0 Contents

1 Theory and Methods

Explain the assumptions behind the theoretical development you are using and the application of the theory to your particular problem. Any heavy algebra or details of computing work should go into an appendix. This section should describe the running of the experiment or experiments and what equipment was used, but should not be a blow by blow account of your work. Experimental accuracy could be discussed here.

1.1 The Inverted Pendulum (IP)

The Inverted Pendulum is an inherently unstable system with highly nonlinear dynamics and is under-actuated.

1.1.1 Dynamics

The full state space equations for the inverted pendulum as defined in fig. 1.1 are given by:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{\left(\frac{2M-m}{m}F_2 - F_1\right)cos\theta + g(M+m)sin\theta - mL\dot{\theta}^2 sin\theta cos\theta}{(M+msin^2\theta)} \\ \dot{\theta} \\ \frac{F_1 + F_2 cos(2\theta) + msin\theta(L\dot{\theta}^2 - gcos\theta)}{L(M+msin^2\theta)} \end{bmatrix} \tag{1.1}$$

Ignoring second order terms and linearising about $\boldsymbol{x}_e = [x_e, \dot{x}_e, \theta_e, \dot{\theta}_e]^T = [0,0,0,0]^T$:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{\frac{2M-m}{m}F_1 - F_2 + g(M+m)\theta}{M} \\ \dot{\theta} \\ \frac{F_1+F_2-gm\theta}{lM} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & g\frac{M+m}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{mg}{lM} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -\frac{1}{M} & \frac{2M-m}{Mm} \\ 0 & 0 \\ \frac{1}{lM} & \frac{1}{lM} \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{1.2}$$

Which, as expected, is unstable since $det(\lambda I - A) = 0 \implies \lambda^2(\lambda^2 + \frac{mg}{lM}) = 0$. Note, for small angles the natural frequency of a non-inverted pendulum is $\omega_n = \sqrt{\frac{mg}{lM}} = \sqrt{\frac{0.1\times 9.81}{0.5\times 1}} \approx 1.40 rad/s$. Therefore, the time constant for the system is $\tau \approx 0.70s$.

