REINFORCEMENT LEARNING FOR CONTROL AND MULTIPLAYER GAMES

IIB Project Investigating the application of Google Deepmind's AlphaZero Algorithm to Classical Control Problems

UNIVERSITY OF CAMBRIDGE
DEPARTMENT OF ENGINEERING

Author Alex Darch

Supervisor Dr. Glenn Vinnecombe

Assessor Dr. Ioannis Lestas

St John's College

Cambridge

May 10, 2019

0 Contents

1 Introduction

1.1 Controlling Dynamical Systems

Dynamical systems have long been of great interest to engineers and scientists due to their ability to handily describe real world phenomena. They describe how a system changes through geometrical space with time and, in principle, their trajectories can be predicted solely from their initial state. In recent years this has become relatively easy, even for non-linear systems, by using numerical methods coupled with increases in computing power. However, often a precise solution is of less import than a prediction of the long term qualitative behaviour of the system. Much effort has been made to find methods to describe this long-term behaviour such as those made by Lyapunov in stability theory.

A corollary of the importance of dynamical systems is that influencing their behaviour is particularly useful. Highly non-linear models are difficult to control optimally and robustly, and the few mathematical techniques developed to deal with these can only handle very small subcategories of problems [?, ?]. The study of more general techniques to solve complex control problems has recently come to the forefront of the field with the advent of machine learning techniques such as reinforcement and deep learning. Many of these are not aimed specifically at control problems and are often designed to play games. This project looks at adapting one such algorithm - AlphaZero - from playing board games to solving general control problems such as the control of an aircraft in flight or the control of under-actuated robotics by starting with a simple inverted pendulum.

1.2 Control Theory

The first attempts at controlling dynamical systems came from classical control theory, which consisted of a series of 'cut-and-try' techniques based largely on adjusting the gains of PID controllers and lead/lag compensators until satisfactory closed loop dynamics were achieved [?].

It wasn't until the unification of the calculus of variations, classical control, random process theory and linear/non-linear programming by bellman in the 1950's [?] that truly optimal control was discovered. Optimal control consists of finding a control law for a specified optimality criterion and can be thought of as a non-linear feedback law, $u(t) = -K(t)x(t)$ based on the initial state, $x(0)$. In discrete time, an optimal control law can be found via dynamic programming (DP). DP is a systematic procedure for transforming an optimisation over a sequence of h inputs into h minimisations over 1 input (but for all states).

$$V(x_k, k) = \underset{u_{k:h-1}}{min}\left(\sum_{i=k}^{h-1} c(x_i, u_i) + J_h(x_h)\right) \tag{1.1}$$

$$= \underset{u_k}{min}\Big(c(x_k, u_k) + V(x_{k+1}, k+1)\Big) \tag{1.2}$$

$$u_k^* = \underset{u_k}{argmin}\Big(c(x_k, u_k) + V(x_{k+1}, k+1)\Big) \tag{1.3}$$

The dynamic programming equations above, where $c(x_i, u_i)$ is the cost as a function of the state and input at time i, $J_h(x_h)$ is the terminal cost, and $V(x_k, k)$ is the value function.

This enables a backwards recursion to find a sequence of value functions. This can be solved for the linear case with quadratic costs analytically (LQR) or, if the system is non-linear, via gradient descent. However, over a finite horizon this is essentially open-loop control. Optimal control of this form has been used to control complex dynamical systems such as spaceflight and aileron folding on aircraft [?, ?]. A closed loop extension to this is Model Predictive Control (MPC), which employs a receding horizon rather than a finite or infinite horizon. MPC can therefore easily deal with plant disturbances and uncertainties, constraints, indefinite horizons and can also be extended to get a control law for non-linear systems. MPC has recently been shown to work in trajectory control for interplanetary rovers [?]. Optimal control is limited in requiring sophisticated models of the environment/plant and generally struggle with highly non-linear models - state of the art is currently linearisation about the predicted trajectory. Furthermore, it is only feasible to 'grid' up to 5/6 dimensions in discrete cases [?].

