
0 Contents

1	Theory and Methods	2
1.1	The Inverted Pendulum (IP)	2
1.1.1	Dynamics	2
1.1.2	Cost and Value Function	3
1.1.3	State Representations	4
1.1.4	Discrete vs Continuous Time	6
1.2	Self Play and Adversaries	7
1.2.1	Cost and Value Functions of the Adversary	7
1.2.2	Choice of Action	7
1.2.3	Episode Execution	8
1.3	Neural Network	8
1.3.1	Loss Functions and Pareto	8
1.3.2	Architectures	9
1.4	MCTS	10
1.4.1	Tree Nodes and Simulations	10
1.4.2	Action Selection	10
1.5	Player and Adversary Evaluation	11
A	Appendices	13
A.1	Inverted Pendulum Dynamics Derivation	13
A.2	Propagation of Quantisation Error	15
	References	16

1 Theory and Methods

1.1 The Inverted Pendulum (IP)

1.1.1 Dynamics

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

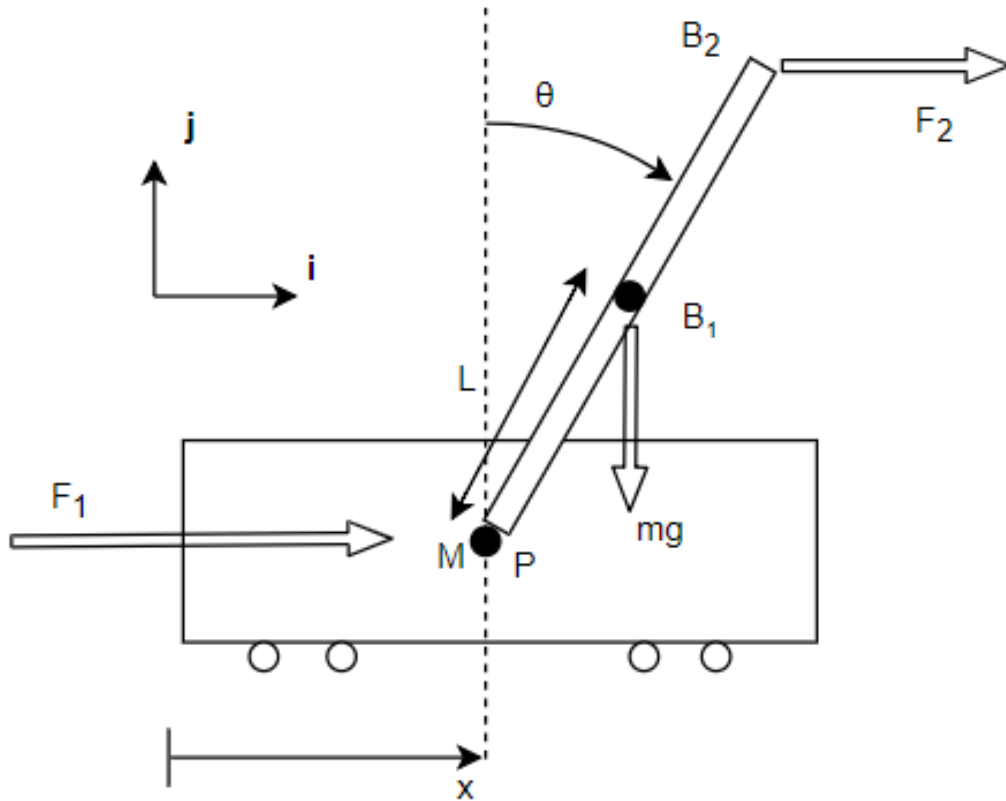


Figure 1.1: A free-body diagram of the inverted pendulum system. For the OpenAI IP the system is in discrete time with a time-step of $\tau = 0.02s$. The other constants are $l = 0.5m$, $m = 0.1kg$, $M = 1kg$, $F = \pm 10N$, $x_{max} = \pm 2.4m$, $\theta_{max} = \pm 12^\circ$.

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

by:

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

Using Lyapunov's indirect method, we can write the linearised equations about the equilibrium, $\mathbf{x}_e = [x_e, \dot{x}_e, \theta_e, \dot{\theta}_e]^T = [0, 0, 0, 0]^T$, as:

$$\begin{bmatrix} \delta\dot{x} \\ \delta\ddot{x} \\ \delta\dot{\theta} \\ \delta\ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(m+M)}{ML}g & 0 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta\dot{x} \\ \delta\theta \\ \delta\dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{1}{M} & -\frac{1}{M} \\ 0 & 0 \\ -\frac{1}{ML} & \frac{2M+m}{mML} \end{bmatrix} \begin{bmatrix} \delta F_1 \\ \delta F_2 \end{bmatrix} \quad (1.2)$$

The eigenvalues are given by $\det(\lambda I - A) = \lambda^2(\lambda^2 - \frac{(m+M)}{ML}g) = 0$. Therefore, the system is unstable about \mathbf{x}_e due to the right half plane pole, $\lambda = \sqrt{\frac{(m+M)}{ML}g}$. Additionally, the time constant of this unstable system is $\tau = \sqrt{\frac{ML}{g(m+M)}} = 0.22s$. Note, if $M \gg m$, $\tau \rightarrow \sqrt{\frac{L}{g}}$, which is the time constant for a simple pendulum.

