Application of Feedforward Control to Pan-Tilt Cameras on Planetary Rovers

by

Jordan Ross

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for
Mechanical and Aerospace Engineering

Department of Mechanical and Aerospace Engineering
Carleton University
Ottawa, Ontario, Canada
June 2014

Copyright ©
2014 - Jordan Ross
The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the Thesis

Application of Feedforward Control to Pan-Tilt Cameras on
Planetary Rovers

Submitted by Jordan Ross
in partial fulfilment of the requirements for the degree of
Master of Applied Science

__________________________
Dr. Alex Ellery, Supervisor

__________________________
Dr. M. Yaras, Department Chair

Carleton University
2014
Abstract

Future rover missions will be enhanced through the addition of science to the planetary traverse phase. Scientific targets are selected through a random search and salient gradient tracking in the visual field, which requires both a search algorithm and a reactive pan-tilt camera controller. This thesis presents a cerebellar-like reactive pan-tilt controller to track salient targets in the visual field as the rover moves based off the cerebellar models and the human vestibulo-ocular reflex. An online neural network using an EKF training law is used as a feed forward controller and it’s performance is compared to standard batch and online neural network training techniques. The controller was then applied to the Barrett WAM to control the manipulator wrist. The online EKF trained network is able to adequately model the internal dynamics of a pan-tilt, while remaining stable due to the continuous learning. This is shown in both simulation and practice.
# Table of Contents

Abstract iii  
Table of Contents iv  
List of Tables vii  
List of Figures viii  
List of Acronyms xii  
List of Symbols xiv  

1 Introduction 1  
1.1 Motivation ......................................................... 1  
1.2 Scope ............................................................. 4  
1.3 Overview ......................................................... 5  

2 Background and Literature Review 6  
2.1 Types of Control .................................................. 6  
2.1.1 Open-loop Control ............................................. 7  
2.1.2 Feedback Control ............................................. 7  
2.1.3 Feedforward Control ........................................ 8  
2.2 Manipulators ..................................................... 9  

iv
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.1 Kinematics</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2 Differential Kinematics</td>
<td>13</td>
</tr>
<tr>
<td>2.2.3 Inverse Kinematics</td>
<td>14</td>
</tr>
<tr>
<td>2.2.4 Dynamics</td>
<td>16</td>
</tr>
<tr>
<td>2.2.5 Manipulator Control</td>
<td>18</td>
</tr>
<tr>
<td>2.3 Cerebellum and VOR</td>
<td>19</td>
</tr>
<tr>
<td>2.4 Pan-Tilt Controllers</td>
<td>22</td>
</tr>
<tr>
<td>2.5 Neural Networks</td>
<td>23</td>
</tr>
<tr>
<td>2.5.1 Neuron Model</td>
<td>24</td>
</tr>
<tr>
<td>2.5.2 Network Structure</td>
<td>25</td>
</tr>
<tr>
<td>2.5.3 Learning Strategies</td>
<td>26</td>
</tr>
<tr>
<td>2.6 State Estimation and Filtering</td>
<td>27</td>
</tr>
<tr>
<td>2.6.1 Kalman Filters</td>
<td>28</td>
</tr>
<tr>
<td>2.6.2 Extended Kalman Filters</td>
<td>29</td>
</tr>
<tr>
<td>2.7 Barrett Whole Arm Manipulator (WAM)</td>
<td>31</td>
</tr>
<tr>
<td>3 Kinematic Model</td>
<td>34</td>
</tr>
<tr>
<td>3.1 Pan-Tilt</td>
<td>36</td>
</tr>
<tr>
<td>3.2 The Barrett WAM</td>
<td>39</td>
</tr>
<tr>
<td>4 Feedforward Control Applied in Simulation</td>
<td>42</td>
</tr>
<tr>
<td>4.1 Controller</td>
<td>43</td>
</tr>
<tr>
<td>4.1.1 Characteristic Behaviour</td>
<td>44</td>
</tr>
<tr>
<td>4.1.2 Neural Network Structure</td>
<td>44</td>
</tr>
<tr>
<td>4.1.3 Model Verification</td>
<td>45</td>
</tr>
<tr>
<td>4.2 EKF Training Algorithm</td>
<td>47</td>
</tr>
<tr>
<td>4.3 Back Propagation</td>
<td>51</td>
</tr>
<tr>
<td>4.3.1 Back Propagation with Momentum</td>
<td>51</td>
</tr>
</tbody>
</table>
## List of Tables

2.1 **Barrett WAM DH Parameters**: Published kinematic parameters for the Barrett WAM 

2.2 **Barrett WAM Dynamic Parameters**: Published dynamic parameters for the Barrett WAM 

4.1 **Neural Network Sizing**: Subset of the resulting verification parameters for different size hidden layers. Table was used to select most appropriate neural network size
# List of Figures

2.1 **Open-loop Controller**: Diagram of a simple open-loop controller. Note controller ignores the disturbance. .......................................................... 7

2.2 **Feedback Controller**: Diagram of a simple feedback controller. Note controller signal acts after the disturbance has affected the system. . . 8

2.3 **Feed Forward Controller**: Diagram of a simple feed forward controller. Note controller signal acts before the disturbance has affected the system. .......................................................... 9

2.4 **DH Convention**: Demonstration of how reference frames and parameters are selected for a link following the DH convention . . . . . . . . . . . 11

2.5 **Basic Cerebellum structure**: Structure of a Cerebellum as proposed by Kawato and Gomi. Skeleto-muscular dynamics are defined by the network and taught by feedback from the body. . . . . . . . . . . . . . . . . 22

2.6 **Neuron structure**: Structure of a forward propagated neuron. Weighted inputs are summed from the previous layer, added to a bias, transformed through the activation function and propagated forward. 24

2.7 **Neural network structure**: Common forward propagation neural network structure. Each neuron is connected to each neuron in the following layer until the outputs are found. . . . . . . . . . . . . . . . . 25
2.8 **Extended Kalman Filter flow diagram**: Flow diagram for the EKF state estimation algorithm. Process starts with the previous state estimate and a new state is estimated based on the system model and measurements. .................................................. 30

2.9 **Barrett WAM**: DH convention defined joint coordinate system for the Barrett WAM. All seven joints are defined. The last three are used in the control algorithms ................................................. 33

3.1 **Rover Coordinate System**: Selection of reference frame orientations and Cartesian vectors used to define the rover and pan-tilt system during traverse. .......................................................... 35

3.2 **Pan-tilt Coordinate System**: Selection of reference frame orientations the pan-tilt camera mount. .......................................................... 36

4.1 **Feedforward Controller**: Structure of the designed feedforward controller .......................................................... 45

4.2 **Simulator output**: Output of the Barrett WAM simulator to track objects in the visual field. Purple $x$ represents the target and blue axis the camera viewing direction. Units are in m. ................................. 57

4.3 **Linear regression of EKF NN**: Linear regression of the EKF trained neural networks. Perfect fit follows the $y = x$ diagonal. .............. 60

4.4 **Results of the EKF trained NN**: Resulting torques for Barrett WAM wrist from the EKF trained neural network (dots) using random inputs compared to the standard dynamic equation output (circles). 61

4.5 **Linear regression of batch back propagated NN**: Linear regression of the batch Levenberg-Marquardt Backpropagation trained neural networks. Perfect fit follows the $y = x$ diagonal. .............. 62
4.6 **Results of the batch MBP trained NN**: Resulting torques for Barrett WAM wrist from the LMB batch trained neural network (dots) using random inputs compared to the standard dynamic equation output (circles). .......................................................... 63

4.7 **Linear regression of online back propagated NN**: Linear regression of the online BPM trained neural networks. Perfect fit follows the \( y = x \) diagonal. .......................................................... 65

4.8 **Results of the online BPM trained NN**: Resulting torques for Barrett WAM wrist from the BPM online trained neural network (dots) using random inputs compared to the standard dynamic equation output (circles). .......................................................... 66

4.9 **\( R^2 \) and MSE during EKF training**: Top figure shows the current \( R^2 \) (unitless) during training with the EKF algorithm. Bottom figure shows the current MSE (\( Nm^2 \)) during training with the EKF algorithm. 68

4.10 **\( R^2 \) and MSE during BPM training**: Top figure shows the current \( R^2 \) (unitless) during training with the BPM algorithm. Bottom figure shows the current MSE (\( Nm^2 \)) during training with the BPM algorithm. 69

4.11 **Barrett WAM Simulation**: Barrett WAM simulation using random inputs controlled by the EKF taught neural network. Units are in m. 70

5.1 **Tracking output**: Visual tracking right and left of the target as seen by the end-effector. Target being track is the steel haptic ball. ......... 76

5.2 **Barrett WAM setup**: Configuration of the Barrett WAM. First four joints represent inputs and the last three are used to correct orientation so the camera can track targets. ........................................... 76

5.3 **Feedback Control Diagram**: Diagram summarizing the PID control of robotic manipulators. ......................................................... 79
5.4 **Linear regression of neural network**: Linear regression of the EKF trained neural network applied to the Barrett WAM. Perfect fit follows the \( y = x \) diagonal. .......................................................... 80

5.5 **Linear regression of feedback**: Linear regression of the feedback control law applied to the Barrett WAM. Perfect fit follows the \( y = x \) diagonal. .......................................................... 81
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronyms</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>BPM</td>
<td>Back Propagation with Momentum</td>
</tr>
<tr>
<td>CMAC</td>
<td>Cerebellar Model Articulation Controller</td>
</tr>
<tr>
<td>DH</td>
<td>Denavit-Hartenberg Convention</td>
</tr>
<tr>
<td>DOF</td>
<td>Degrees Of Freedom</td>
</tr>
<tr>
<td>EKF</td>
<td>Extended Kalman Filter</td>
</tr>
<tr>
<td>FEL</td>
<td>Feedback Error Learning</td>
</tr>
<tr>
<td>IMU</td>
<td>Internal Measurement Unit</td>
</tr>
<tr>
<td>LMB</td>
<td>Levenberg-Marquardt Backpropagation</td>
</tr>
<tr>
<td>MEP</td>
<td>Mars Exploration Program</td>
</tr>
<tr>
<td>MER</td>
<td>Mars Exploration Rovers</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Square Error</td>
</tr>
<tr>
<td>MSL</td>
<td>Mars Science Laboratory</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional/Integral/Differential</td>
</tr>
<tr>
<td>VOR</td>
<td>Vestibulo-Ocular Reflex</td>
</tr>
<tr>
<td>WAM</td>
<td>Whole Arm Manipulator</td>
</tr>
</tbody>
</table>
### List of Symbols

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manipulators</strong></td>
<td></td>
</tr>
<tr>
<td>$d_i$</td>
<td>Link length of the $i^{th}$ joint</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>Joint angle of the $i^{th}$ joint</td>
</tr>
<tr>
<td>$a_i$</td>
<td>Link offset of the $i^{th}$ joint</td>
</tr>
<tr>
<td>$\alpha_i$</td>
<td>Link angle of the $i^{th}$ joint</td>
</tr>
<tr>
<td>$T_{a}^{b}$</td>
<td>Homogeneous transformation matrix from frame $a$ to $b$</td>
</tr>
<tr>
<td>$R_{a}^{b}$</td>
<td>Rotation matrix from frame $a$ to $b$</td>
</tr>
<tr>
<td>$p$</td>
<td>End-effector position</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Link angular velocity</td>
</tr>
<tr>
<td>$J$</td>
<td>End-effector Jacobian matrix</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Joint torque</td>
</tr>
<tr>
<td>$D$</td>
<td>Inertial matrix</td>
</tr>
<tr>
<td>$h$</td>
<td>Centrifugal and Coriolis matrix</td>
</tr>
</tbody>
</table>
$G$  Gravitational torques  
$f$  Link forces  
$\mu$  Link moments  
$I_i$  Mass moment of inertia of the $i^{th}$ link  
$r_{i-1}^i$  Vector from link $i - 1$ to link $i$  
$r_{Ci}^i$  Vector from link $i$ to its center of mass  
$g$  Acceleration due to gravity  
$\tau_d$  Disturbance torque  
$e$  Joint position error vector  
u  Control signal  
w  Disturbance function  
$K_p$  Proportional gains  
$K_i$  Integrative gains  
$K_v$  Differential gains  

---

**Neural Networks**

$g(x)$  Activation energy  
w$_1$  Neuron weights from the first to second layer  
w$_2$  Neuron weights from the second to third layer  
w$_1^0$  Neuron biases for the second layer
$w_2^0$ Neuron biases for the output layer

$y$ Neural network output

---

**Kalman Filters**

$x$ Estimated state

$F$ Process function

$B$ input function

$u$ System input

$w$ process noise

$y$ Measured state

$H$ Measurement function

$v$ Measurement noise

$K$ Kalman gain

$f$ Non-linear process function

$h$ Non-linear measurement function
Chapter 1

Introduction

1.1 Motivation

Rover missions today are managed as highly structured projects. Every mission has a specific set of scientific goals that will be explored in different locations on the planetary surface. The Mars Exploration Rovers (MER) and newest Mars Science Laboratory (MSL) for instance had four primary scientific goals: [1]

1. Determine whether life ever existed on Mars by searching for signs of stable long-standing water supply.

2. Identify the current and past climate of Mars

3. Characterize the geology and the relative roles of wind, water, volcanism, tectonics, and cratering have modified the Martian surface.

4. Prepare for human exploration of the Martian surface

These goals are defined by the Mars Exploration Program (MEP) and are achieved through NASA’s Following the Water and Seek Signs of Life strategy in which scientific targets are selected based on locations where water may exist or may have
existed. One of the more common markers alluding to past or present water formations are sedimentary rocks and it was this marker that scientists used to determine potential landing sites of the most recent Mars lander. The site selection was based on evidence such as topographical lows [2], obtained from orbiters around the planet [3]. While sedimentary rocks mean water, the other primary rock types, including metamorphic and igneous formations, shed light to potential volcanic and tectonic history, layering of the Martian surface and characterization of the past and current climate [4]. In all these cases, the scientifically interesting targets are selected considering the surface geology, and as such sites are selected as being probable areas to find the desired geology [5].

