]$ is the control input at time n. In the case of the unicycle, $\boldsymbol{q}$ consists of the angles and angular velocities of the components of the unicycle, and the position of the unicycle. $\boldsymbol{u}$ consists of the commands sent to the wheel and flywheel motors.

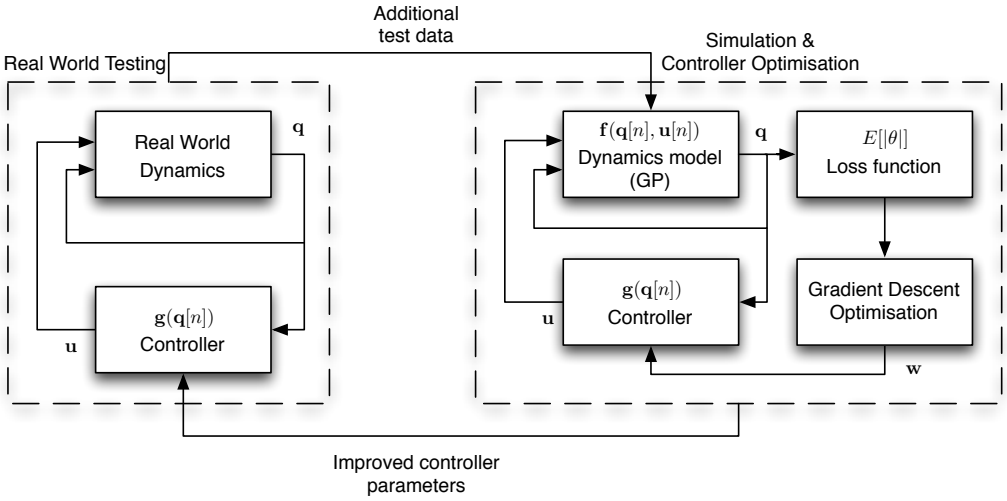


Figure 2: Reinforced Model Learnt Control

This function, $\boldsymbol{f}$, is modelled as a Gaussian Process (GP). By using Gaussian Process Regression (GPR), we can estimate any continuous function from sampled inputs and outputs. For a description of the mechanics of GPR, refer to [12]. When the unicycle runs, we get a series of states and control inputs that can be converted to samples of $\boldsymbol{f}$, and this allows us to use GPR to estimate $\boldsymbol{f}$ at any point. This estimated $\boldsymbol{f}$ is referred to as the **dynamics model**.

By successively applying $\boldsymbol{f}$ to an initial distribution of possible starting states, we can estimate, with confidence bounds, a distribution of states over some finite horizon. This is referred to as **simulation** of the system. Then, a **loss function** is applied to the state distributions—this might find, for example, the expected distance between the top of the unicycle and the upright position. Summing these losses over the horizon gives a numerical score that rates how well the dynamics model believes a given controller will balance the unicycle. This loss score will be lowest for trajectory distributions that are concentrated near the equilibrium.

The gradient of the loss with respect to the controller parameters can be calculated, and this allows standard gradient descent optimisation methods to be used to find a locally optimal controller (for the estimated dynamics model). This process is shown as the right-hand box, "Simulation & Controller Optimisation", in Figure 2, and is referred to as **training** a controller.

Once an optimal controller has been trained on the simulated system, a **trial** is performed on the real system ("Real World Testing" in Figure 2). This generates a log of states and control inputs, which can be converted into more samples of $\boldsymbol{f}$, improving the quality of the dynamics model and allowing a better controller to be trained. This process is repeated iteratively until the dynamics model is sufficiently accurate that the trained controllers perform well on the real system. This process of gathering initial data (see

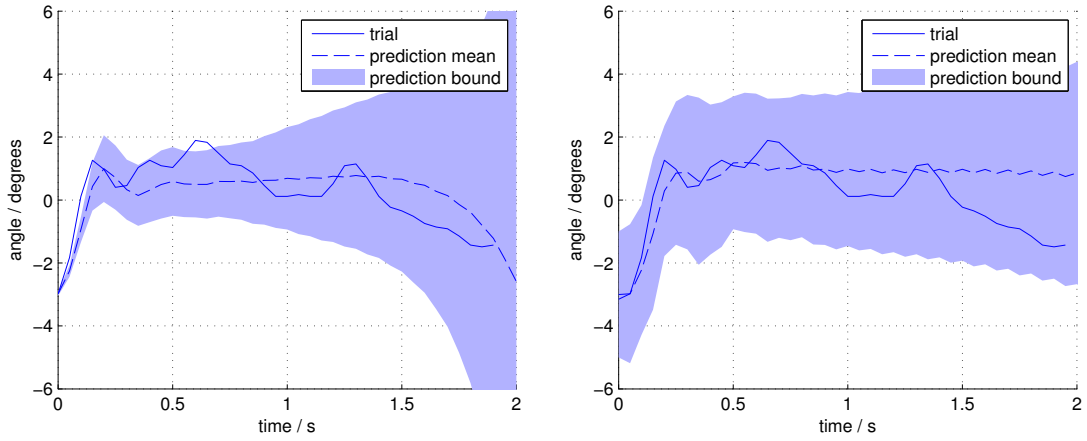
Section 2.2.5) and then iterating between training a new controller and gathering data with it, continuing until steady-state is reached, is referred to as a training **sequence**.

2.1 Prediction Plots

In this approach, knowledge about the state of the system is represented by a multivariate Gaussian distribution over the states. The dynamics model is used to map the distribution at one timestep to the distribution at the next timestep. Repeated application yields a series of distributions at different times representing possible futures of the system. To represent this in a useful fashion, we can marginalise the multivariate Gaussians to get the distribution of a single variable at a given time, and then consider a 95% confidence interval:

$$P(\mathcal{N}(\mu, \sigma^2) \in [\mu - 2\sigma, \mu + 2\sigma]) \approx 95\%$$

We can then plot these confidence intervals as a function of time. This is shown as a filled region in Figure 3. The mean of the predicted distribution is shown as a dashed line. These predictions are for the controller trained by the optimisation procedure described above. After the optimal controller has been trained for the learned dynamics model, a trial on the real system is conducted to gather more data and improve the dynamics model. This trial is shown on both plots as solid line.



(a) Prediction of a specific trajectory

(b) Prediction of a generic trajectory

Figure 3: Predictions of the unicycle's pitch, and the following trial

Note the difference between the two plots. Figure 3(a) shows the prediction when given the initial state of the trial. The uncertainty is limited to measurement error initially, but grows due to uncertainty in the dynamics model. Figure 3(b) shows the prediction starting with a distribution over possible initial states—it is this distribution over trajectories that is optimised by the training process.

2.2 Practical Concerns

There are many different decisions to make when implementing the RMLC strategy, which are detailed in this section. Previously, Rasmussen used Forster's analytical model of the unicycle to simulate the real world trials, and used RMLC to train a controller for this ideal unicycle. Thanks to this, and to McHutchon's work on the real 2D system, we have a lot of information on which choices can work well in these situations, and which are most important.

In many cases, these choices were made after noticing deficiencies in the performance of RMLC on either the 2D system or the 3D unicycle, as explained in Section 4.

2.2.1 GPR Implementation

The GPR system has many configurable parameters, but fortunately the form used previously had proven very robust. This project uses a zero-mean GP with a squared exponential covariance function, with automatic relevance detection. This has the form:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \alpha^2 \exp\left(\sum_{d=1}^{D} -\frac{(x_d - x'_d)^2}{2l_d^2}\right)$$

This expression contains hyperparameters for the signal variance α^2 and the length scales l_d. An additional hyperparameter involved in the regression is the noise variance, σ_ε^2. These parameters are chosen to best fit the data without overfitting with the maximum likelihood (ML) method [12]. The optimal values of these hyperparameters are very useful for interpreting how accurate the dynamics model is: a high SNR $\frac{\alpha}{\sigma_\varepsilon}$ suggests the model can predict very well. Furthermore, automatic relevance detection is provided by the length scales - if a variable is not useful for predicting, the ML length scale will tend to ∞.

2.2.2 Choice of State Vector

The state vector $\boldsymbol{q}[n]$ should be chosen to ensure that the future states are a function only of the current state, and current and future control inputs. In other words, with a known controller the states should form a Markov chain:

$$P(\boldsymbol{q}[n+1]|\boldsymbol{q}[i] \text{ for } i = 1, \ldots, n) = P(\boldsymbol{q}[n+1]|\boldsymbol{q}[n])$$

Forster's analysis suggested the following state to be suitable, which was found by Rasmussen to be sufficient to model the ideal unicycle.

$$\boldsymbol{q}[n] = \begin{bmatrix} \dot{\theta} & \text{roll angular velocity} \\ \dot{\phi} & \text{yaw angular velocity} \\ \dot{\psi}_w & \text{wheel angular velocity} \\ \dot{\psi}_f & \text{pitch angular velocity} \\ \dot{\psi}_t & \text{flywheel (turntable) angular velocity} \\ x_c \\ y_c & \text{position of target in unicycle's reference frame} \\ \theta & \text{roll angle} \\ \psi_f & \text{pitch angle} \end{bmatrix}$$

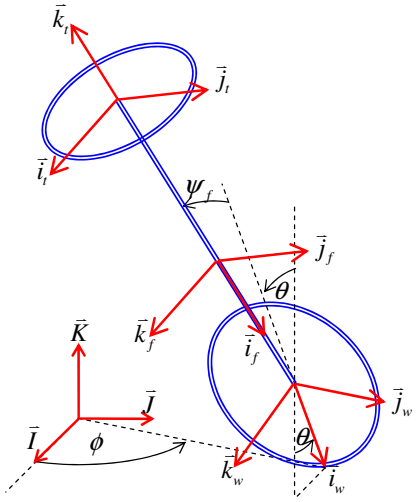


Figure 4: Diagram showing Euler angles for the rotation of the unicycle (from [4])

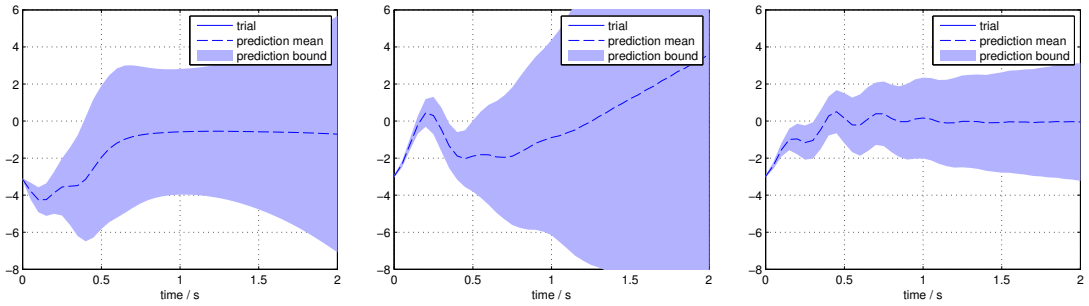
However, this ignores the presence of unobserved states in the system like delays, backlash in the gears, etc. To help the dynamics model deal with these problems, we tried giving it access to the previous state and control input, in effect modelling it as a 2^{nd} order Markov chain:

$$\boldsymbol{q}_2[n] = \begin{bmatrix} \boldsymbol{q}[n-1] \\ \boldsymbol{u}[n-1] \\ \boldsymbol{q}[n] \end{bmatrix}$$

Figure 5(a) shows a prediction for a single trajectory using the 1^{st} order model, and Figure 5(b) shows a prediction for the same trajectory and controller with the 2^{nd} order model.[1] Initially, it is much more certain about the motion of the unicycle, although this

[1]It is puzzling that the predicted trajectories go in the opposite direction initially. It is possible that noise in the measured angular velocity misleads the 1^{st} order model, whereas the 2^{nd} order model gets two consecutive states, making the initial motion of the unicycle more clear. Also, the initial state(s)

certainty quickly diverges. When the controller is optimised for the more accurate 2nd order model, the rate of uncertainty growth is much reduced, as Figure 5(c) shows.

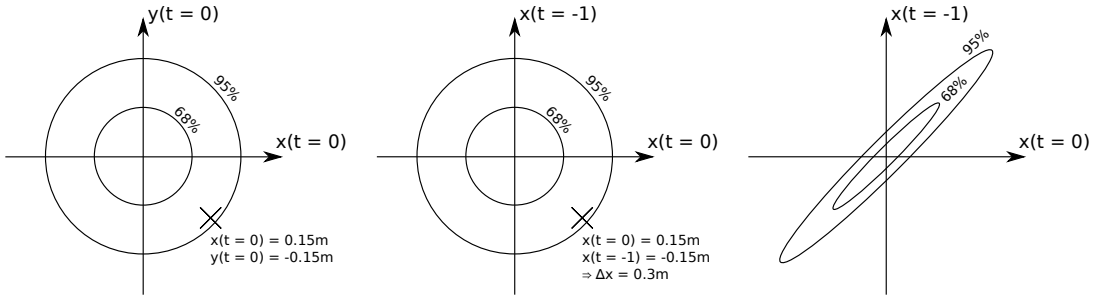


(a) 1st order, optimal controller (b) 2nd order, controller of Figure 5(a) (c) 2nd order, optimal controller

Figure 5: Pitch angle predictions with 1st and 2nd order dynamics models

2.2.3 Initial State Distribution

In order to simulate the unicycle, the dynamics model requires an initial distribution, which it can then transform with the estimated transition function $\boldsymbol{f}$. For the RMLC system used, this must be a sum of a finite number of multivariate Gaussian distributions. Using a 1st order Markov model, a simple and effective choice is a single Gaussian with a diagonal covariance matrix representing the variety of possible initial states. This is represented in Figure 6(a).



