UNIVERSITY OF SOUTHAMPTON

Biologically Inspired Control Techniques for Compliant Reaching

by

R.M. Sunderland

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

March 2006

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by R.M. Sunderland

This research aims to add to the understanding of manipulation and actuator control with the objective of facilitating the future development of fully autonomous manipulating agents. As a first stage along this path, the work presents a simulated implementation of two techniques designed to control a two degree of freedom arm, equipped with six compliant actuators.

After preliminary modelling with a static arm, the first technique, inspired by a biological motor theory called convergent force field control, seeks to use a multi-layer-perceptron (MLP) to steer a custom built dynamic simulation of the arm towards predefined targets within the workspace. The MLP is trained iteratively using mutation hill climbing. The work demonstrates that this approach is capable of creating arm responses that move towards a fixed target from any location within the workspace. That said, the learning approach is not reliable and often gets stuck in local minima. The section concludes by demonstrating the blending of two controllers, to create a smooth range of intermediate results.

The second technique combines convergent force field control with another related biological control theory, called equilibrium trajectory control. Key to the success of this technique is the assumption that smooth natural movements of the tip of the arm can be generated by constant rate movement of the arm's equilibrium point, with respect to the workspace. The model proposed uses a form of convergent force field control to guide an internal representation of the equilibrium point towards a target at a constant rate. Various implementation options are considered with particular attention paid to the way the equilibrium point is encoded internally. Initial development of the trajectory generator and length encoder indicate that, for reasonable sizes of MLP, shoulder centred polar encoding is the most computationally efficient scheme.

It is hoped that the work presented here will provide a foundation for techniques that will bridge the gap between behaviour based control and low level coordination of complex compliant articulated systems.

Contents

Acknowledgements

My wife, Catherine, for her loving support, encouragement and proof-reading.

My supervisors, Richard Crowder and Bob Damper for their guidance, support and considerate criticism.

Nigel Shadbolt, for his help in focusing the project.

Richard Watson, for his thoughtful questions and patient listening.

Chapter 1

Context

This research aims to add to the understanding of manipulation and actuator control with the objective of facilitating the future development of artificial manipulators that can work in a broader range of environments. Inspiration will be drawn from biological and engineering studies, but although cross fertilisation will be likely, the overriding objective will be the development of real-world artifacts.

The study of manipulation has been approached from several angles, including such diverse fields as infant motor development, primate tool use, stroke rehabilitation and industrial equipment design (Paul, 1981; Craig, 1986; Van der Meer et al., 1995; van Schaik et al., 1999; Stevens and Stoykov, 2003). Due to its cross disciplinary nature, the terminology associated with manipulation can be somewhat confusing. This chapter explores the motivation for studying the subject and then outlines the basic manipulation phases, while introducing the key terms.

1.1 Justification of Approach

The goal of this work is to demonstrate that techniques inspired by biological studies of movement may be usefully applied to the control of autonomous robotic manipulation. There are various advantages envisioned for this approach that include robustness to environmental disturbance, improved coordination of complex actuator configurations and increased computational efficiency.

Conventional robotic manipulators are normally constructed to minimise the coupling between actuators, such that each joint may be considered separately. This approach is effective when using conventional motors or actuators, but may be less desirable when building systems using 'soft actuators' such as electro-active polymers (Bar-Cohen, 2002), series-elastic actuators (Pratt and Williamson, 1995), deformable air muscles (Tuffield and Elias, 2003; Boblan et al., 2004) and coiled shape memory alloys (Otsuka

and Ren, 1999). For complex systems, including robotic arms and hands, to be developed using such technologies, effective coordination of actuators is essential. At some stage a desired action must be converted into appropriate actuator activation signals. This conversion will generally require a complicated chain of transformations and processing, all of which must be carefully considered when designing a complete manipulating agent. Although this work will not attempt to analyse all the stages in this chain, they will be considered when defining the interfaces for each stage.

It is tempting to ignore computational requirements, and assume that Moore's law will soon supply the required processing power. While this is to some extent reasonable, it should be remembered that mobile robots must manage their power budget carefully. This means that, as processing power costs battery life, computational requirements must be taken seriously from the design stage.

1.2 Importance of the Problem

The development of industrial tools which are capable of manipulating objects has produced useful results; an excellent example of this is the automation of car production. Although such systems rely on sensor-actuator coupling at a low level (servo-control) they are not usually capable of autonomous behavioural flexibility. Within an industrial setting this does not pose a problem, as each manipulator has a predefined task and is presented with only a very limited range of environmental variation. There are, however, several valuable tasks that require interaction with less well structured, more dynamic environments, where a manipulator will have to cope with a dramatically wider range of input variation. Remote exploration, toxic material handling and ordinance disposal are exemplars of such tasks. The dynamics of delicate manipulation tasks typically have short time constants; tactile feedback is therefore normally used to improve stability. For the tasks considered, signal delay would render a remote operator incapable of using tactile feedback stably (Goldberg, 2000) and it would be too costly or dangerous to employ a human locally. If we are to create machines that can perform remote manipulation, the low-level control of the interacting surfaces must be performed locally, by the machine. Such devices should then be considered at least semi-autonomous. The question then arises, 'How can a machine's controller interact with the world, through its body, to achieve skillful manipulation?' The animal world is replete with examples of systems which have answered exactly that question very successfully, so it seems wise to turn to them for inspiration. Although the human may have very advanced manipulation capabilities, many animals routinely complete difficult manipulation tasks; from warren excavation to nest building, they all are managing materials to improve their environment.

It has been argued that the opposable thumb, or more precisely the manipulation capabilities it affords, played a pivotal role in the development of primate intelligence (Bicchi, 2000). There are two immediate ways in which dexterous manipulation may aid the development of intelligence. Firstly, it facilitates inspection of objects, allowing for more detailed exploration of the environment. Secondly, it permits detailed modification of objects, giving us finer control over our environment. To exploit these advantages, the brain had to learn to connect this range of sensations to the range of available actions in a useful way. It is probable that the brain and the hand developed in tandem, one providing increased sense-act options and the other learning to exploit them. With this in mind, it seems reasonable to think that the development of advanced manipulation capabilities may play a central role in the study of more natural artificial intelligence.

From a more general engineering perspective, the design of manipulating agents is challenging and likely to contribute indirectly to other fields. Manipulation tasks require interrelating multiple control signals, in different reference frames, to coordinate the activities of a high-dimensional system with a range of shifting apparent inertias and time constants.

1.3 Autonomous Agent Approach and Context

Having decided that natural style manipulation is a valuable topic, from both a cognitive science and an engineering perspective, the next step is to clarify what is meant by manipulation. The Oxford English Dictionary defines the verb *manipulate* as "To handle, especially with skill or dexterity; to turn, reposition, reshape, etc., manually or by means of a tool or machine". Within the robotics literature, a manipulator is taken to mean the body, and specifically the arm, of an articulated robot. Within the context of this research *manipulator* will be given the more general definition of 'an autonomous agent that is capable of modifying its physical environment usefully.'

The term *agent* is somewhat ambiguous, and is here considered to mean a system that operates independently, contained within a shell, or 'body', which mediates its interactions with the outside world. The exact interpretation of what constitutes a body is context dependant. For an animal, or a robot, the body is a collective term for its material components, whereas for a symbolic software agent it is the methods through which it can sense and modify the rest of its environment. Although there are interesting parallels between the two, physical agents typically have to deal with significantly more complicated and tightly time-bound environmental interactions.

If a manipulation task requires a human operator, the mechanical and computational components are seen as a tool, or extension, of the human and the whole considered as a manipulator. This more holistic definition results from a desire to see the body and controller as part of a unified whole, rather than as systems that should be assessed and

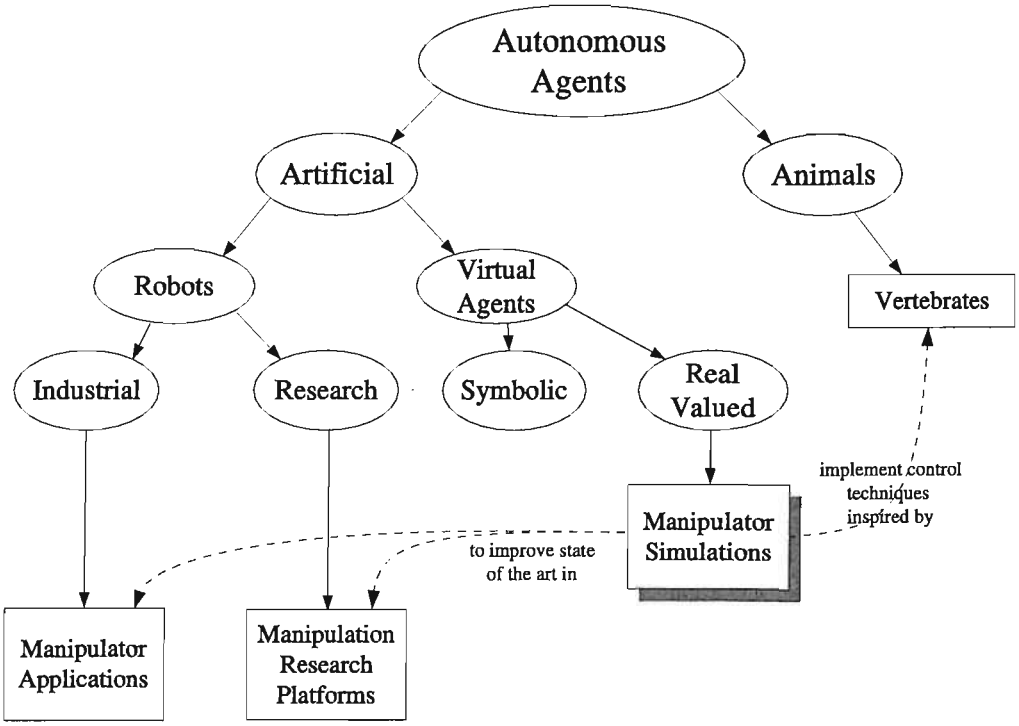


FIGURE 1.1: A partial agent taxonomy, showing the context for the simulation work presented. Solid lines show refinement of a class definition. Dashed lines show the motivation.

developed separately. For more detail on the essential qualities of an agent, see Pfeifer (1996, Chap. 4).