Once landed on the planetary surface, rover missions can be broken into two fundamental phases: the traverse phase and the scientific phase. Rovers cannot be directly placed at the desired scientific sites and as such they are instead landed on the surface some safe distance away. The rover must then traverse across the surface until it reaches the pre-defined location of high scientific interest, where it will deploy the majority of its sensors and instruments to carry out the scientific phase of the mission. During the traverse, the rover at set intervals will image the surrounding area, generate a map and plan a path through the immediate rock formations. Once the optimal path has been determined, the rover executes the path while employing different forms of obstacle avoidance. Once the rover has travelled a set distance, it stops, and repeats the process [6]. As the rover maps the area and executes the planned path trajectory, it is essentially blind and is not actively searching for potential targets in the surrounding environment. This opens the possibility that the rover may pass zones of high scientific yield, potentially more valuable than the predefined locations which are roughly selected. Future rover missions will be enhanced through the opportunistic search for targets during the traverse phase through the addition of active vision.
Active vision in this sense has yet to be used in any planetary rover mission. Active vision itself is a paradigm that suggests the integration of controlled actuation may convert a non-linear and unstable problem from passive observer to a stable solution for an active observer through the use of multiple information sources [7][8]. Its use in future missions has the potential to reduce the computational requirements and resources by focusing only on relevant areas of the visual field instead of on its entirety. This will require capabilities to allow the sensory instruments to track targets in both the real world and the sensor output. In other words, the problem can be broken into a search for salient targets at the image processing level, primarily through tracing image gradients, and a camera control problem. The camera control part acts on the pan-tilt camera mount to ensure that the camera is able to track the salient targets in the visual field as the rover moves. This thesis presents a camera control algorithm to allow for stable, fast tracking at the pan-tilt level in order to track targets in the visual field, taking into consideration the motion of the rover.

Such controller will need to be able to quickly adapt for changes in the orientation and position of the rover as it moves and compensate for the perturbations at the pan-tilt camera mount. The standard approach for any pan-tilt control algorithm is proportional/integral/differential (PID) feedback control. Feedback control however allows disturbances to alter the system before correcting for them, which is non ideal for reactive tracking. A natural starting point in the development of a new controller is to look at similar processes in nature and model a design after known systems. One such process is the vestibulo-ocular reflex (VOR) which occurs in the cerebellum. The cerebellum uses feed forward control, which learns the dynamics of the skeletomuscular system of the body. Feedforward control is able to predict errors in the system dynamics before they occur and correct for them prior to them affecting the system. The designed controller will attempt to track objects in the visual field using a feed forward cerebellar-like model based on the VOR occurring in the brain.
1.2 Scope

The objective of this thesis is to develop a reactive controller for the pan-tilt camera mount of a rover. The controller will allow the camera to track predefined targets as the rover moves. This is done by analysing strategies that are currently found in nature and applying them to the camera mount in the context of rover missions. The controller will be cerebellar-like, and as such will be designed and an extension of the feedback-error-learning (FEL) model. The FEL is essentially a neural network that is actively trained. The presented work is broken into three parts, first the development of a suitable kinematic model to characterize the motion of the required motion of the pan-tilt, second the design and simulation of pan-tilt controller and finally the testing of the designed algorithm in hardware. The simulated and practical models will both be applied to the seven degree of freedom (DOF) Barrett Whole Arm Manipulator (WAM).

1. The kinematic model for the controller is designed to specify the required joint space trajectory needed to track stationary objects in the visual field as the rover is moving. First a general model will be developed for standard 2-DOF pan-tilt camera mounts and a second specifically designed for the 3-DOF wrist of the Barrett WAM.

2. The simulated model will be used to validate the potential of a cerebellar-like controller for use with a pan-tilt camera mount. In simulation, a neural network will be tested to determine its suitability for modelling the pan-tilt dynamics. The second aspect to be simulated is the validation of the efficiency of an active learning training algorithm relative to more standard practises. The stability of the trained network is also presented.

3. Hardware testing will be performed on the Barrett WAM and will be used
to determine whether the cerebellar-like controller developed in simulation is valid for real world tracking applications. It will also be used to quantify the capabilities of a feedforward model compared to a standard feedback controller.

1.3 Overview

The remainder of this thesis will proceed as follows:

Chapter 2 will detail the background information needed to follow the work package performed in this thesis. The material to be covered includes general control theory, manipulators and manipulator control theory, the cerebellum and cerebellar controllers, neural networks, extended Kalman filter and the Barrett WAM.

Chapter 3 details the kinematic model of the pan-tilt camera control algorithm. The kinematic model of a pan-tilt is similar to that of a 2-DOF manipulator, using a vector to the target and the measured motion of the rover. Both a general model for standard pan-tilt mount and for the 3-DOF Barrett WAM wrist is developed; the Barrett WAM wrist will be used in both simulation and hardware testing.

Chapter 4 discusses the development of the cerebellar-like controller. The different training algorithms to be used are explained in this chapter and the simulation environment is detailed. Finally the results of testing and the conclusions that can be drawn from the experiments are presented.

Chapter 5 covers the application of the controller on the Barrett WAM. The basic coding structure is presented and the developed feedback control algorithm used for testing is explained. The results of testing and the conclusions that can be drawn from the experiments are also presented.

Finally chapter 6 summarizes the contributions of this thesis and presents future work that will be performed to follow up on the work presented.
Chapter 2

Background and Literature Review

This chapter will outline some of the background knowledge required to understand and appreciate the results of this thesis. This review will cover the following:

1. Types of control
2. Manipulators and manipulator control
3. The cerebellum and developments in cerebellar-like control
4. Pan-Tilt control
5. Neural networks
6. Extended Kalman filters
7. The Barrett WAM

2.1 Types of Control

Controllers are devices that both monitor and physically alter the operating conditions in a dynamic system. Control theory in essence is a large branch of mathematics which analyses the behaviour of dynamic systems with regards to their inputs. Basic control
theory includes three fundamental control laws: open-loop, feedback and feedforward control.

2.1.1 Open-loop Control

Open-loop control is the most basic form of control. In essence, open-loop control defines a control signal based on the dynamics of the system itself and the system input. Open-loop control assumes that there will be no disturbances acting on the system and therefore they are no measurements of either the system output or environment. Figure 2.1 demonstrates simple open-loop control. Naturally where the output is not being monitored and disturbances are not being accounted for, open-loop control is only efficient for systems that the dynamic model is fully known, as this type of control cannot account for errors.

2.1.2 Feedback Control

Feedback control is the most common controller. Feedback control uses measurements of the system it is controlling to determine the required action. A measurement of the state is compared to the desired system state and the control signal is selected based on the difference. In feedback control the disturbance has already altered the system from its ideal operating conditions, the controller senses that change and acts
Figure 2.2: Feedback Controller: Diagram of a simple feedback controller. Note controller signal acts after the disturbance has affected the system.

to correct it using the scaled difference between the current and desired state. Figure 2.2 summarizes the simplified feedback controller. A major drawback of feedback control is that the state is not corrected until after it has reach an undesirable level, as the control law is a function of the system errors. The classic example is a thermostat attempting to control the temperature of a room. If a window is opened, the temperature of the room will decrease as the environment of the room has now changed. The temperature of the room will drop until it hits some threshold temperature, in which case the thermostat will increase the heat flow into the room until the temperature reaches the acceptable range. The most common form of feedback control is PID control, in which the control signal is defined by the current system error, past system error and a prediction on the future system error.

2.1.3 Feedforward Control

Feed forward control on the other hand is a relatively new branch of control theory, coming into existence with the discovery and development of neural networks. Feedforward controllers measure external properties of the system, or changes to the operating environment that are not controlled, and define a control signal before the
disturbance has a chance to alter the state. The control signal is defined by a prediction of how the disturbance will affect the system and is selected to prevent the state of the system from changing. Figure 2.3 summarizes the structure of a simplified feed forward controller. Using the thermostat example from before, a feedforward controller would sense that the environment has changed when the window was opened, know from some model and past experiences how much heat this would cause to leave the system and pre-emptively increase the heat input to the room to account for these losses. Since the amount of heat added to the system isn’t a function of the current temperature, the room has no chance to cool. Some more advanced control algorithms, such as the Smith predictor or adaptive controller, use a combination of feedforward and feedback control elements to achieve the desired system response [9]. These types of control will be further explained as applied to manipulators in the next section.

2.2 Manipulators

The pan-tilt camera mount can be modelled as a 2-DOF manipulator. The first DOF is represented by the pan or pitch of the camera and the second DOF is represented
by the *tilt* or *yaw* of the camera. Manipulator theory has been extensively studied and is well understood. As such, modelling the pan-tilt as a manipulator greatly simplifies the problem. Manipulators can be broken into two branches. First, the kinematics which quantifies the properties of motion like position, velocity and acceleration without consideration for why the body is moving the way it is. Second, the dynamics which focuses on the interactions between the links of the manipulator, like forces and moments. Manipulator dynamics are used to explain why the kinematic properties are moving the way they are.

### 2.2.1 Kinematics

A manipulator can be represented by a series of rigid links that are connected together by joints. The joint is where the individual transformations occur. A joint can be either prismatic in that it actuates by changing its length, or revolute in that it actuates by changing its angle. The manipulator is constrained on either end by a base and an end-effector. With manipulators, it is usually the trajectory of the end-effector that is to be controlled, depending on the application. The pose of any link, including the end-effector, is defined by both its orientation and Cartesian position. The pose is relative to some world coordinate frame, usually the base of the manipulator. The pose can be fully defined by analysing the specific change at each link, from the previous joint $i - 1$ the current joint $i$, as a combination of elementary rotations and translations. Before this analysis can be performed, the coordinate systems for each link and end-effector have to be defined, and the rotations/translations from link to link need to be quantified. The standard is to use the Denavit-Hartenberg (DH) convention. [10].

To use the DH convention, first the reference frame of each link and end-effector, starting with the base and moving up the manipulator, need to be defined. The base can have any arbitrary reference frame, and the reference frame of all the other
Figure 2.4: DH Convention: Demonstration of how reference frames and parameters are selected for a link following the DH convention.

According to Denavit-Hartenberg theory, the transformations from the reference frame of link \( i - 1 \) to link \( i \) can be completely described by four parameters: two representing translations and two representing rotations. The four parameters are the link length \( d \), joint angle \( \theta \), link offset \( a \) and link angle \( \alpha \). The transformations take the reference frame of link \( i - 1 \) and transform it to match the reference frame of link \( i \) through four steps described by the four DH parameters. First, reference frame is translated along the \( z_{i-1} \) axis by the distance \( d_i \). Second, the resulting reference
frame is rotated about the $z_{i-1}$ axis by angle $\theta_i$. Third, the resulting reference frame is translated along the $x_i$ axis by distance $a_i$. Finally, the resulting reference frame is rotated about the $x_i$ axis by angle $\alpha_i$ so that it matches the reference frame of joint $i$. A revolute joint actuates by changing the joint angle $\theta$, while the prismatic joint actuates by changing the joint length $d$. Figure 2.4 demonstrates this process [11].

$$T_{i-1}^i = T_{z_{i-1}}(d_i) \cdot T_{z_{i-1}}(\theta_i) \cdot T_{x_i}(a_i) \cdot T_{x_i}(\alpha_i)$$  

(2.1)

The series of translations and rotations make up the Homogeneous Transformation matrix. The Homogeneous Transformation matrix describes the pose of the link $i$ with respect to any previous frame.

$$T_{i-1}^i = \begin{bmatrix}
\cos(\theta_i) & -\sin(\theta_i) \cos(\alpha_i) & \sin(\theta_i) \sin(\alpha_i) & a_i \cos(\theta_i) \\
\sin(\theta_i) \cos(\alpha_i) & \cos(\theta_i) & -\cos(\theta_i) \sin(\alpha_i) & a_i \sin(\theta_i) \\
0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\
0 & 0 & 0 & 1
\end{bmatrix}$$  

(2.2)

Post multiplying the Homogeneous Transformation matrices from link to link next will eventually build the Homogeneous Transformation matrix representing the pose of the end-effector with respect to the base reference frame $T_{e}^b$. Assuming the manipulator has $n$ links, the process can be simplified as:

$$T_{b}^e = T_{b}^1 T_{1}^2 T_{2}^3 ... T_{n-1}^n T_{n}^e$$  

(2.3)

As previously mentioned, the Homogeneous Transformation matrix is a $4 \times 4$ matrix representing the pose of a link with respect to one of the previous frames. In
its simplest form, it can be described as:

\[
T_b^e = \begin{bmatrix}
  n_x & s_x & a_x & p_x \\
  n_y & s_y & a_y & p_y \\
  n_z & s_z & a_z & p_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
\] (2.4)

The normal \( n \), slide \( s \) and approach \( a \) make up the rotation matrix \( R \) defining the orientation of the end-effector with respect to a previous frame. In the matrix above, the rotation matrix the matrix highlighted in red. The vector \( p \) represents the end-effector position with respect to a previous frame. The end-effector position is the vector highlighted in blue.

\[ \frac{\partial p}{\partial \theta} \frac{\partial \theta}{dt} = J(\theta)\dot{\theta} \] (2.5)
Where \( \dot{p}_e \) and \( \omega_e \) are the end-effector linear and angular velocities, \( J(\theta) \) is the end-effector Jacobian and \( \dot{\theta} \) are the joint space angular velocities. In this way, the end-effector velocity can be directly related to the joint velocities through the Jacobian. The Jacobian of a manipulator can be broken into two \( 3 \times n \) parts: the Jacobian mapping the joint velocities to the end-effector linear velocity \( J_p \) and the Jacobian mapping the joint velocities to the end-effector angular velocity \( J_o \). The Jacobian can be derived through the differentiation of the rotation matrix with respect to the joint angles and simplified to the following expression.

\[
J(i, \theta) = \begin{bmatrix} J_p(i, \theta) \\ J_o(i, \theta) \end{bmatrix} = \begin{bmatrix} \hat{z}_{i-1} \times (p_e - p_{i-1}) \\ z_{i-1} \end{bmatrix} \quad (2.6)
\]

The relationship between accelerations can be found by differentiating the expression for the velocities with respect to time.

\[
\begin{bmatrix} \ddot{p}_e \\ \dot{\omega}_e \end{bmatrix} = J(\theta)\ddot{\theta} + J(\theta)\dot{\theta} \quad (2.7)
\]

### 2.2.3 Inverse Kinematics

In the forward kinematic model, the joint positions \( \theta_i \) are known and the end-effector pose is to be determined. The inverse problem involves determining the joint angles \( \theta_i \) that are required to give the end-effector a specific pose. This is a difficult problem for manipulators, since there are six world parameters to be controlled and therefore a minimum 6-DOF manipulator is needed to control both orientation and position. Redundant manipulators have more than 6-DOF and can therefore have multiple
solutions for the same pose, which again further complicates the problem. Some
manipulators have joint ranges that need to be taken into account, or kinematic
singularities at the certain orientations which will result in an infinite number of
solutions.