(a) Independent variables (b) Independence between same variable at different times (c) Correlation between same variable at different times

Figure 6: Contours of various Gaussian initial state distributions

When using a 2nd order Markov model, the initial augmented state $\boldsymbol{q}_2[0] = [\boldsymbol{q}[-1]\ \boldsymbol{u}[-1]\ \boldsymbol{q}[0]]^{\mathrm{T}}$ contains two successive states. The simplest choice is to use the same independence assumption as before, as illustrated in Figure 6(b). However, this is an extremely non-physical distribution—the illustration shows how this implies the

come from a rollout using a different controller. As such, the second state provided to the 2nd order model could not be predicted by the 1st order model since it doesn't know the controller.

possibility of crossing the entire region of possible starting positions in a single timestep.[2] This leads to rapid growth in uncertainty in the position, as shown in Figure 7(a), and also means that the behaviour in the trial bore little resemblance to the highly uncertain prediction. However, when given the exact initial state, the dynamics model's prediction is very similar to the trial's trajectory, as demonstrated by Figure 7(b).[3]

Upon realising this, a new initial state distribution was chosen, based on the model that the two previous states are the same, but there is some small measurement error, representing the fact that the states are, in reality, different, and also that this model is not quite true. This distribution is depicted in Figure 6(c).

$$\begin{aligned} q_{i,\text{initial}} &\sim \mathcal{N}(\mu_i, \sigma_i^2) \\ q_i[-1] &\sim q_{i,\text{initial}} + \mathcal{N}(0, \sigma_{\varepsilon,i}) \\ q_i[0] &\sim q_{i,\text{initial}} + \mathcal{N}(0, \sigma_{\varepsilon,i}) \\ \begin{bmatrix} q_i[-1] \\ q_i[0] \end{bmatrix} &\sim \boldsymbol{N}\left(\begin{bmatrix} \mu_i \\ \mu_i \end{bmatrix}, \begin{bmatrix} \sigma_i^2 + \sigma_{\varepsilon,i}^2 & \sigma_i^2 \\ \sigma_i^2 & \sigma_i^2 + \sigma_{\varepsilon,i}^2 \end{bmatrix} \right) \end{aligned}$$

Since at this point several trials had already been performed, we were able to estimate μ_i, σ_i and $\sigma_{\varepsilon,i}$ from the trial data. Using this new initial distribution, the expected behaviour is much more like the observed trial, as shown in Figure 7(c).

2.2.4 Controller Form

One of the most basic forms of controller is a linear controller, $\boldsymbol{g}(\boldsymbol{q}[n]) = \boldsymbol{W}\boldsymbol{q}[n] + \boldsymbol{p}$, where $\boldsymbol{W}$ is a matrix of weights and $\boldsymbol{p}$ is a vector of offsets. This form is capable of stabilising the ideal inverted pendulum, and indeed proved sufficient to stabilise the 2D system.

However, the linear controller cannot generate the correct turning command as shown in Figure 8—this is equivalent to the XOR problem, and can be solved by using a quadratic controller. This takes the form:

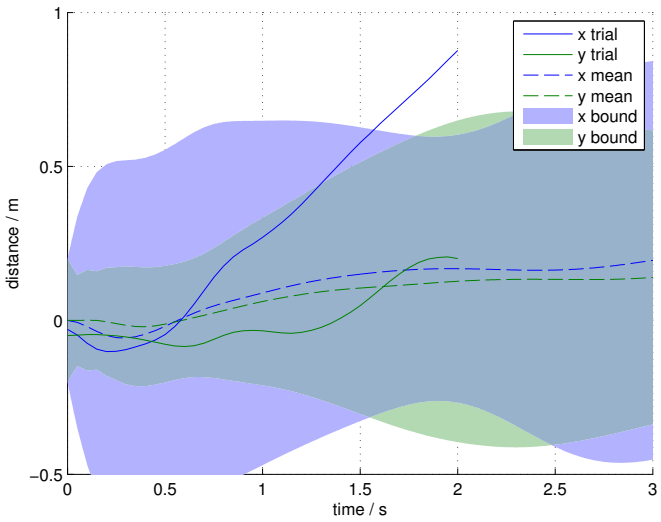
$$g_i(\boldsymbol{q}[n]) = p_i + \sum_{j=1}^{D} w_{i,j} q_j[n] + \sum_{j=1}^{D} \sum_{k=j}^{D} h_{i,j,k} q_j[n] q_k[n]$$

The quadratic controller was sufficient to balance the ideal unicycle in Rasmussen's simulations.

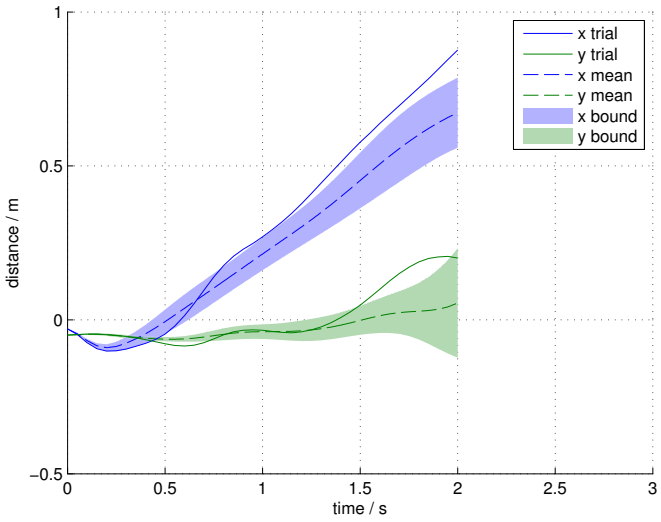
When switching to the 2nd order Markov model, it was found that the controllers performed significantly better when using $\boldsymbol{q}_2[n]$ as input, instead of $\boldsymbol{q}[n]$. To understand

[2] With the 0.05s timestep used in our experiment, this suggests a possible starting velocity of 6 m s^{-1}, which was not that case.

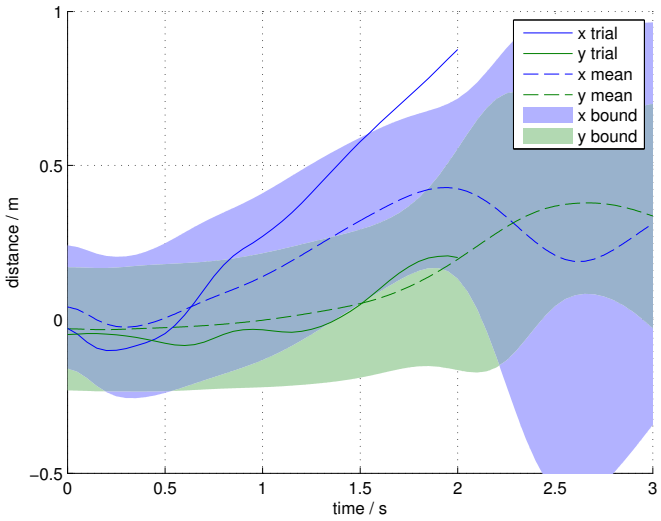
[3] The trial actually follows a trajectory slightly outside the prediction's confidence bounds: this is may be due to measurement noise affecting the initial state estimate, as well as inaccuracy in the dynamics model.



(a) Prediction with original initial distribution



(b) Prediction with trial's initial state



(c) Prediction with new initial distribution

Figure 7: Effect of initial distribution on predictions

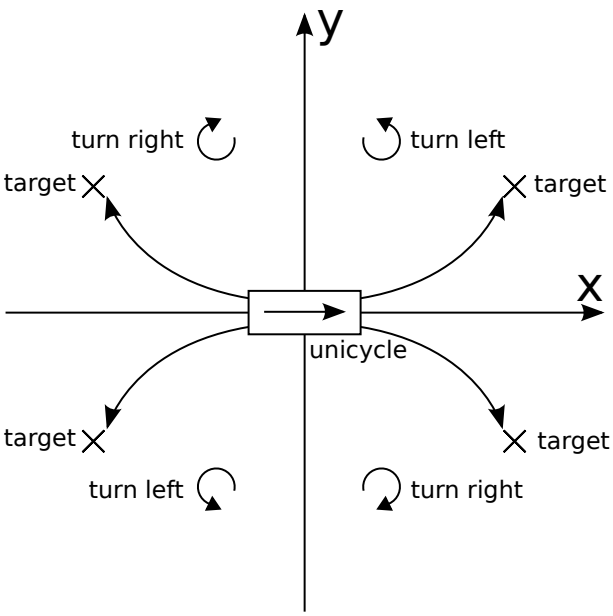
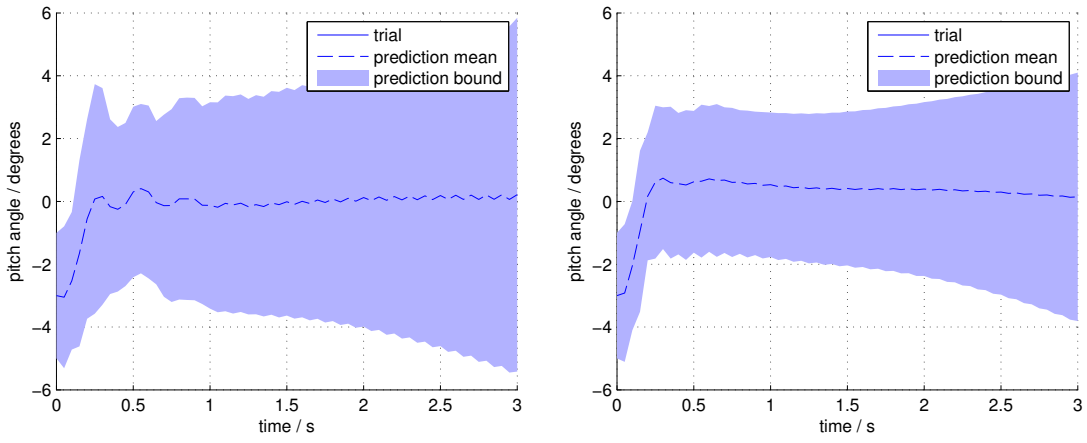


Figure 8: Correct turning command in unicycle-centred coordinates

this, consider the effect with a linear policy: the controller for the augmented state is equivalent to a combination of a two-tap FIR filter and a first-order IIR filter on the original state:

$$\begin{aligned}\boldsymbol{u}[n] &= \boldsymbol{g}(\boldsymbol{q}_2[n]) = \boldsymbol{W}\boldsymbol{q}_2[n] \\ &= \boldsymbol{W}_1\boldsymbol{q}[n-1] + \boldsymbol{W}_2\boldsymbol{u}[n-1] + \boldsymbol{W}_3\boldsymbol{q}[n]\end{aligned}$$

This allows the RMLC system to create basic low or high-pass filters in the controller, and this additional freedom improved both the loss function and the subjective quality of the controllers trained. Figure 9 shows the effect of this change on the optimal controllers.



(a) Control input: $\boldsymbol{q}[n]$

(b) Control input: $\boldsymbol{q}_2[n]$

Figure 9: Effect of providing augmented state to controller

2.2.5 Gathering Initial Data

In order to train a controller, some experience of the system is required. The way this has been done before is with **random trials**: the system inputs are chosen randomly, and the response is recorded. Rasmussen chose to use independent and identically distributed (IID) motor commands when stabilising the ideal unicycle. McHutchon chose to use a controller with randomly chosen weights.

We initially chose to use IID motor commands, chosen uniformly from the full range. However, the dynamics model could not model the resulting data well. We suspect that the same problems that make the 2$^{\text{nd}}$ order Markov model (described in Section 2.2.2) so advantageous mean that if the motor command changes radically between cycles, the behaviour is very unpredictable. Since the dynamics model cannot model signal-dependent noise well, these datapoints cause a very uncertain model to be learnt.

We solved this problem by low-pass filtering the random motor commands: this prevents sudden changes in the motor command, while providing a wider survey of the space than a random controller and avoiding the risk of large positive feedback.

2.2.6 Loss Function

The main concerns when choosing a loss function are to accurately represent what is desired of the controller, and to ensure that different desires are appropriately weighted. When stabilising the ideal unicycle, Rasmussen was successful using the following form of loss function:

$$1 - \mathcal{N}(\boldsymbol{q}[n]; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = 1 - \exp\left(\sum_{d=1}^{D} -\frac{(q_d[n] - \mu_d)^2}{w_d^2}\right)$$

This is configured using w_d, the characteristic widths for each variable. To ignore a variable, take $w_d^{-2} = 0$. While the RMLC system is not too sensitive to these values, it is important to set them appropriately. During a test run on the 2D system, we accidentally set the characteristic pitch angle to 10° and the characteristic distance (from the origin) to around 10cm. This meant that the preferred policy was to fall over immediately, to prevent travelling away from the origin. Loosening the characteristic distance to 30cm led to a controller that balanced the system.