1.3 Reinforcement Learning

Two further generalisations to dynamical systems and optimal control as defined in equation ?? are stochastic dynamics and indefinite horizons (i.e. episodic tasks). This discrete time stochastic control process is known as a Markov Decision Process (MPD). In MDPs the cost function is often written as a reward function and, due to the indefinite nature of the process, the value function for the next step is discounted (where $\lambda \approx 0.9$ typically):

$$V(x_k) = \underset{u_k}{max}\left(\sum_{i=k}^{\infty}\lambda^{i-k}r(x_i,u_i)\right) \tag{1.4}$$

$$= \underset{u_k}{max}\ \mathbb{E}\Big[r(x_k,u_k) + \lambda V(x_{k+1})\Big] \tag{1.5}$$

$$u_k^* = \underset{u_k}{argmax}\ \mathbb{E}\Big[r(x_k,u_k) + \lambda V(x_{k+1})\Big] \tag{1.6}$$

Reinforcement learning (RL) aims to learn the optimal policy, $\pi^*(x_k)$ ($= u^*(x_k)$ in control) of an MDP. This differs from optimal control in its inherent stochastic nature and therefore can lead to intractable search spaces. A solution to this is to learn form sample trajectories. Algorithms such as Q-Learning, SARSA and DYNA have recently had great success in control applications, for example, their use in controlling mobile robots [?, ?]. Furthermore, the advent of neural networks has led to the extension of these to functional approximations from tabula-rasa methods, making the control highly non-linear dynamical systems possible. Notably, Deepminds' recent success with training a robot to gently manipulate objects [?], would not be possible to reproduce using classical or modern control techniques due to dimensional problems.

1.4 AlphaZero

AlphaGo Zero is a revolutionary Reinforcement Learning algorithm that achieved superhuman performance in the game of go, winning 100–0 against the previously published, champion-defeating AlphaGo. It's successor, AlphaZero, is a generalised version that can achieve superhuman performance in many games. There are two key sub-algorithms that form the basis of their success: Monte-Carlo tree search (MCTS) for policy improvement, and a deep CNN for the neural policy and value network. Policy iteration is then implemented through self-play. AlphaGo Zero and AlphaZero differ only in the latter's use of a single neural network that is updated iteratively, rather than evaluated against the previous one, and by not taking advantage of the symmetries of the games.

A key feature of AlphaZero is that it only requires the ability to simulate the environment. It does not need to be told how to win, nor does it need an exact model of the system dynamics, $p(s_{t+1}|s_t, a_t)$, as this can be learnt through self-play. Furthermore, the algorithm often 'discovers' novel solutions to problems, as shown by 'move 37' in a game of Go against the reigning world champion, Lee Sedol. This makes it particularly suitable for learning to control complex dynamical systems where approximate simulations can be made.

1.4.1 Self-Play and The Neural Network

Figure **??**a shows how self-play is performed in AlphaZero. An episode/game, $\{s_1, ..., s_T\}$ is played and for each state, s_t, a Monte-Carlo Tree Search is performed, guided by the current policy f_θ. The MCTS outputs an improved action-*p.m.f*, $\boldsymbol{\pi}_t = [p(a_1|s_t), p(a_2|s_t), ..., p(a_N|s_t)]$. The next move is then selected by sampling from $\boldsymbol{\pi}_t$. The final player then gets scored (e.g in chess $z \in \{-1, 0, 1\}$ for a loss, draw or win respectively). The game score, z, is then appended to each state-action pair depending on who the current player was on that move to give training examples of the form $(s_t, \boldsymbol{\pi}_t, (-1)^{\mathbb{I}(winner)} z)$.

The neural network, $f_\theta(s)$ takes a board state, s, and outputs a *p.m.f* over all actions and the expected outcome, $(\boldsymbol{p}_\theta, v)$ (figure **??**). The networks are initialised with $\theta \sim \mathcal{N}(0, \epsilon)$, ϵ small. The neural network is trained to more closely match the MCTS-informed action-*p.m.f*.s, $\boldsymbol{\pi}_t$, and the expected outcome (state-value), v_t to z.