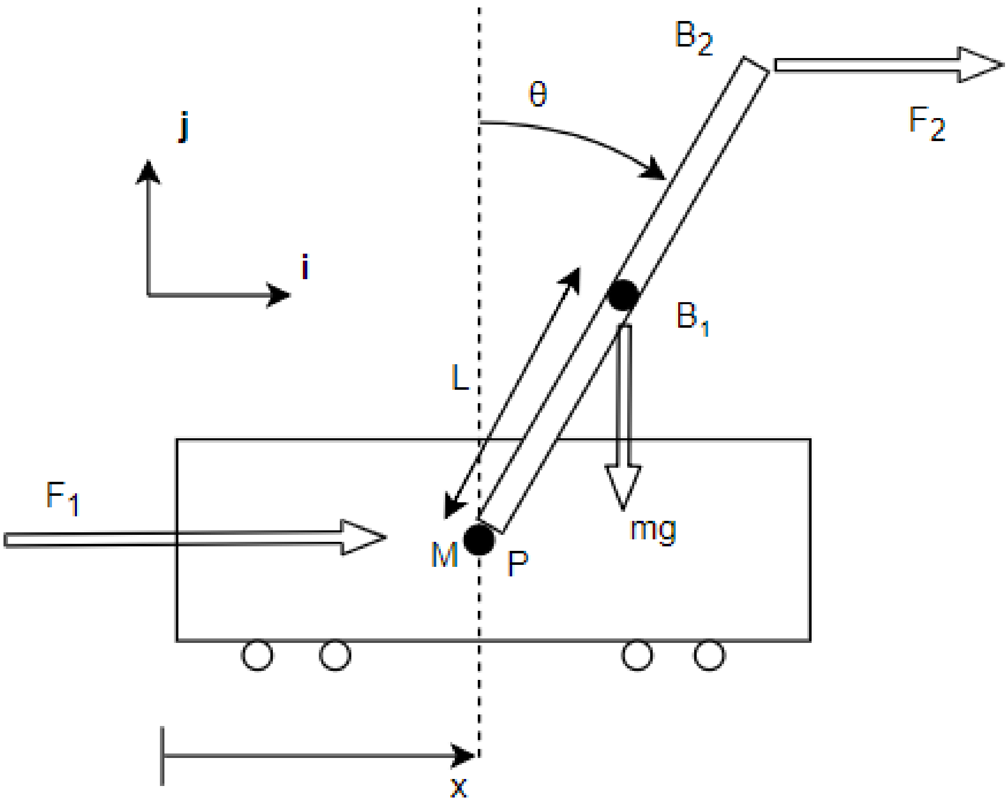

Figure 1.1: A free-body diagram of the inverted pendulum system. For the OpenAI IP the system is in discrete time with a time-step of $\tau = 0.02s$. The other constants are $l = 0.5m$, $m = 0.1kg$, $M = 1kg$, $F = \pm 10N$, $x_{max} = \pm 2.4m$, $\theta_{max} = \pm 12^o$.

OpenAI's gym is a python package that supplies an inverted pendulum environment built-in. This environment was wrapped to use the dynamics above and other extra functionality, whilst providing a rendering function shown in fig. 1.2.

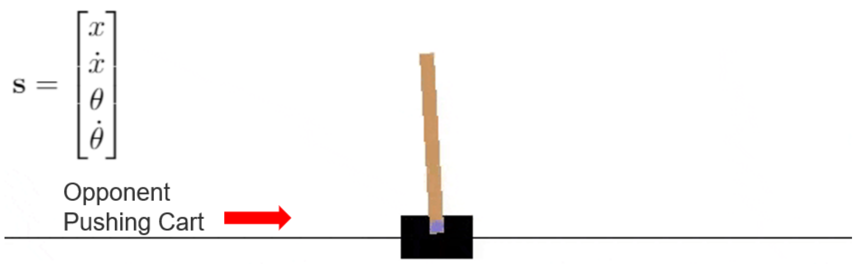

Figure 1.2: The OpenAI gym CartPole environment. The classical state representation is shown in the top left. Actions by the player and the adversary are taken as an impulse to the left or right as defined in fig. 1.1.

1.1.2 Cost and Value Function

For each step/impulse, the 2D state is calculated and a cost, is calculated as:

$$c(x_t, u_t) = -\frac{1}{\sum_i w_i} \boldsymbol{w} \cdot \left[\left(\frac{x_t}{x_{max}}\right)^2, \left(\frac{\dot{x}_t}{\dot{x}_{max}}\right)^2, \left(\frac{\theta_t}{\theta_{max}}\right)^2, \left(\frac{\dot{\theta}_t}{\dot{\theta}_{max}}\right)^2 \right]^T \tag{1.3}$$

Where $\boldsymbol{w}^T = [w_1, w_2, w_3, w_4] = [0.25, 0.1, 0.7, 1]$ and $0 \geq c(x_t, u_t) \geq -1$. The weights, $\boldsymbol{w}$, were chosen through empirical measurement of the the importance of each state ***. Weighting on the inputs was set to zero, as there are only two inputs for this problem, thus the cost can be written as $c(x_t)$. The max values can be approximated experimentally (note, $x_{max} = 2.4$ and $\theta_{max} = 12^o$ are given constraints):