For the ideal unicycle, Rasmussen chose to penalise the pitch and roll angles, to encourage the system to keep the robot vertical. In addition, he chose to penalise the yaw rate $\dot{\phi}$ and flywheel rate $\dot{\psi}_t$ to prevent the system from using a "spinning-top" like approach to balance, and to penalise the distances from the origin, x_c and y_c, to prevent the system from driving the robot very fast to enhance stability.

We used a similar loss function structure, with characteristic widths of 9° on the

angles, and $\frac{1}{2}$ metres on the distances.[4] Unlike the ideal unicycle, the real unicycle has an upper limit on the flywheel speed, so the "spinning-top" strategy is not viable, and it is not important to penalise the flywheel speed.

2.2.7 Timestep

The RMLC approach assumes a discrete-time system - to apply it to a continuous time system, it must be discretised. We used a zero-order hold (ZOH) on the control signal: $u(t) = u[\lfloor \frac{t}{T} \rfloor]$.

Training for the ideal unicycle, Rasmussen found it desirable to choose the largest timestep T with which the system can be stabilised, to reduce the computational cost of the optimisation, and to avoid problems with noise buildup from many repeated applications of the transition function $\boldsymbol{f}$. He chose a timestep of $\frac{1}{7}$ seconds.

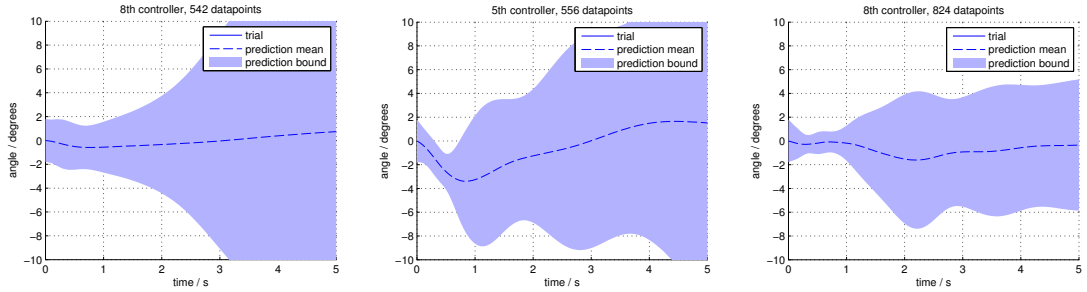
However, for the real unicycle, both McHutchon and we found that the predicted trajectories are more confident and reliable, and controllers perform better, when using a timestep of $\frac{1}{20}$ seconds. It was hoped that this was because the sensors were noisier in real life, and so a shorter time in the ZOH led to several successive noisy control settings being averaged by the low-pass effect of the system, reducing the effect of noise. However, upgrading the sensors didn't seem to change this, so there is possibly another cause.

It is hard to provide a clear demonstration of the advantage of the faster timestep, since trials must be re-conducted and so the datasets are different. In addition, since the training sequence with a 50ms timestep developed better controllers, trials lasted longer and more data was gathered. As such, to compare dynamics models based on similar numbers of datapoints, we must provide fewer trial logs for the 50ms dynamics model. This means that it has had less of a chance to try a variety of strategies, and necessarily is less certain about the dynamics of the system. That said, Figure 10 shows the effect of reducing the timestep, comparing dynamics models with similar numbers of points (Figures 10(a) and 10(b)) and after the same number of trials (Figures 10(a) and 10(c)).

3 Hardware and Software

When the project started, some of the required tasks were already clear, as a result of the previous work on the unicycle. These included extensive changes to the hardware and a complete rewrite of the controller software, both required to fix issues that had been turned up by the previous year's work and to prepare it for 3D balance. A brief summary of the changes is below, followed by greater detail where necessary.

[4] We initially used a distance loss width of 1 metre, but upon realising how little room the unicycle had to manoeuvre we reduced the width to $\frac{1}{2}$ metres to represent the importance of staying within the rope's range. This did not appear to have a dramatic effect on the resulting controllers.



(a) 100ms, 8 trials, 542 points (b) 50ms, 5 trials, 556 points (c) 50ms, 8 trials, 824 points

Figure 10: Predictions with datasets at 100ms timesteps and 50ms timesteps

- The cumbersome FPGA-based controller used previously was replaced with an Arduino microcontroller, allowing changes to be made faster and more easily.
- Older gyroscopes and very noisy angle sensors were replaced by modern MEMS gyroscopes and accelerometers.
- Algorithms for keeping track of angle and position in 3D were written.
- A new motor controller was built for the flywheel, as well as a quadrature encoder for speed sensing. Additionally, a different speed sensing algorithm was implemented.
- A sensor for the battery voltage was added, to ensure it could supply sufficient power to the motors.
- We designed and tested a number of methods of protecting the unicycle from falling, while allowing 3D movement.
- Some faults were identified and could not be fixed, so tests and error-checking were added to ensure these did not corrupt the data.
- To improve safety, the motor is disabled when the operator presses a switch on the chassis or when the unicycle nears the ground.

3.1 Sensing

3.1.1 Angle Sensing

The unicycle had previously used a vibrating crystal rate gyroscope and a pair of infra-red distance sensors to keep track of its angle. However, these were reported to be extremely noisy, and were blamed for a large part of the poor performance of the previous system. Our supervisor had already purchased the ITG-3200, a 3-axis MEMS rate gyroscope with in-built temperature compensation and low-pass filtering. We later purchased an ADXL345, a 3-axis MEMS accelerometer, to determine the absolute angle of the unicycle.

The ITG-3200 rate gyro determines the angular velocity of the chip, and thus the unicycle, around each of its 3 axes. These values are offset by some unknown bias—this is determined before the trial by averaging the output when stationary, and is assumed

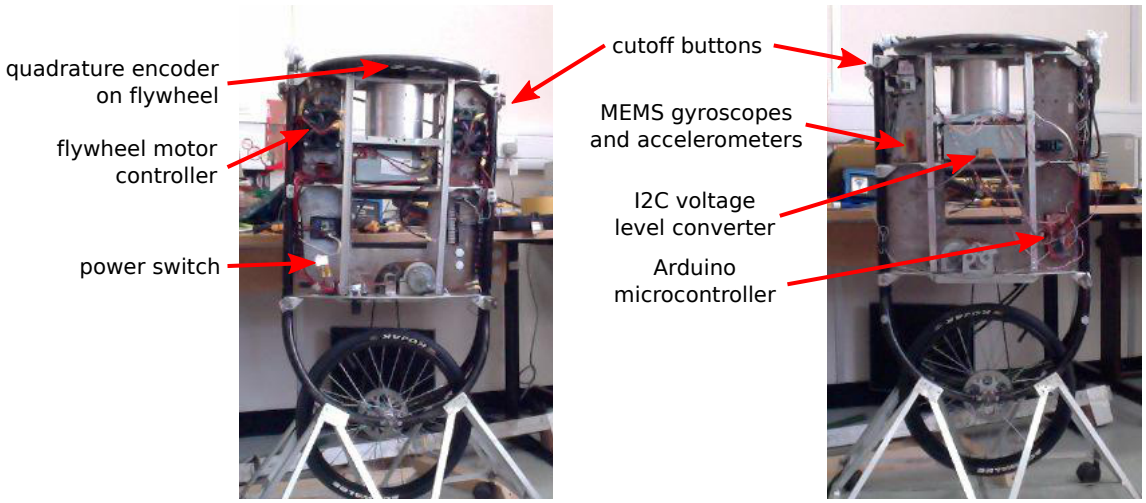


Figure 11: Summary of changes to unicycle hardware

to be approximately constant during a trial.[5] After removing the bias, the readings can be converted to radians per second with a calibration factor from the datasheet [13].

To keep track of the unicycle's orientation, these angular velocities must be integrated. There are 3 commonly used ways to keep track of rotations in 3D, of which unit quaternions were chosen as best suited to the project.

Euler angles These are the yaw, roll and pitch angles shown in Figure 4. They are required for the controller, and so other forms must be converted to them. However, they suffer from gimbal lock—in certain positions, it is impossible to represent a small rotation with a small change in the Euler angles. They also require trigonometric functions in the integration loop: a problem for fast integration on embedded platforms.

Direction Cosine Matrices (DCMs) A DCM is an orthonormal rotation matrix, representing the rotation from the global coordinate system to the body's coordinate system. They require only basic linear algebra to understand. However, when errors in integration accumulate, the matrix will no longer be orthonormal, and there is no clear way to fix this.

Unit quaternions Quaternions extend the complex numbers into 3D. Just as a complex number of unit magnitude can represent a rotation in the 2D plane, a quaternion of unit magnitude can represent a rotation in 3D. Although they may appear confusing, they have the fastest integration step, do not suffer from gimbal lock and can be normalised by simply dividing by the magnitude.

[5]The bias drift over 10 seconds was measured to be negligible. For longer trials, this would have to be taken into account.

Having chosen to use quaternions to track the orientation, various formulae had to be derived to convert to and from this form. These mathematical techniques are summarised here—they are reasonably simple to derive by considering the DCM as a composition of Euler rotations.

Quaternion Integration Given the unit quaternion representing a rotation from the global coordinate system to that of the unicycle, $\boldsymbol{q}[n] = q_w + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k}$, and rate gyro outputs ω_x, ω_y and ω_z, we can integrate the angular velocities with $\boldsymbol{q}[n+1] = \boldsymbol{q}[n]\left(1 + \frac{\omega_x \Delta t}{2}\boldsymbol{i} + \frac{\omega_y \Delta t}{2}\boldsymbol{j} + \frac{\omega_z \Delta t}{2}\boldsymbol{k}\right)$. For more details on this, see [14].

Euler Angles of a Quaternion Unfortunately, the mechanical analysis of Forster, and thus the simulation of the ideal unicycle, uses a different angle convention to all other sources located. We chose to continue with this convention, and so the expressions for converting a quaternion to Euler angles had to be rederived. Using a convention of yaw around the vertical y-axis, roll around the forward x-axis and pitch around the sideways z-axis (the axes of the mounted gyroscope) we get:

$$\begin{aligned}
\phi &= \tan^{-1}\left(\frac{2q_xq_y + 2q_yq_w}{-q_x^2 - q_y^2 + q_z^2 + q_w^2}\right) \\
\theta &= \sin^{-1}\left(2q_wq_x - 2q_yq_z\right) \\
\psi_f &= \tan^{-1}\left(\frac{2q_xq_y + 2q_zq_w}{-q_x^2 + q_y^2 - q_z^2 + q_w^2}\right)
\end{aligned}$$

Angular Velocities To calculate the angular rates, $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}_f$, we convert the quaternion to a DCM, and then apply expressions for the derivatives of the Euler angles of a DCM. This could cause a division by zero in a gimbal lock situation, but fortunately the unicycle never reaches such positions.

$$\begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{bmatrix} = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_zq_w & 2q_xq_z + 2q_yq_w \\ 2q_xq_y + 2q_zq_w & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_xq_w \\ 2q_xq_z - 2q_yq_w & 2q_yq_z + 2q_xq_w & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix}$$

$$\begin{aligned}
\dot{\phi} &= \frac{(d_{12}d_{32} - d_{12}d_{33})\omega_x + (d_{11}d_{33} - d_{13}d_{31})\omega_y}{d_{13}^2 + d_{33}^2} \\
\dot{\theta} &= \frac{d_{22}\omega_x - d_{21}\omega_y}{\sqrt{1 - d_{23}^2}} \\
\dot{\psi}_f &= -\frac{d_{23}(d_{21}\omega_x + d_{22}\omega_y)}{d_{21}^2 + d_{22}^2} + \omega_z
\end{aligned}$$

Initial Angle To determine the initial angle of the unicycle, we take an average accelerometer reading while the unicycle is initially stable. This is a 3D vector, representing the direction of gravity in the reference frame of the rotated unicycle. By considering

the DCM resulting from rotations in pitch and roll, we can determine the pitch and roll angles, and use these to construct an initial rotation quaternion.

$$\boldsymbol{D} = \begin{bmatrix} \cos\psi_f & -\sin\psi_f & 0 \\ \cos\theta\sin\psi_f & \cos\theta\cos\psi_f & -\sin\theta \\ \sin\theta\sin\psi_f & \sin\theta\cos\psi_f & \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} \cos\theta\sin\psi_f \\ \cos\theta\cos\psi_f \\ -\sin\theta \end{bmatrix}$$

$$\theta = -\sin^{-1}(a_z)$$

$$\psi_f = \tan^{-1}\left(\frac{a_x}{a_y}\right)$$

$$\boldsymbol{q} = \left(\cos\left(\frac{\theta}{2}\right) + \boldsymbol{i}\sin\left(\frac{\theta}{2}\right)\right)\left(\cos\left(\frac{\psi_f}{2}\right) + \boldsymbol{k}\sin\left(\frac{\psi_f}{2}\right)\right)$$

Gyro Noise and Drift The approach described above is vulnerable to drift: accumulating errors in the integration of the gyroscope, especially when the zero-offset of the gyroscope changes. To evaluate the effect of this, the gyro was sampled for 10 seconds when stationary. The rate readings are shown in Figure 12. It is clear that the noise and drift are on the same order of magnitude, over a 10 second trial, the zero-offset changed by about 0.1 deg s^{-1}, leading to a drift of below 1°. This was judged as acceptable. If we wished to conduct longer trials, we could use one of a variety of fusion methods such as state observers, Kalman filters or one of many highly tuned implementations developed by UAV hobbyists [15].