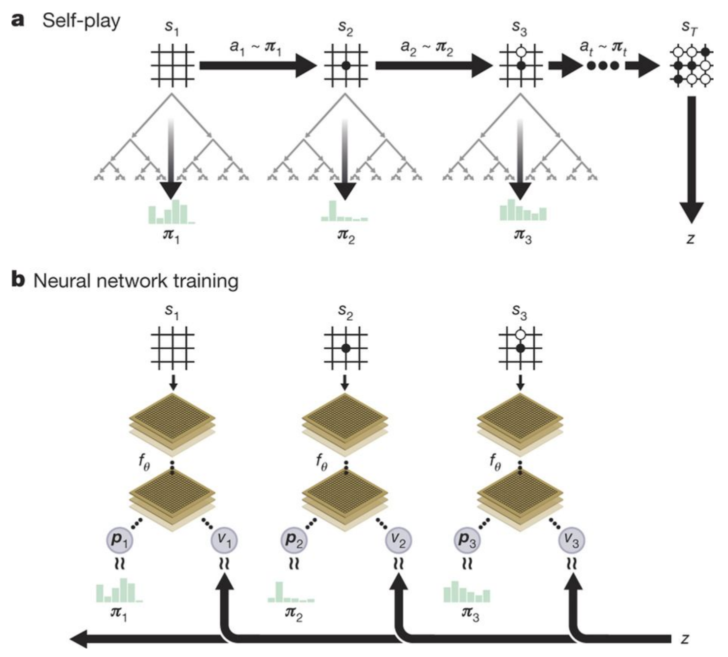

Figure 1.1: A schematic showing how self-play and policy training are performed. Taken from [?].

The loss function for the neural network is given by:

$$\mathcal{L} = (z - v)^2 - \boldsymbol{\pi} \cdot log(\boldsymbol{p}) + c||\theta||^2 \tag{1.7}$$

For chess, the input state consists of an image stack of 119 8x8 planes representing the board state at times $\{t, t-1, ..., t-8\}$, and planes representing repetitions, colours and castling etc. The output action *p.m.f* is a 8x8x73=4672 vector representing every possible move from each piece, illegal moves are then masked. The output value, $v \in (-1, 1)$. The network itself consists of an input convolutional block and two separate 'heads'. The policy head has softmax activation function, preceded by series of ReLU linear layers and batch normalisation layers. The value head also has this but with a tanh output activation. The input convolutional block consists a single convolutional layer followed by 19-39 residual blocks.

1.4.2 Monte Carlo Tree Search

Figure **??**a: A MCTS is performed at each step of an episode. The state at which the tree search starts then becomes the root state, s_{root}. From the root state, the tree search can move to an edge (s, a) by selecting an action, a. Before selection a prior $P(s,a) = (1-\epsilon)\boldsymbol{p}_{\theta,root} + \epsilon\boldsymbol{\eta}$ where $\boldsymbol{\eta} \sim Dir(0.3)$ and $\epsilon = 0.25$ is added to ensure exploration. Each edge stores a prior action-*p.m.f.* output by the policy, P(s, a)=$\boldsymbol{p}_\theta$; a visit count, N(s, a); and an action-value, Q(s, a). Actions are selected by maximising an the action-value plus an upper confidence bound, which encourages exploration. The constant c (~ 1) can be decreased to encourage exploitation.

$$a_{selected} = \underset{\forall a}{argmax}\Big\{Q(s,a) + c \cdot P(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}\Big\} \tag{1.8}$$

Figure **??**b: Once a leaf node (N(s, a) = 0) is reached, the neural network is evaluated at that state: $f_\theta(s) = (p(s,\cdot), v(s))$. The action *p.m.f.* and state-value are stored for the leaf state.

Figure **??**c: The action-values, Q, are calculated as the mean of state-values in the subtree below that action. The state-value are then calculated as the mean of all the action-values branching from that state, and so on.

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{s_{t+1}|s_t,a_t} v(s_{t+1}) \tag{1.9}$$

Figure **??**d: After a set number of simulations (1600), the MCTS-improved action $p.m.f$s, $\boldsymbol{\pi} = p(\boldsymbol{a}|s_{root}) \propto N^{1/\tau}$ are returned, where N is the visit count of each move from the root state and τ controls the sparseness of the probability mass function ($\{\tau = 0\} \rightarrow argmax\{N(s,a)\}$). For self-play, $\tau = 1$ for the first 30 moves and $\tau \rightarrow 0$ thereafter. For evaluation $\tau \rightarrow 0\ \forall t$. If $\tau \rightarrow 0$ then $\boldsymbol{\pi}$ becomes one-hot and the loss function of the neural network makes sense as a prediction of a categorical distribution.