There is no set algorithm to solve the inverse kinematics of a manipulator, due
to the incredible number of potential structures. The function to solve for the joint
angles of the arm are generally highly non-linear and therefore a closed form solution
may not even exist. Solutions may not even be usable, if the determined joint angles
lie outside the joint range of motion, or if the manipulator does not have enough
DOF to control all desired aspects. Computation of closed-form solutions requires
algebraic intuition.

To simplify the inverse kinematics, some manipulators are designed with spherical
wrists. The spherical wrist allows the solution to be decoupled into two sub problems:
position and orientation. In this configuration, the wrist does not contain translations
and therefore doesn’t affect the end-effector Cartesian position. The wrist can be
ignored in the solution for the arm position and used only to achieve the desired
orientation, as is the case for the Barrett WAM testing platform.

The inverse of the differential kinematics on the other hand can be found by simply
inverting the differential kinematic equation presented in the previous section.

\[
\dot{\theta} = J^{-1}(\theta) \begin{bmatrix} \dot{p}_e \\ \omega_e \end{bmatrix}
\]  

(2.8)

Naturally this solution assumes that the Jacobian is always invertible, which is only
the case for 6-DOF manipulators without singularities. To get around this constraint,
approximations of the Jacobian were developed that can be used instead. The most
common approximations being the Jacobian transpose $J^{-1} \sim J^T$ or Jacobian pseudo-inverse $J^{-1} \sim J^T(JJ^T)^{-1}$.

Similarly, the joint accelerations can be solved by inverting the acceleration equation shown in the previous section.

$$\ddot{\theta} = J^{-1}(\theta) \begin{bmatrix} \ddot{p}_e \\ \dot{\omega}_e \end{bmatrix} - J^{-1}(\theta) \dot{J}(\theta) \dot{\theta}$$

(2.9)

### 2.2.4 Dynamics

Manipulator dynamics analyses the interactions between the forces and moments that cause motion. Manipulator dynamics are a function of the desired joint trajectory, $\theta$, $\dot{\theta}$ and $\ddot{\theta}$ and the properties of the arm itself. The most common dynamic models account for inertia, Coriolis, centrifugal and gravitational forces, however more advanced models compensate for motors, gear boxes and friction as well. Since the latter terms are difficult to accurately model, they are often ignored and corrected within the control algorithm. The dynamic equation of a manipulator can be simplified to the following:

$$\tau = D(\theta)\ddot{\theta} + h(\theta, \dot{\theta})\dot{\theta} + G(\theta)$$

(2.10)

Where $D$ is the inertia matrix, $h$ are the Coriolis and centrifugal forces and $G$ are the gravitational forces. There are two common derivations for the manipulator dynamic model. The first is the Lagrange-Euler method which considers the system as a whole. Lagrange method sums the kinetic and potential energy in all of the joints and applies the required torques in order to balance out the energy equation. This algorithm becomes very difficult with the inclusion of more and more joints. An alternative is
to use the Newton-Euler derivation. The Newton-Euler algorithm analyses the forces at each joint separately. This allows for a recursive algorithm, which starts at the base frame moving to the end-effector to calculate the link angular velocity $\omega_i$, angular acceleration $\dot{\omega}_i$, end of link acceleration $\ddot{p}_i$ and center of mass link acceleration $\ddot{p}_{Ci}$:

Forward Recursion ($i = 1 : n$)

\begin{align*}
\omega_i &= \omega_{i-1} + \dot{\theta}z_{i-1} \\
\dot{\omega}_i &= \dot{\omega}_{i-1} + \dot{\theta}z_{i-1} \times \omega_i \\
\ddot{p}_i &= \ddot{p}_{i-1} + \dot{\omega}_i \times \mathbf{r}_{i-1}^i + \omega_i \times (\omega_i \times \mathbf{r}_{i-1}^i) \\
\ddot{p}_{Ci} &= \ddot{p}_i + \dot{\omega}_i \times \mathbf{r}_{Ci}^i + \omega_i \times (\omega_i \times \mathbf{r}_{Ci}^i)
\end{align*}

A second backward recursion is then used to find the forces $f_i$ and moments $\mu_i$ on each joint. The joint toques $\tau_i$ can then be determined from the moments. This recursive algorithm used is as follows:

Backward Recursion ($i = n : 1$)

\begin{align*}
f_i &= f_{i+1} + m_i \ddot{p}_{Ci} - m_i g \\
\mu_i &= \mu_{i+1} - f_i \times (r_{i-1}^i + r_{Ci}^i) + f_{i+1} \times r_{Ci}^i + I_i \ddot{\omega}_i + \omega_i \times I_i \dot{\omega}_i \\
\tau_i &= \mu_i^T z_{i-1}
\end{align*}

The desired toques are converted into electrical commands and are applied to the manipulator joint motors to move the arm to the desired position. [11]
2.2.5 Manipulator Control

There are many problems in regards to manipulator dynamics as it currently stands. In many cases, terms arising from actuators, friction, gearboxes, communication delays, etc, are not considered within the dynamic formulation. In many cases, these terms can be quite large and cause the arm to move in an undesired trajectory, giving rise for the need of controllers [12]. Numerous control schemes have been proposed over the last few decades [13], the most common of which are feedback computed torque controllers [14][15][16]. Computed torque controllers are a form of robust control, which use a fixed structure that yield “acceptable” performance. These fixed structures are often erroneous due to the wide uncertainty in parameter estimations, such as inertias, geometry and loads [17].

A more precise solution which accounts and accommodates for the parameter uncertainties and attempts to accommodate them is adaptive control [17][18][19]. Adaptive manipulator control is an architecture in which the controller is able to adapt to dynamic or changing environments. One of the primary advantages of this structure is that it does not require any prior knowledge regarding the bounds of the uncertain parameters in order to estimate them. Adaptive controllers are able to apply feedforward and feedback control terms, which estimate the uncertain parameters affecting the system dynamics [20]. The downside being the addition of more parameters to the dynamic calculation tends to increase the computational requirements, sensitivities to precision and observation noise [21].

In recent years, a great deal of research has gone into developing simple biologically inspired control algorithms. As previously mentioned, feedforward control attempts to predict the dynamic model of the system, including uncertain or tough to model terms. The most common feedforward control used are computationally simple trained neural networks to predict either the unknown terms in the system dynamics [22] or the entire
dynamic model [23][24][25]. This type of control was developed through the study of the interconnections in the brain, leading to cerebellar-like feed forward controllers. The advantage of these systems is that they can be adapted to any environment and computationally much more simple than modelling each individual term.

2.3 Cerebellum and VOR

A natural starting point in developing a controller to track objects in the visual field is to look to nature for a similar process. An excellent such example is the vestibulo-ocular reflex (VOR). VOR maintains eye fixation during head rotations by counter-rotating the eyes with typically unity gain and very low latency (on the order of 10 ms), making it the fastest visual reflex. Feedback in the body happens quite slowly so the feedforward aspect is needed for stable tracking. The reflex measures head rotations through semicircular canals in the ears and moves the eyes appropriately in the opposite direction, allowing the eyes to track objects in the field of view. The control part of this reflex happens in the cerebellum.

The cerebellum is the feedforward control center for motor commands occurring in the brain. While the Cerebellum itself has been studied for hundreds of years from a scientific perspective [26], it has only been the last 50 years or so that the cerebellum has been modelled and begun to be understood. The cerebellum has an incredible ability to adapt and learn the highly non-linear dynamical models of the skeletomuscular system and in real-time can perform complex functions, provide timing control of opposing muscles and provide force and stiffness muscle control. The human cerebellum consists of two types of neurons: Purkinje cells and granule cells, and three types of axons: mossy fibers, climbing fibers and parallel fibers. The human cerebellum contains approximately 10 million Purkinje cells, each one receiving inputs from around 150 000 granule cells through parallel fibers. Input signals to the granule
cells originate at the spinal cord and are transmitted through mossy fibers. Each granule cell synapses to approximately 200 Purkinje cells, which also receive input from a single climbing fiber, which is strong enough to fire the cell when it is active. The primary neurons also receive inhibitory signals from stellate cells and basket cells ensuring the activation of a single Purkinje cell within a localized neighbourhood. The output of a Purkinje cell is an inhibitory signal, which is processed by the cerebellar nuclei.

The earliest theoretical models of the cerebellum to explain sensorimotor calibration or “learning” were derived by Marr [27] and Albus [28]. Their theory was inspired by the work done by Braitenberg and Atwood [29] and Eccles et al. [30] who considered the cerebellum as a simple timing organ. The models developed by Marr and Albus were based on the observation that each Purkinje cell receives two different types of input signals. The first type is from many relatively weak parallel fibers and the second is a very strong signal from the climbing fiber. The core concept in the models developed by Marr and Albus is that the climbing fiber acts as a teaching signal, which permanently changes the input strength of each of the parallel fibers. These models form the fundamental theory of how learning takes place in the cerebellum and are the first to incorporate a neural network as the base cerebellar structure.

One of the earliest computational models of the cerebellum was also developed by Albus, known as the Cerebellar Model Arithmetic Computer or Cerebellar Model Articulation Controller (CMAC) [31]. The CMAC controller is a derivation of the boxes approach originally proposed by Michie and Chambers [32] and essentially treats the cerebellum as a look-up table, causing the CMAC algorithm to have problems generalizing. Practically the CMAC has been used in a number of control tasks. Barto et al. showed mathematically that this learning network model can be used to solve a problem like the cart and pendulum [33]. Lang later used a variation of the CMAC
to control a 3-DOF manipulator [34] and Miller used a similar model to control 5-DOF manipulators [35][36]. Li and Leong applied the CMAC controller developed by Albus to a 5-DOF redundant manipulator to solve the inverse kinematic problem [37], as opposed to the control problem.

Miall et al. proposed a slightly different version of the cerebellum, where it acts as a Smith Predictor [38][39] since sensory feedback in the body is such a slow process. The Smith Predictor is designed for use in systems with pure time delay and adds an additional feedback loop with a forward model to predict the outcomes of the time delayed feedback. The forward model can be trained and adjusted to properly predict the generated feedback and correct for it preemptively. This is needed to prevent the time delayed feedback from becoming quickly outdated. The Smith predictor removes the time delay from the system by substituting it with its prediction. In essence, the Smith predictor generates a rapid prediction of a motor command and a delayed copy of the prediction to match the actual feedback.

Paulin on the other hand theorized that the cerebellum is fundamentally a simple sensory processing organ [40]. As such, he viewed the cerebellum as an analogue to either a Bayesian state estimator [41] or a specific subset, the Kalman filter [42][43]. This theory breaks what would normally be extended smooth motions with many DOF into smaller subsections. It also would suggest that models of the cerebellum could be derived from adaptive arrays [44].

Kawato later proposed a feedback-error-learning (FEL) model of the cerebellum [45][46][47]. The feedback-error-learning model proposed that the cerebellum learns actual internal models of the propriooskeleto-muscular structure as well as the environment to be used in optimal control. Kawato and Gomi devised a generalized model of the cerebellum for each of its four major subsections, shown in figure 2.5 [48]. The model for each subsection is fundamentally the same, explaining how different parts of the cerebellum are able to adapt in cases of brain damage. The FEL model
has been used in a few robotic applications. Miyamoto et al. used the model for trajectory control of a 3-DOF PUMA manipulator [49] and Katayama and Kawato applied the controller to a 5-DOF rubber-actuator-arm [50]. Shibata and Schaal later applied the FEL to a humanoid robot face to emulate the VOR using a recursive least squares training law [51]. Some current investigations focus on using different feedback learning algorithms to optimize the process [52], testing different network structures [53] and applying the FEL model to adaptive control [54].

### 2.4 Pan-Tilt Controllers

Most in-use pan-tilt camera controllers model the camera mount as a manipulator, and use simple feedback PID of PD controllers. There have been few attempts to create reactive feedforward pan-tilt controllers that use VOR as the design basis. Park et al developed such a feed forward controller, the controller was not adaptive and used visual information as feedback to correct the error [55]. As such, the feedforward signal is not correcting for errors prior to them affecting the system, however it was shown to be quite successful.
Nakamura et al. used a variation of Kawato’s model to build a feedforward tracking algorithm for stationary security camera’s [56]. During their experimentation, it was show that the feedforward controller provided a better solution than the feedback controller within the context of reactive control. This was because the internal learned model was eventually able to become accurate enough to properly estimate the system dynamics.

Panerai et al. used VOR as the basis for more complex tracking systems using camera system that modelled the human head. They were able to show the geometric relationship between cameras for both monocular and binocular vision, as well as applying a form of adaptive gain control [57][58] and eventually the addition on neural control [59].