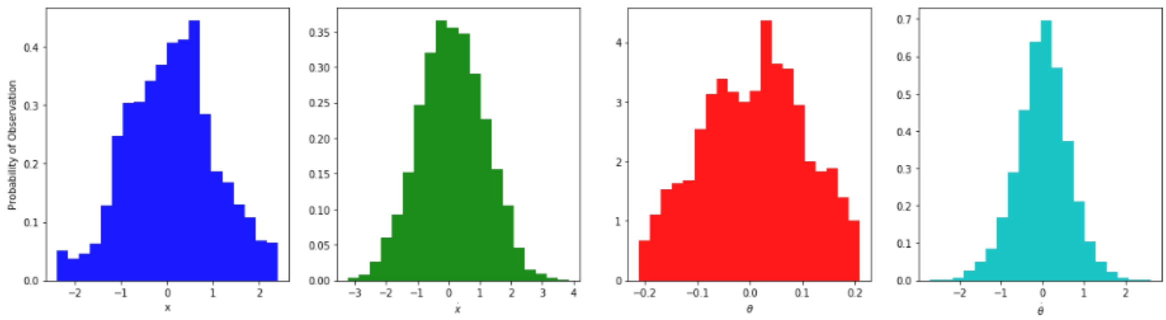

Figure 1.3: Histograms of typical state values. The frequencies greatly depend on the quality of the controller, with better controllers giving much narrower distributions. However, these are typical for a controller of medium efficacy over many episodes where the starting state is randomised (possibly to an uncontrollable state).

Suitable estimates for the the values of $\dot{x}_{max}$ and $\dot{\theta}_{max}$ are thus ≈ 3 and 2 respectively.

The value function is computed after an episode has completed as the discounted future losses at each state with the constraint that $\gamma^k < \frac{1}{20}$, where $\frac{1}{20}$ was chosen as it is a standard factor for insignificance. Since steps_beyonds_done (= k) must be defined in the CartPoleWrapper class, this is a constant, and therefore γ is calculated as $\gamma < \frac{1}{20}^{\frac{1}{k}}$. The discounted values are calculated using a geometric series:

$$v_0 = \frac{\sum_{\tau=0}^{k} \gamma^\tau c(x_\tau)}{\sum_{\tau=0}^{k} \gamma_\tau}, \qquad \text{where } \gamma^k < \frac{1}{20} \tag{1.4}$$

Where for simplicity of notation, $v_0 = v(t)$, the state value at step t.

1.1.3 State Representations

The state can be represented in a number of ways, most simply this would just be feeding $\boldsymbol{x} = [x, \dot{x}, \theta, \dot{\theta}]$ into the neural network. This has a number of advantages such as lower computational cost, greater numerical accuracy (if the process is fully observable) and simpler implementation. Conversely, following Silver et. al, a 2D representation can be used. There are several possibilities for this, all of which first require binning $\boldsymbol{x}$:

(1) A matrix stack of x vs $\dot{x}$ and θ vs $\dot{\theta}$, both of which would only have one non-zero entry. This scales as b^n where b = number of bins and n = number of states.

(2) A matrix stack of x_t vs x_{t-1} for all states. Similarly this scales as b^n, however the derivative states do not need to be plotted as these can be inferred. This has the advantage that, if the derivatives are not observable, we can build them into the 2D representation, however, if they are observable then this is less accurate than (1).

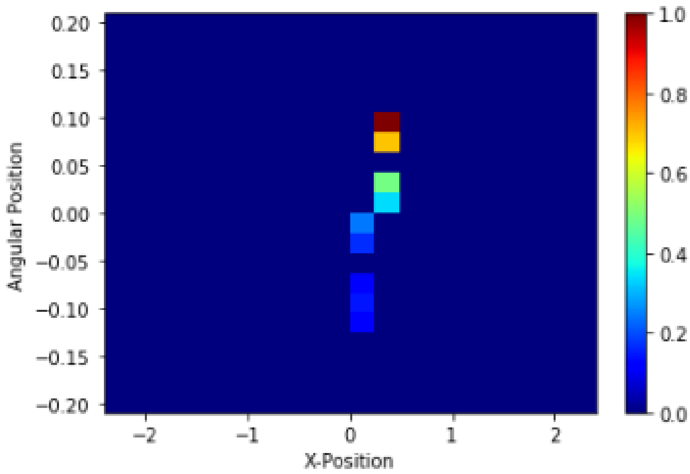

Figure 1.4: An example of a 2D state representation where there are 20 bins and 17 random actions have been taken.

(3) A matrix of discounted previous states forming a motion history image. This is the formulation used, and shown in fig. 1.4. algorithm 1 shows the implementation details.

A 2D representation like this allows us to use a convolutional neural network, which has the benefit of various transformation invariances - these are particularly useful for the inverted pendulum since it is symmetric.