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

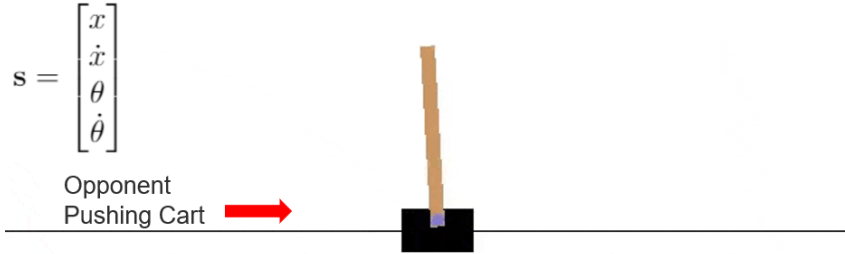


Figure 1.2: The OpenAI gym CartPole environment. The classical state representation is shown in the top left. Actions by the player and the adversary are taken as an impulse to the left or right as defined in fig. 1.1.

1.1.2 Cost and Value Function

For each step/impulse, the 2D state is calculated and a cost, is calculated as in eq. (1.3), where $\mathbf{w}^T = [w_1, w_2, w_3, w_4] = [0.25, 0.1, 0.7, 1]$ and $0 \geq c(x_t, u_t) \geq -1$. The weights, \mathbf{w} , were chosen through empirical measurement of the the importance of each state ***.

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

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 $\theta_{max} = 12^\circ$ are given constraints):

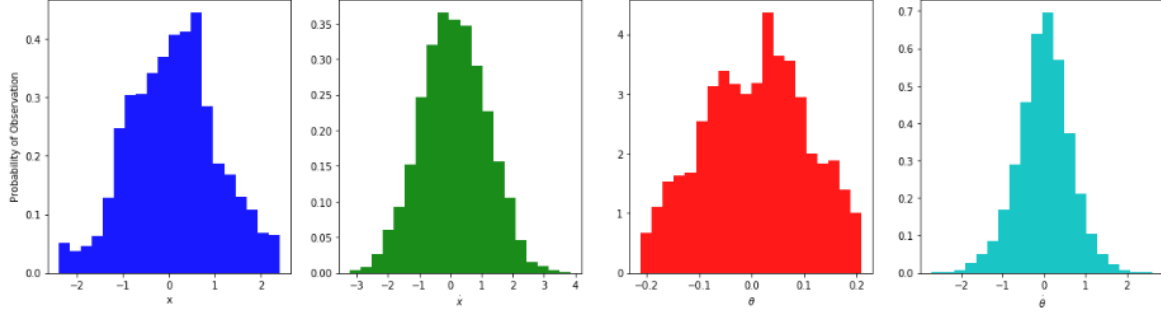


Figure 1.3: Histograms of typical state values. The frequencies greatly depend on the quality of the controller, with better controllers giving much narrower distributions. However, these are typical for a controller of medium efficacy over many episodes where the starting state is randomised (possibly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

Suitable estimates for the the values of \dot{x}_{max} and $\dot{\theta}_{max}$ are thus approximately 3 and 2 respectively. The value function is computed after the episode has completed, and is the discounted future losses at each state. A horizon constraint of $\gamma^k < \frac{1}{20}$ was chosen as it is a standard factor for insignificance. Since the steps taken after a terminal state has been reached, k , must be defined with the game dynamics (the `CartPoleWrapper` class), it is a constant. Therefore γ is calculated as $\gamma < \frac{1}{20^{\frac{1}{k}}}$. The discounted future values are calculated using a geometric series:

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

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. The simplest method would be $\mathbf{x} = [x, \dot{x}, \theta, \dot{\theta}]$. 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 [14], a 2-dimensional (2D) representation may be used. There are several possibilities for this, all of which first require binning \mathbf{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, they are built into the 2D representation, however, if they are observable then this is less accurate than (1).

(3) A matrix of discounted previous states, forming a motion history image. An example of this is shown in section 1.1.3, and Algorithm 1 shows the implementation details. This was the chosen representation.

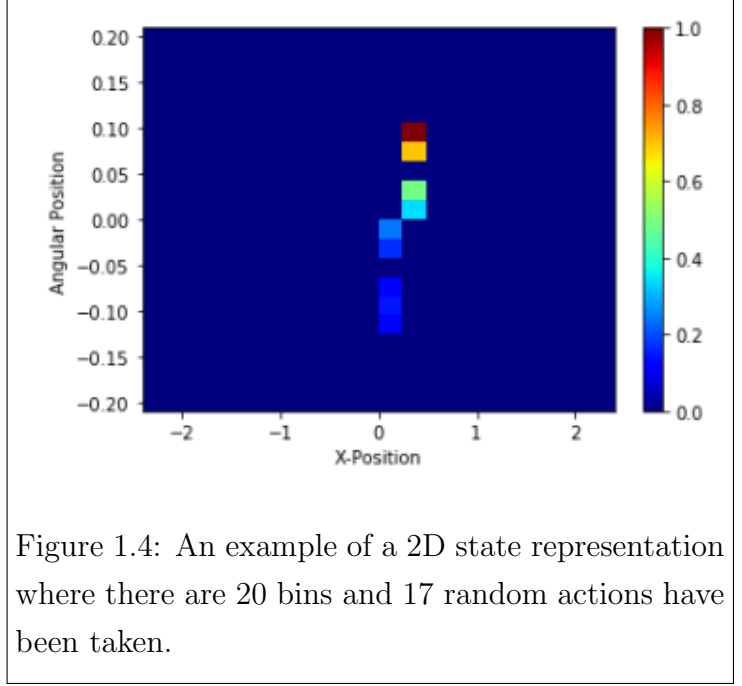


Figure 1.4: An example of a 2D state representation where there are 20 bins and 17 random actions have been taken.