## 2.5 Neural Networks

Most studies model the structure of the cerebellum as an artificial neural network (ANN). An ANN can be best described as a function mapper or emulator. The neural network uses a series on non-linear computations to replicate a non-linear function of varying degrees of complexity. ANNs were originally designed by McCulloch and Pitts in 1943 [60] to model how the human brain performs complex calculations. While the original network was simply a forward multilayer perception (MLP) network, neural networks have seen a lot of development expanding into different types of networks, structures and learning algorithms. A collection of major papers highlighting the development of neural networks has been assembled and published by Anderson and Rosenfeld [61]. The following section reviews the basic theory of the neural network, different structures and network types that exist and learning strategies that are used to train the network weights.
2.5.1 Neuron Model

The most fundamental concept in neural networks is the neuron. The network itself can be summarized as a collection of neurons and the primary difference in the different network structures is how the neurons are connected together. Each neuron has a set number other neurons connected to it. Each neuron in the previous layer inputting to the current neuron has both an output value and a weight associated with the neuron it is feeding into. The weighted output of each input neuron is summed at the current neuron and added to a bias specific to the current neuron. This sum is known as the neuron activation energy.

The output of a neuron is determined by the activation energy of a neuron and the network activation function. The activation function scales the activation energy and outputs the scaled binary output (between $[0,1]$).

$$g(x) = \frac{1}{1 + e^{-Cx}}$$ (2.18)

In this case $g$ is the activation energy of the neuron, $C$ is the neuron scaler and $x$ is the neuron input. Some functions perform better with a bipolar activation function (between $[-1,1]$). In these cases, a hyperbolic tangent function is used.
Figure 2.7: Neural network structure: Common forward propagation neural network structure. Each neuron is connected to each neuron in the following layer until the outputs are found.

\[ g(x) = \tanh(Cx) = \frac{1 - e^{-2Cx}}{1 + e^{-2Cx}} \quad (2.19) \]

### 2.5.2 Network Structure

The most common structure for a neural network is the multilayer perception (MLP). The MLP neural network has an input layer, one or more hidden layers, and an output layer; it has been shown that one hidden layer of sufficient neurons is able to replicate any function [62]. Each hidden layer consists of a defined number of neurons selected to optimize the capabilities of the designed network. The number of neurons in the input layer and the output layer are constrained by the number of system inputs and outputs. Each neuron in the current layer is connected to every neuron in the following layer.

Since each neuron has an associated bias, output and weight array, once the input
layer is specified, the network can quickly calculate the resulting output. The MLP neural network with a single hidden layer can be summarize into a single convenient equation.

\[ y = W_2 \tanh(W_1 x + b_1) + b_2 \]  

(2.20)

Where \( W_1 \) the vector of weights from the input layer to the hidden layer, \( W_2 \) is the vector of weights from the hidden layer to the output layer, \( b_1 \) are the biases for the hidden layer and \( b_2 \) are the biases for the output layer.

Other network structures may also be used depending on application. Recurrent neural networks feed neuron outputs back onto themselves or previous layers. These feedback loops act as a memory for time variant systems. Modular neural networks do not connect every neuron in the previous layer to every neuron in the current layer, but a subset. This thesis only uses feedforward MLP ANNs.

### 2.5.3 Learning Strategies

Tuning the weights and biases of a neural network to proper values allows the ANN to approximate functions, which is what makes neural networks so attractive. The process of selecting weights and biases is known as training the network. There are three learning paradigms for neural networks, supervised learning, unsupervised learning and reinforcement learning.

Supervised learning models are provided with learning signals consisting of both the inputs and the desired outputs. The network then processes the results with the same input and compares them to the desired output. This error vector is then propagated back through the network and adjusts its weights. Most algorithms break the training data into training, validation and testing set and the process is repeated
over and over, tweaking the weights until it best matches the validation and testing data sets. With most supervised learning algorithms, once the network has finished learning, the weights become frozen and are no longer tweaked. The most common supervised learning algorithm is the back propagation algorithm [63], in which the weights of the network are updated by propagating the desired results backwards through the network.

In unsupervised learning, the network is only presented with the inputs and not the desired outputs. In this case, the network itself must decide what features it will use to adjust and group the network inputs. This is often referred to as self-organization or adaptation. Unsupervised learning currently is not well understood, however it opens the possibility of continuous learning algorithms where networks are able to continuously adapt to new situations and new environments.

Finally, reinforcement learning does not provide a training data set but generates its own signals through interactions with its environment. The environment is modelled as a Markov decision process (MDP) in which at each point in time, the agent performs an action and the environment generates an observation and a cost. The goal of the learning paradigm is to perform its actions through minimizing the long term cost.

\section*{2.6 State Estimation and Filtering}

Neural network training can be regarded as a state estimation problem. The state of an ANN in this case is its weights and biases. State estimation is a widely studied topic with many different algorithms available. State estimators in the purest sense provide an estimate of the state of a system that is subject to noise and disturbances. One state estimation algorithm for linear systems that are subjected to additive Gaussian white noise is the Kalman filter [64], which is a recursive Bayesian
filter for multivariate Gaussian distributions.

### 2.6.1 Kalman Filters

The Kalman filter provides a recursive solution to the state-space linear system filtering problem. It is recursive in that, at every time step, it estimates the state using only the previous state estimation and the new input. Since the Kalman filter is a function of only the last time step, it is only that state which needs to be stored, and not the entire history.

Consider a linear, discrete time system. The state vector $x_k$ is defined as the minimal set of data that uniquely describes the unforced behaviour of the discrete time system at time step $k$ and $y_k$ is some measurement of the state, related by the matrix $H_k$. The state of the system as the following time step $k+1$ can be found by propagating the current state through the model transition matrix $F_k$ and adding the influence of the user input signal $B_k u_k$. The system can then be defined by: [64]

\[
\begin{align*}
    x_k &= F_k x_{k-1} + B_k u_k + w_k \\
    y_k &= H_k x_k + v_k
\end{align*}
\]  

where $w_k$ and $v_k$ are process and measurement Gaussian white noise with mean 0 and covariance $Q$ and $R$. That is to say $w_k \sim N(0, Q)$ and $v_k \sim N(0, R)$. The best estimate of the state combines information about the model or the predicted state and data taken from sensors measuring the state or part of the state. The filtering takes place over two distinct phases, the prediction phase and update phase.

The prediction phase estimates the current state $x_{k|k-1}$ based on the dynamic model of the system and the previous state $x_{k-1|k-1}$. It also uses the uncertainty of the model to determine the covariance of the state $P_{k|k+1}$ based on the previous
covariance estimate $P_{k-1|k-1}$ and model noise $Q$.

$$x_{k|k-1} = F_k x_{k-1|k-1} + B_k u_k \quad (2.23)$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q \quad (2.24)$$

The predicted state $x_{k|k-1}$ and predicted covariance $P_{k|k-1}$ are then updated to a more precise estimate of the state $x_{k|k}$ and covariance $P_{k|k}$ using sensor measurements of the system $y_k$. To simplify the update, the Kalman Gain $K_k$ is usually defined first. The update phase can be summarized in three steps.

$$K_k = P_{k|k-1} H_k^T [H_k P_{k|k-1} H_k^T + R]^{-1} \quad (2.25)$$

$$x_{k|k} = x_{k|k-1} + K_k (y_k - H_k x_{k|k-1}) \quad (2.26)$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \quad (2.27)$$

The Kalman gain acts as a weighting system, allowing more emphasis to be put on either the model or measurements, depending on which is giving a better estimate of the system.

### 2.6.2 Extended Kalman Filters

Since most engineering systems are non-linear, the Kalman filter cannot be used as is. To estimate the state of a non-linear system, the extended Kalman filter (EKF) was developed. The EKF attempts to linearize the non-linear system through a first-order Taylor series expansion. This method defines the state space model $F_k$ as the first order approximation of the process function $f$ and the measurement model $H_k$ as the first order approximation of the measurement function $h$. The state equations can
then be written.

\begin{align}
  x_k &= f(\hat{x}_{k-1}, u_k) + w_k \tag{2.28} \\
  y_k &= h(x_k) + v_k \tag{2.29}
\end{align}

Using the approximations, the same process is followed for the EKF as was followed for the Kalman Filter. First a prediction of the state is made based on the system model.

\begin{align}
  x_{k|k-1} &= f(x_{k-1|k-1}, u_k) + w_k \tag{2.30} \\
  P_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + Q \tag{2.31}
\end{align}
After the state has been predicted, it is updated in the update phase with measurements of the state.

\[
K_k = P_{k|k-1}H_k^T [H_k P_{k|k-1}H_k^T + R]^{-1}
\]

\[
x_{k|k} = x_{k|k-1} + K_k(y_k - h(x_{k|k-1}))
\]

\[
P_{k|k} = (I - K_kH_k)P_{k|k-1}
\]

The EKF provides an estimation of the non-linear system. The EKF procedure can be seen in figure 2.8.

2.7 Barrett Whole Arm Manipulator (WAM)

The Whole Arm Manipulator (WAM) was developed by the Massachusetts Institute of technology (MIT) in 1990. It has since been taken over and further developed by Barrett Technology Incorporated. The arm is unique in that it is the only manipulator with direct-drive motors and joints thanks to its cable driven technology, making its joint torque control more stable, reduces backlash and limits the effects of friction. The WAM arm is highly back drivable and made to be able to control contact forces, enables force control, haptics hybrid control and tele-operation.

The Barrett WAM will be used as the platform to test the designed controller, both in simulation and in hardware. The developed controller will be based on the published DH parameters, found by following the DH convention shown above. The individual joint reference frames can be defined as shown in figure 2.9 [65]. The DH parameters for the Barrett WAM are listed in table 2.1 and the dynamic parameters are listed in table 2.2.
Table 2.1: Barrett WAM DH Parameters: Published kinematic parameters for the Barrett WAM

<table>
<thead>
<tr>
<th>Joint i</th>
<th>Mass (kg)</th>
<th>( \theta_i ) (rad)</th>
<th>( d_i ) (m)</th>
<th>( a_i ) (m)</th>
<th>( \alpha_i ) (rad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.77</td>
<td>( \theta_1 )</td>
<td>0</td>
<td>0</td>
<td>(-\frac{\pi}{2})</td>
</tr>
<tr>
<td>2</td>
<td>3.87</td>
<td>( \theta_2 )</td>
<td>0</td>
<td>0</td>
<td>(\frac{\pi}{2})</td>
</tr>
<tr>
<td>3</td>
<td>1.80</td>
<td>( \theta_3 )</td>
<td>0.55</td>
<td>0.045</td>
<td>(-\frac{\pi}{2})</td>
</tr>
<tr>
<td>4</td>
<td>2.40</td>
<td>( \theta_4 )</td>
<td>0</td>
<td>-0.045</td>
<td>(\frac{\pi}{2})</td>
</tr>
<tr>
<td>5</td>
<td>0.12</td>
<td>( \theta_5 )</td>
<td>0.3</td>
<td>0</td>
<td>(-\frac{\pi}{2})</td>
</tr>
<tr>
<td>6</td>
<td>0.42</td>
<td>( \theta_6 )</td>
<td>0</td>
<td>0</td>
<td>(\frac{\pi}{2})</td>
</tr>
<tr>
<td>7</td>
<td>0.07</td>
<td>( \theta_7 )</td>
<td>0.06</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.2: Barrett WAM Dynamic Parameters: Published dynamic parameters for the Barrett WAM

<table>
<thead>
<tr>
<th>Joint i</th>
<th>Center of Mass (m)</th>
<th>Inertia (kg·m²)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( r_{i,x} )</td>
<td>( r_{i,y} )</td>
</tr>
<tr>
<td>1</td>
<td>-0.0044</td>
<td>0.1219</td>
</tr>
<tr>
<td>2</td>
<td>-0.0024</td>
<td>0.0311</td>
</tr>
<tr>
<td>3</td>
<td>0.0067</td>
<td>-0.3425</td>
</tr>
<tr>
<td>4</td>
<td>-0.0400</td>
<td>-0.0002</td>
</tr>
<tr>
<td>5</td>
<td>0.0001</td>
<td>-0.2949</td>
</tr>
<tr>
<td>6</td>
<td>-0.0001</td>
<td>-0.0170</td>
</tr>
<tr>
<td>7</td>
<td>-0.0001</td>
<td>0.0002</td>
</tr>
</tbody>
</table>
Figure 2.9: Barrett WAM: DH convention defined joint coordinate system for the Barrett WAM. All seven joints are defined. The last three are used in the control algorithms.
Chapter 3

Kinematic Model

The pan-tilt camera controller will be based on the model presented by Kawato and Gomi. The controller uses a neural network to predict the required joint torques of the pan-tilt to create the trajectory need to track objects in the visual field. The dynamics are primarily a function of the joint trajectory ie: the joint angles $\theta$, joint velocity $\dot{\theta}$ and joint acceleration $\ddot{\theta}$. These values are the inputs to the neural network. The required joint trajectory is defined by the pan-tilt kinematics. The following chapter defines the kinematic model for the designed controller of a pan-tilt camera mount. The kinematic model is generated using only the DH parameters of the pan-tilt and can therefore be generalized to fit each individual case or rover platform. The kinematic model of the pan-tilt is required to calculate the desired joint space trajectory based on the measured motion of the rover body via the internal measurement unit (IMU) in order to allow the camera to track objects in the visual field. The perturbations are measured at the rover base, not on the camera. Since the testing platform is the 3-DOF wrist of the Barrett WAM as opposed to a 2-DOF pan-tilt, both solutions will be presented. It can be assumed that if the controller can adapt to the more complex Barrett WAM, it will also be able to adapt to the simpler 2-DOF pan-tilt.
Figure 3.1: Rover Coordinate System: Selection of reference frame orientations and Cartesian vectors used to define the rover and pan-tilt system during traverse.
3.1 Pan-Tilt

Perturbations on the pan-tilt camera are not being directly measured and therefore they need to be inferred from the motion measured at the base of the rover by the IMU. In order to simplify the system, the pan-tilt is kinematically modelled as a 2-DOF manipulator. As such, its mathematical model can further be broken into two sub parts, the kinematics and differential kinematics.