Figure 1.1 shows a partial agent taxonomy, highlighting the context for this work, in terms of both technology and motivation. The work presented in this thesis involves the implementation of real-valued virtual agents, that are designed to test the suitability of control techniques inspired by vertebrate studies for use in both industrial and research robots. In this context, *real-valued* is taken to imply that no symbolic logic is performed by the agent, and that the state of this system is continuously variable (rather than discrete). The work presented here uses virtual agents, rather than physical ones, so care has been taken to restrict the information exchanged between the controller and the environment/body to ensure authenticity. As well as considering the amount of information available to a controller, it is also important to consider the quality of the information. Physical agents must normally contend with sensor and actuator noise; for those using analogue controllers, processing noise must also be considered.

1.4 Manipulation Cycle

Most manipulation tasks can be considered as a series of 'cycles' through four behavioural phases: *Approach*, *Closure*, *Modification* and *Release* (Fig. 1.2). The objectives of each

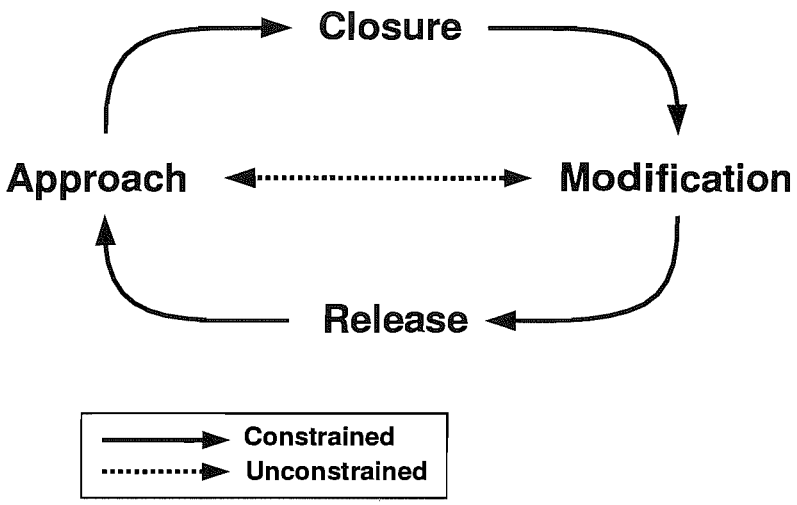


FIGURE 1.2: The stages of the constrained and unconstrained manipulation cycles.

phase are quite distinct and are discussed below.

Approach: The object of the approach phase is to reconfigure the manipulator so that a graspable portion of the target object lies within the end-effector's working space. There are two main components to this phase: reaching and pre-shaping. Reaching describes the movement of the end-effector from its current position toward the target. Pre-shaping describes the process of arranging the end-effector to allow close approach without premature collision of the target and the digits.

Closure: During the closure phase, the manipulator attempts to reconfigure the end-effector to create a series of contacts between it and the target object. The force and location of these contacts will depend on the nature of the modification required, but generally should not damage the manipulator or the target.

Modification: The actual work of the manipulation task is performed during the penultimate stage: modification. This could be as simple as a rotary or translational repositioning, or as complex as reshaping clay. Most of the force exerted during this phase is provided by the relatively small, finely controlled, actuators of the end-effector, but where this is insufficient the more powerful arm actuators may be recruited, for example when opening a door or pulling a rope. This recruitment process allows the manipulator to trade precision for power, in order to optimise task performance. As robotic manipulators must also handle tasks with differing power and precision requirements, it seems likely that similar behavioural tactics will be beneficial.

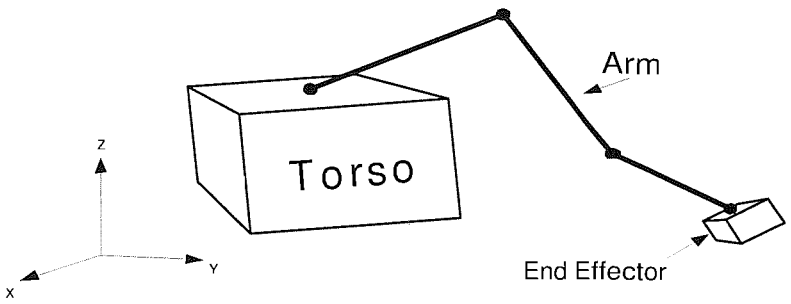


FIGURE 1.3: The components of an abstract manipulator: torso, arm and end-effector.

Release: Once the required modification has been achieved, the end-effector must usually disengage from the target object without disturbing it inadvertently. It may be desirable for the object to be left in a stable configuration, or it could be released during a high speed movement (i.e., thrown). In either case, the coordinated, timely release of contact points is important.

A significant subset of manipulation tasks do not require a closure phase (for example spinning a pen on a desk, pushing a large box or punching something). Without a closure phase, it is not possible for the end-effector to exert any pulling forces. There are several reasons why an unconstrained manipulation cycle may be preferred. Firstly, it may offer favourable system dynamics; only a non-constrained pen will keep spinning between flicks. Secondly, it may be faster as the closure phase will take time, for example deflecting a missile. Lastly it may be selected as a second best because no hand holds are available.

1.5 Dexterous End-Effectors

Manipulation is often assumed to require some form of dexterous end-effector, such as a hand or a robotic gripper; this is, however, not always the case. Observe a human trying to retrieve their car keys from a trouser pocket while carrying a large pile of books and papers and you may see chins, arm-to-torso pins and even teeth in use. If we consider an octopus' tentacle, it does not have an end-effector as such, but is certainly capable of manipulation tasks. We should therefore recognise the potential for many different parts of a body to perform useful manipulation and resist the rush to categorise rigidly each component. That said, many animals and robots have specialised end-effectors, that are equipped with relatively small, high resolution actuators and their accompanying sensors. Typically, they are not suitable for moving the body itself, and are restricted to manipulating objects that are an order of magnitude smaller than the body. Figure 1.3 shows an abstract manipulator, with an end-effector.

FIGURE 1.4: Robonaut Hand, taken from Robonaut Website (2005). The robot is shown screwing a nut onto a bolt under telerobotic control.

Development of dexterous end-effectors was initially inspired by work on prosthetic hands and then by the need to handle objects in hazardous environments. Generally, end-effectors designed for use as prosthetic hands have a very low load carrying capacity and would only be suitable for a small range of manipulation tasks (Dubey, 1997).

There have been a number of significantly successful research robotic hands, including the Stanford/JPL hand, Utah/MIT hand and Belgrade/USC hand. The Stanford/JPL hand (Salisbury, 1982; Mason and Salisbury, 1985) has three fingers and is driven by twelve DC motors, via tension cables. The use of tendons allows the hand itself to be lighter but adds to overall drive complexity and weight. The Utah/MIT hand (Jacobsen et al., 1984) has three fingers and an opposable thumb, each with four degrees of freedom. It is equipped with optical and capacitive touch sensors. The Belgrade/USC hand (Bekey et al., 1990) has the most fingers (four and an opposable thumb) but the simplest drive system, relying on only four motors. These are built directly into the wrist, so the entire end-effector is self-contained. This reduces its dexterity, but would make integration with commercial robotic arms easier. Recent work by NASA (Lovchik and Diftler, 1999), focused on developing robots capable of extra vehicular activity (space walks), has produced a very advanced self-contained hand, with near human dimensions (Fig. 1.4). For a review of the development of robotic hands, including a discussion of key terms, see Bicchi (2000).

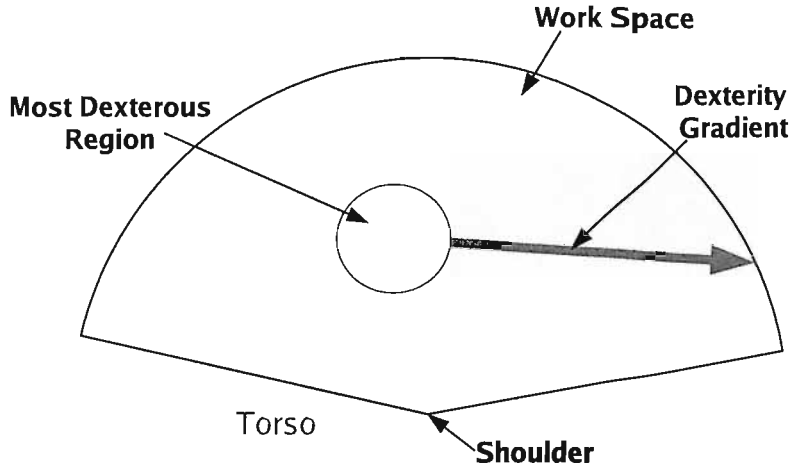


FIGURE 1.5: Section through a manipulator's workspace. The area of most dexterity is shown in white, with an arrow indicating the gradual reduction in dexterity toward the workspace perimeter. The workspace configuration will depend on the manipulator's physical and control characteristics.

1.6 Workspace

The volume that a manipulator can reach, without moving its body, is referred to as its workspace. A manipulator will not be able to reach its entire workspace with equal dexterity, so we must consider the quality of any given location. Complex tasks that require fine control must be performed near the optimal workspace location, whereas simple tasks like pick-and-place can be performed over a large range of locations. Figure 1.5 shows a section through a manipulator's torso and workspace. Robotics literature presents a formal definition of workspace volume, based on robotic structure (Craig, 1986, p. 102), but this does not give any measure of the varying quality of the space. Many industrial robots are rigidly anchored, which means that world-coordinate space and body-coordinate space are locked together. It is, however, worth noting that the workspace is defined in body-coordinates and mobile manipulators are capable of moving their bodies to align their optimum manipulation location to be near the target object.

