Machine Learning for Control

by

Rodrigo Queiro (DOW)

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: _____

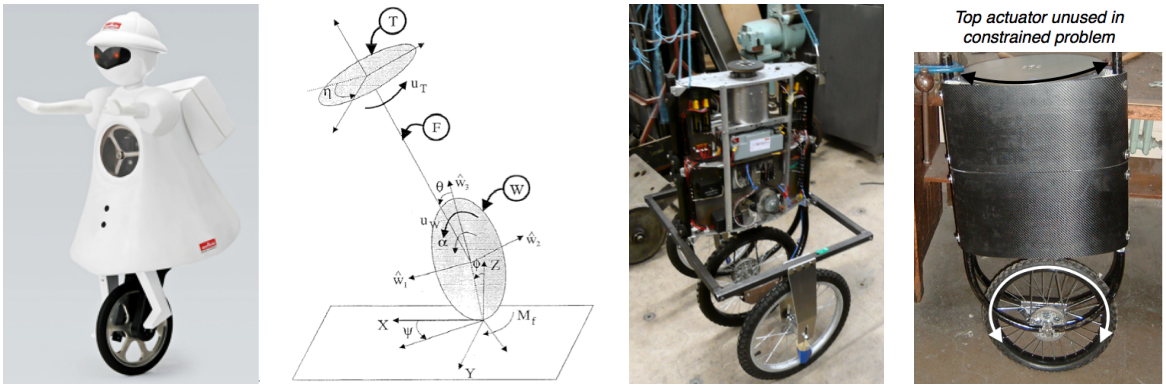
Technical Abstract

technical abstract...

Contents

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 [1, 2], as in Figure 1(a). Other use a more realistic (and challenging) model of a flywheel in the horizontal plane [3, 4], as in Figures 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 [5, 6] 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 it to 2D dynamic control: 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 [7]. 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 [3, 4], 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 faster enough. Vos and Naveh had to use complex, approximate techniques to model the unicycle.

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 [8, 9].

In 2009/2010, McHutchon successfully applied RMLC to the 2D system [10]. 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 principle 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.

2 Reinforced Model Learnt Control

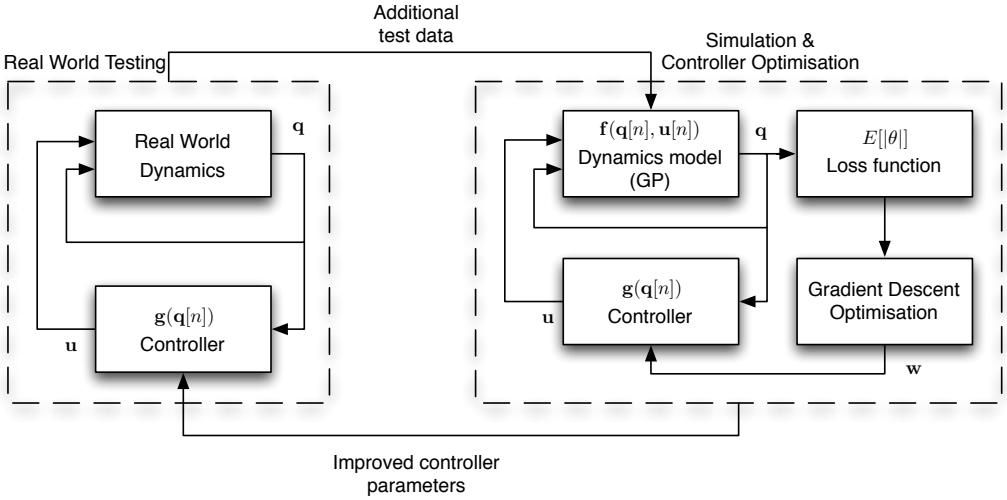


Figure 2: Reinforced Model Learnt Control

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 control input at time n. In the case of the unicycle, $\boldsymbol{q}$ consists of 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.