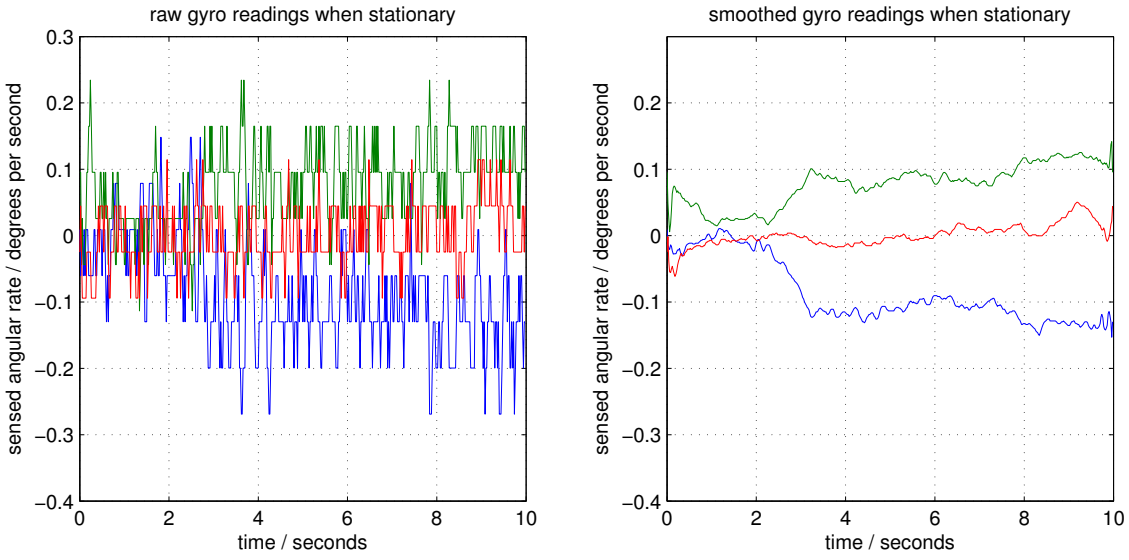


Figure 12: Gyro readings when stationary

Accelerometer Noise Figure 13 shows the accelerometer noise when stationary (the mean reading has been subtracted).[6] These noises correspond to noise standard deviations for pitch and roll of about 0.5°, but this can be reduced by averaging many readings. More detail on the use of the accelerometer can be found in Douglass's report.

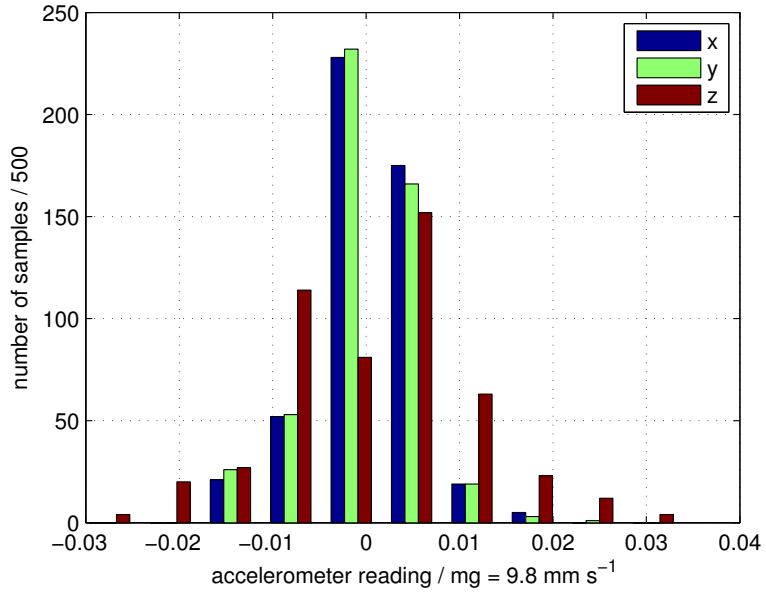


Figure 13: Accelerometer readings when stationary

3.1.2 Position Sensing

In order to reduce the number of state variables necessary, Rasmussen used self-centred coordinates to keep track of the position of the target. In these coordinates, the unicycle is at (0, 0) and faces along the positive x-axis, as shown in Figures 8 and 14. At each timestep, we must use the change in yaw angle $\Delta\phi$ and the change in wheel angle $\Delta\psi_w$ (along with wheel radius r_w) to calculate the new target position. From Figure 14 we can see that x_c and y_c must be modified as follows:

$$\begin{bmatrix} x_c[n+1] \\ y_c[n+1] \end{bmatrix} = \begin{bmatrix} \cos(-\Delta\phi) & -\sin(-\Delta\phi) \\ \sin(-\Delta\phi) & \cos(-\Delta\phi) \end{bmatrix} \begin{bmatrix} x_c[n] \\ y_c[n] \end{bmatrix} + \begin{bmatrix} -r_w\Delta\psi_w \\ 0 \end{bmatrix}$$

The shaft encoder used to detect wheel angle changes has a quantisation error of around 1mm. The buildup in yaw error over the trial should be less than 1°, so when the distance from the target is around 1m, we can expect errors of around $1\text{mm} + 1^\circ \cdot \frac{\pi}{180} \cdot 1\text{m} \approx 2\text{cm}$. This should be more than accurate enough for the application.

[6]Note that the noise characteristic is very different for one axis compared to the other 2 - this is because this axis is perpendicular to the chip, and as such is built differently.

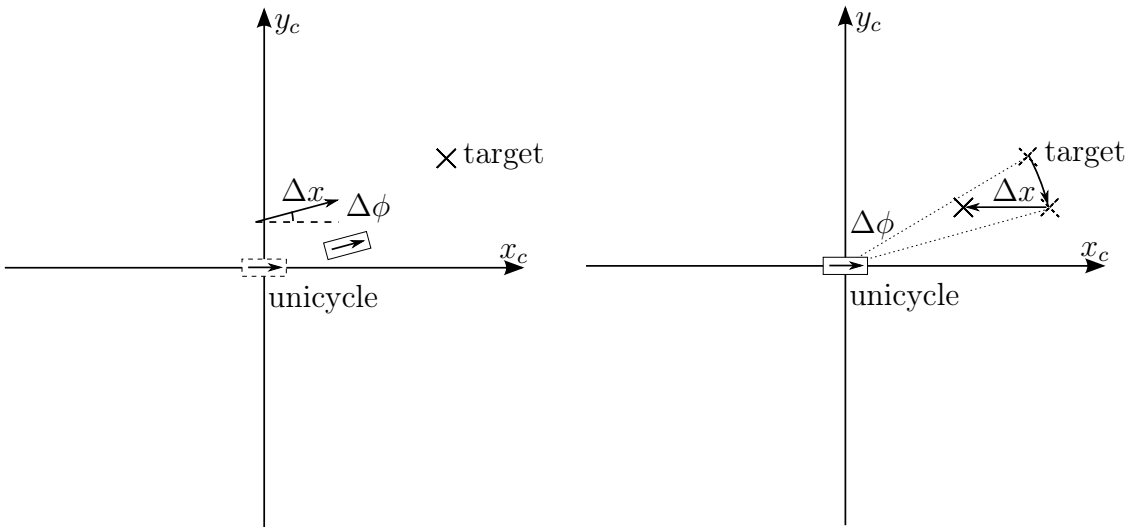


Figure 14: Effect of unicycle movement on target position in self-centred coordinates

3.1.3 Wheel Speed Sensing

The unicycle uses a quadrature encoder to measure the position and speed of the wheel. This is a digital sensor, which generates a pulse every time the wheel moves through $\frac{1}{512}$ of a revolution. Merely counting these pulses gives extremely good measurements of position (accurate to around 1mm) but it is harder to measure speed.[7] The approach being used previously was to measure the number of pulses in 40ms, but this leads to a quantisation error of 25 pulses per second, or 2.5cm s^{-1}. An alternative method is to measure the amount of time between the last two pulses, but at high speeds this time may be very short, and this time may be hard to measure accurately.

This became more important when we added a quadrature encoder for the flywheel: the sensors used are much less accurate than those on the wheel. There are only 96 pulses per revolution, and there is significant non-uniformity around the wheel. This means that the first method would have a quantisation error of 94°/s, and the second method would only be accurate to within around 50%. This is unacceptable accuracy.

After reading a comparative analysis of various methods [17], we decided on a method that combines the benefits of the two approaches above. It adapts the size of the 40ms window to contain an integer number of pulses, eliminating quantisation error. At high speeds, it is similar to the first scheme, considering the length of many pulses to reduce the effect on non-uniformity and timing errors. At sufficiently low speeds, it adapts to only using the most recent pulse, just as the second scheme does. Figure 15 explains how it works—note that some thought is required about what should be returned when no pulses are observed in the window. Implementing this scheme approximately doubled the

[7] More details on how the quadrature encoders are used to sense position can be found in Douglass's report.

signal-to-noise ratio, as determined from the GP hyperparameters.

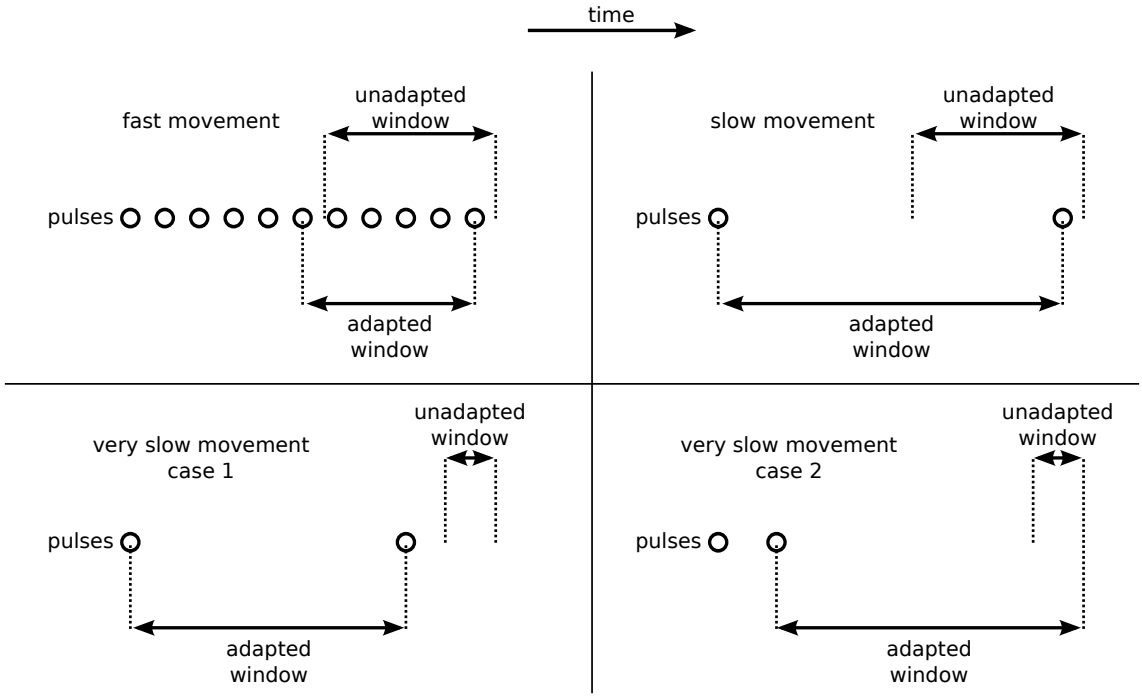


Figure 15: Adaptive-window encoder speed measurement

3.2 Fall Protection

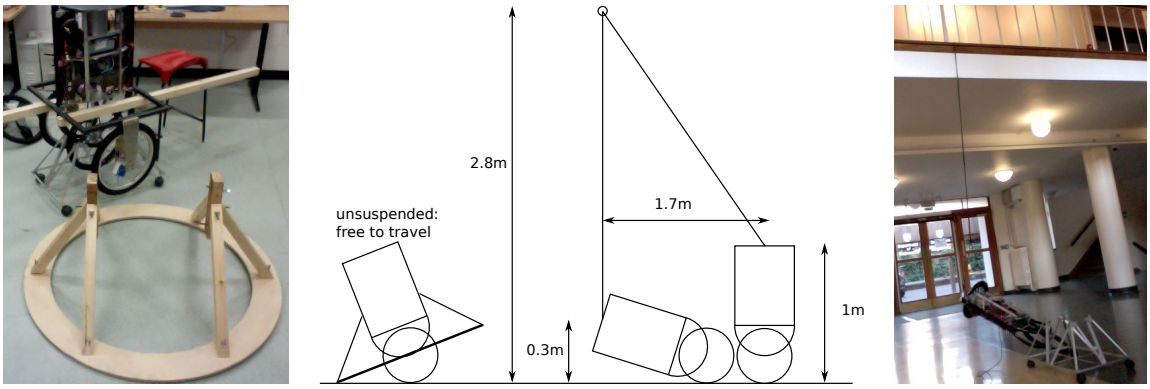
Due to the trial-and-error nature of the RMLC method, the unicycle must be able to survive using a controller that may be worse than none at all. When the dynamics model is inaccurate, it may choose a controller that throws the 30kg unicycle towards the ground with the combined forces of gravity and the motor. Figure 16 shows the effect of repeated trials on the aluminium bar used in the previous year.