Algorithm 1 Create 2D State

```
function GETSTATE2D(x̂_{t-1}^{(2D)}, binEdges, nBins)
    x̂ ← getNormedState()
    for all x_i ∈ x̂_t do
        x_i ← argmin|binEdges − x_i|          ▷ get the index of the nearest bin edge.
    end for
    HistEdges ← linspace(−0.5, nBins − 0.5, nBins + 1)          ▷ centre by shifting -0.5
    x̂_t^{(2D)} ← histogram2d(x, θ, bins = (histEdges, histEdges))          ▷ Inbuilt function
    x̂_{t-1}^{(2D)}[x̂_{t-1}^{(2D)} < λ^{−(T+1)}] ← 0          ▷ Only keep x̂_{t-1}^{(2D)} from t < T
    return x̂_t^{(2D)} + λx̂_{t-1}^{(2D)}
end function
```

The state space model for the quantisation of the linearised inverted pendulum (valid for small time steps) can be modelled as:

$$\boldsymbol{x}_t^{(2D)} = C\boldsymbol{x}_t + \boldsymbol{V}_t \qquad \boldsymbol{V}_t \sim WN(0, \sigma_v^2 I) \tag{1.5}$$
$$\boldsymbol{x}_t = A\boldsymbol{x}_{t-1} + B\boldsymbol{u}_t \tag{1.6}$$

The initial mean squared error, and the propagation of the error (when estimated optimally) are given by eqs. (1.7) and (1.8) (derivation details can be found in appendix A.2).

$$\sigma_0^2 = \sigma_v^2 = \frac{\delta x^2}{12} \tag{1.7}$$
$$\sigma_{n+1}^2 = A\sigma_n^2\left(1 - \frac{A\sigma_n^2}{A\sigma_n^2 + \sigma_v^2}\right) \tag{1.8}$$

If the spectral density of eq. (1.8) is less than one then the error will decay to zero, and the neural network (if acting as an optimal filter) should be able to recover $\boldsymbol{x}$ without loss. The number of steps needed for this recovery decreases with smaller bin sizes since $\lim_{\delta x \to 0} \sigma_v^2 = 0$, and σ_v^2 decays with δx^2.

Work out how small vs how many steps (possibly with a second order approx?)

With limited memory and a non-zero bin size the overall error can be reduced by binning more densely in regions in which we expect to spend more time. Figure 1.3 shows that the state visit frequencies roughly correspond to normal distribution, therefore by transforming the bins with an inverse gaussian c.f.d. a flattened distribution can be obtained with a greater density of bins in the centre (fig. 1.5). This has the additional benefit of allowing finer control where it is needed. For example, if the pole is far from the equilibrium the optimal action more easily determined, and subject to less change with small state variations.

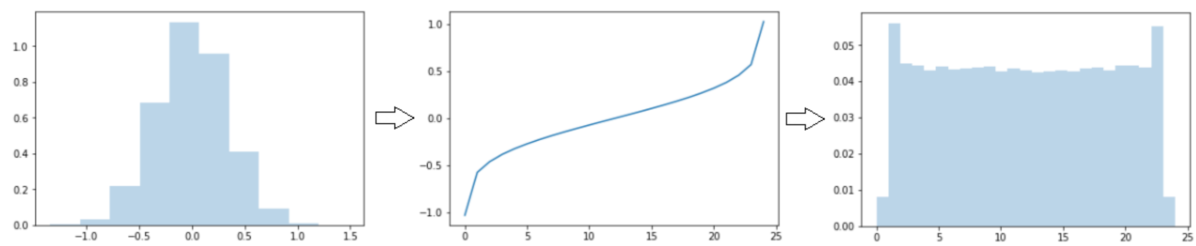

Figure 1.5: Binning of a gaussian distribution with bin edges scaled with an inverse gaussian c.f.d. For this example there are 25 bins.