A 2D representation such as 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.

The state space model for the quantisation of the linearised inverted pendulum (valid for small time steps) can be modelled as (where \mathcal{U} is a uniform distribution, centred on 0):

The state space model for the quantisation of the linearised inverted pendulum can

Algorithm 1 Create 2D State

```

1: function GETSTATE2D( $\hat{\mathbf{x}}_{t-1}^{(2D)}$ , binEdges, nBins)
2:    $\hat{\mathbf{x}} \leftarrow getNormedState()$ 
3:   for all  $x_i \in \hat{\mathbf{x}}_t$  do
4:      $x_i \leftarrow argmin|binEdges - x_i|$  ▷ get the index of the nearest bin edge.
5:   end for
6:    $HistEdges \leftarrow linspace(-0.5, nBins - 0.5, nBins + 1)$  ▷ centre by shifting -0.5
7:    $\hat{\mathbf{x}}_t^{(2D)} \leftarrow histogram2d(x, \theta, bins = (histEdges, histEdges))$  ▷ Inbuilt function
8:    $\hat{\mathbf{x}}_{t-1}^{(2D)}[\hat{\mathbf{x}}_{t-1}^{(2D)} < \lambda^{-(T+1)}] \leftarrow 0$  ▷ Only keep  $\hat{\mathbf{x}}_{t-1}^{(2D)}$  from  $t < T$ 
9:   return  $\hat{\mathbf{x}}_t^{(2D)} + \lambda \hat{\mathbf{x}}_{t-1}^{(2D)}$ 
10: end function

```

be written as:

$$\mathbf{x}_t^{(2D)} = C\mathbf{x}_t + \mathbf{V}_t \quad \mathbf{V}_t \sim \mathcal{U} \left(\begin{bmatrix} \frac{1}{\delta x} \\ \frac{1}{\delta \theta} \end{bmatrix} \right) \quad (1.5)$$

$$\mathbf{x}_t = A\mathbf{x}_{t-1} + B\mathbf{u}_t \quad (1.6)$$

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} .

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

$$\Sigma_0 = \sigma_v^2 I = \begin{bmatrix} \frac{\delta x^2}{12} & 0 \\ 0 & \frac{\delta \theta^2}{12} \end{bmatrix} \quad (1.7)$$

$$\Sigma_{n+1} = (I - \bar{\Sigma}_{n+1} C^T (\sigma_v^2 I + C \bar{\Sigma}_{n+1} C^T)^{-1} C) \bar{\Sigma}_{n+1} \quad (1.8)$$

Where $\bar{\Sigma}_{n+1} = A \Sigma_n A^T$. When A is the linearised inverted pendulum dynamics and C is an arbitrary matrix to convert \mathbf{x}_t into a 2D state, it can be shown (see appendix A.2) that as $t \rightarrow \infty$, the covariance of quantisation decays to zero and therefore this form for a 2D state becomes lossless (assuming that the neural network acts as an optimal kalman filter). The number of steps needed for this recovery decreases with smaller bin sizes since $\lim_{\delta x \rightarrow 0} \sigma_v^2 = 0$, and σ_v^2 decays with δx^2 .

With limited memory and a non-zero bin size the overall error can be reduced by binning more densely in regions in which the IP is expected 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.

1.1.4 Discrete vs Continuous Time

For a small enough discrete time step, the simulation will approximate the continuous process very well. Additionally pseudo-continuous actions can be achieved 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 $\dot{x}_{i,max} \times \frac{\tau}{x_{i,max}} = 2.5\%$

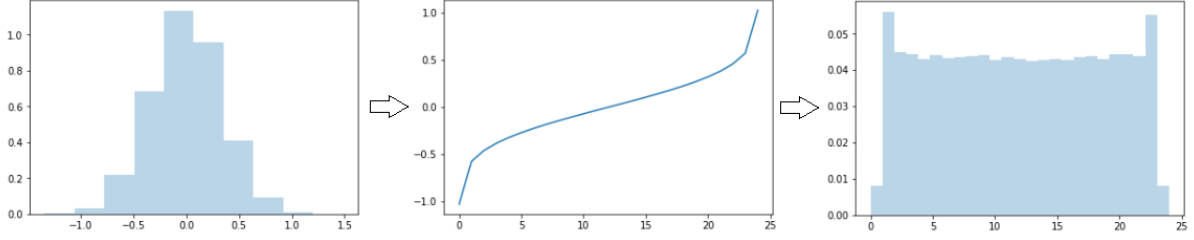


Figure 1.5: Binning of a gaussian distribution with bin edges scaled with an inverse gaussian c.f.d. For this example there are 25 bins.

and 30% respectively. In practice a 30% change in θ would only occur around θ_{max} and would be uncontrollable, therefore the change can be assumed to be less than this. Therefore, the quick succession of actions can be modelled as a continuous one:

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

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 whereas the player is working to maximise it. For board games, where agents are on equal footing, a value function ($-1 < v < 1$) representing the expected outcome can be used. This has the advantage of symmetry - when the board is viewed from the other agent's perspective the value can be multiplied by -1, giving the expected outcome for that agent.