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 [11]. 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 penalises uncertainty as well as falling.

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 a 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.

2.1 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 it, 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.

2.1.1 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 expo-

nential 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 [11]. 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 inf.

2.1.2 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, 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.

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

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:

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

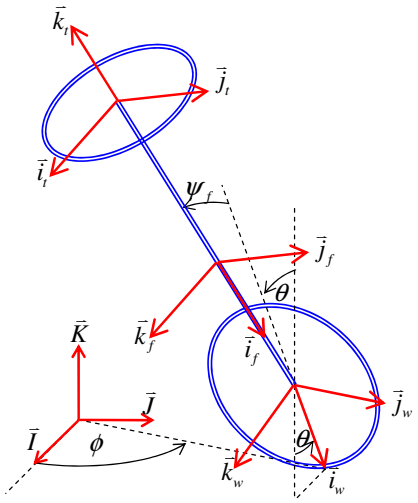


Figure 3: Diagram showing Euler angles for the rotation of the unicycle (from [7])

This significantly improved the accuracy of the dynamics model, which in turn suggests that unobserved states are significant in the behaviour of the real unicycle.

2.1.3 Controller Form

The most basic form 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 4—this is equivalent to the XOR problem, and can be solved by using a quadratic controller. This takes the form:

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

It was found that the controllers performed significantly better when using $\boldsymbol{q}_2[n]$ as input, instead of $\boldsymbol{q}[n]$. To understand 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,

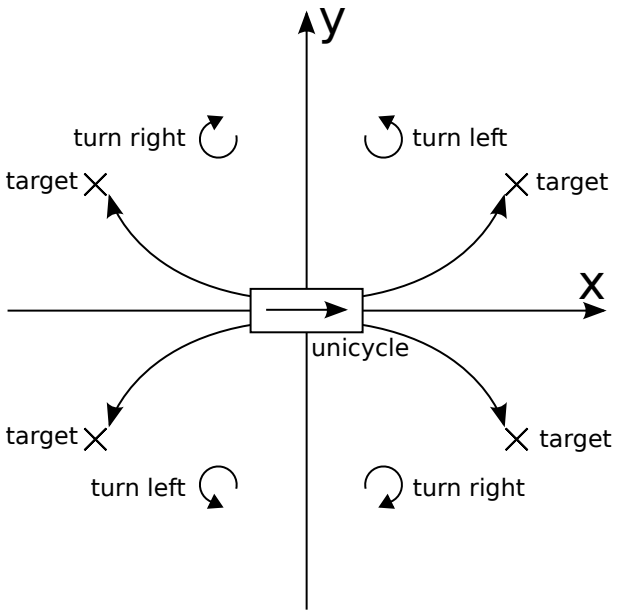


Figure 4: Correct turning command in unicycle-centred coordinates

and this additional freedom improved the subjective quality of the controllers trained.

2.1.4 Loss Function

The main concerns when choosing a loss function are 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 the same loss function structure, with characteristic widths of 9° on the

angles, 1 metre on the distances and 1 rps & 3 rps on the yaw & flywheel rates respectively. Unlike the ideal unicycle, the real unicycle has an upper limit on the flywheel speed, so the "spinning-top" strategy is not viable, so it is possible that the yaw & flywheel rates need not be penalised. This was not tested.

2.1.5 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.

3 Hardware and Software

When the project started, some of the requirements were already clear, as a result of the previous work on the unicycle. As such, extensive changes to the hardware and a complete rewrite of the controller software were 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.

- 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.
- 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 a user presses a switch on the chassis or when the unicycle nears the ground.