The vector of desired joint angles \( \theta \) is a vector of two joint angles, representing the pan and tilt of the camera. The orientations of the base, rover and pan-tilt coordinate system can be defined as shown in figure 3.1 and figure 3.2. Then desired orientation of the pan-tilt system can be represented by a rotation matrix \( R \) in the form of the following.

\[
R = \begin{bmatrix}
    x_x & y_x & z_x \\
    x_y & y_y & z_y \\
    x_z & y_z & z_z 
\end{bmatrix}
\]  

\[ (3.1) \]
This rotation matrix represents the changes in orientation from the base coordinate system to that of the camera. The $x$, $y$ and $z$ columns of $R$ represent the normalized vectors in base coordinates that make up the axis of the camera coordinate system. Since the $x$ axis of the camera orientation was defined to be the viewing direction of the camera, then the normalized vector from the camera to the target in base coordinates $p_t$ becomes the $x$ column of the rotation matrix $R$. The vector $p_t$ can be found from known vectors from the origin to the target $p_o$, from the origin to the rover $p_r$ and from the rover to the camera $p_p$, all in base coordinates.

$$p_t = p_o - p_r - p_p \quad (3.2)$$

Since $R$ represents the desired orientation of the camera in base coordinates, the pan-tilt must correct the random motions of the rover body to give this desired final orientation. If $R_o$ is defined as the rotation matrix of the rover body in base coordinates, and $R_p$ as the rotation matrix of the camera in rover coordinates, then total rotation matrix $R$ can be defined as.

$$R = R_o^r R_p^r \quad (3.3)$$

Pan-tilts are designed to be essentially spherical, and therefore the translation and orientation can be separated. $R_o^r$ acts as the random translation/rotations and $R_p^r$ is a function of the pan-tilt joint angles alone which can act to correct the orientation due to the errors created by $R_o^r$. The rotation matrix $R_p^r$ can be isolated and used to solve for the required joint angles $\theta$ that will give the desired orientation.
\[ R_p^o = R^{o T}_o R \]  \hspace{1cm} (3.4)

Since the pan-tilt is being modelled as a manipulator, the motions from the two joints \( \theta_1 \) and \( \theta_2 \) must yield to a rotation matrix \( R_p^o \) in the following form, given following the DH multiplication of a 2-DOF manipulator.

\[
R_p^o = \begin{bmatrix}
\cos(\theta_1) \cos(\theta_2) & -\sin(\theta_1) & \cos(\theta_1) \sin(\theta_2) \\
\sin(\theta_1) \cos(\theta_2) & \cos(\theta_1) & \sin(\theta_1) \sin(\theta_2) \\
-\sin(\theta_2) & 0 & \cos(\theta_2)
\end{bmatrix}
\hspace{1cm} (3.5)
\]

It can be seen that the joint angles \( \theta_1 \) and \( \theta_2 \) can be solved for using only the \( x \) column of matrix \( R_p^o \). This simplifies the problem as now the \( y \) and \( z \) columns of \( R \) do not have to be defined in order to solve for the joint angles \( \theta_1 \) and \( \theta_2 \). This also means that the orientation of the camera cannot be controlled, only the viewing direction for the 2-DOF pan-tilt. This naturally makes sense as 2-DOF is under defined to control a 3-DOF world orientation. The joint angles \( \theta_1 \) and \( \theta_2 \) can be found by solving the \( x \) column of \( R_p^o \).

\[
\theta_1 = \arctan\left(\frac{x_y}{x_x}\right) \hspace{1cm} (3.6)
\]

\[
\theta_2 = \arctan\left(\frac{\sqrt{x_x^2 + x_y^2}}{-x_z}\right) \hspace{1cm} (3.7)
\]

This solution forces the joint angle \( \theta_1 \) to remain between \( -\pi : \pi \) and joint angle \( \theta_2 \) to
remain between 0 : π. This however does not limit the field of view, as the constraint in the second joint overlaps the region covered by the first joint.

The joint angular velocity \( \dot{\theta} \) and acceleration \( \ddot{\theta} \) are found by differentiating the joint position and velocity. If the correction in orientation is to take place in a single time step \( dt \), then the required joint velocity and joint acceleration can be found as.

\[
\dot{\theta}_i = \frac{\theta_i - \theta_{i-1}}{dt} \quad (3.8)
\]
\[
\ddot{\theta}_i = \frac{\dot{\theta}_i - \dot{\theta}_{i-1}}{dt} \quad (3.9)
\]

### 3.2 The Barrett WAM

The hardware testing for the pan-tilt controller will be performed using the Barrett WAM. The base frame for the manipulator will model the base frame in the rover application. The first four joints will be used to model inputs to the system; in the case of the rover/pan-tilt these represent the perturbations on the rover as it is moving. The orientation of the fourth link will represent the orientation of the rover body \( R^r_o \) and the position of the fifth link represents the position of the pan-tilt camera on the rover body \( p_r + p_o \). Finally the last three joints model the camera pan-tilt itself. In this case, and extra degree of freedom was used to show the robustness of the system, as the solution becomes simpler as the DOF decreases.

In the case of the Barrett WAM, the mathematical model used varied slightly from that of a pan tilt, namely as there is an extra joint to solve for and will allow for an extra constraint. The addition joint also changes the orientation of the camera. With the extra joint, it is not the \( z \) axis of the camera frame that represents its viewing direction and therefore the vector \( p_t \) now is the \( z \) column of the rotation.

The full rotation matrix in rover coordinates \( R^p_r \) for the 3-DOF WAM wrist end
The effector is.

\[
R^e_o = \begin{bmatrix}
  c(\theta_1)c(\theta_2)c(\theta_3) - s(\theta_1)s(\theta_3) & -s(\theta_1)c(\theta_3) - c(\theta_1)c(\theta_2)s(\theta_3) & c(\theta_1)s(\theta_2) \\
  s(\theta_1)c(\theta_2)c(\theta_3) + c(\theta_1)s(\theta_3) & c(\theta_1)c(\theta_3) - s(\theta_1)c(\theta_2)s(\theta_3) & s(\theta_1)s(\theta_2) \\
  -s(\theta_2)c(\theta_3) & s(\theta_2)s(\theta_3) & c(\theta_2)
\end{bmatrix}
\] (3.10)

The joint angles \(\theta_1\), \(\theta_2\) and \(\theta_3\) can then be found.

\[
\theta_1 = \arctan\left(\frac{z_y}{z_x}\right) \tag{3.11}
\]

\[
\theta_2 = \arctan\left(\frac{\sqrt{z_x^2 + z_y^2}}{z_z}\right) \tag{3.12}
\]

\[
\theta_3 = \arctan\left(\frac{y_z}{-x_z}\right) \tag{3.13}
\]

Since the orientation can now be fully controlled, in order to solve for \(\theta_3\), the \(x\) and \(y\) columns of \(R^e_o\) need to be known, and by extension, the \(x\) and \(y\) columns of \(R\) now have to be solved as well. The remaining two columns can be found using the fundamental property of rotation matrices, \(R^T R = R R^T = I\), where \(I\) is the identity matrix. An additional constraint can also be added to the rotation matrix \(R\) as well to control both the looking direction and camera orientation. In this application, the variable \(y_z\) will be set to 0, forcing the camera to remain level through the trajectory. Solving for the other 6 components of the rotation matrix \(R\) give:
The differential kinematics can be solved in the same manner presented in the previous section, through differentiation.

\[
\dot{\theta}_i = \frac{\theta_i - \theta_{i-1}}{dt} \tag{3.20}
\]
\[
\ddot{\theta}_i = \frac{\dot{\theta}_i - \dot{\theta}_{i-1}}{dt} \tag{3.21}
\]
Chapter 4

Feedforward Control Applied in Simulation

The problem of pan-tilt dynamic control has been shown to be highly non-linear and complex [66]. The most successful systems are high-computational algorithms with feedback correction loops. These systems estimate the required joint torques to generate the desired joint trajectory. Since the dynamic model of the pan-tilt cannot adequately compensate for all forces acting on the system, ie there will always be terms that are not properly accounted for in the estimations due to the sheer complexity and number of factors. These errors are commonly corrected for using either PD or PID control schemes. In the standard feedback PID controller, a control signal acts on the error in trajectory that occurred in a previous time step. In this way, the system does not attempt to predict unknown dynamic terms during the motion, but instead attempt to compensate for them after they have already caused errors in the system. Feedback control essentially waits for the errors to occur, instead of trying to prevent them from occurring. Feedback control is also limited in that these systems are not able to correct position in a single time-step, as it does not have enough time to build its error vector and begin to try and roughly correct its position, limiting its usefulness as a reactive controller.
A feedforward controller on the other hand would be able to predict the unmodelled terms and compensate for their effects before they happen. Neural networks are well known for emulating complex non-linear functions [67], and therefore are a common branch in feed forward control. They are able to be designed and trained in a variety of ways to cater to specific needs, and are able to perform function mapping with relatively few computational resources. This chapter outlines the designed neural network, its desired performance and designed structure to determine its suitability in this type of control application.

4.1 Controller

This section outlines the details of the designed controller and the verification parameters that will be used to analyse the network capabilities. The controller will be required to estimate the required joint torque commands for a pan-tilt system to track salient objects in the visual field, similar to how it is done in the human cerebellum. Since the pan-tilt is being simulated using the Barrett WAM, the controller will be designed to control the last three joints of the WAM. The controller will also be designed an extension of the feedback-error-learning model of the cerebellum proposed by Kawato. This type of model will allow the controller to continuously learn its own dynamics, every converging towards a better solution and allowing the controller to adapt for potential changes or estimation errors in the working environment. In Kawato’s model, he used a PID feedback control signal to train the network. The controller designed for this thesis will also use a feedback signal however the training algorithm used will instead be an extended Kalman filter. The Kalman filter algorithm will add robustness to the system by allowing the controller to put more or less emphasis on either the current network weights or the current training data set. This will allow the system to be more stable as the changes in networks weights will
be more controlled during use.

4.1.1 Characteristic Behaviour

The desired behaviour from the neural network will be to emulate the vestibulo-ocular reflex seen in the human cerebellum to track pre-defined salient targets in the visual field during the traverse phase of planetary rover missions. Using the known required joint trajectory defined in Chapter 3, the network will predict the required joint torques needed for a pan-tilt camera mount, simulated by the Barrett WAM, to quickly and accurately track the target. Since the dynamic model of the pan-tilt is a function of its orientation and trajectory $\theta$, $\dot{\theta}$ and $\ddot{\theta}$, these kinematic properties will serve as the network inputs. The network will output the joint space torque commands, and the pre-built software in the arm will convert these commands to electrical commands. Once the arm has moved according to the predicted torques, the actual motion will be measured. An EKF will then merge the new training data set with the existing network weights to give a new estimate of the proper network weights [68] [69].

4.1.2 Neural Network Structure

Growing and pruning a MLP neural network has previously been explored [70] and it has also been shown that a single hidden layer will adequately model any system. It is well known that more complex systems require more hidden neurons for the network to converge. One the other hand, as the number of hidden neurons increases, so the number of states and computational time to propagate the network. This is especially true given that the training algorithm is using an EKF, meaning the number of mathematical operations during training increases exponentially. Larger networks can also be prone to local minima during the training which can prevent
Figure 4.1: Feedforward Controller: Structure of the designed feedforward controller

the network from generalizing to the system as a whole. For this reason, the network must contain a minimum number of hidden neurons to accurately represent the non-linear system. To limit the number of hidden neurons, a conservative quantity will be initially selected and slowly increased until the network is able to adequately converge. The overall network structure for the simulated pan-tilt will be a 9-input (θ, ˙θ and ¨θ for each joint), 1-layer n-neuron, 3-output (τ for each joint) neural network trained by an EKF. The overall control scheme can be seen in Figure 4.1.

4.1.3 Model Verification

The accuracy and precision of the model has to be verified in order to quantify whether the neural network solution adequately models the pan-tilt dynamics and can be used in this form of control. The performance of a neural network is analysed using two parameters, the coefficient of determination $R^2$ [71] and the mean square error MSE [72]. In the simulation, the neural network will be trained to model the Newton-Euler manipulator dynamic equations. In this case, the Newton-Euler dynamics represent the $x$ axis of the regression and the network output will be the $y$ axis. Therefore the desired network performance is for a $x = y$ diagonal fit.
The coefficient of determination indicates how well data points fit a statistical model. In the case of a neural network, the $R^2$ value relates how well the output of the neural network maps to the desired output. The $R^2$ value is a measure of the accuracy of the neural network. Mathematically, it can be described as the following.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$ (4.1)

Where $y_i$ is the output of the neural network, $\hat{y}_i$ is the desired response, $\bar{y}$ is the mean value of $y_i$ and $n$ is the number of data points used. The best possible fit has a $R^2$ value of 1 meaning the network perfectly fits the data. A very poor fit has a $R^2$ value of 0 meaning the network is unable to model the data.

The second neural network analysis technique to be used is the mean square error. The mean square error is a measure of the precision of the network output and describes how close the network output values are to the desired values. This differs from the $R^2$ measurement in that the $R^2$ is really a rescaling of the MSE. The MSE is a measure of the network precision and mathematically be described as the following.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$ (4.2)

With the MSE parameter, the smaller the number value the better the network estimations. However, since MSE is a relative value, a mean value is needed for comparison. Through simulation of the arm, it was found that the mean square torque command for the wrist of the Barrett WAM is $3.7400 \times 10^{-3}$, which is the value that will be used to quantify the MSE by acting as a reference value in which to compare the errors.
4.2 EKF Training Algorithm

The desired response of the network is to have it predict the required joint spaces torque to have the camera pan-tilt follow a random set of input trajectories. To build the set of training data, a series of random joint angles, velocities and accelerations were propagated through the Newton-Euler dynamic calculations for the last three joints of the Barrett WAM manipulator. These torques had Gaussian white noise added to them to model inaccuracies in the model and joint space kinematics. There is a desire for the cerebellar-like controller to learn actively and still be robust and stable; therefore the most suitable learning algorithm is the EKF learning model.[73][74]

If the vector $w_k$ contains the weights and biases of the neural network, then it can also be defined as the state of the system. In this way, the output of the neural network itself is a function of the system state, and can therefore be considered a measurement of the state, meaning the network itself can be considered to be the measurement model. Therefore, at each time step, the weights of the system can be updated based on a correct signal generated by the Newton-Euler dynamics using the EKF procedure.