1.1.4 Discrete vs Continuous Time

For a small enough discrete time step, the simulation will approximate the continuous process very well. Additionally, we can achieve pseudo-continuous actions within the constraint of $u \in \{-1, 1\}$ via pulse-width modulation. A time step of 0.02s is 35x smaller than the time constant of the linearised system ($\tau \approx 0.70s$). The positional change per time step is therefore $\sim \frac{1}{35}\% = 3\%$. Alternatively the average maximum velocities are $3m/s$ and $2rad/s$, thus the maximum positional change per time step is $\dot{x}_{i,max} \times \frac{\tau}{x_{i,max}} =$ 2.5% and 30% respectively. In practice a 30% change in θ would only occur around θ_{max} and would be uncontrollable, therefore we can take the change to be far less than this . Therefore, we can model the quick succession of actions as a continuous one:

$$F(t) = g(t) * \int_0^\infty u(t)dt \approx g(t) \sum_{n=0}^{\infty} I\tau\delta(t - n\tau) \tag{1.9}$$

can we? Also proof of uncontrollabiltiy?

work out expression for force? Convolution? Just avg with decay term? Ask vinnicombe

1.2 Self Play and Adversaries

1.2.1 Cost and Value Functions of the Adversary

In AlphaZero, the adversary is working to minimise the value function and the player is working to maximise it. For board games where players are on equal footing a value function representing the expected outcome, $-1 < v < 1$ can be used. This has the advantage of symmetry - when the board is viewed from the other player's perspective the value can be multiplied by -1, and we get the expected outcome for that player.

The adversary for the inverted pendulum has the additional advantage of gravity, making the play un-symmetric. Given that both the state-value and cost are $-1 < v, c < 0$, multiplying by -1 would mean the adversary is maximising a value function $0 < v < 1$. State-values outside these ranges have no meaning. If a single neural network is used, values close to equilibrium may be predicted into the wrong range. Consequently, both the adversary and the player must predict true state-values. This also has the advantage of maintaining an intuitive meaning of the state-value.

1.2.2 Choice of Action

The inverted pendulum system is symmetric in both x and θ allowing for the possibility of taking advantage of this, however as shown with AlphaZero [13], this provides minimal benefit and also hinders generalisation. The adversarial point of action was chosen to be at the top of the pole, acting horizontally (fig. 1.1). Thus ensuring that two distinct policies must be learnt, rather than one being just inverse probabilities of the other. For simplicity $u_{adv} \in \{-1, 1\}$ was chosen.

There are two good ways of representing when it is the adversaries turn for the neural network:

Multiply the board representation by -1 such that opponent pieces are negative. This can only take two players and; A network that outputs both state-values and action *p.m.f.*'s should predict the same state values for both player and adversary, but predict vastly different actions. Therefore, the values will be decoupled from the actions, which was one of the major benefits of using a single neural network. However, a single network is simpler, and this more closely follows AlphaZero's methodology.

Record the player number with each example and use player-labeled examples to train different neural networks. Using a neural network for each player causes half of the examples to be lost as only the relevant player's examples are used to train each network. However, this doesn't suffer from the problems above and can cope with agents with a different number of possible actions more easily.

*** Currently method 2 is being used, but subject to change ***

Note, in the case of the inverted pendulum, the optimal action is the inverse of the worst action. However, this is not a general result, for example, in a system with non-linear and asymmetric dynamics it is possible to have the target perpendicular to the limit of stability, thus for the adversary it is better to push the system into instability rather than away from equilibrium.

1.2.3 Episode Execution

Pseudocode for episode execution following the sections above is shown in algorithm 2.

1.3 Neural Network

1.3.1 Loss Functions and Pareto

do we use mse or not? how do we balance action and value loss if not symmetric? Alpha zero sets them equal, can we use the same principals here?

