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

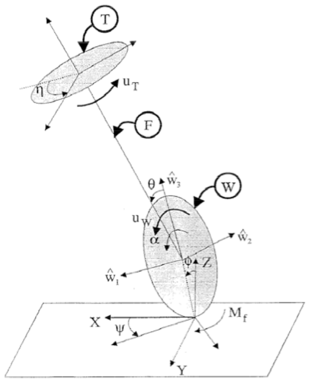
technical abstract...

Contents

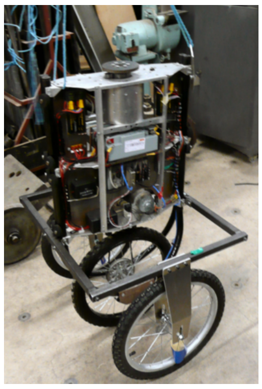
1 Introduction

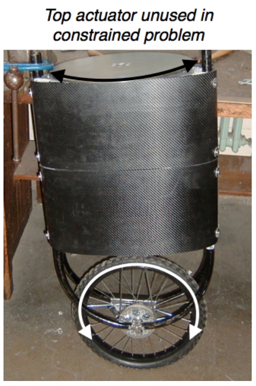
Balancing a unicycle is a very challenging task for a human rider. Many attempts have been made to achieve this task, using a variety of models for the action of the rider. Some represent the rider as a flywheel or pendulum in the coronal plane, allowing direct compensation of falling to the side [1, 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

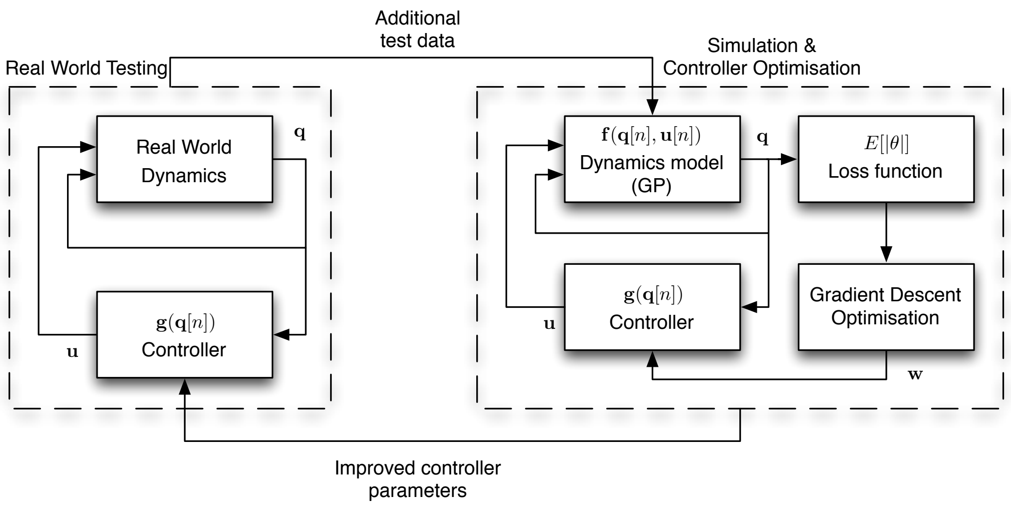


Figure 2: Reinforced Model Learnt Control

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 **rollout** 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 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 [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.

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

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 2nd order Markov chain:

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

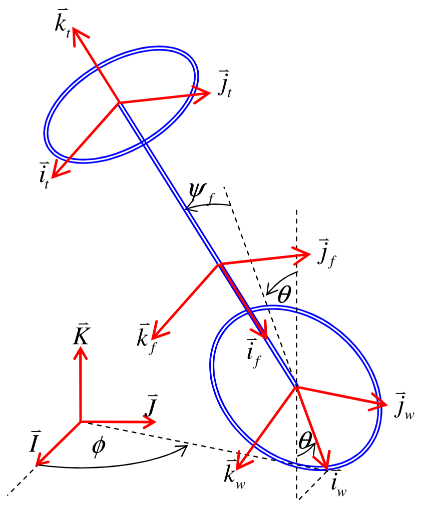


Figure 3: Diagram showing Euler angles for the rotation of the unicycle (from [7])

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,

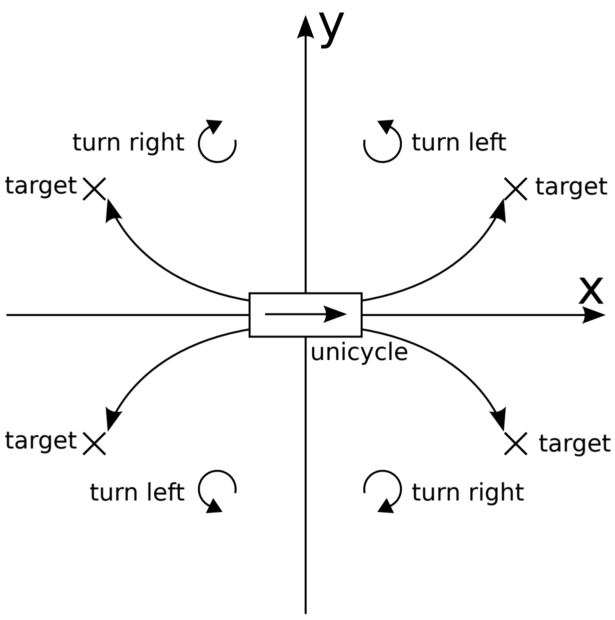


Figure 4: Correct turning command in unicycle-centred coordinates

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 Apparatus and Experimental Results

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. `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.

A Extra Stuff