The adversary for the inverted pendulum has the additional advantage of gravity, making the play asymmetric. 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 . By taking advantage of this the number of training examples could be doubled by reflecting episodes along the axis of symmetry. However, as shown with AlphaZero [14], this provides minimal benefit

can
it?
Also
proof
of un-
con-
trolla-
bilty?

work
out
ex-
pres-
sion
for
force?
Con-
volu-
tion?
Just
avg
with
decay
term?
Ask
vinni-
combe

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 whether it is the adversary’s turn for the neural network:

Multiply the board representation by -1 such that opponent pieces are negative.

This has the disadvantages that it can only take two agents 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 could 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 negating the board more closely follows AlphaZero’s methodology.

Record the agent with each example and use agent-labeled examples to train different neural networks. Using a neural network for each agent causes half of the examples to be lost as only the relevant agent’s examples are used to train each network. However, this does not 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 closest edge of the invariant set, 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

The following loss function is minimised when training the neural networks:

$$\mathcal{L} = \sum_t (v_\theta(\mathbf{x}_t^{(2D)}) - v_t)^2 + c_{pareto} \cdot \|\mathbf{p}_\theta(\mathbf{x}_t^{(2D)}) - \boldsymbol{\pi}_t\|_2^2 \quad (1.10)$$

Algorithm 2 Execute Episode

```
1: function EXECUTEEPISODE
2:    $example \leftarrow []$ 
3:    $\mathbf{x}, \mathbf{x}^{(2D)}, c \leftarrow resetEpisode()$   $\triangleright$  Set initial  $\mathbf{x}$  randomly and initialise the cost
4:    $agent \leftarrow 0$ 
5:   repeat
6:      $\boldsymbol{\pi} \leftarrow getActionProb(\mathbf{x}, \mathbf{x}^{(2D)}, agent)$   $\triangleright$  Perform MCTS Simulations
7:      $example.append((\mathbf{x}^{(2D)}, \boldsymbol{\pi}, c, agent))$ 
8:      $u \sim \boldsymbol{\pi}$   $\triangleright$  Sample action
9:      $\mathbf{x}, c \leftarrow step(\mathbf{x}, u)$   $\triangleright$  Take next true episode step
10:     $\mathbf{x}^{(2D)} \leftarrow getState2D(\mathbf{x}, \mathbf{x}^{(2D)})$ 
11:     $agent \leftarrow nextAgent(agent)$ 
12:  until  $episodeEnded(\mathbf{x})$ 
13:   $example \leftarrow convertCostsToValues(example)$ 
14:  return  $example$ 
15: end function
```

Where c_{pareto} is a constant. During training the agents move probabilistically throughout, rather than switching to acting greedily after 30 moves as in AlphaZero [14]. As such, mean squared error was used for the action-losses.

is this actually a good idea? Why not use greedy?

Due to the asymmetric nature of the problem, the value and action-losses can be of very different magnitudes. A constant factor, c_{pareto} , was used to prevent this.

1.3.2 Architectures

Two neural network architectures are used, one that takes the raw state, $\mathbf{x} = [x \dot{x} \theta \dot{\theta}]^T$ as input, and another that takes the 2D state, $\mathbf{x}^{(2D)}$. The adversary and player have separate networks, both predicting the state-value and action $p.m.fs$.

For the network with $\mathbf{x}^{(2D)}$ as input, there are 2 fully connected convolutional layers with max-pooling, followed by 2 feedforward layers with *ReLU* activations. The two “heads” split here and have 2 fully-connected layers each, with the value head outputting v_θ and the action head outputting \mathbf{p}_θ . The network with \mathbf{x} as input is the same, except the convolutional layers are replaced with 2 feedforward layers.

Both architectures perform training with the *Adam Optimiser*, a mini-batch size of 8, a dropout of 0.3 and batch normalisation. The networks are implemented in pytorch¹.

The neural network is evaluated once every step in the MCTS, therefore in an episode of length L and S MCTS simulations/step, the neural network is evaluated more than

¹www.pytorch.org

$L \times S$ times. The GPU used to run experiments is a single NVIDIA GeForce GTX 1050. The convolutional layers of a neural network are the most computationally expensive part and, increasing with the image size, are more than 50% of the computational time in the architectures defined above. Therefore, there is a trade off between the number of convolutional layers used and computation time. The number of layers was chosen such that running the program takes less than 12hrs.

1.4 MCTS

The general MCTS algorithm, outlined in ??, was implemented for the inverted pendulum as in Algorithm 3.

1.4.1 Tree Nodes and Simulations

The state for the inverted pendulum is continuous ($\mathbf{x} \in \mathbb{R}^4$), as opposed to the discrete nature of board games. In order to be able to compare nodes, the states are multiplied by 10^{12} and converted to integers. Each MCTS simulation ends with a leaf node being expanded, therefore after S simulations from a node, S states will be visited. Over the whole episode, $S \times L$ distinct states will be visited in total. For an episode length of 200, the probability of revisiting a state again from a different trajectory is $(S \times L)10^{-124}$: impossibly small.

Binning in this manner means that S is limited only in computing power. Having $S > 50$ leads to computing times of days on the NVIDIA GeForce GTX 1050 GPU used for this project ($S = 1600$ for AlphaZero).