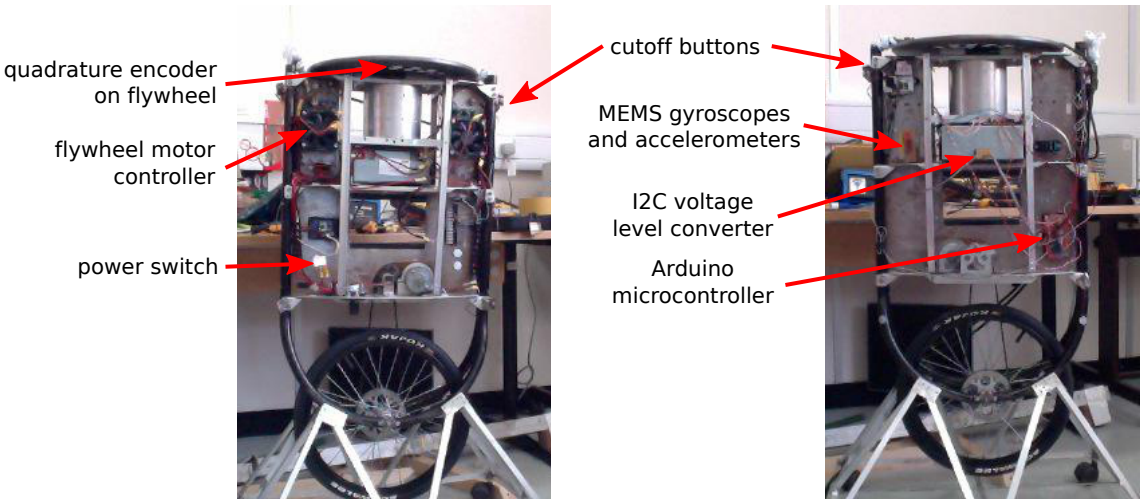


Figure 5: Summary of changes to unicycle hardware

3.1 Sensing

3.1.1 Angle Sensing

The unicycle had previously used a MEMS 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 to be approximately constant during a trial. They can then be converted to radians per second with a calibration factor from the datasheet [12].

To keep track of the absolute angle, these angular velocities must be integrated. There are 3 possible 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 3. 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 tightest integration loop, do not suffer from gimbal lock and can be normalised by simply dividing by the magnitude.

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 [13].

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.

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

$$\dot{\phi} = \frac{(d_{12}d_{32}-d_{12}d_{33})\omega_x + (d_{11}d_{33}-d_{13}d_{31})\omega_y}{d_{13}^2+d_{33}^2}$$

$$\dot{\theta} = \frac{d_{22}\omega_x - d_{21}\omega_y}{\sqrt{1-d_{23}^2}}$$

$$\dot{\psi}_f = -\frac{d_{23}(d_{21}\omega_x + d_{22}\omega_y)}{d_{21}^2+d_{22}^2} + \omega_z$$

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 6. 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 [14].

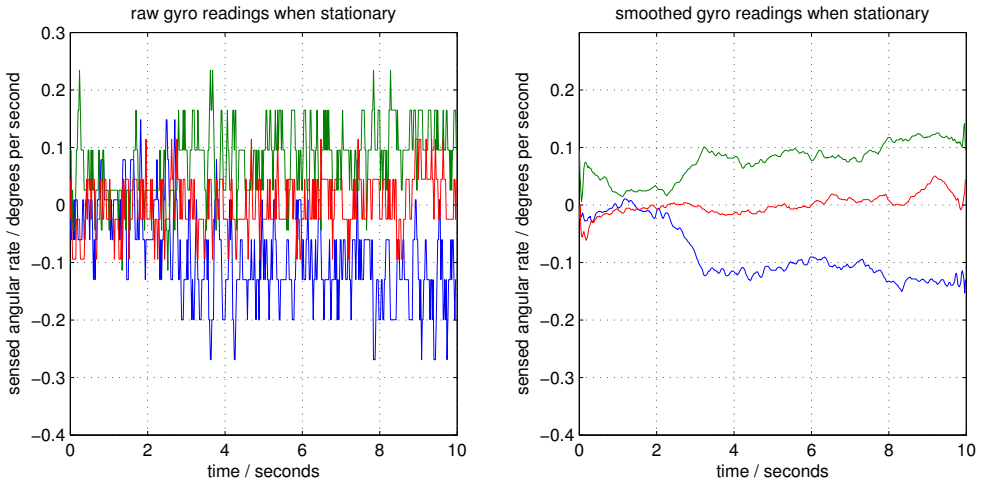


Figure 6: Gyro readings when stationary

Accelerometer Noise Figure 7 shows the accelerometer noise when stationary (the mean reading has been subtracted). (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 is built differently.) This corresponds to noise standard deviations for pitch and roll of about 0.5°, but this can be reduced by averaging many readings.