1.7 Grasp Types

A review by Bicchi and Kumar (2000) defines the standard grasp types. Initial studies in this field were driven by the requirement to build jigs that clamped work pieces securely. Reuleaux (1875) defined *form closure*, the condition under which a positive combination of the contact forces and moments acting on an object, without assuming any friction, can resist any disturbing force. Further analysis determined the minimum number of

contacts required to restrain planar and spatial bodies (Somoff, 1897; Markenscoff and Papadimitriou, 1989; Markenscoff et al., 1990).

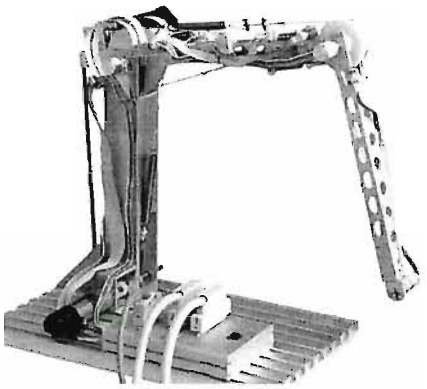
An alternative grasp type, called *force closure*, is achieved if and only if any arbitrary external force or torque acting on the object can be balanced by pressing the finger tips against the object at the selected grip points (Mishra et al., 1987). It is usually possible to achieve force closure with fewer contact points than form closure. This means it can be achieved with fewer fingers (assuming one contact per finger) and therefore a simpler and cheaper end-effector can be used.

There are two broad categories of force closure, *pinch grip* and *enveloping grip*. With a pinch grip, the only contact between the object and the end-effector occurs at the tips of the fingers, while an enveloping grip (sometimes referred to as a power grip) also uses the inside surface of the palm and fingers. An enveloping grip may be achieved with even fewer actuators, but this is generally at the expense of dexterity.

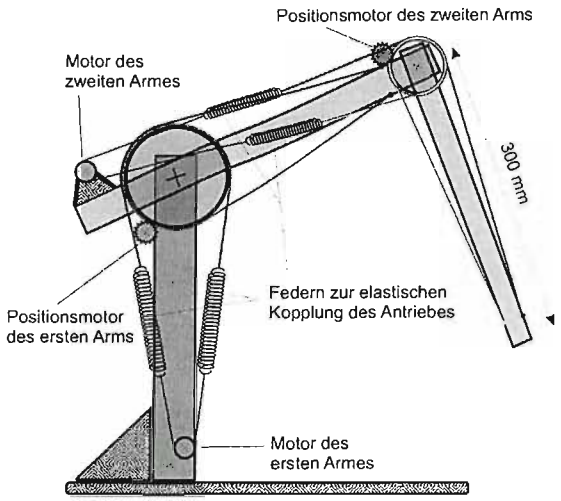
1.8 Planar Reaching

As discussed, the approach phase of the manipulation cycle has two distinct components: reaching and pre-shaping. The work contained in this thesis focuses on control of reaching behaviours for compliant arms with two degrees of freedom. This aspect of manipulation was chosen for four reasons. Firstly, reaching is commonly used by researchers studying animal motor control, so there will be literature available for inspiration and assessment of results. Secondly, it allows a physical model to be built with a manageable number of configuration parameters. Thirdly, constraining the movement to two dimensions dramatically simplifies the graphing, and therefore analysis, of results. Lastly, a reaching phase is required for nearly all manipulation tasks, and as such should not be overlooked.

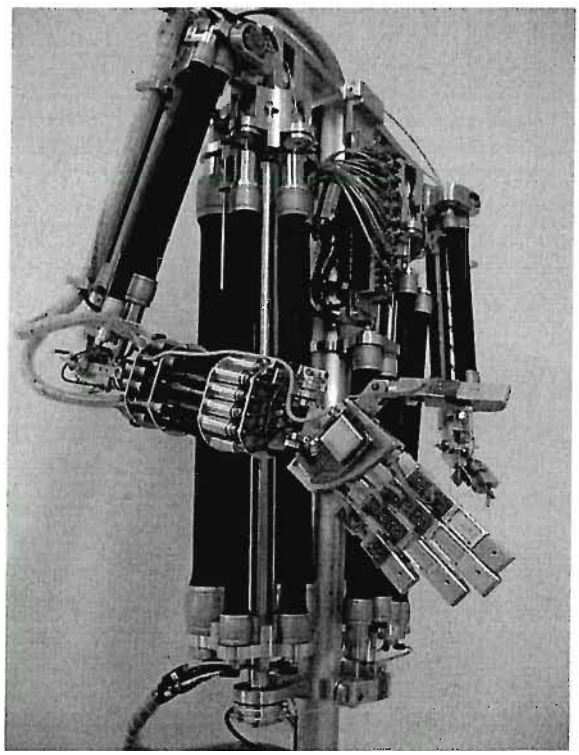
Figure 1.6 shows photographs of two research robots that are capable of compliant reaching. The Möhl (1997) arm used series elastic actuators to control a simple two degree of freedom reaching arm. In contrast Zar 5 (Boblan et al., 2003), uses FESTO fluidic muscles (deformable air muscles) with pulse width modulated switching valves. Both these robots would provide interesting platforms on which to develop compliant reaching techniques. Within the time scale of a single PhD it is not possible to build robotic machinery of a similar complexity to Zar 5, but its existence is worth noting as evidence that the control techniques explored in this thesis will have real-world applications in the medium term.



(a)



(b)



(c)

FIGURE 1.6: (a) and (b), Compliant arm driven by antagonistic series elastic actuators. SOURCE: Der bionische Roboterarm of Technische Universität Darmstadt Website (2005). (c) Zar5 robot driven by FESTO fluidic actuators. The actuators' pressures are continually adjusted using pulse width modulated values. The hand control signals are generated remotely using a data glove. SOURCE: Humanoider Muskelroboter ZAR of Technischen Universität Berlin Website (2005).

1.9 Structure of the Thesis

The following chapters explore the topic of manipulation, first from a robotics perspective and then a biological one. A preliminary modelling experiment is then presented.

Chapter 5 then introduces a more complicated arm model and justifies its parameters. The subsequent chapter presents a dynamic simulation of the arm model that is used to train a series of independent field controllers.

Chapter 7 introduces a more advanced controller model and details some potential configuration options. Finally, chapters 8 and 9 present an implementation of two key controller modules.

The work reported in this thesis contributed to the following papers:

- An approach to the simulation of robotic systems using XML-based configuration files. (Sunderland et al., 2004a) *Proceedings of DETC'04, Design Engineering Technical Conferences, Salt Lake City, UT.*
- Flexible XML-based configuration of physical simulations. (Sunderland et al., 2004b) *Software Practice and Experience.*
- A framework for biologically-inspired control of reaching motions. (Sunderland et al., 2005) *Proceedings of 3rd International Symposium on Adaptive Motion in Animals and Machines (AMAM 2005).*

Chapter 2

Artificial Manipulation

The development of artificial manipulators serves two purposes: deepening our understanding of manipulation theory and creating machines that can do useful work. From a research perspective the former is more attractive because, although it will take longer to produce commercially-exploitable results, proper development of manipulation theory may eventually allow the construction of autonomous artificial systems capable of robust manipulation behaviours. This would pave the way for more physically interactive independent robots that would further our understanding of intelligent action and perform useful work.

2.1 From Action to Actuation

For a manipulator to achieve useful work, a 'task' must be converted into a series of actions, which in turn must lead to timely activation of the actuators. It is not essential for the manipulation system to perform all these stages simultaneously, usually tasks are converted into a series of Cartesian or joint coordinates prior to manipulation. Such prior calculation may be appropriate where a manipulator frequently repeats the same task. However, where the tasks must be continuously modified to compensate for environmental variation these calculations must be performed before each action, and should therefore be considered part of the main manipulation process.

There are several approaches to decomposing a task into separate actions. However, this work will focus on the conversion of actions into actuator activations. If we let $\mathcal{A}$ be the space of all possible actions and $\mathcal{M}$ be the space of all possible actuator activations (both spaces may be continuous or discrete) we can define the mapping function, t_{am}, from $\mathcal{A}$ to $\mathcal{M}$:

$$t_{am} : \boldsymbol{a} \rightarrow \boldsymbol{m} \text{ for } \boldsymbol{a} \in \mathcal{A} \text{ and } \boldsymbol{m} \in \mathcal{M} \tag{2.1}$$

The actuators are connected to the body, and so it is usually possible to define a mapping, t_{mb}, from $\mathcal{M}$ to body configuration $\mathcal{B}$. This mapping is environmentally sensitive, (i.e., depends on v, a member of $\mathcal{V}$; the space of environmental configurations.)

$$t_{mb} : \boldsymbol{m}, \boldsymbol{v} \rightarrow \boldsymbol{b} \text{ for } \boldsymbol{b} \in \mathcal{B} \text{ and } \boldsymbol{v} \in \mathcal{V}$$

$$t_{bv} : \boldsymbol{b} \rightarrow \boldsymbol{v}$$

For navigational robots, surface slip may affect t_{mb}. Equally for manipulation systems, interaction with the environment (sustained contact or collision) will also affect t_{mb}. The final transform (t_{bv}), a property of the environment rather than the robot, defines how the world will change as a result of the body movements, bringing the goal of the manipulation nearer to, or possibly further from, completion.

The mapping t_{am} is very simple for navigational robots (consider *steer-hard-left* or *drive-straight*) compared to manipulation robots (consider *grasp-fragile-object* or *turn-object-clockwise-in-body-frame*). This results from three factors; three-dimensionality, actuator coupling and frame-transformations, which we will consider in turn.