1.3.2 Architectures

Player vs Adversary Architectures? Combined? GPU, computing power and complexity

1.4 MCTS

outline + pseudocode

Algorithm 2 Execute Episode

function EXECUTEEPISODE
  $example \leftarrow []$
  $\boldsymbol{x}, \boldsymbol{x}^{(2D)}, c \leftarrow resetEpisode()$ ▷ Set initial $\boldsymbol{x}$ randomly and initialise the cost
  $player \leftarrow 0$
  repeat
    $\boldsymbol{\pi} \leftarrow getActionProb(\boldsymbol{x}, \boldsymbol{x}^{(2D)}, player)$ ▷ Perform MCTS Simulations
    $example.append((\boldsymbol{x}^{(2D)}, \boldsymbol{\pi}, c, player))$
    $u \sim \boldsymbol{\pi}$ ▷ Sample action
    $\boldsymbol{x}, c \leftarrow step(\boldsymbol{x}, u)$ ▷ Take next true episode step
    $\boldsymbol{x}^{(2D)} \leftarrow getState2D(\boldsymbol{x}, \boldsymbol{x}^{(2D)})$
    $player \leftarrow nextPlayer(player)$
  until $episodeEnded(\boldsymbol{x})$
  $example \leftarrow convertCostsToValues(example)$
  return $example$
end function

Algorithm 3 MCTS

function GETACTIONPROB$(boldsymbolx, \boldsymbol{x}^{(2D)}, player)$
end function

function SEARCH
end function

1.4.1 State and Player Representation

1.4.2 Terminal States and Suicide***

1.4.3 Modified UCB

1.5 Player and Adversary Evaluation

1.5.1 Elo Scoring

A Appendices

A.1 Inverted Pendulum Dynamics Derivation

We can find the state space equations for the Inverted Pendulum using d'Alembert forces. Firstly we define the distance and velocity vectors to the important points:

$$
\begin{aligned}
\boldsymbol{r}_P &= x\boldsymbol{i} \\
\boldsymbol{r}_{B_1/P} &= Lsin\theta\boldsymbol{i} + Lcos\theta\boldsymbol{j} \\
\boldsymbol{r}_{B_1} &= (x + Lcos\theta)\boldsymbol{i} + L\dot{\theta}sin\theta\boldsymbol{j} \\
\dot{\boldsymbol{r}}_{B_1} &= (\dot{x} + L\dot{\theta}cos\theta)\boldsymbol{i} - L\dot{\theta}sin\theta\boldsymbol{j}
\end{aligned}
$$

Linear Momentum, $\boldsymbol{\rho} = \sum_i m_i\dot{\boldsymbol{r}}_{i/o} = m\dot{\boldsymbol{r}}_{B_1} + M\dot{\boldsymbol{r}}_P$:

$$
\boldsymbol{\rho} = \begin{bmatrix} (M+m)\dot{x} + ml\dot{\theta}cos\theta \\ -ml\dot{\theta}sin\theta \\ 0 \end{bmatrix}
$$

Moment of momentum about P, $\boldsymbol{h}_P = \boldsymbol{r}_{B_1/P} \times m\dot{\boldsymbol{r}}_{B_1}$:

$$
\begin{aligned}
\boldsymbol{h}_P &= -mL(L\dot{\theta} + \dot{x}cos\theta)\boldsymbol{k} \\
\therefore \dot{\boldsymbol{h}}_P &= -mL(L\ddot{\theta} + \ddot{x}cos\theta - \dot{x}\dot{\theta}sin\theta)\boldsymbol{k}
\end{aligned}
$$

We can balance moments using $\dot{\boldsymbol{h}}_P + \dot{\boldsymbol{r}}_P \times \boldsymbol{\rho} = \boldsymbol{Q}_e$ and $\boldsymbol{Q}_e = \boldsymbol{r}_{B_1/P} \times -mg\boldsymbol{j} + \boldsymbol{r}_{B_2/P} \times F_2\boldsymbol{i}$:

$$
\dot{\boldsymbol{h}}_P + \dot{\boldsymbol{r}}_P \times \boldsymbol{\rho} = \begin{bmatrix} 0 \\ 0 \\ -mL(\ddot{x}cos\theta + L\ddot{\theta}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -L(mgsin\theta + 2F_2cos\theta) \end{bmatrix} = \boldsymbol{Q}_e
$$