Figure 16: Damaged aluminium box section from previous year's testing

Two approaches had been used previously: suspending the unicycle from a crane in the Structures Lab, and equipping the unicycle with a support structure that hits the ground and stops the fall before it becomes dangerous. Suspending the unicycle limits its range of motion, as if it travels too far the rope will pull it over, as shown in Figure 17(b).

We suspected that the ability to travel freely would be important for the unicycle, since if an RMLC system is unable to experience control policies that travel a long way,

the dynamics model will never learn that this can happen and that it should try to avoid it. As such, we chose to build a wooden skirt which would travel with the unicycle and protect it from falling, as shown in Figure 17(a). More details on the design of the skirt, including a CAD drawing of the mounted skirt, can be found in Douglass's report.



(a) Wooden skirt (b) Movement range (c) Suspended

Figure 17: Unicycle fall protection methods

3.3 Inability to Spin

After building the skirt and preparing the necessary hardware and software, we started training the unicycle. However, it made very little progress, and would consistently seem to fall to the side, without turning in that direction and driving forwards to pick itself up. We tried modifying the controller to send the maximum motor command to the flywheel at all times, and found that it could only turn around 30° before falling.

It is very important that the robot be able to turn sufficiently fast to be able to point in the direction in which it is falling, before it has fallen too far to recover. We considered various possible causes for the slow turning:

- Adding the skirt had increased the moment of inertia (MoI), making it spin slowly.
- Friction with the ground was causing it to turn too slowly.
- The gear ratio for the flywheel was chosen badly.
- The flywheel had insufficient moment of inertia to turn the robot fast enough, even without the above effects.
- The flywheel motor was insufficiently powerful.

To try to quantify this, we suspended the robot above the ground, and tested the response of the system to a step increase in flywheel command. We then used a mathematical model to predict how changing various variables would affect the spin speed.

3.3.1 Spin Analysis

The response of a DC motor can be well modelled as [18]:

$$\frac{\Omega_{\text{motor}}(s)}{V(s)} = \frac{K_t}{(R + Ls)(Js + b) + K_t^2}$$

where we define:

Symbol	Description
$\omega_{\text{motor}}(t)$	Speed of motor shaft
$v(t)$	Motor input voltage
$\Omega_{\text{motor}}(s), V(s)$	Laplace transforms of the above
K_t	Motor torque constant
R	Motor armature resistance
L	Motor armature inductance (usually negligible)
J	Moment of inertia of object on motor shaft
b	Damping on motor shaft (sometimes negligible)

The effect of the gear train is analogous to that of a transformer: the flywheel MoI, J_f appears to the motor as an MoI of $J = \frac{J_f}{n^2}$. More obviously, it has the effect that $\omega_{\text{flywheel}} = \frac{\omega_{\text{motor}}}{n}$.

So, neglecting L and b, we have:

$$\begin{aligned}\frac{\Omega_{\text{flywheel}}(s)}{V(s)} &= \frac{K_t}{R\frac{J_f}{n^2}s + K_t^2} \\ &= n^{-1}K_t^{-1}\frac{1}{\frac{RJ_f}{n^2K_t^2}s + 1}\end{aligned}$$

The corresponding step response is of the form:

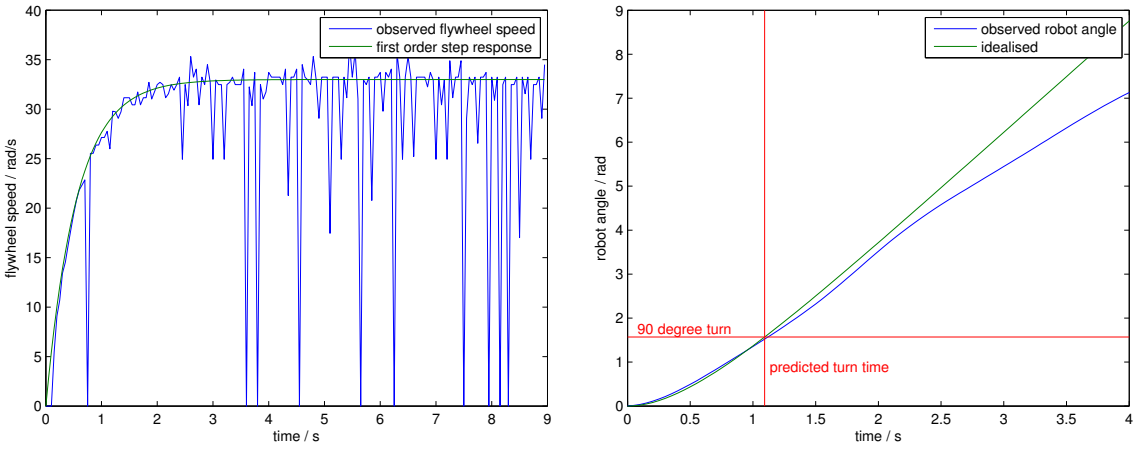
$$\omega_{\text{flywheel}}(t) = K(1 - e^{-\frac{t}{\tau}})$$

Fitting this to the observed step response, as in Figure 18(a), tells us that $K = 33$ and $\tau = 0.55$ seconds.[8]

If we integrate to get the flywheel angle, and then assume that there are no external torques on the robot and that the robot has MoI J_r:[9]

$$\begin{aligned}\omega_{\text{flywheel}}(t) &= K(1 - e^{-\frac{t}{\tau}}) \\ \theta_{\text{flywheel}}(t) &= K(t + \tau(e^{-\frac{t}{\tau}} - 1)) \\ \theta_{\text{robot}}(t) &= \frac{J_f}{J_r}K(t + \tau(e^{-\frac{t}{\tau}} - 1))\end{aligned}$$

[8] The egregious noise in the flywheel speed trace is caused by the problem described in Section 3.4.3.
[9] Including the skirt, if attached.



(a) Flywheel speed

(b) Robot angle

Figure 18: Response to step in flywheel command

By fitting (see Figure 18(b)) we get $\frac{J_r}{J_f} = 13$ (including the skirt). As the angle increased, the rope supporting the unicycle twisted, slowing the robot's spin, but the initial fit is good. We can now solve for the time to turn 90°. This model predicts a turn time of 1.1 seconds for the suspended robot: this is far too slow, and so clearly contact with the ground is not entirely to blame—we must look elsewhere to solve the problem.

We can check the observed value of $\frac{J_r}{J_f}$ by estimating the moments of inertia. Modelling the flywheel as a uniform disc of mass 8kg, the robot as a uniform cuboid of mass 24kg, and the skirt as mass concentrated at 55cm (the ring) and 40cm (the struts) from the body:

$$\begin{aligned}
J_f &= \frac{1}{2}MR^2 = \frac{1}{2}\cdot 8\cdot (0.2)^2 \\
&= 0.16 \text{ kg m}^2 \\
J_r &= \frac{1}{12}ML^2 = \frac{1}{12}\cdot 24\cdot (0.4)^2 \\
&= 0.32 \text{ kg m}^2 \\
\text{mass of skirt} &= 4.4 \text{ kg} \\
\text{mass of structs} &= 3.5 \text{ kg} \\
J_{\text{skirt}} &= 3.5\cdot 0.4^2 + 4.4\cdot 0.55^2 \\
&= 1.89 \text{ kg m}^2 \\
\frac{J_r}{J_f} &= 2 \\
\frac{J_r + J_{\text{skirt}}}{J_f} &= 13.8
\end{aligned}$$

This backs up the result of the previous analysis, which suggested $\frac{J_r+J_{\text{skirt}}}{J_f} = 13$.

With this model, we can predict the effect on the turn time of various changes to the suspended robot—Table 3.3.1 shows various possibilities, and makes it clear that reducing or removing the skirt is most valuable improvement to make.

gear ratio	flywheel MoI	robot MoI	turn time
1x	1x	1x	1.09s
1x	2x	1x	0.94s
1.6x (optimum)	2x	1x	0.87s
1x	10x	1x	0.85s
4.8x (optimum)	10x	1x	0.51s
1x	1x	$\frac{3.6}{13}$x (lightweight skirt)	0.50s
1x	1x	$\frac{2}{13}$x (removed skirt)	0.36s

Table 1: Ideal turn time under various conditions

3.3.2 Solving the Spin Problem

Having realised this, we redesigned the skirt to reduce its MoI. We reduced the width of the rim from 12cm to 2cm, and replaced the wooden struts with aluminium members. This reduced the MoI of the skirt by a factor of 7x, and could still support the weight of the unicycle. However, after a few falls, the wood had started to crack and one of the aluminium struts broke at the joint. Despite reinforcing the wood with glue and aluminium struts, we were unable to reach a design for the skirt that was both sturdy and light.

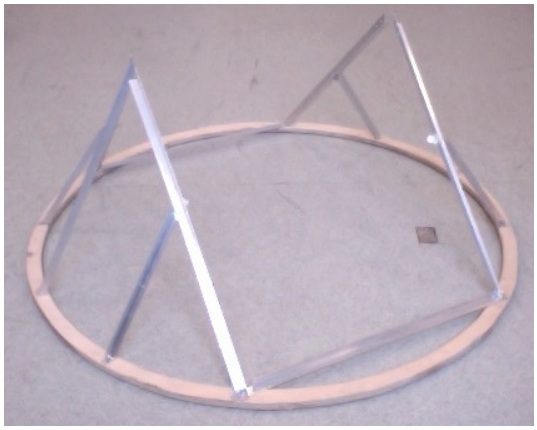


Figure 19: Reduced-weight skirt

Eventually, we chose to use a rope tied to the mezzanine in the foyer of the Engineering department, shown earlier in Figure 17(c). In addition to imposing an unwelcome limit on the range of the unicycle, this is a very busy area, and so testing can only be done outside of the department's normal opening hours. Unfortunately, this was the only way that the problematic effects of the skirt could be avoided while maintaining a safe testing procedure.

3.4 Known Issues

Throughout the design process we found some problems with the unicycle's hardware and software that we were unable to fix. When this occurred, we did our best to stop them from interfering with the testing process.

3.4.1 Toothed Belt

The robot uses a toothed belt to transfer torque to the wheel, shown in Figure 20. During tests, it would frequently slip when the controller demanded large changes in motor velocity. This leads to a mismatch between the encoder reading and the actual position, which would cause great problems for the dynamic model. As such, it was critical to make sure that we observed carefully for trials in which the belt slipped, and removed the corrupted data from the logs. Towards the end of the testing process, we installed an extra strut, also shown in Figure 20, to maintain tension in the belt.

Figure 20: Toothed belt and tensioning strut

3.4.2 Motor Control

Unpredictably, one or both of the motors would fail move at all. This proved extremely hard to debug, since sometimes even touching the circuit with a single probe of a voltmeter would cause them to start working again. At one time, we were able to investigate the problem with an oscilloscope—it appeared that the inputs to the motor controller were behaving correctly, and that while the PWM input went through to the output, both terminals of the motor were driven with exactly the same PWM signal, with no potential difference across the motor.

The cause of this is unknown, but likely an electrical problem judging from the unpredictable behaviour and sensitivity to contact from the voltmeter probe. Because we were unable to prevent it from happening, we modified the software to test both motors before every trial. This allowed us to power cycle the system to avoid the problem whenever it was detected.

3.4.3 Flywheel Encoder

The flywheel encoder is prone to generating spurious pulses—these can manifest as sudden changes in direction, which are perceived as sudden stops by the speed-sensing algorithm described earlier. Figure 21 shows the effect of these spurious transmissions—clearly extremely severe, and very problematic for the dynamic model to predict.