1.4.2 Action Selection

Action selection moving down the tree was modified to reflect the asymmetry of the state-value:

$$u_{player}^* = \underset{u \text{ in } \mathcal{U}_{player}}{\operatorname{argmax}} \left\{ Q(\mathbf{x}, u) + c \cdot \mathbf{p}_\theta(\mathbf{x}, u) \frac{\sqrt{\sum_b N(\mathbf{x}, b)}}{1 + N(\mathbf{x}, b)} \right\} = \underset{u \text{ in } \mathcal{U}_{player}}{\operatorname{argmax}} U_{player}(\mathbf{x}, u) \quad (1.11)$$

$$u_{adv}^* = \underset{u \text{ in } \mathcal{U}_{adv}}{\operatorname{argmax}} \left\{ -Q(\mathbf{x}, u) + c \cdot \mathbf{p}_\theta(\mathbf{x}, u) \frac{\sqrt{\sum_b N(\mathbf{x}, b)}}{1 + N(\mathbf{x}, b)} \right\} = \underset{u \text{ in } \mathcal{U}_{adv}}{\operatorname{argmax}} U_{adv}(\mathbf{x}, u) \quad (1.12)$$

Where $c = 1$ was used. Negating Q in \mathcal{U}_{adv} causes the adversary to minimise the state-value whilst still maximising the upper confidence bound, thus ensuring exploration.

When the inverted pendulum exceeds a constraint the tree search should return -1, and at that state the neural network should also return -1. Additionally, when the search

reaches 200 steps (the arbitrarily set “end” of the episode), the neural network will not be aware of this fact. Therefore, when any terminal state is reached, the neural network is evaluated and v is returned.

1.5 Player and Adversary Evaluation

For problems in which one agent has a distinct advantage, determining how “good” their policy is compared to other policies becomes difficult. For example, if the pendulum stays up for longer, is it difficult to determine whether the player has improved because it is controlling the system better, or the adversary is less effective, or both have improved but one improved more. Handicapping the adversary such that it is equal to the player is not practical due to the non-linear relationships between external forces, and gravity, on the dynamics. Furthermore, the inverted pendulum must stay up for at least some amount of time, otherwise the network would not have the training examples to learn from. Therefore handicapping the adversary with $F_2 = \alpha F_1$, and limiting the adversary to push every K steps were both used ($\alpha \sim 0.05$ and $K \sim 20$ initially). The score used is the number of steps the agent stayed up for divided by the max number of steps. For the player this ranges between 0 and 1, and for the adversary $score = (200 - steps)/200$ is used.

Over one policy iteration, there are 4 different policies: the two current player and adversary policies, $f_{\theta, curr}^{(player)}$ and $f_{\theta, curr}^{(adv)}$, and the challenger policies, $f_{\theta, chal}^{(player)}$ and $f_{\theta, chal}^{(adv)}$. The improvement of the player can be determined by pitting the current player against the current adversary, and then the challenging player against the current adversary; and vice-versa for the adversary improvement². This gives the four equations in ??.

does
this
make
sense?

$$\mathbb{E}[s_{curr}^p] = \frac{1}{1 + 10^{\wedge \left(\frac{\mu_{curr}^a - \mu_{curr}^p}{400} \right)}} \quad \mathbb{E}[s_{curr}^a] = \frac{1}{1 + 10^{\wedge \left(\frac{\mu_{curr}^p - \mu_{curr}^a}{400} \right)}} \quad (1.13)$$

$$\mathbb{E}[s_{chal}^p] = \frac{1}{1 + 10^{\wedge \left(\frac{\mu_{curr}^a - \mu_{chal}^p}{400} \right)}} \quad \mathbb{E}[s_{chal}^a] = \frac{1}{1 + 10^{\wedge \left(\frac{\mu_{curr}^p - \mu_{chal}^a}{400} \right)}} \quad (1.14)$$

$$(1.15)$$

where $\mathbb{E}[s_{curr}^p] = \mathbb{E}[s(f_{\theta, curr}^{(player)})]$ denotes the score for the current player, and $\mu_{curr}^p = \mu(f_{\theta, curr}^{(player)})$ is the rating of the current player. The expected scores can be estimated via monte-carlo. Rearranging gives the relation between successive agent ratings as:

$$\mu_{chal}^p = \mu_{curr}^p + 400 \log_{10} \left(\frac{E[s_{chal}^p]}{1 - E[s_{chal}^p]} \cdot \frac{1 - E[s_{curr}^p]}{E[s_{curr}^p]} \right) \quad (1.16)$$

²Note, if the ratings were correctly updated in the last policy iteration then $\mathbb{E}[\mathbb{E}[s_{curr}^{agent}] - \mu_{curr}^{agent}] = 0$