The process equation, or the state of an EKF is defined by the following.

$$w_k = f(w_{k|k-1}, u_k) + q_k$$  \hspace{1cm} (4.3)

Where $w_{k|k-1}$ is the predicted state at time $k$, $u_k$ represents the user input to the system and $q_k$ is the model noise normally distributed about 0 with covariance $Q$. The function $f$ is the process function that transforms the state from time step $k - 1$ to time step $k$ denoted by $w_{k-1|k-1}$. Since the weights of a neural network are not a function of time or are affected by user input, the process equation can be simplified.
\[ w_k = w_{k-1} + q_k \] (4.4)

The measurement equation of an EKF is defined by the following.

\[ y_k = b(w_k) + r_k \] (4.5)

Where \( y_k \) is the measured state at time \( k \) and \( r_k \) is the measurement noise normally distributed about 0 with covariance \( R \). However, since the neural network is being trained to match an output and it is a least-squares estimate, the measurement \( y_k \) will be replaced with the training signal corresponding to the training data. The measurement model \( b \) is represented by the neural network itself. A neural network can be written as the following.

\[ b = w_2 \tanh(w_1 x + w^0_1) + w^0_2 \] (4.6)

The weights \( w_{k|k-1} \) can be broken into four components, \( w_1, w_2, w^0_1 \) and \( w^0_2 \) depending on their location in the neural network. \( w_1 \) are the weights acting from the input layer to the hidden layer, \( w^0_1 \) are the biases of the hidden layer, \( w_2 \) are the weights acting from the hidden layer to the output layer and \( w^0_2 \) is the bias of the output layer. The input to the network is defined by \( x \).

At this point, the EKF theory can be applied to update the network weights. The first phase determines the predicted state \( w_{k|k-1} \) and the predicted covariance \( P_{k|k-1} \).
\[ w_{k|k-1} = F_{k-1|k-1}w_{k-1|k-1} \quad (4.7) \]
\[ P_{k|k-1} = F_{k-1|k-1}P_{k-1|k-1}F_{k-1|k-1}^T + Q \quad (4.8) \]

Since there is no process function for a neural network, the process Jacobian \( F_{k-1|k-1} \) can be found by differentiating the state vector by itself, giving the identity matrix \( I \). This simplifies the prediction to the following.

\[ w_{k|k-1} = w_{k-1|k-1} \quad (4.9) \]
\[ P_{k|k-1} = P_{k-1|k-1} + Q \quad (4.10) \]

The second phase of the training algorithm, the update step, can be calculated at this point as well. First the Kalman gain is computed.

\[ K_k = P_{k|k-1}B_k^T (B_kP_{k|k-1}B_k^T + R)^{-1} \quad (4.11) \]

The Jacobian of the measurement equation or neural network \( B_k \), with respect to the weights \( w \) can be broken into four equations depending on the location of the weights and biases in the network. The Jacobian of each of the four elements \( w_1 \), \( w_2 \), \( w^0_1 \) and \( w^0_2 \) can be found as the following.
\[
\frac{\partial b}{\partial w_1} = w_2x(1 - \tanh^2(w_1x + w_0^0)) \quad (4.12)
\]
\[
\frac{\partial b}{\partial w_2} = \tanh(w_1x + w_0^1) \quad (4.13)
\]
\[
\frac{\partial b}{\partial w_1^0} = w_2(1 - \tanh^2(w_1x + w_0^1)) \quad (4.14)
\]
\[
\frac{\partial b}{\partial w_2^0} = 1 \quad (4.15)
\]

Finally the updated weights \(w_{k|k}\) and covariance \(P_{k|k}\) can be determined based on the desired output training signal \(y_k\) and current output \(b(w_{k|k-1})\) of the neural network.

\[
w_{k|k} = w_{k|k-1} + K_k(y_k - b(w_{k|k-1})) \quad (4.16)
\]
\[
P_{k|k} = P_{k|k-1} - K_kP_{k|k-1}B_k^T \quad (4.17)
\]

The advantage of the EKF training algorithm is that it is able to be updated actively and does not require an entire training set of data, but is able to amend the current training set to the system though he current covariance estimate. It also has parameters that can be adjusted to control how fast the system learns, primarily the model and measurement noise \(Q\) and \(R\). A high \(Q\) and low \(R\) would put more emphasis on the training data and less on keeping the current state, forcing the change in the weights to be higher. On the other hand, once the network has been trained, a lower \(Q\) and higher \(R\) puts more value to the current state of the network and prevents it from varying to a large degree, making the network more stable.
4.3 Back Propagation

The back propagation learning algorithm is a standard and widely accepted and used training algorithm for neural networks. There are a few different varieties of the back propagation algorithm that each has their own merits and disadvantages. In the applications of the Kawato FEL model, an online gradient decent moment algorithm (BPM) was used [75]. On the other hand, this model is not as efficient as some of the other structures. The Levenberg-Marquardt Backpropagation (LMB) on the other hand is known to be one of the fastest back propagation training laws with very accurate results. This algorithm cannot be used online and can only be used in batch applications. The back propagation learning algorithm uses a large set of training data which it propagates through the network in reverse to calculate a weight gradient, which it then uses to update the weights in the network.

4.3.1 Back Propagation with Momentum

The algorithm begins by defining the sum square error $E$ to evaluate the training process.

$$E(x_k, w_k) = \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} (y_{n,m} - b(w_{n,m}))^2$$

(4.18)

Where $N$ is the number of training samples, $M$ is the number of network outputs, $y$ is the desired output and $b(w_{n,m})$ is the current output. The gradient vector $g$ is the first-order derivative of the total error $E$ with respect to the current network weights $w_k$.

$$g = \frac{\partial E(x_k, w_k)}{\partial w_k}$$

(4.19)
The training law is then defined by following the error gradient.

\[ w_{k+1} = w_k - \eta g + \alpha \delta w_k \]  

(4.20)

Where \( \eta \) is the learning rate and \( \alpha \) is the momentum factor. The BPM algorithm will be used to test the EKF learning algorithm in terms of online learning.

### 4.3.2 Levenberg-Marquardt Backpropagation

The LMB learning algorithm is a derivative of Newton’s Method and the Gauss-Newton Algorithm. Newton’s Method assumes that all gradient components \( g_n \) are functions of the weights \( w_k \) and are linearly independent. This gives rise to the relation.

\[ g = \frac{\partial E(x_k, w_k)}{\partial w_k} \]  

(4.21)

\[ \frac{\partial g}{\partial w_k} = -\frac{\partial^2 E(x_k, w_k)}{\partial w_k^2} \delta w \]  

(4.22)

The second-derivative of the total error is defined as the Hessian matrix \( H \) and is highly complex and difficult to calculate. However, if it is known, the gradient of the weights \( \delta w \) can be found and used to update the network weights.

\[ \delta w = -H^{-1} g \]  

(4.23)

The Gauss-Newton algorithm introduces an approximation to the Hessian matrix and gradient calculation by defining an easier to calculate Jacobian.
The approximated Hessian matrix and gradient vector can then be defined as

\[ g = J e \]  \hspace{1cm} (4.25) \]
\[ H = J^T J \]  \hspace{1cm} (4.26) \]

and the network weight gradient \( \delta w \) is now

\[ \delta w = -(J_k^T J_k)^{-1} J_k e \]  \hspace{1cm} (4.27) \]

Up to this point, there is nothing forcing the Hessian matrix to be invertible, and therefore stable. The Levenberg-Marquardt Backpropagation algorithm introduces another term to the approximated Hessian matrix, which insures that it is always invertible and therefore always stable.

\[ H = J_k^T J_k + \mu I \]  \hspace{1cm} (4.28) \]

Where \( \eta \) is the learning rate and \( \alpha \) is the momentum factor.

where \( I \) is the identity matrix and \( \mu \) is called the combination coefficient. The new network weights can now be found as:

\[ w_{k+1} = w_k - (J_k^T J_k + \mu I)^{-1} J_k e_k \]  \hspace{1cm} (4.29) \]
The LMB algorithm can only be used in batch training. Batch training changes the weights only once and converges to the entire set of training data, resulting in very low residuals. The batch trained LMB algorithm is used to test the network performance robustness.

4.4 Simulator

For a manipulator, the joint torques would be applied to motors themselves as voltages and the resulting joint motion would be measured. To test the controller as a whole, a simulated version Barrett WAM was designed to emulate the response of the arm to input joint torques. The forward dynamics will be solved by using the predicted torques from the neural network to find the resulting joint accelerations. The accelerations can be integrated with the known initial conditions to find the resultant joint velocities and position.

Solving for the acceleration from the input torque is not trivial. The most common method is to calculate the Coriolis forces $h$ and gravitational forces $g$ separately, subtract them from the predicted torques and solve for the D inertial matrix, which can be inverted to find the joint accelerations.

$$\ddot{\theta} = D(\theta)^{-1}(\tau - h(\theta, \dot{\theta}) - g(\theta))$$ (4.30)

The inertial matrix $D$ can be found by differentiating the inverse dynamic model by acceleration of each joint which creates a computationally heavy algorithm and fails to take advantage of the properties of the D matrix. A more computationally efficient method to solve the D matrix is to use the composite mass algorithm. In this three-stage recursive algorithm, one joint is isolated at a time to create a 1-DOF
system where all links prior to and after the moving joint are considered 1 link. Each link is analysed to determine its effect on the inertia relative to the moving link. Since the D matrix is symmetric, only half of the D matrix has to be found computationally.

The first recursion starts at the end-effector and defines the kinematic parameters of each single joint system.

Backward Recursion \((i = N : 1)\)

\[
M_i = M_{i+1} + m_i
\]
\[
c_i = \frac{m_i(r_i^{i-1} + r_i^{i}) + M_{i+1}(c_{i+1} + r_{Ci}^i)}{M_i} \tag{4.32}
\]
\[
E_i = E_{i+1} + I_i
\]
\[
+ M_{i+1}[(c_{i+1} + r_{Ci}^i - c_i)^T(c_{i+1} + r_{Ci}^i - c_i)I - (c_{i+1} + r_{Ci}^i - c_i)(c_{i+1} + r_{Ci}^i - c_i)^T] + m_i[(r_i^{i-1} + r_{Ci}^i - c_i)^T(r_i^{i-1} + r_{Ci}^i - c_i)I - (r_i^{i-1} + r_{Ci}^i - c_i)(r_i^{i-1} + r_{Ci}^i - c_i)^T] \tag{4.33}
\]

Where \(m_i\) is the mass of link \(i\), \(M_i\) is the total mass of links \(i : N\), \(c_i\) is the center of mass of links \(i : N\), \(r_i^{i-1}\) is the vector from link \(i - 1\) to \(i\), \(r_{Ci}^i\) is the vector from \(i\) to the center of mass of link \(i\), \(E_i\) is the mass moment of inertia of links \(i : N\), \(I - i\) is the mass moment of inertia of link \(i\) and \(I\) is the identity matrix.

The forward recursion is then able to move from the base to the end-effector defining the forces and moments at each joint.

Forward Recursion \((i = 1 : N)\)

\[
F_i = z_{i-1} \times M_i c_i \tag{4.34}
\]
\[
\Psi_i = E_i z_{i-1} \tag{4.35}
\]
Where $F_i$ is the total force acting on link $i$, $\Psi_i$ is the total moment acting on link $i$ and $z_{i1}$ is the $z$ component of the rotation matrix of link $i$.

Finally a second backward recursion defines the inertial components of each joint.

Backward Recursion ($i = N : 1$, $j = N : 1$)

\begin{align*}
  f_{i,i} &= F_i \\
  f_{i,j} &= f_{i+1,j} \\
  \mu_{i,i} &= \Psi_i + c_i \times F_i \\
  \mu_{i,j} &= \mu_{i+1,j} + r_{i-1}^i \times f_{i,j} \\
  D_{i,j} &= \mu_{i,j} \cdot z_{i-1}
\end{align*}

Where $f_{i,j}$ is the force exerted on link $i$ by link $j$ and $\mu_{i,j}$ is the moment acting on link $i$ by link $j$.

This algorithm is able to quickly replicate inertial matrix $D$ of the Barrett WAM at different orientations. To approximate the torque contribution from just the Coriolis and gravitational forces, the total torque calculation is done a second time, this time by setting the input joint acceleration term to 0, thus cancelling out the inertial component. Once the resulting joint acceleration is calculated, the joint velocity and joint position can be found by integration. The resulting joint trajectory is plotted and the orientation of the pan-tilt camera is highlighted to emphasis which direction it is facing during the move.

Figure 4.2 is the output of the Barrett WAM simulator. The blue line represents the viewing direction of the camera and the green and red lines represents the $y$ and $x$ axis of the pan-tilt reference frames respectively. The purple $x$ represents the target being tracked. The simulator applies random trajectories to the first 4-DOF of the arm, finds the resulting required wrist trajectory, converts it into a torque command,
Figure 4.2: Simulator output: Output of the Barrett WAM simulator to track objects in the visual field. Purple $x$ represents the target and blue axis the camera viewing direction. Units are in m.

back to a trajectory and plots this trajectory. In the prefect system, the blue line representing the viewing direction of the camera will always be intersecting the target $x$ and the green $y$ axis of the pan-tilt will remain parallel to the $x - y$ plane in the base frame.

4.5 Results

This thesis proposes a neural network trained online using an EKF training law to predict the dynamics of a pan-tilt. The results can be broken into four parts: the suitability of a neural network, performance of the EKF training law, stability of the EKF training law and a full system simulation. To give some bench mark for the EKF trained network, 2 other networks were also generated. These networks were built to the same specifications of the EKF trained network. The first was build using
a batch LMB training law. The batch algorithm is the most common neural network
training law and is widely used and accepted to give robust results. This network
will be used to compare the overall performance of the EKF trained network. The
desire is for the EKF trained network to have similar or better performance than the
batch LMB trained network. The second network was trained using an online LMB
algorithm. The online algorithm is an online version of the batch LMB algorithm.
The desire is for the EKF trained network to have a better performance and better
stability than the online LMB network.