Three-dimensionality increases the number of degrees of freedom a rigid body may possess. For a rigid body in a system, the move from two to three dimensions adds a further three degrees of freedom. The description of a vehicle driving on a surface requires only three coordinates, whereas a three dimensional, six element manipulator requires (before other constraints are applied) 36 coordinates. The situation is further complicated if one considers deformable surfaces, whose configuration cannot be fully described by a reference frame.

Manipulation systems are generally very reliant on actuator coupling; the final movement of the end-effector is the net result of all the actuators in the supporting kinematic chain. This coupling results in a non-linear mapping between joint velocities and end-effector velocity (Paul, 1981).

Finally, manipulation tasks are further complicated by the number of coordinate reference frames that must be considered. These do not relate directly to any physical quantity, but are intrinsic to most mathematical descriptions. From a design point of view, the mapping between frames is possible, assuming that we have good information about the system configuration. In the absence of such information, the mapping becomes very difficult. It is therefore very important to consider 'in what frame' a sensor reading originates. This can be more easily appreciated when we consider two pressure-sensing digits of a mechanical hand touching each other. The controller may want to know if the fingers are touching each other, or if they are pinching some intermediate free-moving relatively light body (Fig 2.1). The forces experienced are

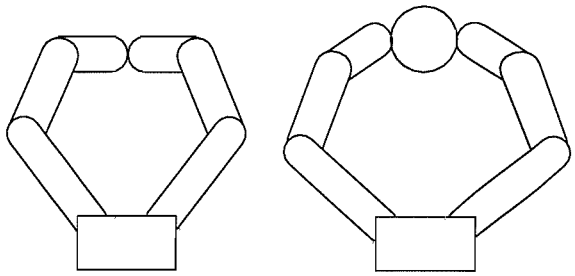


FIGURE 2.1: Two fingered hand, closed and gripping a light object

similar in both cases, assuming that the system is static and no other forces are acting on the body. The solution requires transforming the contact point for each finger into a shared reference frame, probably the palm's, and testing for their proximity. In a system where the mapping from the digit contact points to a palm frame is not fully determined (consider deformable contacts controlled by low-resolution-sensed compliant actuators), this approach becomes impossible. It may be that there are other approaches that would create similar results to a palm frame remapping with more approximate, or joint related information, but as yet they are under-developed. The same kind of frame-translation problems are encountered in all of the manipulation phases.

2.2 Conventional Approach

When considering 'robotic arms' most people imagine the devices used for welding and assembly by automated factories. This class of robots, normally termed industrial manipulators, has been commercially successful and contributed significantly to the current flexibility and quality of the manufacturing industry. They were developed from a blend of two technologies: second world war teleoperation devices for handling radioactive materials and numerically-controlled machine tools (Paul, 1981). Initially, they were taught directly using joint coordinates, either computed by hand or captured from an initial human-controlled run through. The first systems were not equipped with any form of sensor, and this lack of task-related information limited their performance. The addition of touch sensors improved things a little. However, these were soon surpassed by camera-based systems which could describe the relationship between objects using homogeneous transforms (Wichman, 1967). For this approach to work, a process called inverse kinematics, which was capable of translating homogeneous transforms into joint control values, was developed.

2.2.1 Inverse kinematics and dynamics

Denavit and Hartenberg (1955) proposed a system that could describe a rigid robotic arm as a series of homogeneous transforms. Each homogeneous transform is a four by

four matrix which allows for arbitrary rotational and translational mapping between two Cartesian coordinate frames:

$$\boldsymbol{v}_b = {}^{b}_{a}H\boldsymbol{v}_a$$

where

$$\boldsymbol{v}_a = \begin{bmatrix} x_a \\ y_a \\ z_a \\ 1 \end{bmatrix} \quad \boldsymbol{v}_b = \begin{bmatrix} x_b \\ y_b \\ z_b \\ 1 \end{bmatrix} \quad {}^{b}_{a}H = \begin{bmatrix} u_1 & v_1 & w_1 & p_1 \\ u_2 & v_2 & w_2 & p_2 \\ u_3 & v_3 & w_3 & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\boldsymbol{v}_a$ is a mapping of vector $\boldsymbol{v}_b$. The equation can perform two roles, either to describe the relationship between two points in the same space, or to define the same point as it occurs in two different spaces.

For each joint it is possible to define a transform that maps between its reference frame and the next: ${}^{i+1}_{i}R$. So for a conventional industrial arm with six degrees of freedom it is possible to write a series of transforms that map from the base all the way to the tool interface (i.e., the point of connection between the end-effector and the arm):

$${}^{6}_{0}R = {}^{1}_{0}R\,{}^{2}_{1}R\,{}^{3}_{2}R\,{}^{4}_{3}R\,{}^{5}_{4}R\,{}^{6}_{5}R \tag{2.2}$$

This defines the forward-kinematic chain, i.e., for any set of joint-angles we can calculate ${}^{6}_{0}R$. For an industrial manipulator many elements of the intermediate transformations will be trivial and so it is possible to write the components of ${}^{6}_{0}R$ using simpler equations than those derived from directly expanding Equation (2.2). For simple manipulators it is often possible to describe their workspace using a subset of the homogeneous transform, R_w. This workspace description can then be used to select target poses for the end-effector. It is then possible to equate the components of R_w with ${}^{6}_{0}R$ and solve for joint angles. These inverse-kinematic equations allow us to determine the correct joint angles to achieve a given end-effector location.

There are, however, a few complications. It is possible that the location specified by R_w does not lie within the workspace of the manipulator, in which case the inverse-kinematic equations will be insoluble. It is therefore normal to test certain parts of the equations to show that a solution is possible. When two joint axes align, the manipulator is said to be in a singular position. Mathematically this means there are an infinite number of possible joint combinations that would lead to the same end-effector configuration. Such configurations are generally avoided by applying constraints at the task planning stage. The final problem occurs when several discrete solutions exist (Fig. 2.2), i.e., a

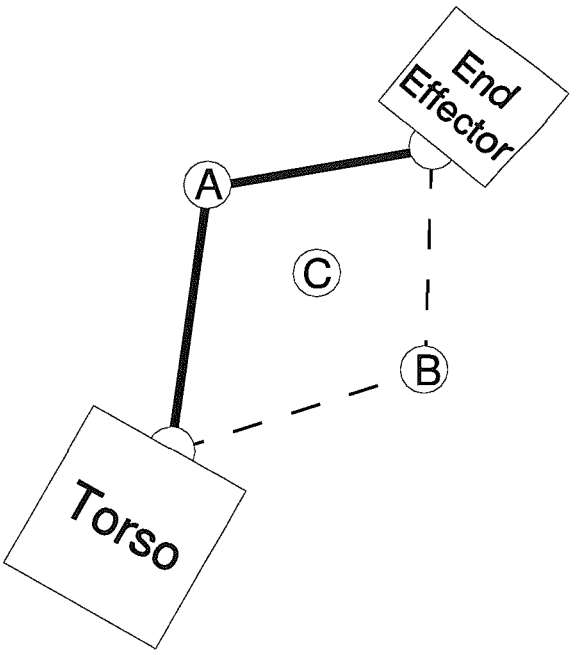


FIGURE 2.2: Plan view of reaching poses, where two discrete solutions (A and B) exist, but where intermediate solutions fail (e.g., point C).

given limb can be in configuration A or B, but intermediate solutions would be invalid (e.g., C in this case). In such situations, it is normal either to artificially constrain the robot's movement to being non-redundant or to use the 'nearest' solution to the current position, with a weighting to prefer movement of the smaller joints.

As computational power has increased, a numerically based solution for a general robotic manipulator (no closed form solution need exist) was developed (Goldenberg et al., 1985). This approach also suffered from instability around singular configurations.

For some tasks it is only necessary to complete point to point movements without considering intermediate stages. However, many tasks require more advanced 'trajectory management' or 'contouring'. This allows for smooth acceleration and deceleration of the end-effector, which is essential for tasks such as painting and welding. For a simple kinematic chain, it was possible to calculate a Jacobian matrix; one that maps an infinitesimal movement in joint space to an infinitesimal movement of the end-effector. To execute smooth trajectory control, the movement can be expressed as a series of splines in time and global coordinate space. At each instant, the Jacobian is calculated and then used to convert the desired end-effector velocity into appropriate joint velocities. This form of control is also sensitive to a second form of singularity, with joint-speed demand tending toward infinity as the manipulator approaches the edge of the workspace.

A separate Jacobian must be calculated for every point of interest, which means that the analysis of a pinch grip is significantly simpler than that of an enveloping grip,

where a separate Jacobian would have to be calculated for each finger joint. Jacobian matrices can also be used to calculate the effect of a static force being transmitted down a kinematic chain (Craig, 1986, p. 175).

2.2.2 Grasp metrics

Given an accurate model of an end-effector and a target object, it is possible to calculate the wrench (i.e., combination of orthogonal and torsional forces) applied at each contact point. The wrenches can be analysed to produce various descriptions of the grasp. Although there are many metrics defined, Shimoga (1996) showed that they could all be considered as combinations of four basic properties:

1. **Dexterity:** defined as the ability of a grasp to achieve sub-objectives while holding the object.
2. **Equilibrium:** a measure of how well the forces and torques acting on the object are balanced.
3. **Stability:** a measure of how disturbance-induced errors in applied force and torque decay with time.
4. **Dynamic-behaviour:** a measure of how the fingers interact with the object in terms of their natural frequencies and damping ratios.

A classical approach to grasp planning would be to generate a range of potential end-effector configurations and use a combination of grasp metrics to select the optimal one. This can be a computationally intensive task and grasp metrics have to be efficiently designed. For more details about conventional approaches to manipulation, see the overview by Okamura et al. (2000).