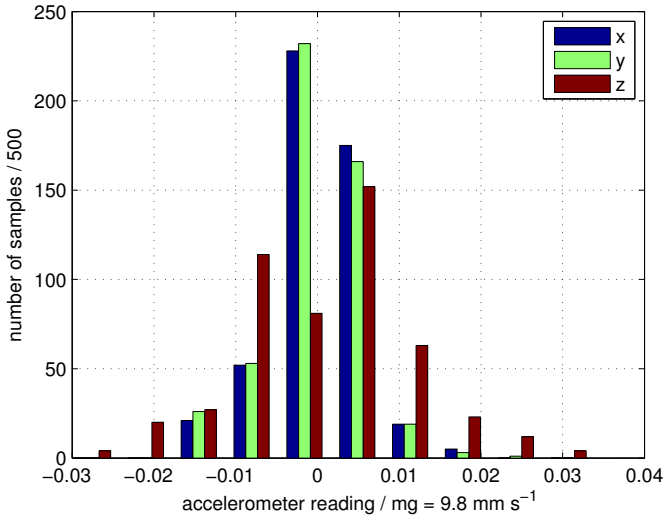


Figure 7: Accelerometer readings when stationary

3.1.2 Position Sensing

In order to reduce the number of state variable 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 Figure 4. 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 8 we can see that x_c and y_c must be modified as follows:

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

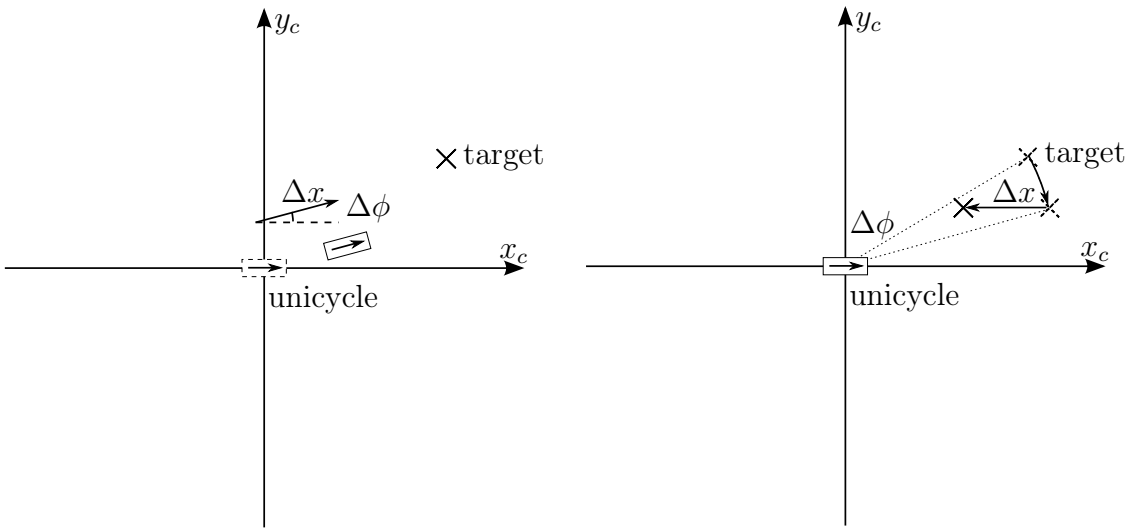


Figure 8: Effect of unicycle movement on target position in self-centred coordinates

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.