Algorithm 3 MCTS (based on S.Nair’s implementation [15])

```
1: function GETACTIONPROB( $\mathbf{x}_{root}, \mathbf{x}^{(2D)}, agent$ )
2:   for  $i$  in nMCTSSims do ▷ Simulate nMCTSSims episodes from  $\mathbf{x}_{root}$ 
3:      $reset(\mathbf{x}_{root})$ 
4:      $MCTSSearch(\mathbf{x}^{(2D)}, agent)$ 
5:   end for
6:    $N(\mathbf{x}_{root}, u) \leftarrow getCounts()$  ▷ Count the times each edge,  $(\mathbf{x}_{root}, u)$ , was visited.
7:    $N(\mathbf{x}_{root}, u) \leftarrow N(\mathbf{x}_{root}, u)^\tau$  ▷ Control Sparsity with the temperature,  $\tau$ 
8:   return  $\pi \leftarrow norm(N(\mathbf{x}_{root}, u))$ 
9: end function

10: function MCTSSearch( $\mathbf{x}^{(2D)}, agent$ )
11:   if  $\mathbf{x}$  is terminal then
12:      $\pi, v \leftarrow f_\theta(\mathbf{x}^{(2D)})$ 
13:     return  $v$  ▷ Use  $v_{nnet}$  whether fallen or past the step limit
14:   end if
15:
16:   if  $\mathbf{x} \notin Tree$  then ▷ Expand Leaf Node
17:      $\pi, v \leftarrow f_\theta(\mathbf{x}^{(2D)})$ 
18:      $N(\mathbf{x}, \cdot) \leftarrow 0$ 
19:      $P(\mathbf{x}, \cdot) \leftarrow \pi$ 
20:     return  $v$ 
21:   end if
22:
23:   if  $agent = player$  then ▷ Get best action using UCB
24:      $u^* \leftarrow \underset{u \in \mathcal{U}_{player}}{argmax} U_{player}(\mathbf{x}, u)$ 
25:   else
26:      $u^* \leftarrow \underset{u \in \mathcal{U}_{adv}}{argmax} U_{adv}(\mathbf{x}, u)$ 
27:   end if
28:    $\mathbf{x}, c \leftarrow step(\mathbf{x}, u)$  ▷ Take next MCTS simulated step
29:    $\mathbf{x}^{(2D)} \leftarrow getState2D(\mathbf{x}, \mathbf{x}^{(2D)})$ 
30:    $agent \leftarrow nextAgent(agent)$ 
31:    $v \leftarrow MCTSSearch(\mathbf{x}^{(2D)}, agent)$  ▷ Recursion to next node
32:
33:    $Q(\mathbf{x}, u^*) \leftarrow \frac{N(\mathbf{x}, u^*)Q(\mathbf{x}, u^*) + v}{N(\mathbf{x}, u^*) + 1}$  ▷ Backup Q-values up the tree
34:    $N(\mathbf{x}, u^*) \leftarrow N(\mathbf{x}, u^*) + 1$ 
35:   return  $v$ 
36: end function
```

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:

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

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

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

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

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

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

$$\dot{\mathbf{h}}_P + \dot{\mathbf{r}}_P \times \boldsymbol{\rho} = \begin{bmatrix} 0 \\ 0 \\ -mL(\ddot{x}\cos\theta + L\ddot{\theta}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -L(mg\sin\theta + 2F_2\cos\theta) \end{bmatrix} = \mathbf{Q}_e$$

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

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

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

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

$$\ddot{x}(M + m\sin^2\theta) = F_1 - F_2\cos(2\theta) + m\sin\theta(L\dot{\theta}^2 - g\cos\theta) \quad (\text{A.2})$$

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

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{\left(\frac{2M+m}{m}F_2 - F_1\right)\cos\theta + g(M+m)\sin\theta - mL\dot{\theta}^2\sin\theta\cos\theta}{(M+m\sin^2\theta)} \\ \dot{\theta} \\ \frac{F_1 - F_2\cos(2\theta) + m\sin\theta(L\dot{\theta}^2 - g\cos\theta)}{L(M+m\sin^2\theta)} \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{x}, F_1, F_2) \\ f_2(\mathbf{x}, F_1, F_2) \\ f_3(\mathbf{x}, F_1, F_2) \\ f_4(\mathbf{x}, F_1, F_2) \end{bmatrix} \quad (\text{A.3})$$

Using Lyapunov's indirect method, we can write the linearised equations about the equilibrium, $\mathbf{x}_e = [x_e, \dot{x}_e, \theta_e, \dot{\theta}_e]^T = [0, 0, 0, 0]^T$, as:

$$\begin{bmatrix} \delta\dot{x} \\ \delta\ddot{x} \\ \delta\dot{\theta} \\ \delta\ddot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x} \big|_{\mathbf{x}_e} & \frac{\partial f_1}{\partial \dot{x}} \big|_{\mathbf{x}_e} & \frac{\partial f_1}{\partial \theta} \big|_{\mathbf{x}_e} & \frac{\partial f_1}{\partial \dot{\theta}} \big|_{\mathbf{x}_e} \\ \frac{\partial f_2}{\partial x} \big|_{\mathbf{x}_e} & \frac{\partial f_2}{\partial \dot{x}} \big|_{\mathbf{x}_e} & \frac{\partial f_2}{\partial \theta} \big|_{\mathbf{x}_e} & \frac{\partial f_2}{\partial \dot{\theta}} \big|_{\mathbf{x}_e} \\ \frac{\partial f_3}{\partial x} \big|_{\mathbf{x}_e} & \frac{\partial f_3}{\partial \dot{x}} \big|_{\mathbf{x}_e} & \frac{\partial f_3}{\partial \theta} \big|_{\mathbf{x}_e} & \frac{\partial f_3}{\partial \dot{\theta}} \big|_{\mathbf{x}_e} \\ \frac{\partial f_4}{\partial x} \big|_{\mathbf{x}_e} & \frac{\partial f_4}{\partial \dot{x}} \big|_{\mathbf{x}_e} & \frac{\partial f_4}{\partial \theta} \big|_{\mathbf{x}_e} & \frac{\partial f_4}{\partial \dot{\theta}} \big|_{\mathbf{x}_e} \end{bmatrix} \begin{bmatrix} \delta x \\ \delta\dot{x} \\ \delta\theta \\ \delta\dot{\theta} \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial F_1} \big|_{\mathbf{x}_e} & \frac{\partial f_1}{\partial F_2} \big|_{\mathbf{x}_e} \\ \frac{\partial f_2}{\partial F_1} \big|_{\mathbf{x}_e} & \frac{\partial f_2}{\partial F_2} \big|_{\mathbf{x}_e} \\ \frac{\partial f_3}{\partial F_1} \big|_{\mathbf{x}_e} & \frac{\partial f_3}{\partial F_2} \big|_{\mathbf{x}_e} \\ \frac{\partial f_4}{\partial F_1} \big|_{\mathbf{x}_e} & \frac{\partial f_4}{\partial F_2} \big|_{\mathbf{x}_e} \end{bmatrix} \begin{bmatrix} \delta F_1 \\ \delta F_2 \end{bmatrix} \quad (\text{A.4})$$