These metrics are entirely reliant on values derived from an abstract geometric model of the grasp, rather than on measurements of an actual grasping process. As such, it would not be possible to use them to assess the performance of a real robot directly. Although reaching behaviours are significantly simpler than those used for grasp management, stability and dynamic behaviour are essential to both. To facilitate future comparison, alternative model-free metrics should be developed that are suitable for both simulated and real robots.

2.3 Behaviour-Based Robotics

The traditional approach can be seen as a variant of the *sense-model-plan-act* (SMPA) framework for intelligent action that has been proposed by classical AI theory. Unfortunately, such approaches must assume access to accurate geometric descriptions of the manipulator body and its environment, something that is much harder in practice than in theory. SMPA works well when the transformation from sensing to model building is trivial and the world does not change much during an SMPA cycle. However true this may be for a game of chess, it does not hold when dealing with the washing-up. The success of SMPA based manipulators is largely due to reliance on pre-calculation of their body and environment models. This unfortunately limits them to applications where such pre-calculation is possible, and does not need significant updating, for example a laboratory or production line. For such a system to work in an unstructured environment, it must be capable of selectively disregarding and rebuilding its model as the world around it changes. Referred to as *the frame problem*, this process is computationally intractable (Dennett, 1984). Even if a system maintains a reasonable approximation of the environment, it will always contain errors. The system must be able to deal with this misalignment.

Brooks (1991a,b) proposed an alternative approach to developing systems capable of intelligent action. His approach discards the SMPA framework and replaces it with a system comprised of many parallel independent 'behaviours'. Each behaviour maps directly from sensor readings to actuator activations, with only very limited 'message-passing' interaction between them. Although this approach seems unnecessarily restrictive, it has been very successful in producing robust mobile robotic behaviours that respond in a sensible and timely way to their environment. Such systems, by their very nature, perform no planning, and have minimal internal state, but instead perform complex procedures by responding only to their current sensor state. For an introduction to the field, see Arkin (1998).

2.3.1 Embodied intelligence

Although Brooks helped formalise behaviour-based robotics, he was building on a tradition that dates back to Grey Walter (1950, 1951, 1963) who developed a series of small mobile 'tortoise' robots. Even though these tortoises were controlled by simple, two valve, analogue circuits, they were capable of displaying complex 'behaviours'. One tortoise, dubbed 'Machina Speculatrix', performed phototaxis (i.e., following lights) but was averse to very bright lights until its battery began to run low. Since a strong light source was placed inside its recharge station (or 'hutch'), it would return there only as required. Speculatrix demonstrates two concepts: emergent behaviour and ecological niches. Its behaviour is considered emergent because it results from the interaction

between the robot's electronics, mechanics and environment, and would be sensitive to significant changes in any of them. This contrasts with the traditional view of intelligence as being 'seated' in the brain, with the body merely being a convenient way of interacting with the world. For more detail about Grey Walter's work see Holland (1996, 2003).

A robot's 'ecological niche' defines the range of environments within which it can operate successfully. For Machina Speculatrix, this was a room with a flat floor and a recharge station. From a manipulation perspective, we often glibly talk about 'general purpose' end-effectors – something that in the strictest sense will never exist. The human hand itself is very far from the ideal of a general-purpose manipulator. It can function in only a limited range of temperatures, air-pressure, acidity and radiation levels. It only functions sensibly with a limited range of object sizes (try picking up a water melon single-handed or fixing a clockwork watch without a pair of tweezers). Equally the hand has a limited range of forces that it can apply and that it can withstand. These limitations combine to form the hand's environmental niche. A robotic hand may be designed for the same niche as its human counterpart, but equally may be designed for demolition work, satellite maintenance or even Martian environments. Pfeifer and Scheier (1999) cover the foundational concepts and the important design considerations for embodied intelligence.

2.3.2 The value of learning

Often characterised as advocating absolutely no representations of the world within a behaviour-based robot, Brooks (1991) counters that "individual layers extract only those *aspects* of the world which they find relevant–projections of a representation into a simple subspace". He is however determinedly against traditional AI schemas and explicate representations of goals. So how can these representations be learnt?

Generally 'learning' can happen in two ways: per species (evolution) or per individual. Within this research, learning will be used to refer solely to individual-level improvement. Although evolutionary development is demonstrated by all animals, individual learning is also present to varying degrees, usually correlated to complexity of animal behaviour. There are at least four classes of things that a behaviour based robot can learn (Brooks, 1991):

1. representations of the world that help in some task;
2. aspects of instances of sensors and actuators (sometimes called calibration);
3. the ways in which individual behaviours should interact;
4. new behavioural modules.

Behaviour based robots have demonstrated many of the above (with the exception of 4). As discussed, for manipulating robots, the challenge of converting a desired action into appropriate actuator activations is far from trivial. Although traditional techniques may be appropriate for rigid robots that work in highly structured environments, the real world will require a more robust and flexible approach. Reduced reliance on internal model construction, if possible, will go some way to meeting this need, but it seems likely that some form of automatic learning of the mapping between desired action and actuator activation will be necessary. Within the above groupings, this would be considered a form of calibration, but it is significantly more complicated than that classification implies.

As well as having to learn to handle the abstract transformations involved, a robot would have to deal with the idiosyncrasies of its own particular sensors and actuators. Elliott and Shadbolt (2001) proposed a neurologically-justifiable model that allows a system to compensate for such variation which may prove a useful foundation for this part of the learning process.

2.4 Conclusion

Robotic systems that perform articulated manipulation using compliant actuators currently lack a satisfactory motor control methodology. Such controllers need to be computationally efficient, robust to environmental disturbance and capable of integration with other control architectures. It is possible that behaviour-based and biologically-inspired techniques may have a role to play in the development of such controllers. The following chapter explores four biological motor control theories, with particular reference to reaching behaviours.

Chapter 3

Biological Perspective

When considering biological systems as inspiration for artificial control systems, one must carefully consider the differences between the natural and artificial morphologies, e.g., overall physical structure, actuator dynamics, transducer characteristics and controller substrate. Where possible one should distinguish those properties that the system is able to exploit from those for which it must compensate. Naturally, there will be some overlap, but it is important to avoid seeing imitation as the objective, without trying to understand the role of the various properties. To establish the context for future discussions of control techniques, this chapter begins by describing first the overall neurological context and then the mechanical context for mammalian motor control. It concludes by presenting two control techniques which may be suitable for adaptation to autonomous robots. As this is an engineering thesis, the treatment of this material will naturally be somewhat simplified.

3.1 Neurological Context

The mammalian central nervous system (CNS) can be divided into four distinct portions; the cerebrum, the cerebellum, the brain stem and the spinal cord, as shown in Figure 3.1. The cerebrum is a large soft area at the front of the brain which is divided into two hemispheres. It is particularly developed in primates and is thought to control higher order processing tasks. The cerebellum is a denser portion located at the base of the cerebrum. It is primarily a movement control unit and is well connected to the spine and cerebrum. Both the cerebrum and the cerebellum are mounted on the brain stem, which performs two main functions; communication to and from the spine, and maintenance of basic systems like breathing and body temperature. The spinal cord provides a link between the CNS and the peripheral nervous system (Bear et al., 2001).

The peripheral nervous system (PNS) extends from the dorsal (back) and ventral (front) roots in the spinal column. The PNS can be considered to comprise two parallel

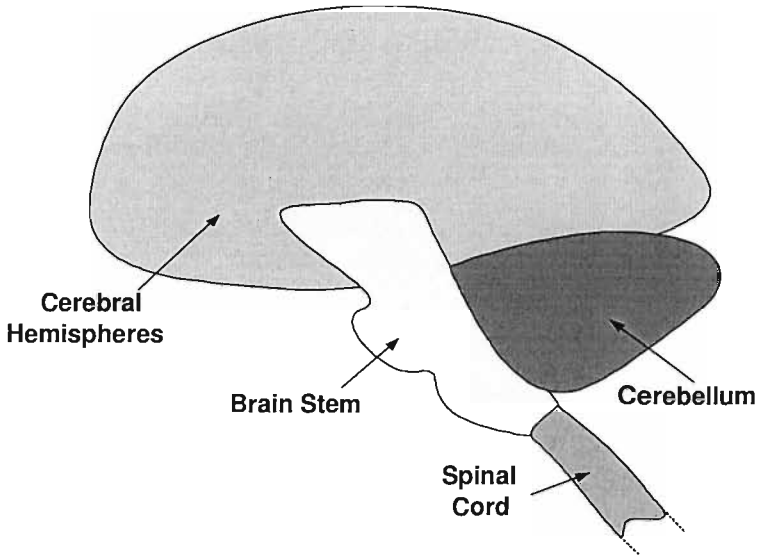


FIGURE 3.1: Cross section of the human brain, showing the arrangement of the cerebrum, cerebellum, brain stem and spinal cord.

LEVEL	FUNCTION	STRUCTURES
High	Strategy	Association areas of neocortex, basal ganglia
Middle	Tactics	Motor cortex, cerebellum
Low	Execution	Brain stem, spinal cord

TABLE 3.1: The hierarchy of biological motor control, from Bear et al. (2001).

systems; the somatic PNS, which controls voluntary movement, and the autonomic nervous system, which maintains involuntary systems, including blood vessel dilation and internal organ operation.