4.5.1 Suitability of a Neural Network

The dynamics of a pan-tilt camera mount are highly complex, and therefore a larger
network is needed to cope with the system. The network was initially selected to
be a simple 1-layer 6-neuron forward propagated network. 1000 random trajectories,
each with 50 points were selected to make up the training data. Each trajectory
was propagated through the Newton-Euler dynamic formulation and used to train
the network. The 1000th move was used to compare the neural network output to
the desired output using the $R^2$ algorithm described above. The 1-layer 6-neuron
network had an $R^2$ value of 0.6152 and was unsuccessful in modelling the pan-tilt
dynamics. The number of hidden neurons was then increased and the process was
repeated until the $R^2$ was close to 1 and additional neurons did not significantly
improve the network results. Table 4.1 summarizes the results of the network sizing
process. Both the $R^2$ and MSE were calculated for each point, however the primary
deciding factor on deciding if the network converged was the $R^2$. The MSE is used
later to determine the network suitability in modelling the pan-tilt dynamics.

Table 4.1 demonstrates a subset of the network testing process, in which networks
using different numbers of hidden neurons were trained and tested. It can be seen
that the fewest numbers of neurons needed to adequately mimic the dynamics of
Table 4.1: Neural Network Sizing: Subset of the resulting verification parameters for different size hidden layers. Table was used to select most appropriate neural network size.

<table>
<thead>
<tr>
<th>Neurons</th>
<th>(R^2)</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.6152</td>
<td>1.5012 (\times 10^{-3})</td>
</tr>
<tr>
<td>10</td>
<td>0.7668</td>
<td>8.8760 (\times 10^{-4})</td>
</tr>
<tr>
<td>12</td>
<td>0.9998</td>
<td>7.5530 (\times 10^{-7})</td>
</tr>
<tr>
<td>15</td>
<td>0.9991</td>
<td>3.2601 (\times 10^{-6})</td>
</tr>
<tr>
<td>25</td>
<td>0.9999</td>
<td>2.5097 (\times 10^{-7})</td>
</tr>
</tbody>
</table>

The camera pan-tilt mount is 12. The 1-layer 12-neuron network had a \(R^2\) value of 0.9998, while any fewer neurons drastically reduced the accuracy of the network. On the other hand, since the \(R^2\) value is so close to 1 additional neurons are needless and will only increase the computational requirements of the network. In general, the \(R^2\) value improved as more neurons were added, however the 15-neuron network saw a small dip in the \(R^2\). While this dip is essentially negligible, it can be attributed to the difficulties in training larger neural networks due to the potential of local minima and the difficulties in getting larger networks to converge. The MSE analysis of each network also shows the significant decrease in the mean square error for the 12-neuron network compared to the networks with slightly fewer neurons and a small decrease for networks with additional neurons. The MSE for the 12-neuron network was found to be 7.5530 \(\times 10^{-7}\). Recalling that the mean-square torque command used in the training set was 3.7400 \(\times 10^{-3}\), this means that the average error caused by the network is 0.02\% which can be considered negligible. Therefore, it can be determined that a 1-layer 12-neuron neural network is able to adequately model the dynamics of a 3-DOF pan-tilt controller.
4.5.2 Performance of the EKF Training Law

Figure 4.3 demonstrates the linear regression of the network output torques compared to the desired torques for the optimal 12-neuron structure. The black line represents the desired $x = y$ fit from the network. The green, blue and purple dots represent the joint torques for each of the first, second and third wrist joints respectively. The plotted torques together are relatively linear and are dotted out along the black diagonal, which is expected due to $R^2$ being so close to 1. They are also grouped very close to the line which was expected due to the MSE being so low.

The specific results of the neural network trained with the EKF can be seen in figure 4.4. This figure better represents the performance of the network to specific commands instead of a generalized analysis shown in figure 4.3. Specific joint trajectories are propagated through both the Newton-Euler dynamics and the neural network. The green, blue and purple circles represent the desired torque outputs for the specific trajectory for the first, second and third joints respectively. The light blue, yellow and red dots represent the network predicted torques for the specific move.
Figure 4.4: Results of the EKF trained NN: Resulting torques for Barrett WAM wrist from the EKF trained neural network (dots) using random inputs compared to the standard dynamic equation output (circles).
Figure 4.5: Linear regression of batch back propagated NN: Linear regression of the batch Levenberg-Marquardt Backpropagation trained neural networks. Perfect fit follows the $y = x$ diagonal.

for the first, second and third joints respectively. It can be seen that the network is able to model specific joint torques accurately. The first few plots represent commands during training, while the latter represent trajectories tested once the network has had time to converge.

While the neural network trained by the EKF has been shown to adequately approximate the dynamics of a pan-tilt, another network of the same structure was designed and taught using the Levenberg-Marquardt Backpropagation learning algorithm. This was done to be able to compare the capabilities of the EKF training algorithm by giving it a standard algorithm to be compared to. The back propagated network used the same 1-layer 12-neuron structure, same training data and was analysed using the same techniques. The back propagation taught network was found to have a $R^2$ value of 1.0000 and a MSE of $5.3189 \times 10^{-8}$. This correlates to an average error of 0.001N in the joint torque prediction. Figure 4.5 demonstrates the linear regression fit of this network. The green, blue and purple dots represent the joint torques for each of the first, second and third wrist joints respectively and
Figure 4.6: Results of the batch MBP trained NN: Resulting torques for Barrett WAM wrist from the LMB batch trained neural network (dots) using random inputs compared to the standard dynamic equation output (circles).
the black line once again represents the desired network response. It can be seen that
the predicted torques are once again very linear, however it can also be seen that the
grouping of the torques are a little bit tighter than that of the EKF trained network,
which is expected given the MSE for the LMB network is smaller. However, since the
batch network is trained to a batch data set and is not corrected for new motions it
is more susceptible to outliers, as can be seen in a few cases for joint 1.

The results of the neural network trained by LMB due to specific moves were also
plotted similar to the EKF and can be seen in figure 4.6. Specific joint trajectories
were propagated through both the Newton-Euler dynamics and the neural network.
The green, blue and purple circles represent the desired torque outputs for the specific
trajectory for the first, second and third joints respectively. The light blue, yellow
and red dots represent the network predicted torques for the specific move for the first,
second and third joints respectively.

A third network, being trained with an online BPM training algorithm was also
generated. This network was used to compare the EKF trained network with a
standard online training algorithm. The BPM online algorithm is the algorithm
currently used in the FEL control systems. The online BPM network was found
to have an $R^2$ of 0.9716 and a MSE of $1.127 \times 10^{-4}$. In terms of a neural network
training algorithm, the online BPM performed much worse than both the batch BPM
and EKF algorithms. The primary cause for the poor results is due to the learning
algorithm forgetting past training sets. Figure 4.7 demonstrates the linear regression
of the BPM online trained network. The green, blue and purple dots represent the
joint torques for each of the first, second and third wrist joints respectively and the
black line once again represents the desired network response. It is easy to see from
this figure that the torques are poorly estimated leading to the large $R^2$ and MSE
values.

The results of the neural network trained by the online BPM using to specific
Figure 4.7: Linear regression of online back propagated NN: Linear regression of the online BPM trained neural networks. Perfect fit follows the $y = x$ diagonal.

moves as inputs were also plotted similar to the EKF and can be seen in figure 4.8. Specific joint trajectories were propagated through both the Newton-Euler dynamics and the neural network. The green, blue and purple circles represent the desired torque outputs for the specific trajectory for the first, second and third joints respectively. The light blue, yellow and red dots represent the network predicted toques for the specific move for the first, second and third joints respectively. This figure shows that the LMB online algorithm never really converges to a robust solution. Once again, the first few plots demonstrate network performance during training, while the latter plots are using a network that has had time to converge.

It can be inferred from the comparison of the first two learning algorithms, demonstrated by their MSE, $R^2$ and performance in figures 4.4 and 4.6 that a neural network is able to predict the dynamics of a pan-tilt. Furthermore, an EKF trained network is able to perform very closely to a standard set by the widely used Levenberg-Marquardt Backpropagation batch training algorithm. While the back propagation algorithm was able to converge to a slightly better solution, the differences between it and the EKF training algorithm were small. The $R^2$ fit differed by only .0003 which
Figure 4.8: Results of the online BPM trained NN: Resulting torques for Barrett WAM wrist from the BPM online trained neural network (dots) using random inputs compared to the standard dynamic equation output (circles).
can be considered negligible and goes away with the inaccuracies in the system. On the other hand, the MSE was approximately a factor of ten better than that of the EKF training law, showing that training to a set of data as opposed to single points may yield a better solution. However, in comparison to the average joint torque command it only represents a 0.02% MSE relative to the back propagated network at 0.001%. The MSE is small enough that the difference between the EKF and back propagated network can be considered irrelevant in the application of a pan-tilt controller. This shows that the EKF online training algorithm can be applied to a neural network and achieve similar results to a batch trained network. In terms of online training, the EKF online network was much better than the standard LMB online algorithm. The LMB algorithm wasn’t able to converge to a solution close to the level achieved by the batch and EKF algorithms, making it unsuitable for the purposes of a pan-tilt camera mount.

4.5.3 Stability of the EKF Training Law

The stability of the controller would be a natural analysis as well however feedforward controllers cannot be analysed using conventional stability techniques. Instead the stability analysis will focus on the convergence stability.

Since the controller is continuously changing its weights, there is the potential for it to become unstable and converge to certain moves instead of generalizing to the entire dynamics. There is also a desired for the network to converge quickly and reduce the time needed for the solution to converge. Figure 4.9 demonstrates the $R^2$ value and MSE of the EKF trained network as a function of the number of 50 point trajectories it has seen. Both the MSE and $R^2$ follow a similar trend which is expected since they are somewhat related. The network was able to converge to a usable solution after only approximately 10 sets of trajectories. It can also been seen from this figure that the network needed to see approximately 10-20 different
Figure 4.9: $R^2$ and MSE during EKF training: Top figure shows the current $R^2$ (unitless) during training with the EKF algorithm. Bottom figure shows the current MSE ($Nm^2$) during training with the EKF algorithm.

trajectories before the solution was able to have a $R^2$ above .995 and MSE below $0.2 \times 10^{-5}$. After it converged the network became more and more stable as time when on, represented by the smaller and smaller changes to the MSE and $R^2$ as more and more training data is considered. During the final 700 training data sets, the $R^2$ did not vary by more than 0.0004.

As previously mentioned, figure 4.4 demonstrates the specific torque commands predicted by the EKF trained neural network during training. It can be seen from the training plots in this figure that the initial joint torques estimates from the network are very poor. However after only 5 trajectories are considered the estimate gets much better and after 25 the network can be said to be quiet accurate. However it was shown in figure 4.9 that the network is still a little unstable in this region.

A similar training analysis was performed for the online BPM algorithm. Figure 4.10 demonstrates the $R^2$ value and MSE of the EKF trained network as a function of the number of 50 point trajectories it has seen. Both the MSE and $R^2$ follow a similar trend which is expected since they are somewhat related. A wide variance in both the $R^2$ and MSE values can also be seen during training. The large jumps
Figure 4.10: $R^2$ and MSE during BPM training: Top figure shows the current $R^2$ (unitless) during training with the BPM algorithm. Bottom figure shows the current MSE ($Nm^2$) during training with the BPM algorithm.

In performance show instability in the training algorithm, caused by the constant forgetting and retraining of the network. In most cases, because the network only trains to single points, the correction made to update the weights is too large and it cause bigger errors in future calculations. In the 1000 training points used, the online BPM algorithm is unable to converge to a robust solution.

Figure 4.8 demonstrates the specific torque commands predicted by the online BPM trained neural network during the training process. It can be seen from the training plots in this figure that the initial joint toques estimates from the network are very poor. The predicted torques begin to converge during training, but are not able to converge to the solution shown by the EKF online training algorithm.

4.5.4 Full System Simulation

The trained EKF network was then put into simulation of the WAM arm, which is used to simplify the transfer to hardware. A target was defined somewhere in the workspace and the first four joints were given a random trajectory. The simulator
Figure 4.11: **Barrett WAM Simulation**: Barrett WAM simulation using random inputs controlled by the EKF taught neural network. Units are in m

determined the required joint trajectory using the methods shown in Chapter 3 and used the designed neural network to estimate the required torques to produce the trajectory. The torques were then converted back into the joint space trajectory using the inverse dynamics shown in the previous section. The desired response from the simulator is for the blue axis representing the viewing direction of the camera to constantly be intersecting the purple $x$ representing the target. The green axis representing the orientation of the camera should remain parallel to the $x - y$ plane in base coordinates.

Figure 4.11 demonstrates the output of the simulator for one of the tested trajectories. It can be seen from this figure that the camera was able to track the target while remaining horizontal throughout the move. It was noted that there was a lot of noise generated by the inverse dynamic model, due to the amount of integration. However, the simulator was still able to give a good representation of the network
4.6 Chapter Summary

A neural network is able to accurately and precisely model the dynamics of a 3-DOF spherical manipulator. The neural network in this sense has many advantages. Not only is it computationally more efficient but it is also able to be trained for a variety of complex and non-linear system components that most models do not currently account for, such as friction [77]. Neural networks are able to generalize to situations they have not been directly trained for and are very easy to apply. The ability to generalize is especially valuable in noisy conditions like manipulators where most input parameters are integrated from noisy data. The downside to using neural networks is the amount of effort required for training and design. Both structure and training rules will need to be adapted depending on the results, or even repeated if the dynamic system changes in anyway, such as dust build up in joints or inaccuracies in determining actually operating parameters. Conventional training laws put limits on the training and are not adaptable to a changing system, however an active learning system can continuously be trained to new data while in use and converge to new solutions if the dynamic system changes from that which it was during training.