$$\begin{bmatrix} \delta\dot{x} \\ \delta\ddot{x} \\ \delta\dot{\theta} \\ \delta\ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(m+M)}{ML}g & 0 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta\dot{x} \\ \delta\theta \\ \delta\dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{1}{M} & -\frac{1}{M} \\ 0 & 0 \\ -\frac{1}{ML} & \frac{2M+m}{mML} \end{bmatrix} \begin{bmatrix} \delta F_1 \\ \delta F_2 \end{bmatrix} \quad (\text{A.5})$$

The eigenvalues are given by $\det(\lambda I - A) = \lambda^2(\lambda^2 - \frac{(m+M)}{ML}g) = 0$. Therefore, the system is unstable about \mathbf{x}_e due to the right half plane pole, $\lambda = \sqrt{\frac{(m+M)}{ML}}g$. Additionally, the time constant of this unstable system is $\tau = \sqrt{\frac{ML}{g(m+M)}}$. Note, if $M \gg m$, $\tau \rightarrow \sqrt{\frac{L}{g}}$, which is the time constant for a simple pendulum.

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

$$\text{rank}[\mathbf{B} \ \mathbf{A}\mathbf{B} \ \mathbf{A}^2\mathbf{B} \ \mathbf{A}^3\mathbf{B}] = 4 \quad (\text{A.6})$$

Therefore for any initial condition we can reach \mathbf{x}_e in finite time under these linear assumptions.

A.2 Propagation of Quantisation Error

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

$$\mathbf{x}_t^{(2D)} = C\mathbf{x}_t + \mathbf{V}_t \quad \mathbf{V}_t \sim \mathcal{U} \left(\begin{bmatrix} \frac{1}{\delta x} \\ \frac{1}{\delta \theta} \end{bmatrix} \right) \quad (\text{A.7})$$

$$\mathbf{x}_t = A\mathbf{x}_{t-1} + B\mathbf{u}_t \quad (\text{A.8})$$

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} .

Assuming the quantisation bin sizes, δx and $\delta \theta$, are small and that x and θ are independent within the bin, the quantisation noise can be modelled as uniform random variables with covariance, $\text{cov}(\mathbf{V}, \mathbf{V}) = \mathbb{E}[\mathbf{V}\mathbf{V}^T]$:

$$= \mathbb{E} \begin{bmatrix} x^2 & x\theta \\ \theta x & \theta^2 \end{bmatrix} = \begin{bmatrix} \int_{-\delta x/2}^{\delta x/2} x^2 \cdot \frac{1}{\delta x} dx & 0 \\ 0 & \int_{-\delta \theta/2}^{\delta \theta/2} \theta^2 \cdot \frac{1}{\delta \theta} d\theta \end{bmatrix} = \begin{bmatrix} \frac{\delta x^2}{12} & 0 \\ 0 & \frac{\delta \theta^2}{12} \end{bmatrix} \quad (\text{A.9})$$

For simplicity, let $\delta x = \delta \theta$, and therefore, $\text{cov}(\mathbf{V}, \mathbf{V}) = \sigma_v^2 I$.

Kalman filtering can be used to find an optimal estimate $\hat{\mathbf{x}}_n = K[\mathbf{x}_n | \mathbf{y}_{1:n}]$ using the algorithm (derived in [3]).

Algorithm 4 Multivariate Kalman Filtering

1: **Given:** $\hat{\mathbf{x}}_n = K[\mathbf{x}_n | \mathbf{y}_{1:n}]$ and $\Sigma_n = \mathbb{E}[(\mathbf{x}_n - \hat{\mathbf{x}}_n)(\mathbf{x}_n - \hat{\mathbf{x}}_n)^T]$

Prediction:

2: $\bar{\mathbf{x}}_{n+1} = K[\mathbf{x}_n | \mathbf{y}_{1:n}] = A\hat{\mathbf{x}}_n + B\mathbf{u}_{n+1}$

3: $\bar{\Sigma}_{n+1} = \mathbb{E}[(\mathbf{x}_{n+1} - \bar{\mathbf{x}}_{n+1})(\mathbf{x}_{n+1} - \bar{\mathbf{x}}_{n+1})^T] = A\Sigma_n A^T + \Sigma_w \quad \triangleright \Sigma_w = \mathbf{0}$

Update:

4: $\tilde{\mathbf{y}}_{n+1} = \mathbf{y}_{n+1} - C\bar{\mathbf{x}}_{n+1} \quad \triangleright \text{Calculate Innovation residual}$