The central motor system is organised into a hierarchy of control layers as shown in Table 3.1. Although this hierarchy will not necessarily relate directly to an appropriate control structure for an artificial manipulator, it does suggest that this approach is worth considering. Although this may first appear to be a horizontal SMPA framework, it can be viewed as a more vertical sense-act framework if each layer is considered to be a separate behaviour unit. Where this diverges from a Brooksian model is in the amount of inter-behavioural communication and the amount of feed-forward control. If we consider the cerebellum, although it does receive commands for voluntary movement (vertical input), it integrates this with direct sensor information (horizontal input) to maintain balance. Manipulation tasks sometimes require *ballistic movements* (i.e., a movement that must be planned and subsequently enacted without time for significant in-process feedback and correction). The learning for ballistic processes must happen after the event and therefore requires a mechanism for working out which parts of the movement contributed to, and which detracted from, the action's overall success. This

type of system has been studied formally and is usually managed using a process called reinforcement learning (Sutton and Barto, 1998).

3.2 Muscles

One of the most exciting things about biological motor control is the distinctive nature of the actuators used, namely muscles. Traditional robot drives are stiff (i.e., contain very limited compliance) and relatively dense, with a low power to weight ratio. This has led to many manipulators being designed with a rigid base, so that heavier parts of the system can be removed from the arm itself.

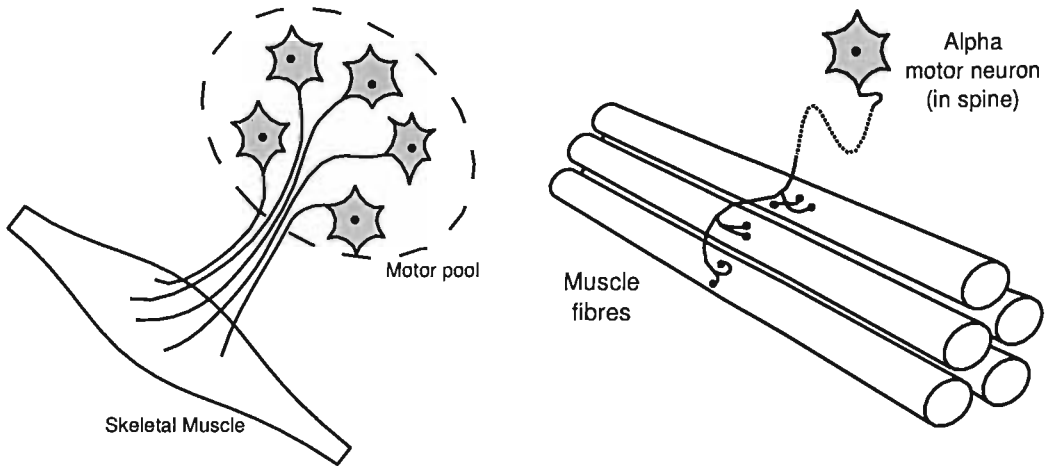
In contrast, the human arm is substantially self contained, with the exception of a few shoulder muscles, power supply and main control centre. Also, rather than being heavy and stiff, the arm itself is light and has mechanically variable compliance (as opposed to some advanced robotic drives, which can use feedback to simulate variable compliance). This mechanical compliance has natural advantages both in terms of protecting the manipulator and its environment as well as the potential to conserve energy.

So it would appear that natural style body mechanics may have advantages over traditional drives when it comes to designing autonomous robots, but are they achievable? Sadly current actuation technology lags significantly behind that achieved by nature, as the meagre performance of the robotic contenders in the first round of Yoseph Bar-Cohen's *Arm-wrestling Match of EAP Robotic Arm against Human* so aptly demonstrated (AMERAH Website, 2005). That said, there is significant research activity focused on the development of novel actuation methods, and it is the author's hope that this will yield practical results in the next few years.

3.2.1 Motor units

A muscle contraction is the result of the action of many motor units, each of which contains a single control neuron (termed an *alpha motor neuron*) and a bundle of muscle fibres which it activates. The collection of alpha neurons that controls a single muscle is termed a motor neuron pool (Fig. 3.2).

There are two mechanisms through which the central nervous system controls the amount of muscle contraction. The first is to vary the firing rate of the alpha neurons and the second is to recruit muscle units according to size. Some muscle units have up to 1000 fibres per alpha neuron (in the leg for example), while some have as few as three (e.g., finger or eye rotation muscles). Within any muscle, small units are selected first, with the larger units only being recruited as the amount of force required increases. This naturally leads to a system that is capable of great precision at low loads, yet is still capable of generating large forces, albeit at a lower accuracy.



(a) Motor pool (b) Motor unit

FIGURE 3.2: (a) skeletal muscle is made from many parallel motor units. A group of alpha neurons that control a single muscle is termed a *motor pool* and is located in the spine. (b) each motor unit has an alpha motor neuron and a bundle of between approximately three and a thousand motor fibres. SOURCE: Bear et al. (2001, Fig 13.6, p. 442)

3.2.2 Renshaw cell inhibition

Work by Akasawa and Kato (1990) and Uchiyama and Akazawa (1998) developed a neural network that demonstrated that muscle unit recruitment could be controlled by feedback mediated via a series of *Renshaw* cells. This would allow the CNS to use only one control signal per muscle (rather than a different one per alpha neuron), a configuration referred to as the common drive hypothesis (DeLuca et al., 1982). In their model, Renshaw cells are activated from the same signals as the alpha neurons, but act to inhibit other local motor neurons (including the original alpha neurons). This creates a multi-loop negative feedback system that serves to linearise the lumped muscle response.

From an engineering perspective, this kind of network will only have application in configurations where actuators are composed of many independent strands. The work presented in this thesis assumes that any actuators are either naturally homogeneous, with a single control input, or have been configured so that they may be treated as such.

3.2.3 Spindle and Golgi feedback

Skeletal muscles contain specialised sensory structures called muscle spindles. Each spindle runs the entire length of the muscle and has a group of high speed sensory axons wrapped around the fibres in the middle of the spindle, that give feedback about changes in the muscle's length. The feedback from the spindle is connected to virtually every

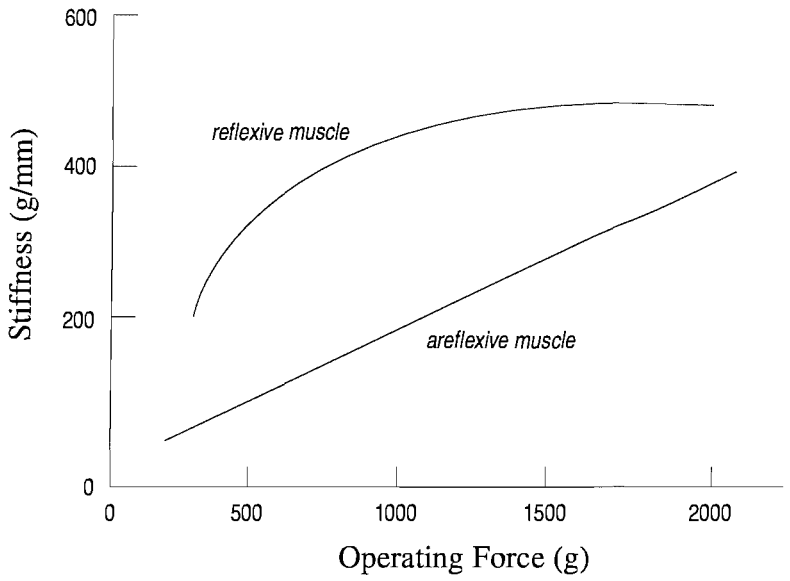


FIGURE 3.3: Comparison of the stiffness (in grams per millimetre) to operating force (in grams) of a reflexive and areflexive muscle (cat soleus), from Hoffer and Andreassen (1978).

alpha motor neuron in the muscle's control pool. This feedback plays an important role in the myotatic reflex (*myo* from the Greek for 'muscle', and *tatic* from the Greek for 'stretch'), which is generally considered important for maintaining muscle tone (co-activation of antagonistic muscle groups). To maintain spindle tension, and hence feedback signal, over a range of muscle lengths, a set of intrafusal fibres is used to contract/relax the ends of the spindle. These intrafusal fibres are controlled by a second class of motor neuron (gamma motor neurons). Varying the amount of spindle tension will affect the behaviour of the myotatic reflex, and could be seen as leading to a variation in the rest length of the muscle.

A second form of muscle feedback is provided by Golgi tendon organs, which measure the amount of tension developed by the muscle. Although less thoroughly connected than the spindle, the Golgi sensor axons inhibit some of the muscle alpha neurons, creating the reverse-myotatic reflex, which protects the muscles during heavy loads, and fragile objects during manipulation.

Using work by Hoffer and Andreassen (1978, 1981), which explores the stiffness/force characteristic of reflexive (i.e., with spindle and Golgi feedback) and areflexive muscles, Shadmehr and Arbib (1992) demonstrates that reflexive muscles are best modelled as non-linear springs with *variable rest length*, rather than as non-linear springs with *variable stiffness*. Figure 3.3 shows a comparison of the stiffness of reflexive and areflexive muscles plotted against operating force. For a reflexive muscle stretched to length λ,

the stiffness developed, $\frac{d\phi}{d\lambda}$, can be approximated as follows;

$$\frac{d\phi}{d\lambda} = k(1 - \exp(-\alpha\phi)) \tag{3.1}$$

The model has two constant parameters, k and α. It can be solved to give the force developed, ϕ, in terms of λ and an integration constant β, giving:

$$\phi = \frac{1}{\alpha}\ln\big(\exp(\alpha k(\lambda - \beta)) + 1\big) \tag{3.2}$$

For any given length, the only way to change the force developed is to vary β, which can therefore be thought of as a controllable parameter (Shadmehr and Arbib, 1992).

3.3 Control Models

It has been argued that it is unlikely that our brain solves the precise equations used by traditional inverse kinematics to control our limbs (Alexander and Crutcher, 1990). So what are the alternatives? As previously discussed, biology presents many systems that have learnt to control articulated structures (i.e., vertebrates). In this work we will focus on mammals as this class contains many of the species which are capable of complex manipulation.