And also balance linear momentum using $\boldsymbol{F}_e = \dot{\boldsymbol{\rho}}$:

$$
\dot{\boldsymbol{\rho}} = \begin{bmatrix} (m+M)\ddot{x} + mL(\ddot{\theta}cos\theta - \dot{\theta}^2sin\theta) \\ -mL(\ddot{\theta}sin\theta + \dot{\theta}^2cos\theta) \\ 0 \end{bmatrix} = \begin{bmatrix} F_1 + F_2 \\ R - mg \\ 0 \end{bmatrix} = \boldsymbol{F}_e
$$

Finally we can write the system dynamics in terms of $\ddot{\theta}$ and $\ddot{x}$:

$$\ddot{\theta}\big(M + msin^2\theta\big)L = \left(\frac{2M-m}{m}F_2 - F_1\right)cos\theta + g(M+m)sin\theta - mL\dot{\theta}^2 sin\theta cos\theta \tag{A.1}$$

$$\ddot{x}(M + msin^2\theta) = F_1 + F_2cos(2\theta) + msin\theta(L\dot{\theta}^2 - gcos\theta) \tag{A.2}$$

Simplifying this for our problem by substituting in constants, we can write the full state space equation:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{\left(\frac{2M-m}{m}F_2 - F_1\right)cos\theta + g(M+m)sin\theta - mL\dot{\theta}^2 sin\theta cos\theta}{(M+msin^2\theta)} \\ \dot{\theta} \\ \frac{F_1 + F_2cos(2\theta) + msin\theta(L\dot{\theta}^2 - gcos\theta)}{L(M+msin^2\theta)} \end{bmatrix} \tag{A.3}$$

It can be proved that the inverted pendulum system is controllable by showing:

$$rank[\boldsymbol{B}\ \boldsymbol{AB}\ \boldsymbol{A^2B}\ \boldsymbol{A^3B}] = 4 \tag{A.4}$$

Therefore for any initial condition we can reach $\boldsymbol{x}_e$ in finite time under these linear assumptions. However, for $\theta \approx 0$ we need a more sophisticated model.

A.1.1 Swing Up Control

One way to get the cart to swing the pendulum up to the linear-range is to find a homoclinic orbit (a trajectory that passes though an unstable fixed point). I.e. we must find a controller that that drives the pendulum to the unstable equilibrium. This can be done using energy shaping, and in the context of the inverted pendulum, this constitutes applying force to maximise the potential energy and minimise kinetic. Once in the linear region we then switch to an LQR controller to complete the task.

A.2 Propagation of Quantisation Error

The state space model for the quantisation of the linearised inverted pendulum can be written as:

$$\boldsymbol{x}_t^{(2D)} = C\boldsymbol{x}_t + \boldsymbol{V}_t \qquad \boldsymbol{V}_t \sim WN(0, \sigma_v^2 I) \tag{A.5}$$

$$\boldsymbol{x}_t = A\boldsymbol{x}_{t-1} + B\boldsymbol{u}_t \tag{A.6}$$

Where A and B are the linearised system dynamics (valid for small time steps), and C is the linear transformation to a 2D state space, with quantisation noise $\mathbf{V}$.

*** This derivation here is for 1D and also follows the derivation from 4F7, which does not take into account actions (B=0), and has additional noise on the second equation.

However, because actions are deterministic, it should have no effect on the propagated MSE. Could fairly easily do a full derivation for this, but is it that necessary? Also σ_v^2 is not true, as this is only a 1D error. Finally, possible spanner... Limiting the memory of the system to only t-k could place limits on the accuracy? Should do full derivation if time allows. Note, is this even the correct way to go about this? Can we assume a neural network will act as a kalman filter?***

Assuming the quantisation bin size, δx, is small, the quantisation noise can be modelled as uniform random variable with a mean squared error:

$$\sigma_v^2 = \mathbb{E}[V_n^2] = \int_{-\delta x/2}^{\delta x/2} u^2 \cdot \frac{1}{\delta x} du = \frac{\delta x^2}{12} \tag{A.7}$$

Kalman filtering can be used to find an optimal estimate $\hat{X}_n = K[X_n|Y_{n-k:n}]$ using the algorithm (derived in [3]).