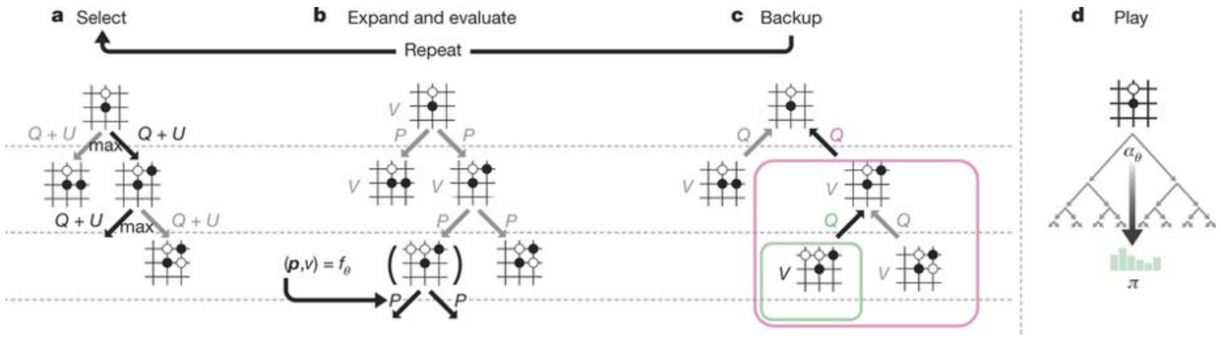

Figure 1.2: A schematic outlining the steps involved in a monte-carlo tree search. Taken from [?].

1.4.3 Policy Iteration

AlphaZero uses a single neural network that continually updates, whether it is better or worse than the previous network. Whereas AlphaGo Zero games are generated using the best player from all previous iterations and then only replaced if the new player wins $> 50\%$ of evaluation games played.

Evaluation is done by playing 400+ greedy ($\tau \rightarrow 0$) games of the current best neural network against the challenging neural network. The networks are then ranked based on an elo scoring system.

The application of AlphaZero to dynamical systems a number of potential benefits above traditional optimal control or reinforcement learning:

Robust Disturbance Handling - The Adversary in AlphaZero is effectively worst case scenario disturbance modelling. By incorporating this into the training process, the controller should be able to cope with situations well outside normal operating conditions.

Non-Linear Systems - Neural networks are universal function approximators and, hence with the correct architecture, therefore should be able to model any system.

General Framework - AlphaZero is a general algorithm that should be able to be applied to many different systems with minimal changes and still model the control well.

This project investigates these.

2 Theory & Methodology

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.

2.1 The Inverted Pendulum (IP)

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

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

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{\left(\frac{2M-m}{m}F_2 - F_1\right)cos\theta + g(M+m)sin\theta - mL\dot{\theta}^2 sin\theta cos\theta}{(M+msin^2\theta)} \\ \dot{\theta} \\ \frac{F_1 + F_2 cos(2\theta) + msin\theta(L\dot{\theta}^2 - gcos\theta)}{L(M+msin^2\theta)} \end{bmatrix} \tag{2.1}$$

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

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{\frac{2M-m}{m}F_1 - F_2 + g(M+m)\theta}{M} \\ \dot{\theta} \\ \frac{F_1+F_2-gm\theta}{lM} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & g\frac{M+m}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{mg}{lM} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -\frac{1}{M} & \frac{2M-m}{Mm} \\ 0 & 0 \\ \frac{1}{lM} & \frac{1}{lM} \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{2.2}$$

Which, as expected, is unstable since $det(\lambda I - A) = 0 \implies \lambda^2(\lambda^2 + \frac{mg}{lM}) = 0$. 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 $\tau \approx 0.70s$. A discrete time step of 0.02s is 35x smaller than this and therefore we expect an impulse to cause $\sim 3\%$ change in the state values. This is far below the threshold for pulse-width modulation, i.e. the actions are fast enough for the input forces to be modelled as continuous **** Is this right?? experimentally, the largest velocities give even better results ****