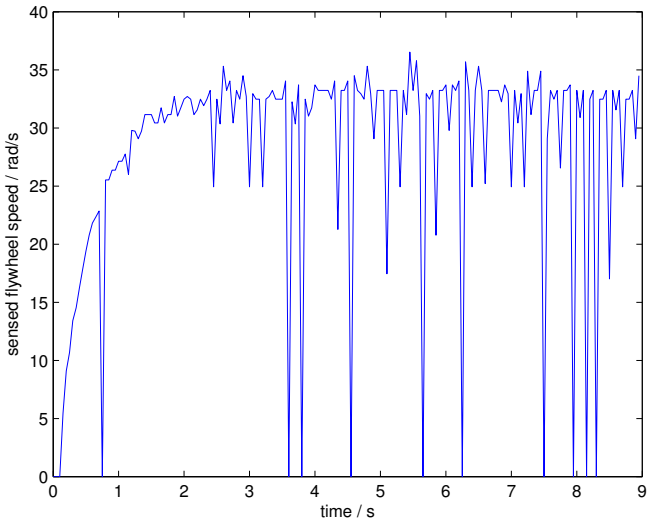


Figure 21: Corrupted flywheel speed measurements

As described in Douglass's report, the flywheel encoder uses Schmitt triggers to digitise the analog encoder outputs. Figure 22 shows a trace of the input and output to this signal cleaning circuit (the two waveforms have been traced in an image editor for clarity). There appears to be a negative spike in the signal, part-way through the high cycle. This is due to interference between the two quadrature sensors.[10] Unfortunately, the low hysteresis voltage on the model of Schmitt trigger used[11] means that this is enough to reach the threshold and cause problems. The extent of this problem was limited slightly by placing a larger piece of cardboard between the two sensors.

The other problem visible is that the range of the analog signal is not 0–5V, but around 0–3.4V. This is because the photodiode in the sensor is not fully conducting when pointed at a white part of the pattern. To ensure that the Schmitt trigger acted properly, we reduced its supply voltage to 3.3V. While this meant that the input exceeded the supply voltage and that the 3.3V logic output was not matched to the 5V input on the AVR (the processor on the Arduino), both the Schmitt trigger and the AVR's datasheets stated that these were acceptable [19] [20]. This does have the effect of further reducing the hysteresis voltage, but combined with the added occlusion, it helped control the problem.

Despite these changes, the problem is still very sensitive to the orientation of the

[10] Covering one emitter removed the spike.
[11] 0.4V minimum, 0.7V typical

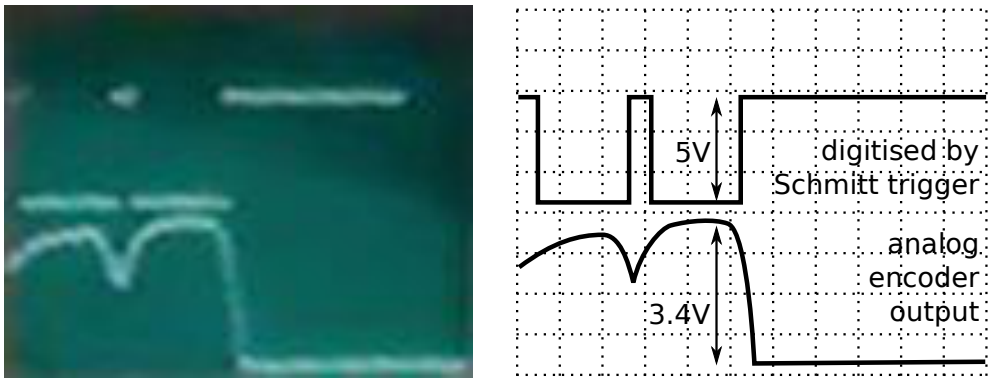


Figure 22: Oscilloscope trace of flywheel encoder outputs

sensors, and the shocks undergone during trials can cause it to resurface. To avoid this, we adapted the software to try to test for this problem before every trial, and we carefully inspected the logs after the trials, in an attempt to detect signs of corrupted data.

3.4.4 I^{2}C Communications

The ITG-3200 and ADXL345 both reside on the same I^{2}C bus, a digital communications protocol that allows the Arduino to read and write data to its peripherals. We chose to use the I^{2}C driver provided by the Arduino project, which served us well initially. However, we found that occasionally, and unpredictably, the Arduino would stop responding. We narrowed this down to an infinite loop in the I^{2}C driver, waiting for a response from the peripheral.

We were unable to identify or remove the cause of this problem, which could be anything from a race-condition in the software to damage to one of the involved ICs.[12] Fortunately, the problem occurred reasonably rarely, so we chose to implement timeouts in the I^{2}C driver. Whenever one of these timeouts occurs, the error is signalled, and the data can be discarded.

4 Results of Learning

Michaelmas term was spent installing the new hardware and rewriting the software—about 2000 lines of code. In particular, about a week was spent trying to debug the I^{2}C communications problem described in Section 3.4.4. We also discussed potential solutions for protecting the unicycle from falls, and then designed the wooden skirt, which took several days in the workshop to construct.

In Lent term, we learned how to use the RMLC system, and then applied it to the 2D system. We required 4 full training sequences to reach satisfactory performance, each

[12] Soldering mistakes led to a number of short circuits and over-volting of the ICs during the development process.

requiring several days of trials interspersed with controller training. When they failed, we had to look into the results to determine exactly why they failed and how the performance could be improved. After a few weeks, we had investigated the changes necessary to achieve good performance, and started working on the hardware and software necessary for 3D balance: another 1500 lines of code, along with the flywheel controller and sensor, installing the accelerometer, etc. At the end of the Lent term, we attempted to start training, but had to rebuild the skirt to make it lighter, requiring another few days in and out of the workshop. When this failed, we chose to use the rope, and by the middle of the Easter vacation we had started training in earnest.

These training sequences took much longer—around 1–2 weeks each due to increased computational requirements—and when term re-started we were only able to test the system before 9am and after 5pm. After the causes of the failure of the first two training sequences had been determined, resolved and tested, we were able to run a third sequence, although by this point very little time remained.

4.1 Learning the 2D System

McHutchon's report describes the processes that the RMLC system goes through when successfully learning to balance the 2D system. Our aim in learning the 2D system was to verify that our sensors and controller worked correctly, and that we were using the machine learning system appropriately. We ran 3 training sequences before deciding to move onto the 3D unicycle.

Due to a calculation error, the first sequence used a loss function that very heavily penalised the distance moved (see Section 2.2.6)—a movement of 10cm was considered "as bad as" an angle of 10°. This led to a controller which would intentionally fall as fast as possible to stop the unicycle without moving. Figure 23 shows the effect of this on the trajectory of the trials: by the end, the distance moved is consistently extremely small. Although the optimal controller would be one that kept both distance and angle close to zero, the problem is that intentionally falling is a local optimum: initially, it is more attractive to fall than to try to balance, and it never gathers any data suggesting that it can balance.

The second sequence fixed this error and, as shown in Figure 24, this sequence successfully generated controllers than could balance the unicycle for 15 seconds, with no sign of growing instability.[13] However, this is despite the fact that the uncertainty of the prediction grows extremely rapidly. Clearly, the dynamics model has managed to capture enough of the dynamics of the real system to allow balancing the unicycle, yet is very uncertain about its predictions.

We suspected this was due to the high levels of noise in the sensors. In particular, the

[13]These prediction plots are described in Section 2.1.

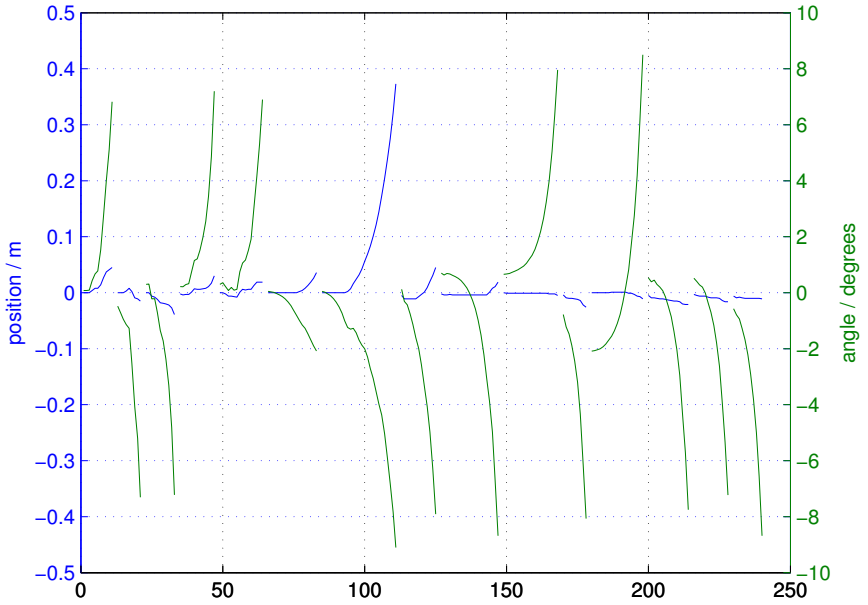


Figure 23: Trials from the first sequence

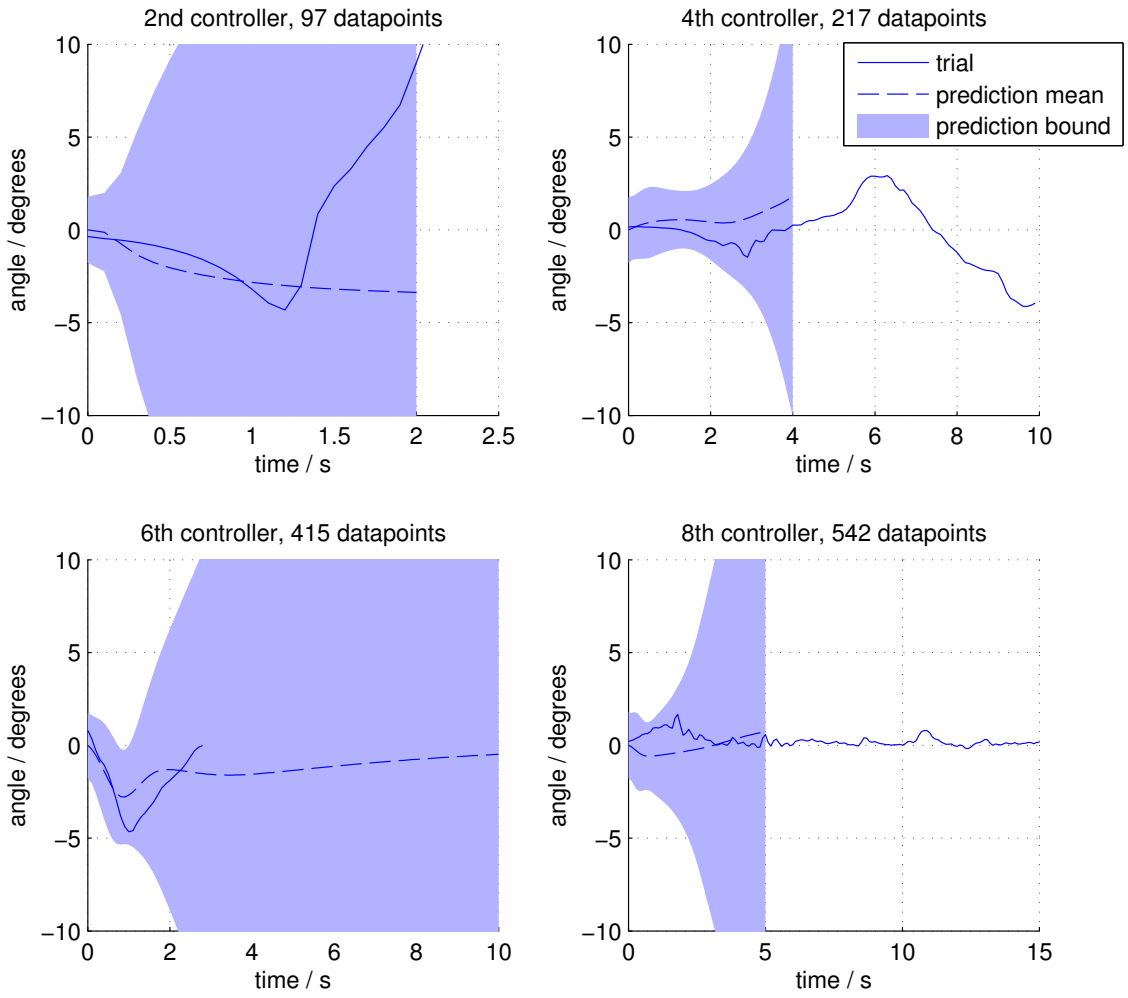
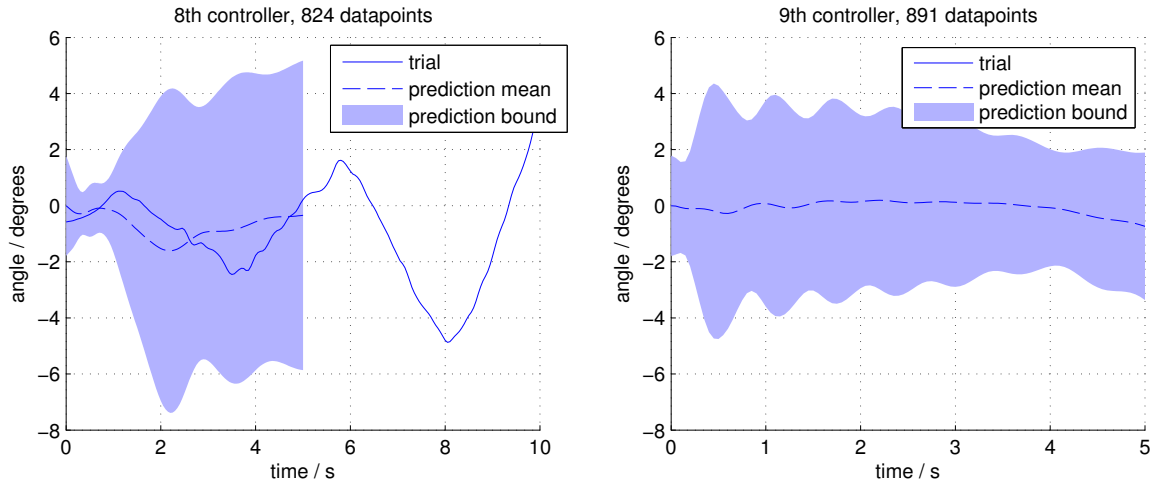


Figure 24: Sampled angle predictions and trials from 2nd training sequence

wheel speed is detected by differentiating the output of a digital position sensor, which dramatically amplifies the quantisation noise. McHutchon reported that decreasing the timestep (Section 2.2.7) help address problems with sensor noise, so we started another sequence with the timestep reduced from 100ms to 50ms. This produced the controller and prediction shown in Figure 25(a)—much improved, but still inadequate.