3.1.3 Wheel Speed Sensing

When we started the project, a quadrature encoder was being used 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. 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}.

(TODO: explain how shaft encoders track position? Refer to Alan's report?)

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 72 pulses

per revolution, and there is significant non-uniformity around the wheel. This means that the first method would have a quantisation error of 125°/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 [15], 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, avoiding quantisation error. Figure 9 explains how it works—note that some thought is required about what should be returned when no pulses are observed in the window.

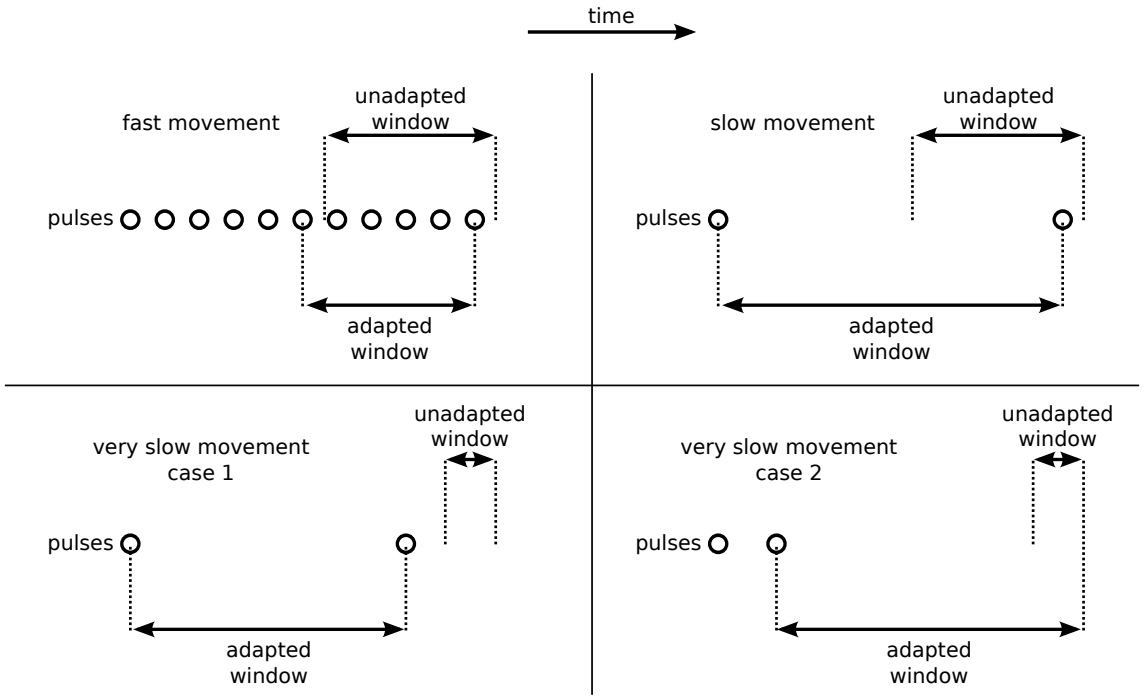


Figure 9: Adaptive window encoder speed measurement

4 Results and Discussion

5 Conclusions

References

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

[2] 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.

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

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

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

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

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

[8] Carl Edward Rasmussen and Marc Peter Deisenroth. Recent advances in reinforcement learning. chapter Probabilistic Inference for Fast Learning in Control, pages 229–242. Springer-Verlag, Berlin, Heidelberg, 2008.

[9] Marc P. Deisenroth and Carl 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.

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

[11] Carl Edward Rasmussen. Gaussian processes for machine learning. MIT Press, 2006.

[12] ITG-3200 data sheet.

[13] 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.

[14] Tom Pycke. gluonpilot's attitude estimation. Retrieved 20/05/2011 from `http://www.gluonpilot.com/wiki/Matlab_attitude_estimation`, 2010.

[15] 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.

A Extra Stuff