The network developed for this thesis was able to achieve the goal of mimicking the dynamics of a 3-DOF manipulator. The network, given a required trajectory was able to predict the required joint torques need to achieve the desired motion. In order to alleviate the problems with separate training and online phases, an active learning algorithm was tested and compared to that of a known dedicated training algorithm in back propagation. The active learning algorithm was taught online with an EKF training law. It was also able to adequately mimic the dynamics of a 3-DOF manipulator, however the back propagated network was slightly better. The
small differences in the final network fits are due to the biasing effect of the online training algorithm. The biasing effect was the found trend for the network to more accurately model the more recent trajectories. In the long term, it is likely that with more training data the EKF trained network will continue to converge to a better solution as the data showed continuous improvement. The network was also found to converge to a usable solution quickly, needed only approximately 10 moves before it was able to learn the system dynamics. The constantly changing weights made the network a little unstable for the first few trajectories as well but were able to converge to a solution after only 50-60 moves. In terms of an online training algorithm, the EKF algorithm was able to converge to a more robust solution than the LMB online algorithm, as well as a more stable solution. Convergence was also much faster for the EKF trained network. The primary difference between the EKF training law and the online BPM training law is that the EKF law is able to keep some measure of the past training results, while the BPM discards part training sets.
Chapter 5

Feedforward Control Applied to Barrett WAM

The previous chapter showed that neural networks can be used to predict the dynamics of a 3-DOF spherical wrist, in the context of tracking targets in the visual field. It was also shown that an active learning training algorithm can be used to give similar results to those generated by more standardized back propagation techniques while still providing a stable solution. This chapter will discuss the results of the designed controller as applied in hardware to the Barrett WAM. The network will continue to learn actively as it attempts to model the real dynamics of the system and the results of such will be compared to traditional PD feedback control.

5.1 Code Structure

Since the simulator designed in Chapter 4 was based on the Barrett WAM, the kinematic model stays the same. The training model on the other hand is no longer valid. The training model in simulation was taught by a correct signal, representing the exact required torques that are need to produce the desired trajectory. In practise, this value cannot be determined. It isn’t trivial to determine the exact required torques
for a specific desired move. That is to say, there is no way to exactly determine how
much torque is needed to produce a certain joint trajectory. However, the inverse
problem can be measured. The joint space trajectory of a known applied torque can
be measured and used as the training data set. However, this give rise to the pos-
sibility of the data set remaining outside of the operating conditions. For example,
if the control torques is not enough to cause the arm to move, the network will be
continuously taught that that output torque will cause no motion and it will never
be able to adapt to moving data. To overcome this problem, the wrist was hit with
a feedback torque based on a simple proportional gain law. this way, the feedback
torque will cause the arm to move, allowing it to learn motion dynamics. As the
network gets more precise, the feedback torque becomes negligible as it is a function
of the joint error.

The control algorithm runs at a rate of 50 Hz, the bottleneck factor being the
rate at which the EKF can run. For each iteration, the arm performs the following
actions:

1. The positions of the first four joints are measured

2. The required joint angles for the last three joints are determined and differenti-
ated to determine the required trajectory to correct for the change in orientation
ciaused by the first four joints

3. The desired trajectory is propagated through the network to determine the
required joint torques. These are added to the feedback torques and applied to
the arm.

4. After the time step $dt$, the actual trajectory of the arm is measured

5. The actual trajectory and applied torque is used to update the network weights
to the new data.
5.2 Testing Environment

To test the control algorithms, the arm was given the vector in base coordinates of a target to track. The desired trajectory and resulting trajectory were monitored to mathematically quantify the results. In addition a raspberry pi and camera module were also fitted to the end-effector of the arm to visually quantify the resulting trajectory. Figure 5.1 demonstrates the images from the camera module at different arm configurations and figure 5.2 demonstrates the arm configuration. In order to have something to compare the feed forward controller to, a feedback controller was also tested. The same validation parameters used in Chapter 4, $R^2$ and MSE, were used to compare the results. The comparison will focus on the joint angle positions instead of joint torque as was done previously, due to the non existence of the correct torque command. The first four joints of the arm were moved manually, proving the system knew nothing of the motion beforehand.

The arm was given the vector from its base to a target, and the end-effector was to track the position of the target as the arm moved. The only measurement the arm is able to take is joint position, so in order to determine the velocity and acceleration, the joint positions were integrated.

5.3 Feedback PID Control

The most common feedback controller uses the computed torque method, which is a form of proportional/integral/differential (PID) control acting on the dynamic equation of a manipulator introduced in (2.10). The control law starts by defining the torque disturbance, $\tau_d$ that is causing the error in motion and adding it to (2.10).
Figure 5.1: Tracking output: Visual tracking right and left of the target as seen by the end-effector. Target being track is the steel haptic ball.

Figure 5.2: Barrett WAM setup: Configuration of the Barrett WAM. First four joints represent inputs and the last three are used to correct orientation so the camera can track targets.
\[ \tau = D(\theta)(\ddot{\theta}) + h(\theta, \dot{\theta})\dot{\theta} + G(\theta) + \tau_d \]  \hspace{1cm} (5.1)

Next we define an error vector \( e \).

\[ e(t) = \theta_d(t) - \theta(t) \]  \hspace{1cm} (5.2)

where \( \theta_d \) is the desired trajectory and \( \theta \) is the actual manipulator trajectory. To demonstrate the effect \( \tau \) has on the tracking error, the error vector \( e \) is differentiated twice.

\[ \dot{e}(t) = \dot{\theta}_d(t) - \dot{\theta}(t) \]  \hspace{1cm} (5.3)

\[ \ddot{e}(t) = \ddot{\theta}_d(t) - \ddot{\theta}(t) \]  \hspace{1cm} (5.4)

If we solve for \( \ddot{\theta} \) in equation 5.1 and sub it into the equation for \( \ddot{e}(t) \), we get

\[ \ddot{e} = D^{-1}(\theta)(h(\theta, \dot{\theta})\dot{\theta} + G(\theta) + \tau_d - \tau) \]  \hspace{1cm} (5.5)

If we define a control signal \( u(t) \) as

\[ u = \ddot{\theta}_d + D^{-1}(\theta)(h(\theta, \dot{\theta})\dot{\theta} + G(\theta) - \tau) \]  \hspace{1cm} (5.6)

and the disturbance function \( w \) that we are trying to correct for as

\[ w = D^{-1}(\theta)\tau_d \]  \hspace{1cm} (5.7)

then the dynamic equation 5.1 can be re-written as
\[ \tau = D(\theta)(\ddot{\theta}_d - u) + h(\theta, \dot{\theta})\dot{\theta} + G(\theta) \]  

(5.8)

Now a control signal \( u(t) \) can be selected to drive the error vector \( e(t) \) to 0. As previously stated, the most common manipulator controller uses a PID control signal. The PID control law defines the signal \( u \) as.

\[ u = -K_v\dot{e} - K_p e - K_i \int_0^t e dt \]  

(5.9)

and by extension

\[ \ddot{e} + K_v\dot{e} + K_p e + K_i \int_0^t e dt = w \]  

(5.10)

where \( K_p, K_i \) and \( K_v \) are the proportional, differential and integral gain matrices. These matrices are \( n \times n \) diagonal matrices of the gains \( k_p, k_i \) and \( k_v \) for each joint. In state space form this becomes

\[
\begin{bmatrix}
\frac{d}{dt} e \\
\dot{e} \\
\end{bmatrix} = \begin{bmatrix}
0 & I & 0 \\
0 & 0 & I \\
-K_i & -K_p & -K_v \\
\end{bmatrix} \begin{bmatrix}
e \\
\dot{e} \\
\end{bmatrix} + \begin{bmatrix}0 \\
0 \\
1 \\
\end{bmatrix} w
\]  

(5.11)

where \( \dot{e} = e \). The state space representation makes the characteristic equation of the system easy to determine.

\[
\delta_c = |s^3I + K_v s^2 + K_p s + K_i| = \prod_{i=1}^{n} (s^3 + k_{v,i}s^2 + k_{p,i}s + k_{i,i})
\]  

(5.12)
Since the roots of the characteristic equation represent the pole positions of the system in the $s$ plane, and therefore the response of the system, the gains $K_p$, $K_i$ and $K_v$ can be selected to give any desired system response [79]. Figure 5.3 summarizes the feedback manipulator control structure.

5.4 Results

It was shown in the previous chapter that the network becomes unstable for the first few trajectories. To therefore the arm was initially taught using a feedback controller and did not affect the control loop in any way until the network was trained. This was done to prevent the arm from over torquing during the initial training process from the rapidly changing network weights. Once the network was trained, it was added into the control loop with the PD controller.

The network was able to very quickly adapt to the dynamics of the 3-DOF wrist. The same analysis presented in the previous chapter was used to quantify the results.
Figure 5.4: Linear regression of neural network: Linear regression of the EKF trained neural network applied to the Barrett WAM. Perfect fit follows the $y = x$ diagonal.

The mean square joint trajectory command for the WAM arm was found to be 0.2859. The feed forward controller was found to have a $R^2$ value of 0.9931 and a MSE of 0.0018. Figure 5.4 demonstrates the linear regression of the feed forward controller. The black line represents the desired $x = y$ fit from the network. The green, blue and purple dots represent the joint trajectory for each of the first, second and third wrist joints respectively. The plotted trajectories together are very linear and are dotted out along the black diagonal, which is expected due to $R^2$ being so close to 1. They are also grouped very close to the line which was expected due to the MSE being so low. The second joint was constrained to remain between 0 and $\pi$ which is why the joint angle never drops below 0.

A feedback controller was also tested in order to give a relative comparison for the feed forward cerebellar-like controller. The feedback controller had an $R^2$ value
Figure 5.5: **Linear regression of feedback**: Linear regression of the feedback control law applied to the Barrett WAM. Perfect fit follows the $y = x$ diagonal.

of 0.9750 and an MSE of 0.0098. Figure 5.5 demonstrates the linear regression of the feedback controller. The green, blue and purple dots represent the joint trajectory for each of the first, second and third wrist joints respectively. The plotted trajectories together are quite linear and are dotted out along the black diagonal, which is expected due to $R^2$ being still quite high. However the grouping of the feedback controller is quite poor. While the fit may be linear, it is not very precise which can be seen from the distance the points are from the line.

### 5.5 Chapter Summary

It was previously shown that a neural network is able to accurately and precisely model the dynamics of a 3-DOF spherical manipulator. The testing done on the actual arm itself confirms this simulated result. However, it was found that the network was less accurate in practice than in simulation. This was found to be due to the noise in
the joint velocity and acceleration that was used to train the network. Since the arm only measures joint positions, the joint velocity had to be integrated over the time step once and acceleration was done so twice, which explodes the relatively small estimation error in the joint encoders. However, even with the large errors in the measurements, the network was able to adequately predict the required joint space trajectory need for the reactive controller to track objects in the visual field.

In comparison with the feedback controller, the feed forward controller was found to be both more accurate and precise. The feed forward controller was able to predict the required joint torques more accurately due to the fact that it is able to predict the exact required torques as opposed to a torque that will give the approximate motion and correct for errors later.

It was found that the network would cause the end-effector to have a slight tremor, which can be attributed to the network attempting to compensate for the friction in the joints as this is one of the largest dynamic contributions and has linear components based on the joint velocity. The small errors in the joint velocity measurement cause the arm to want to move slightly and therefore the network creates torque commands to generate the desired motion.
Chapter 6

Conclusions

This thesis presents an analogy between the control of a pan-tilt camera mount to the vestibulo-ocular reflex exhibited by the human cerebellum, in the context of tracking visuals targets during the traverse phase of rover missions. The solution presented was an extension of the theoretical feedback-error-learning model of the cerebellum proposed by Kawato. The model consisted of feed-forward neural network actively trained by an extended Kalman filter, instead of the more traditional back propagation training algorithm. It was found that a neural network was able to adequately predict the joint torque commands required to track objects in the visual field both in simulation and in practise. The online training EKF training algorithm was able to converge to a solution close to those achieved by the batch LMB algorithm and over 3 orders of magnitudes better than the online BPM currently used in the few FEL applications. The EKF online algorithm was also shown to converge to a usable solution much faster than the BPM algorithm as well as remain stable during the consternated altering of the network weights. In practise, the feed forward controller was able to track objects in the visual field faster and with fewer errors than the conventional PD controller.

The presented feedforward controller was used as the control algorithm for
panoramic cameras during the traverse phase of rover missions. This capability represents the first component towards the application of active vision and the opportunistic search of salient targets and the expansion of the potential yield from planetary rovers. Furthermore, feedforward control also has the potential to enhance other aspects of robotic planetary missions, such as manipulator tele-operation and force control.

6.1 Summary of Contributions

1. The generation of a mathematical model to allow for visual tracking of targets using only information readily available on the rover itself through a standard IMU. The model is a function of the DH parameters of the rover and pan-tilt system alone making it easily adaptable to any system.

2. The design and simulation of a cerebellar-like reactive controller to predict the required joint space torque commands to actively track objects in the visual field. The controller was neural network that was able to be trained actively online to new data sets with a performance similar to that achieved by more standardized practices while proving to be much more stable.

3. The application of the designed controller in hardware on the Barrett WAM. The arm was able to track a given target faster and with fewer errors in orientation than conventional feedback PD controllers.

6.2 Future Work

1. The cerebellar-like controller was able to track objects well on the Barrett WAM, however using the arm limited the range of motion of the platform. Applying the controller to a rover with a wider range of motion could further solidify the performance of the work package.
2. Past studies have shown that potentially the cerebellar controller may perform better if the dynamics are broken into two different components: first a network to predict the non moving torques and a second to predict the required moving torques.

3. The cerebellar-like controller can easily be applied to other applications. Future works could include applying the controller to larger systems, such as full manipulator control.

4. The controller was designed as the control component of an active vision like salience tracking system to actively search for targets into the visual field during the traverse. With the development of the camera end gradient tracking elements the two components will need to be tested as a unit.
References