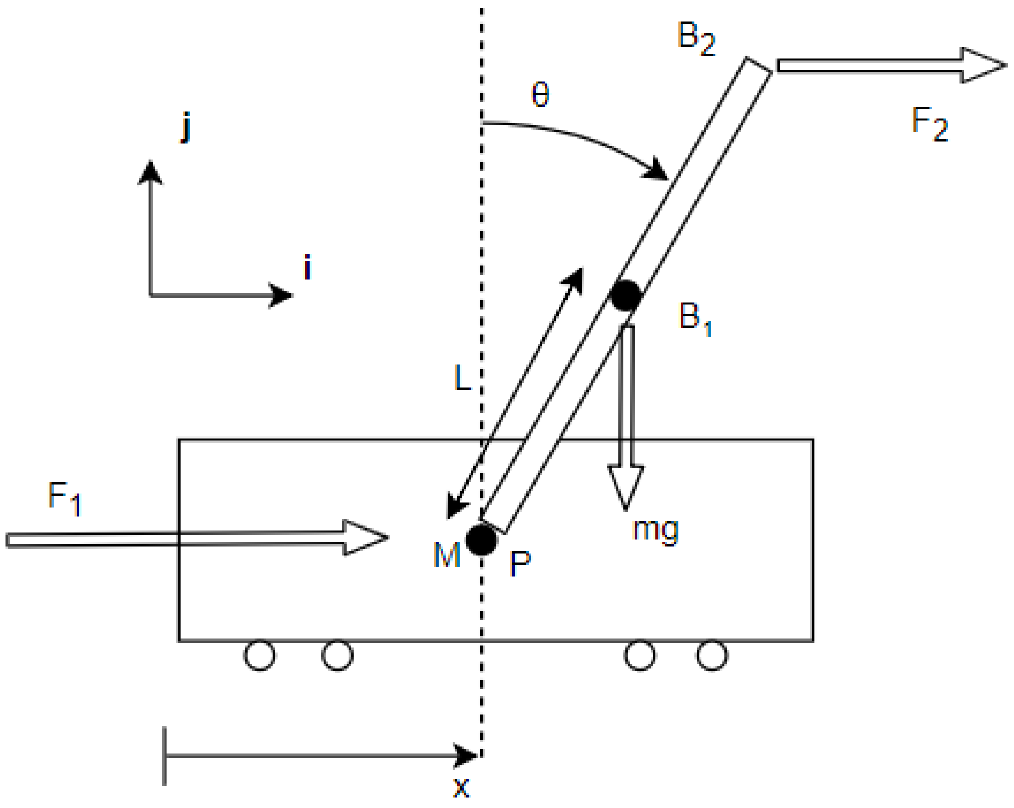

Figure 2.1: A free-body diagram of the inverted pendulum system. For the OpenAI IP the system is in discrete time with a time-step of $\tau = 0.02s$. The other constants are $l = 0.5m$, $m = 0.1kg$, $M = 1kg$, $F = \pm 10N$, $x_{max} = \pm 2.4m$, $\theta_{max} = \pm 12^o$.

2.2 Neural Network

2.2.1 Architectures

2.2.2 Loss Functions and Pareto

2.3 Scoring and Comparison

3 Experimental Techniques

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.

3.1 blah blah

4 References

[1] M.C. Smith, I Lestas, *4F3: 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] Arthur E. Bryson Jr, *Optimal Control - 1950 to 1985.* IEEE Control Systems, 0272-1708/95 pg.26-33, 1996.

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

[5] 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

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

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

[8] 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

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

[10] 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.

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

[12] 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.

5 Appendicies

5.1 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 $\boldsymbol{F}_e = \dot{\boldsymbol{\rho}}$:

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

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

$$\ddot{\theta}\big(M + msin^2\theta\big)L = \left(\frac{2M-m}{m}F_2 - F_1\right)cos\theta + g(M+m)sin\theta - mL\dot{\theta}^2 sin\theta cos\theta \quad (5.1)$$

$$\ddot{x}(M + msin^2\theta) = F_1 + F_2cos(2\theta) + msin\theta(L\dot{\theta}^2 - gcos\theta) \quad (5.2)$$

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

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{\left(\frac{2M-m}{m}F_2 - F_1\right)cos\theta + g(M+m)sin\theta - mL\dot{\theta}^2 sin\theta cos\theta}{(M+msin^2\theta)} \\ \dot{\theta} \\ \frac{F_1 + F_2cos(2\theta) + msin\theta(L\dot{\theta}^2 - gcos\theta)}{L(M+msin^2\theta)} \end{bmatrix} \quad (5.3)$$

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

$$rank[\boldsymbol{B}\ \boldsymbol{AB}\ \boldsymbol{A^2B}\ \boldsymbol{A^3B}] = 4 \quad (5.4)$$

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 sophistocated model.

5.1.1 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 cartpole, 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.