3.3.1 Equilibrium control

Originally proposed by Feldman (1966, 1986), equilibrium point control (EPC) suggests that rather than driving the muscles directly, the control of animal reaching movements is mediated via the selection of an equilibrium point (EP) to which the end-effector travels (Bizzi et al., 1991, 1992). Initially, this seems very similar to differential position control, and indeed has been approximated as such (Hsiao et al., 2003). Such approximations, however, used non-elastic actuators and traditional inverse kinematics to translate between world and joint coordinates and as such suffer from their inherent problems. There are two main models of equilibrium control: α-model and λ-model. These two alternative models make different assumptions about the control signals sent to the muscles. The α-model assumes that the control signals do not change during a reaching motion, i.e., when a target point is selected the muscle's rest-lengths are set once and the rest of the motion is entirely the result of the combined muscle dynamics. It is clear that if such an approach worked, it would significantly reduce the amount of processing the brain would have to do during a motion.

The role of α-model EPC in animal motor control is supported by clinical trials that studied the forearm movement in normal and deafferenated (surgery that lesions the proprioceptive feedback at the base of the neck) rhesus monkeys conducted by Polit and Bizzi (1978). The monkeys sat in a primate chair with their right forearm fastened to an apparatus that permitted flexation and extension in the horizontal plane. During the trials the monkeys, whose view of their arm had been obstructed, were trained to point toward a light (one of several arranged in a semicircle at 5° intervals) and maintain that posture to receive a reward. A motor was attached in series with the elbow clamp and was used to apply randomised disturbing forces in some trials.

Monkeys performed the test well, even after surgery, which implies that the correction to the disturbing forces was performed at a spinal level. However, the normal monkeys were able to adapt far more successfully when the elbow fixing was moved with respect to their body. This suggests that the short and long proprioceptive loops are not required for learnt pointing tasks but are necessary for adaptation to significant changes in the task dynamics.

The alternative λ-model assumes that the control signals vary continuously during the arm movement. It suggests a compromise between direct velocity control of the arm and the α-model's complete reliance on arm mechanics. Rather than altering the rest-lengths of the muscles to move the arm's equilibrium point directly to the target location, the rest-lengths are changed smoothly, so that the arm's equilibrium point moves with constant rate in body space, hence the λ-model is sometimes referred to as equilibrium trajectory control (ETC). Equilibrium trajectory control still relies on the mechanics of the muscles, but only to convert a constant-rate equilibrium point movement into the smooth bell-shaped velocity movements typical of vertebrate reaching motions.

If we return to our formalisations for a moment, ETC suggests the mapping from action to actuation, t_{am} (Eqn. 2.1), may be mediated by continuous redefinition of an equilibrium point. If we define the space of potential equilibrium points as $\mathcal{E}$, we can then express this mapping:

$$t_{ae} : \boldsymbol{a} \rightarrow \boldsymbol{e} \quad t_{em} : \boldsymbol{e} \rightarrow \boldsymbol{m} \quad \text{for} \quad \boldsymbol{e} \in \mathcal{E}$$

Work by Flash (1987) attempted to use ETC to model human reaching behaviours. A series of reaching tasks was performed by three human subjects whose view of their arm had been obstructed and who were not given any feedback during the trial. It had previously been observed that the kinematic features of unconstrained planar point-to-point movements are not sensitive to workspace region, being roughly straight with a bell shaped velocity profile (Morasso, 1981; Abend et al., 1982). Although the hand movement was straight, to achieve this the arm joint velocities must follow a significantly more complex path. Flash's research showed that the hand trajectories contained

distinct curvatures, depending on workspace region. She then developed an ETC model of human reaching that incorporated previously measured static human arm stiffness readings. The model's results accurately predicted many of the features measured by the human trial, including the characteristic deviations from straight-line movement for each workspace region.

Neither EPC nor ETC are universally accepted explanations of the neurological control of reaching movements. Two key objections are raised by Lackner and Dizio (1994) and Gomi and Kawato (1996).

The first objection is based on studies that involve reaching movements performed in a slowly revolving room (the rotation of which the subject was unaware). Results from this trial suggest that subjects use real time feedback to adjust their reaching movements, which runs against the conclusions drawn from the deafferenated monkeys. It can be argued that a spinning room is not the right context in which to study the movement control of primates as they evolved in an environment substantially devoid of Coriolis forces. However, it is possible that this unusual context reveals control that would never be visible in a normal environment. Studies involving a force feedback manupendulum (that also creates velocity dependant forces) have shown similar results (Shadmehr and Mussa-Ivaldi, 1994).

The second objection comes from studying human arm stiffness during reaching movements. Reaching tasks were performed whilst grasping a low-friction low-inertia, yet stiff, air-bearing manupendulum. During some movements, the arm was disturbed and its deviation measured. The amount of deviation could then be used to measure the level of stiffness developed during the movement. This is important because as stiffness increases ETC tends toward conventional position feedback. These studies have shown results that do not fit with some ETC models and hence undermine it as an explanation of motor control. It has been countered that the muscle models considered were overly simplified and the results would align with ETC predictions if more authentic muscle models were used (Gribble et al., 1998).

Both these objections point toward a more centrally planned approach that uses real time feedback to correct for deviations. The following section outlines a common formulation of this approach.

Although ETC may play a significant role in the management of redundant systems, it should be noted that most of the experiments to date have involved constrained limb movements that have been artificially made non-redundant. This simplifies the mathematics involved but may be hiding a weakness in the approach. Typically two types of reaching task are considered: point-to-point and primitive-shape-drawing. These both present significant challenges, but are trivial compared to many basic manipulation tasks.

3.3.2 Force control

Work within the cybernetics field has more recently been focused on an *internal representation* approach, sometimes called the *force control hypothesis*. Originating more than 20 years ago (Hollerbach, 1982), this approach assumes that inverse dynamics plays an explicit role in neuromuscular control. More recently, the theory has been extended to use forward models to assist in predictive control (Conditt et al., 1997). Control is assumed to follow a three stage process:

1. Motion is planned kinematically and required forces are specified centrally.
2. Actuator activations are calculated using inverse dynamics.
3. Forward and inverse kinematics models are used for predictive control.

Such approaches do not normally suggest neurological details, but use behavioural studies of reaching in unusual environments to support the notion that such models exist and are re-trainable. Critics from the equilibrium point community have refuted the viability of this approach suggesting that it does not explain certain reflex responses satisfactorily and, furthermore, is not the only explanation of the data presented (Ostry and Feldman, 2003).

It is interesting to note how closely the stages outlined above align with conventional robotic control approaches. This closeness has lead some to criticise the more recent integration of biological and engineering research endeavours, suggesting that insights from the robotics fields are at best distracting and at worst misleading when applied to the understanding of biological systems (Balasubramaniam and Feldman, 2001; Latash and Feldman, 2004).

From the point of view of this work, force control does not present much useful insight, as it draws heavily from conventional control theory and does not readily suggest any low level implementation strategies. That said, some force control literature does suggest that λ-model control could be used to simplify the control task, with some form of force control being used to compensate for the dynamics of the manipulated objects (Mah and Mussa-Ivaldi, 2002). It is hard to see how these conflicting approaches could be combined, but the combination may yield useful results in the future.

3.3.3 Convergent force field control

A review of this topic is presented by Bizzi et al. (1991), Flash and Sejnowski (2001) and Mussa-Ivaldi and Bizzi (2000). Studies based on deafferenated frogs show that it is still possible to elicit coordinated muscle movements by direct stimulation of the spine

(Giszter et al., 1993). This implies that the spine is actively involved in this coordination, rather than just being a relay for signals from the brain stem. By varying the region of the spine stimulated, distinct *force fields* of muscle activation were observed in the leg muscle. That is to say, if the stimulation of the spine remained constant, whenever the leg was moved to a given location, it would always exert the same force with the same direction. A range of force fields was analysed, and through a variant of principal component analysis, it was discovered that 94% of the fields could be constructed from the vector superposition of just five 'basis' fields (d'Avella and Bizzi, 1998). Given the right control system, this could allow for just five neurons to control the gross actuation of the entire leg. In terms of our previous analysis, this means that the leg is not being controlled in equilibrium point space, $\mathcal{E}$, but rather in field space, $\mathcal{G}$, where $\boldsymbol{g}$ is a vector of the gains for each of the basis fields:

$$t_{ag} : \boldsymbol{a} \rightarrow \boldsymbol{g} \quad t_{gm} : \boldsymbol{g} \rightarrow \boldsymbol{m}$$

Irrespective of whether convergent force field control (CFFC) accurately describes the vertebrate reaching control process, before attempting to use it in an engineering context it is important to consider what potential benefits it may offer.

CFFC allows the actuators of the manipulator and end-effector to be treated as a whole and imposes limitations on the ways in which they can be controlled. This may help mask the problems presented by redundant configurations, effectively creating preferences of pose at a very low level.

If we assume that the function mapping the force field gains to the actuator activations (t_{gm}) is well designed, it removes the potential for an action to be expressed in a way that is outside the mechanical range of the manipulator. It is, however, quite possible for the field space to express a force demand that is beyond the capabilities of the actuators. Effectively field gain space is automatically bounded to be within the working volume of the robot. This removes the requirement to test that the incoming demand is within the arm's workspace, and the requirement to have a default policy to implement when it is not.

It may be possible to increase the orthogonality of $\mathcal{G}$ with respect to actions, so rather than treating it as smooth, $\boldsymbol{g}$ would generally have only a small number of non-zero components. For example a pose could be comprised from a *close-hand* and *extend-index-finger* field. The first would affect all the fingers and second would counteract the action of the first for only the index finger. This may result in an action-orientated meaning for each element of the control vector.

As the mechanical and sensor performance of the system changes over time it may be possible to adjust the fields to compensate. This would allow higher level control behaviours to be used without continuous modification. There have been studies on

human adaptation to different reaching environments which suggest a similar process is happening (Shadmehr and Mussa-Ivaldi, 1994). Lastly, if the purpose of each field can be determined, there is a small possibility that a behaviour could be shared between robots with different mechanical configurations but similar basis fields.