Algorithm 4 Kalman Filtering

1: **Given:** $\hat{X}_n = K[X_n|Y_{n-k:n}]$ and $\sigma^2 = \mathbb{E}[(X_n - \hat{X}_n)^2]$

Prediction:

2: $\bar{X}_{n+1} = K[X_n|Y_{n-k:n}] = f_n\hat{X}_n$

3: $\bar{\sigma}_{n+1}^2 = \mathbb{E}[(X_{n+1} - \bar{X}_{n+1})^2] = f_n\sigma_n^2 + \sigma_w^2$ $\triangleright\ \sigma_w^2 = 0$

Update:

4: $I_{n+1} = Y_{n+1} - g_{n+1}\bar{X}_{n+1}$

5: $\hat{X}_{n+1} = \bar{X}_{n+1} + \frac{g_{n+1}\bar{\sigma}_{n+1}^2}{g_{n+1}^2\bar{\sigma}_{n+1}^2 + \sigma_v^2} I_{n+1}$ $\triangleright\ \sigma_v^2 = \frac{\delta x^2}{12}$

6: $\sigma_{n+1}^2 = \bar{\sigma}_{n+1}^2\left(1 - \frac{g_{n+1}^2\bar{\sigma}_{n+1}^2}{g_{n+1}^2\bar{\sigma}_{n+1}^2 + \sigma_v^2}\right)$

Thus we can find the optimal mean squared error of the next value from algorithm 4 line 6:

$$\sigma_{n+1}^2 = f_n\sigma_n^2\left(1 - \frac{f_n\sigma_n^2}{f_n\sigma_n^2 + \sigma_v^2}\right) \tag{A.8}$$

B References

[1] M.C. Smith, I Lestas, *4F2: Robust and Non-Linear Control* Cambridge University Engineering Department, 2019

[2] G. Vinnicombe, K. Glover, F. Forni, *4F3: Optimal and Predictive Control* Cambridge University Engineering Department, 2019

[3] S. Singh, *4F7: Statistical Signal Analysis* Cambridge University Engineering Department, 2019

[4] Arthur E. Bryson Jr, *Optimal Control - 1950 to 1985.* IEEE Control Systems, 0272-1708/95 pg.26-33, 1996.

[5] I. Michael Ross, Ronald J. Proulx, and Mark Karpenko, *Unscented Optimal Control for Space Flight.* ISSFD S12-5, 2014.

[6] Zheng Jie Wang, Shijun Guo, Wei Li, *Modeling,Simulation and Optimal Control for an Aircraft of Aileron-less Folding Wing* WSEAS TRANSACTIONS on SYSTEMS and CONTROL, ISSN: 1991-8763, 10:3, 2008

[7] Giovanni Binet, Rainer Krenn and Alberto Bemporad, *Model Predictive Control Applications for Planetary Rovers.* imtlucca, 2012.

[8] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction (2nd Edition).* The MIT Press, Cambridge, Massachusetts, London, England

[9] I. Carlucho, M. De Paula, S. Villar, G. Acosta . *Incremental Q-learning strategy for adaptive PID control of mobile robots.* Expert Systems with Applications. 80. 10.1016, 2017

[10] Yuxi Li, *Deep Reinforcement Learning: An Overview.* CoRR, abs/1810.06339, 2018.

[11] Sandy H. Huang, Martina Zambelli, Jackie Kay, Murilo F. Martins, Yuval Tassa, Patrick M. Pilarski, Raia Hadsell, *Learning Gentle Object Manipulation with Curiosity-Driven Deep Reinforcement Learning.* arXiv 2019.

[12] David Silver, Julian Schrittwieser, Karen Simonyan et al, *Mastering the game of Go without human knowledge.* Nature, vol. 550, pg.354–359, 2017.

[13] David Silver, Thomas Hubert, Julian Schrittwieser et al, *A general reinforcement learning algorithm that masters chess, shogi and Go through self-pla.* Science 362:6419, pg.1140-1144, 2018.