(a) 50ms timestep

(b) 2nd order Markov

Figure 25: Final predictions of 2D training sequence

During trials, we observed the unicycle shaking and a lot of noise coming from the gear train. We suspected that problems like gear backlash were contributing to the poor performance of the system, and so we gave the dynamic model access to two consecutive states when predicting (as described in Section 2.2.2). This further improved the confidence of the prediction, shown in Figure 25(b), and the quality of the controller. Unfortunately, the log data for this trial was lost, but a video remains: the unicycle is observed to remain balanced, but to be in a limit cycle oscillation, slowly falling one way, before picking itself up and returning to slowly falling.

This poor performance of the resulting controller is likely due to the large dead-zone in the motor: on a PWM scale from -100% to 100%, there is a dead-zone from -10% to 10%. Figure 26 shows a trial exhibiting stable limit cycle behaviour, along with the motor command and dead-zone. The motor command is within the dead zone for quite a significant duration of the cycle. There are a number of possibilities for remedying this problem, such as manually compensating for the dead-zone, or allowing the RMLC system to use non-linear controllers that could learn to compensate for the dead-zone. Due to time constraints, however, we accepted the performance as satisfactory and moved on to the 3D unicycle.[14]

[14]There is a technique for dealing with motor dead-zones known as "dithering", which involves adding a high-frequency periodic signal to the motor command, which will push the motor in and out of its dead-

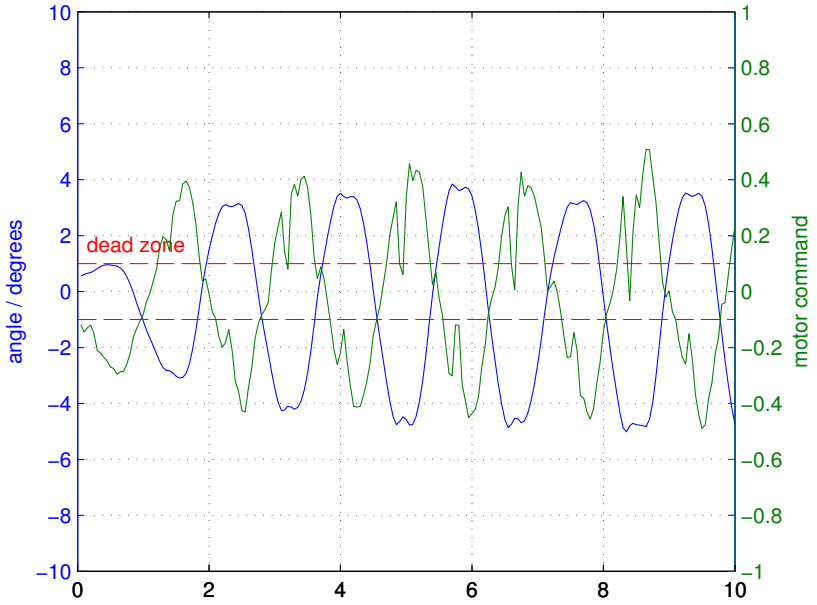


Figure 26: Trial exhibiting limit cycle behaviour

4.2 Learning the 3D Unicycle

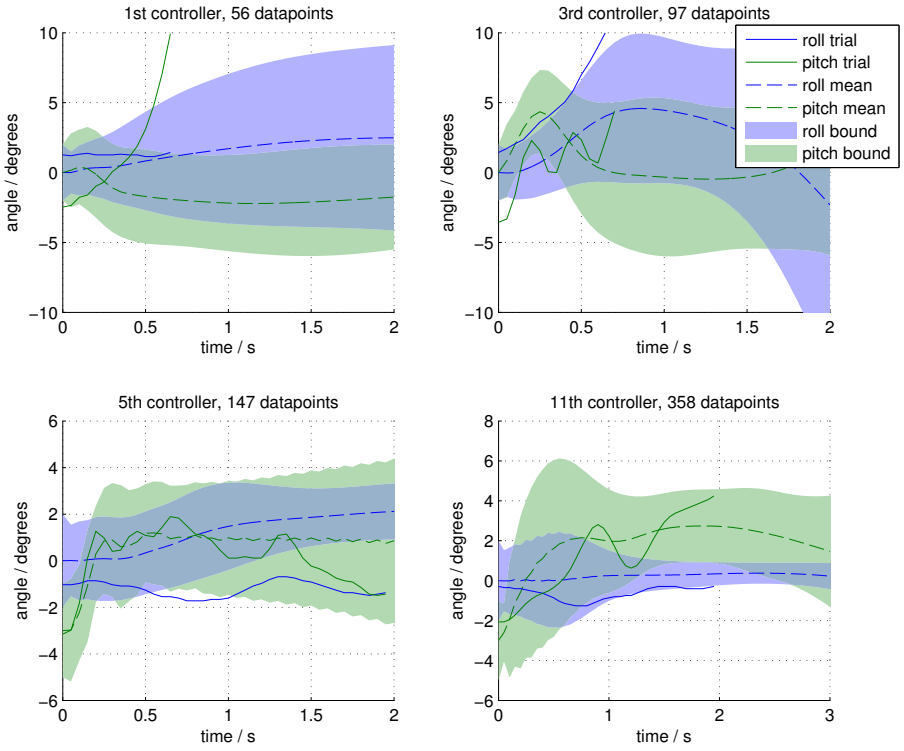
Once all necessary work was complete, we initiated a training sequence for the 3D unicycle. As detailed in Section 3.2, we realised that we couldn't use the skirt for fall protection, and decided to use a loose rope to protect the unicycle. We then iterated through a training sequence, and observed that the system learned to balance the robot very quickly.

As shown in Figure 27(a), though initial trials disagreed greatly with the prediction, by the 3$^{\text{rd}}$ trial the system was learning to balance in pitch, and the trials start to resemble the predictions. By the 5$^{\text{th}}$ trial the unicycle could control both pitch and roll, and when the sequence ended after 11 trials, the system could confidently predict both roll and pitch angles.

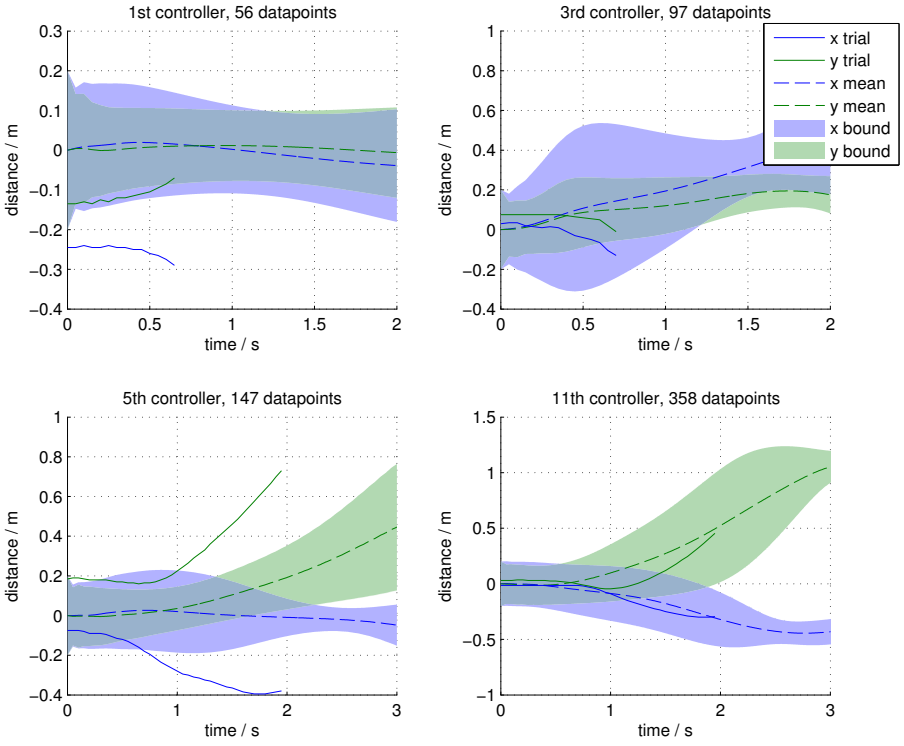
However, the length of the trials is limited by the rope: once the unicycle travels a certain distance to the side, the rope becomes tight and pull the unicycle over, interfering with the dynamics in an unpredictable way. If the unicycle is to balance for any length of time it must learn to stay within this limited region. Figure 27(b) shows that the unicycle failed to achieve this: by the 11$^{\text{th}}$ controller, the dynamics model predicts reliably the position trajectory, but the system is unable to keep itself near the centre.

Upon investigation, we realised that there was an error in the position representation: the self-centred position integration described in Section 3.1.2 was not being performed properly, and so there was no way for the quadratic controller to generate the correct commands to return to the origin. We fixed this and proceeded with a second training

zone. The system will filter this rapidly oscillating force out, effectively smoothing the dead-zone out. McHutchon observed much better performing controllers for the 2D system—ironically, it is possible that the high noise levels of his angle sensors had a dithering effect on the motor, improving the performance.



(a) Unicycle angles



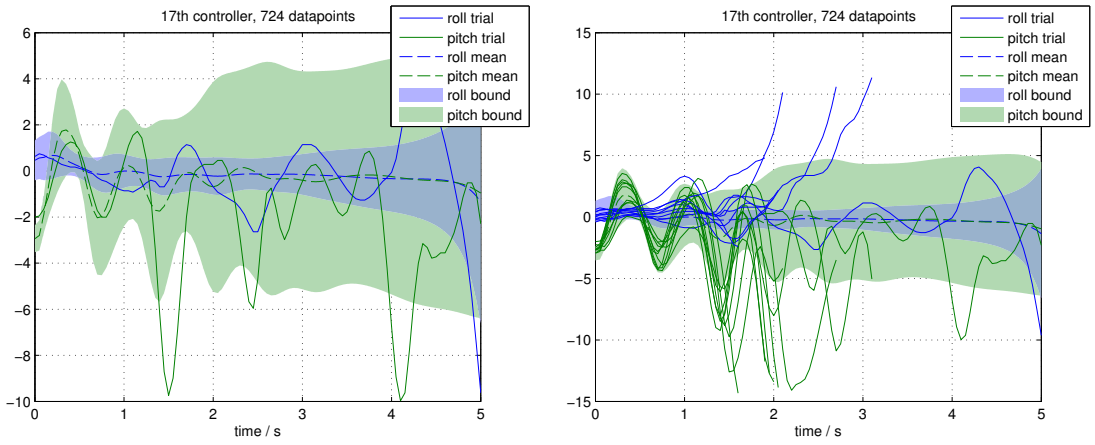
(b) Unicycle position

Figure 27: 1st sequence of 3D trials

sequence, but it failed to learn a satisfactory dynamics model—we looked for causes, and found the problem with the flywheel speed sensor described in Section 3.4.3. We resolved this and proceeded with a 3rd sequence.

At the start of the 3rd sequence, we observed a learning process similar to the 1st sequence: initially, the robot quickly learned to balance in pitch, and then to balance in roll too, but still tended to wander off until the support rope would allow it no farther. Changing the loss function to penalise off-centre positions more heavily did not appear to significantly affect the produced controllers.

At this stage, after testing 13 controllers and gathering around 30 seconds worth of experience with the system, we noticed the problem with the initial state distribution described in Section 2.2.3. After fixing this problem, the simulations represented reality much better, and the RMLC system was able to train better controllers. After 3 more trials, the trained controller successfully kept the unicycle upright for the optimisation horizon of 5 seconds, as shown in Figure 28(a). However, it is clear that the dynamics model is not fully capturing the behaviour of the unicycle: the real trial's trajectory makes large excursions outside of the predicted confidence intervals. Furthermore, trials will consistently follow this unpredicted trajectory, as shown in Figure 28(b), which shows the trajectories of 9 trials with the same controller. It also shows that the 5 second trial was unrepresentative of the performance of the controller.