5: $S_{n+1} = \Sigma_v + C\bar{\Sigma}_{n+1}C^T \quad \triangleright \text{Calculate Innovation Covariance}$

6: $\hat{\mathbf{x}}_{n+1} = \bar{\mathbf{x}}_{n+1} + \bar{\Sigma}_{n+1}C^T S_{n+1}^{-1} \tilde{\mathbf{y}}_{n+1} \quad \triangleright \Sigma_v = \sigma_v^2 I$

7: $\Sigma_{n+1} = (I - \bar{\Sigma}_{n+1}C^T S_{n+1}^{-1}C)\bar{\Sigma}_{n+1}$

Thus we can find the optimal linear estimate of the covariance of Equation (A.7) from algorithm 4 line 7:

$$\Sigma_{n+1} = (I - \bar{\Sigma}_{n+1}C^T S_{n+1}^{-1}C)\bar{\Sigma}_{n+1} \quad (\text{A.10})$$

$$= (I - \bar{\Sigma}_{n+1}C^T(\sigma_v^2 I + C\bar{\Sigma}_{n+1}C^T)^{-1}C)\bar{\Sigma}_{n+1}, \quad \text{where } \bar{\Sigma}_{n+1} = A\Sigma_n A^T \quad (\text{A.11})$$

For the univariate case this reduces to:

$$\sigma_{n+1}^2 = \bar{\sigma}_{n+1}^2 \left(1 - \frac{C^2 \bar{\sigma}_{n+1}^2}{\sigma_v^2 + C^2 \bar{\sigma}_{n+1}^2}\right) \quad (\text{A.12})$$

$$= A^2 \sigma_n^2 \left(1 - \frac{C^2 A^2 \sigma_n^2}{\sigma_v^2 + C^2 A^2 \sigma_n^2}\right) \quad (\text{A.13})$$

It is obvious from this that as $\sigma_v^2 \rightarrow 0$, $\sigma_{n+1}^2 \rightarrow 0$. Furthermore, if the spectral radius, $\rho\left(A^2\left(1 - \frac{C^2 A^2 \sigma_n^2}{\sigma_v^2 + C^2 A^2 \sigma_n^2}\right)\right) < 1$, then the mean squared error from quantisation will reduce to zero. Since $0 < \left\|\frac{C^2 A^2 \sigma_n^2}{\sigma_v^2 + C^2 A^2 \sigma_n^2}\right\|_2^2 < 1$ if $\sigma_v^2 > 0$, a sufficient condition is that $A^2 \leq 1 \implies \rho(A) \leq 1$.

For the multivariate case, it can similarly be shown (by Gelfand's Theorem, $\rho(M_1 \dots M_n) \leq \rho(M_1) \dots \rho(M_n)$) that $\rho(\bar{\Sigma}_{n+1} C^T (\sigma_v^2 I + C \bar{\Sigma}_{n+1} C^T)^{-1} C) < 1$, therefore a sufficient condition for the decay of the covariance is $\rho(A \Sigma_n A^T) \leq \rho(\Sigma_n) \implies \rho(A A^T) \leq 1$. For the linearised dynamics derived in eq. (A.5) the eigenvalues, λ are given by $\lambda^2(1 - \lambda)^2 = 0 \implies \rho(A A^T) \leq 1$. Therefore, the covariance decays to zero in the linearised multivariate case and the 2D state becomes a lossless estimate of the state.

The rate of decay for the univariate case can be determined with a first order approximation about $\sigma_v^2 = 0$:

$$\sigma_{n+1}^2(\sigma_v^2) = A^2 \sigma_n^2 \left(1 - \frac{C^2 A^2 \sigma_n^2}{\sigma_v^2 + C^2 A^2 \sigma_n^2}\right) \quad (\text{A.14})$$

$$\approx \sigma_{n+1}^2(0) + \delta \sigma_v^2 \frac{\partial \sigma_{n+1}^2(\sigma_v^2)}{\partial \sigma_v^2} \Big|_{\sigma_v^2=0} \quad (\text{A.15})$$

$$= \frac{\delta \sigma_v^2}{(A C \sigma_n)^2} \quad (\text{A.16})$$

As shown above, the mean square error decays with the square of the bin-size. Similarly, linearising about $\sigma_n^2 \approx 0$ gives $\sigma_{n+1}^2 = A^2$ for the linear case, which is constant.... Need to go to second order...

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] Russ Tedrake, *Underactuated Robotics*. MIT OpenCourseWare, Ch.3, Spring 2009.
- [9] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction (2nd Edition)*. The MIT Press, Cambridge, Massachusetts, London, England. 2018.
- [10] 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

- [11] Yuxi Li, *Deep Reinforcement Learning: An Overview*. CoRR, abs/1810.06339, 2018.
- [12] 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.
- [13] David Silver, Julian Schrittwieser, Karen Simonyan et al, *Mastering the game of Go without human knowledge*. Nature, vol. 550, pg.354–359, 2017.
- [14] 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.
- [15] S. Thakoor, S. Nair and M. Jhunjhunwala, *Learning to Play Othello Without Human Knowledge* Stanford University Press, 2018 <https://github.com/suragnair/alpha-zero-general>
- [16] HowlingPixel.com, *Elo Rating System*. https://howlingpixel.com/i-en/Elo_rating_system acc: 11/03/2019. published 2019