(a) Single trial

(b) Many trials

Figure 28: Prediction and trials for the 17th controller of the 3rd 3D training sequence

By this point, the trials cumulatively represented about 35 seconds of experience with the system, and performing a single simulation took more than 20 minutes—typically, several hundred simulations are required to train an optimal controller. We conducted 2 more trials, but performance did not improve significantly.

4.3 Possible Reasons for Failure of Long-Term 3D Balance

The RMLC system did achieve limited success in the 3D balance problem. In the 1st sequence, when it was incapable of controlling its position, the controllers were capable of controlling angle well, balancing until the unicycle reached the end of its rope. In the 3rd sequence, it initially repeated the feat of balancing, then produced controllers which would stay within the range of the rope, but could only rarely remain upright for more than 2–3 seconds.

In contrast, in the simulations conducted by Rasmussen, RMLC was able to develop a quadratic controller that could balance the ideal unicycle indefinitely with a comparable amount of experience. Since the ideal unicycle model was developed to approximate the dynamics of the real unicycle, it was hoped that these results would be replicated. However, there are a number of differences between the ideal unicycle and the real system which may be to blame for the unsatisfactory performance.

4.3.1 Restricted Range

The smooth balance, achieved in the 1st and 3rd sequences, was very promising. It showed that the system was capable of balancing the unicycle, while also being able to predict horizonal motion, as demonstrated by Figure 27. This would suggest that it would be reasonably simple for the system to generate the correct turning commands to follow a circular path around the target.

It is possible that the reason the system then seemed to lose the ability to reliably balance the unicycle after this point is that it was unable to follow such a path and remain within the very limited range of the rope. However, the failure of the dynamics model to predict the unreliable balance, shown by Figure 28(b), suggests that this is not entirely to blame for the performance of the system.[15]

4.3.2 Unmodeled Dynamics

The success of the 2nd order Markov model shows that the idealisation neglects some hidden states in the system such as gear backlash or sticking points. It is very possible that some hidden states remained even with the 2nd order model, leaving the dynamics model incapable of predicting the effect of these states. Unfortunately, since the computational requirements grow quadratically with the number of consecutive states used for prediction, increasing this further is not currently viable, so other approaches to countering these hidden states would have to be explored.

[15] Of course, it is possible that the only reason the dynamics model failed towards the end of the 3rd sequence is that we changed the initial state distribution and loss function during the sequence, and it just needed more data to get a handle on the dynamics. However, the computational cost of the RMLC system means that adding more data was impossible.

4.3.3 Changes in the Dynamics

A core assumption of the RMLC system is that the dynamics of the system, $\boldsymbol{f}$, stay constant through the training sequence. However, there are many physical aspects of the unicycle which can change from trial to trial: tyre inflation, battery charge, component positions,[16] initial angle,[17] etc. It is unknown how much these small changes may have affected the performance of RMLC, but it is possible that the failure of the dynamics model at the end of the 3$^{\text{rd}}$ sequence can be attributed to these.

4.3.4 Sensor Noise and State Estimation

While some of the sensors are very high quality, some are extremely bad—in particular, the flywheel encoder is one of the least reliable ways of sensing speed—and these could be causing serious problems for the dynamics model. Despite this, the dynamics model appeared to be capable of predicting the unicycle's behaviour well in the 1$^{\text{st}}$ sequence, so perhaps the state estimation is not as large a problem as might be expected.

McHutchon was able to significantly improve the performance of a trained controller by adding using the dynamics model to generate a linear state filter, and using this in conjunction with the final controller. However, the effect of state filtering on the learning process has not been investigated.

5 Conclusion

After successfully adding a variety of new sensors, replacing the controller, reimplementing the unicycle's software and installing a motor controller for the flywheel, the unicycle was brought into a suitable state for balancing. Even with this work, the unicycle remains a very hard system to stabilise, due both to the low quality of the sensors and actuators, and the complexity of the ideal mechanics of the system.[18]

Despite these difficulties, RMLC was capable of designing a controller that could balance the unicycle until it pulled its support rope taut, and was sufficiently robust to maintain balance after this point in some trials. It also designed a controller that, in one trial, balanced the unicycle for 5 seconds, although it appears that the system was forced to sacrifice angular stability to stay near the centre of the testing area.

When using conventional control, the focus is on modelling and stabilising idealised system dynamics. With RMLC, the focus is shifted to minimising the effect of noise to ensure a reliable dynamics model can be learned, and accurately representing the real

[16] The violent shocks the unicycle undergoes when falling were capable of shifting the battery in its mounting, although much effort was put into minimising the extent of this.

[17] The approach of using the accelerometer only before the trial, and not during, increases the sensitivity to the initial position of the unicycle.

[18] As evidenced by the difficulties encountered by Vos and Naveh in designing a controller for it.

world problem to ensure that trained controllers perform well. Techniques such as shorter timesteps, 2^{nd} order Markov modelling and carefully avoiding corrupted data were found to be critical for the former concern, and choosing an appropriate loss function and initial state distribution were important for the latter. For difficult systems, these problems may be much simpler than dealing directly with the dynamics.

In the case of the unicycle system, previous attempts to stabilise it had problems modelling the gyroscopic and Coriolis effects that are inherent in the system's dynamics. RMLC frees the designer from these worries, by learning the dynamics directly from experience with the system. In particular, it can learn the dynamics with remarkably small amounts of experience with the system, in most cases producing a reasonable controller with less that 10 seconds of experience.

However, it failed to produce a controller that could reliably balance the controller for more than 3 seconds. There are a number of possible reasons for this—most obvious is the restricted range of the unicycle, preventing the unicycle from travelling more than a metre in any direction. RMLC's initial success balancing the unicycle suggests that it would be capable of indefinite balance if some of the current difficulties could be resolved.

6 Further Work

Since the main objective of the project, indefinite balance of the unicycle, was not achieved, the obvious avenue for future work is improving either the RMLC system or the unicycle to the point that this is possible. Such an achievement would provide very compelling evidence of RMLC's suitability for difficult control problems.

For further investigation of the performance of RMLC on the unicycle system, it would be very useful to build a smaller unicycle. This would completely solve the fall-protection and range issues, and it could be built without the design flaws that have plagued this project. A system like Lego would allow the mechanical design to be varied easily, allowing the investigation of different dynamic systems.

If future projects intend to continue with the current system, they would do well to focus on improving the sensors. Obviously, physical improvements to the sensors would allow more accurate estimation of the state, and thus more accurate predictions from the dynamics model. Further to this, McHutchon had success using a Kalman filter based on the dynamics model to improve the state observations, and suggested using a non-linear approximation to the dynamics model to further improve performance—using the full dynamics model is currently impossible, since with 900 datapoints it simulates at 150x slower than real-time on a very fast processor.

In order to balance indefinitely, it may be necessary to use a more complicated controller than the quadratic controller used here. Naveh used different control schemes in different regions of state space to improve performance. More generally, Gaussian Pro-

cesses have previously been used for the controller as well as the dynamic model to allow radically different control strategies to be used in different regions of state space. This is a necessity for tasks such as swinging up an inverted pendulum from the downwards position, and it seems likely that the greater flexibility of such a controller would improve balancing performance, and allow the unicycle to recover from a greater range of position & angle offsets. However, this would almost certainly require a more powerful processor than the AVR used for this project.

When the unicycle was first built by Mellors and Lamb in 2004/2005, they included a wireless transmitter and receiver for the purpose of allowing a human to remotely control the unicycle. Once the unicycle is capable of reliably balancing and controlling its position, it would be interesting to see how well the controller deals with changes in its target position. Since it has not been optimised for this—McHutchon noted the poor disturbance rejection of RMLC-trained controllers—it seems likely that it will not be able to track the target position well.

Additionally, while the RMLC system requires little experience with the data to generate a good controller, the current implementation requires huge amounts of computational time to generate the controllers. The 3$^{\text{rd}}$ 3D training sequence used 10 days of CPU-time on a high-end server. In addition to optimisations to the training code such as approximations of GPR, it would be interesting to investigate the how fast the system learns when the amount of optimisation time is more tightly limited.

Despite the flaws of heavy computational requirements and sensitivity to disturbances, this approach to control has proven itself by both its partial success on this system, and its success balancing the ideal unicycle in Rasmussen's simulations, with extremely little experience with the system. Addressing these concerns would be very interesting, and could make the RMLC method significantly more useful.

References

[1] M. Mellors. Robotic unicycle: Mechanics & control. CUED Master's Project, 2005.

[2] A. Lamb. Robotic unicycle: Mechanics & control. CUED Master's Project, 2005.

[3] N. D'Souza-Mathew. Balancing of a robotic unicycle. CUED Master's Project, 2008.

[4] D. Forster. Robotic unicycle. CUED Master's Project, 2009.

[5] D. Vos. Nonlinear control of an autonomous unicycle robot: Practical issues. MIT PhD Thesis, 1992.

[6] Y. Naveh et al. Nonlinear modelling and control of a unicycle. *Dynamics and Control*, 9:279–296, 1999.

[7] C. E. Rasmussen and M. P. Deisenroth. Recent advances in reinforcement learning. chapter Probabilistic Inference for Fast Learning in Control, pages 229–242. Springer-Verlag, Berlin, Heidelberg, 2008.

[8] A. McHutchon. Machine learning for control. CUED Master's Project, 2010.

[9] D. Zenkov et al. The Lyapunov-Malkin theorem and stabilization of the unicycle with rider. *Systems and Control Letters*, 46:293–302, 2002.

[10] Murata Manufacturing Co. Development of the unicycle-riding robot: Murata girl. Retrieved 18/05/2011 from `http://www.murata.com/new/news_release/2008/0923.html`, 2008.

[11] M. P. Deisenroth and C. E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA, June 2011.

[12] C. E. Rasmussen and C. Williams. Gaussian processes for machine learning. MIT Press, 2006.

[13] ITG-3200 data sheet.

[14] J. Favre, B. M. Jolles, O. Siegrist, and K. Aminian. Quaternion-based fusion of gyroscopes and accelerometers to improve 3D angle measurement. *Electronics Letters*, 42(11):612–614, 2006.

[15] T. Pycke. gluonpilot's attitude estimation. Retrieved 20/05/2011 from `http://www.gluonpilot.com/wiki/Matlab_attitude_estimation`, 2010.

[16] A. Douglass. Machine learning for control. CUED Master's Project, 2011.

[17] R. Petrella, M. Tursini, L. Peretti, and M. Zigliotto. Speed measurement algorithms for low-resolution incremental encoder equipped drives: a comparative analysis. In *Electrical Machines and Power Electronics, 2007. ACEMP '07. International Aegean Conference on*, pages 780 –787, sept. 2007.

[18] K. Lundberg. Notes on feedback systems. 6.302 Lecture Notes, 2008.

[19] NXP HEF4093B data sheet.

[20] ATmega1280/V data sheet.

A Evaluation of Risk Assessment

The main concern in our risk assessment was the mechanical risk to the unicycle, the operator and their surroundings. The trial-and-error nature of RMLC means that initial controllers may well have positive feedback, and the unicycle will throw itself toward the ground much faster than it would fall naturally. Additionally, even when this is not the case, the unicycle is composed of more than 30kg of steel in a very unstable configuration.

As described in the body of the report, a very large proportion of the work on this project was spent trying to determine a way to test the unicycle without risking it hitting the ground when it fell. Fortunately, all the methods used did successfully protect the unicycle during trials. Although the technique used to protect the unicycle changed (switching from the skirt to the rope) we were careful to ensure that the rope would not allow the unicycle to hit the ground or any bystanders.

When we were using the rope, the range of the unicycle was limited, but when using the skirt and testing the 2D system, it was free to travel as far as it wanted. This presented the risk of the unicycle building up speed and driving into a wall, or one of the desks in the Control Lab. Previously, the only way to disable the unicycle during a trial was to use a nearby computer—to address this, we added cutoff buttons to the unicycle, allowing us to quickly disable it before it damaged itself or the surroundings. This made testing easy and safe.

During the project, we discovered a risk that we didn't foresee. When we left the unicycle on its stand in the Control Lab, the cleaner tried to move it. Since, even on the stand, the unicycle is difficult to handle, it was knocked over. Fortunately, the cleaner was not hurt and damage to the unicycle and floor were inconsequential. To prevent this happening again we left a "Do Not Touch" sign on the unicycle whenever it was unattended, and made sure that the unicycle was left safely out of the way.

There was also a small electrical risk in the project. Although the unicycle's power supply is only 12V, it has a very high current capacity and can produce sparks capable of welding contacts together. Previously, the key to the unicycle's power switch had been lost and to turn the power on, the unicycle had to be "hot-wired" by directly connecting the main power supply wires every time. Early in the project, we installed a power switch, removing this risk. We also endeavoured to insulate much of the exposed wiring on the unicycle to prevent short circuits.