3.4 Conclusion

There has been, and will continue to be, much debate within the cybernetics community as to which of the competing models best describes the reaching control process used by vertebrates. The currently available models all have data supporting and contradicting them.

From an engineering perspective the study of this kind of literature has more value as inspiration than for use in making hard and fast conclusions. Even if an accurate model of vertebrate reaching is developed, there is no reason to assume that such a model would be in any way optimal. It would, however, be at least worth detailed consideration. The following chapters explore two of the techniques outlined (CFFC and ETC) and present simulations of them using simplified actuator models.

Chapter 4

Static Modelling of Convergent Force Field Control

This chapter presents a static implementation of convergent force field control (CFFC). The arm model presented is significantly simpler than that commonly used in cybernetic modelling studies (Hogan, 1985; Lukashin et al., 1996a,b), but does model the variation of the actuator's moment arms. This is a feature that is normally overlooked but that may have important implications for stability (Shadmehr and Arbib, 1992). A schematic of the simplified arm is presented in Figure 4.1.

To explore the static model, experiments were performed which compared three proprioceptive configurations in terms of computational efficiency. The objective was to train an MLP to convert the proprioceptive information into actuator activations that would create forces at the tip of the arm that changed smoothly across the workspace. To train the MLP, a set of training input and output vectors was required, which were generated by a two stage process. The first stage used a mutation hill climbing algorithm to calculate the actuator activations that would best align the force at the tip of the arm with the desired field pattern at each point. The second stage used these learnt actuator activations to train a range of neural networks. The following sections explain the procedure in more detail, and a summary of the process is given in Figure 4.2a.

4.1 System Model

The simplified arm model used in this initial modelling work has four actuators and is built from two symmetrical stages. The configuration presented was considered the simplest form of compliant actuator driven articulated structure that was capable of two dimensional movement, and was therefore chosen for this initial study. The approach used to calculate the force at the tip of the arm is presented in Appendix C. The

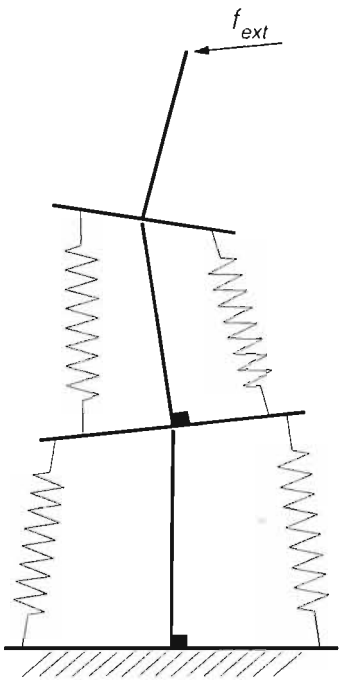


FIGURE 4.1: Simplified arm model. The springs shown represent series-elastic actuators, (springs with a motor driven base offset, allowing them to exert a range of forces at any given length). The structure comprises 3 T-frames, but as the base frame is rigidly clamped, there are only 6 moving components (2 T-frames and 4 actuators). The simulation process aims to control the force exerted at the tip of the arm, f_{ext}.

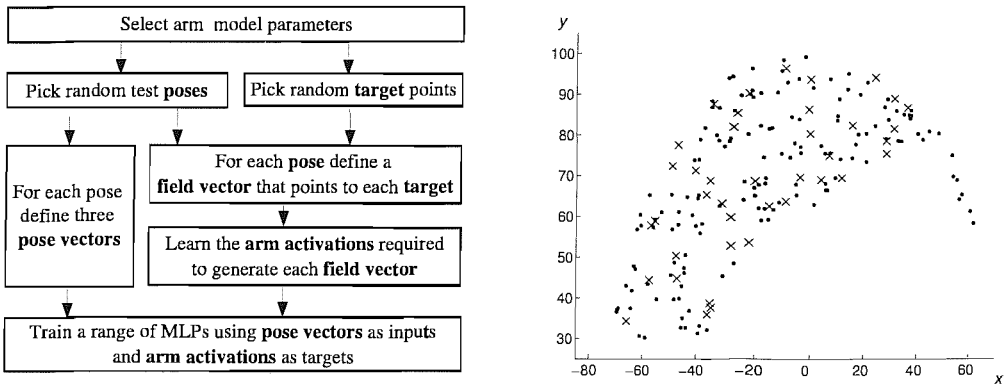
equations are developed in a way that would allow them to be extended to systems with more stages.

As previously discussed, the non-linear nature of muscles has been demonstrated to have important effects on system stability (Shadmehr and Arbib, 1992). In order to maintain the potential for real-world implementation, the actuators have been modelled as linear springs with constant stiffness (k) and variable rest length (d):

$$f = k(l_{\text{ext}} - d) \tag{4.1}$$

This more closely matches the characteristics of a series-elastic actuator, while providing the possibility that the final system could achieve similar performance to biological systems by using non-linear feedback. The model is constrained only to generate forces when extended and not when compressed.

In this chapter, the field controller is implemented using a fully connected MLP, with sigmoid activation of the hidden layer and linear activation of the output nodes. The feedback from the arm model is normalised and the MLP output is scaled up to match the input ranges of the actuator models.



(a) Simulation work-flow

(b) Targets and test points

FIGURE 4.2: (a) summary of the simulation process undertaken. (b) arm's workspace showing the 40 randomly chosen targets (crosses) and 150 test pose endpoints (dots). Note that the workspace is not symmetric because the joint rest positions are not aligned with the arm links.

4.2 Selecting Target Positions and Test Poses

A range of target positions and initial test poses was selected randomly from within the workspace of the arm. These were chosen uniformly with respect to Cartesian workspace coordinates, and then rejected if they fell outside the workspace of the arm. The joints were restricted to lie within ± 1.45 radians of their rest positions. This reduced the likelihood that the selection of target points or test poses would bias the learning stages. Figure 4.2b shows the selected targets and test poses. For the test poses, only the location of the end point is plotted, even though the entire pose was actually defined. For each pose, the joint angles required were calculated using conventional inverse kinematics.

4.3 Calculating Ideal Field Vectors

A field controller is responsible for generating a single target field, which must therefore be selected before training can begin. The literature shows that these can be quite varied (Bizzi et al., 1991), but for this work, a simpler sub-set of fields was used. The fields were chosen such that, at each point, the field vector should be directed towards a single target point with a magnitude proportional to the distance from the point:

$$\boldsymbol{f} = G(\boldsymbol{e} - \boldsymbol{o}) \tag{4.2}$$

where $\boldsymbol{o}$ is the location of the target point, G is a scaling constant and $\boldsymbol{f}$ is the ideal field vector at test point $\boldsymbol{e}$. Figure 4.3 shows two examples of ideal fields. At this

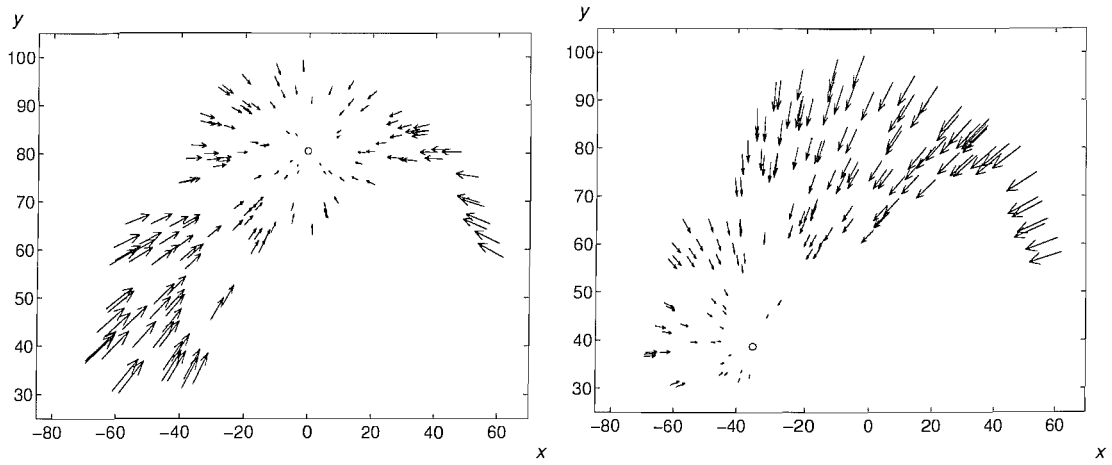


FIGURE 4.3: Two example sets of ideal field vectors. The circles mark the target point for the respective fields. Each arrow represents the ideal field vector at that test point.

point it is worth saying a few words abouts the units used in this simulation. In the actuator equation (Eqn. 4.1), there are two fundamental units (length and force) and one composite unit (stiffness). The same fundamental would naturally be used in the field equation (Eqn. 4.2). As these equations, taken alongside a geometrical description of the arm, describe the complete system, there need be no physical grounding to their values. The value of actuator stiffness is therefore only meaningful with respect to the value of G, which is held constant throughout.

4.4 Learning Arm Activations

For each of the test points a variant of mutation hill climbing method was used to find the actuator activations that generated a force at the arm tip that was best aligned with the ideal field vector. While searching for sources of error in the training data, a clear limitation of the hill climbing method was exposed. The static arm model used in this work was significantly more sensitive to cross-activation of muscle pairs than to co-activation. This meant that there was little training pressure on co-activation, effectively leaving two unconstrained variables in the training process, which were randomly disturbed by the hill climber. As the actuator activations were going to be used to train the MLP controller later, it was important that the random variation was removed. This was achieved by imposing an extra constraint on the hill climbing process that ensured constant co-activation throughout the workspace.

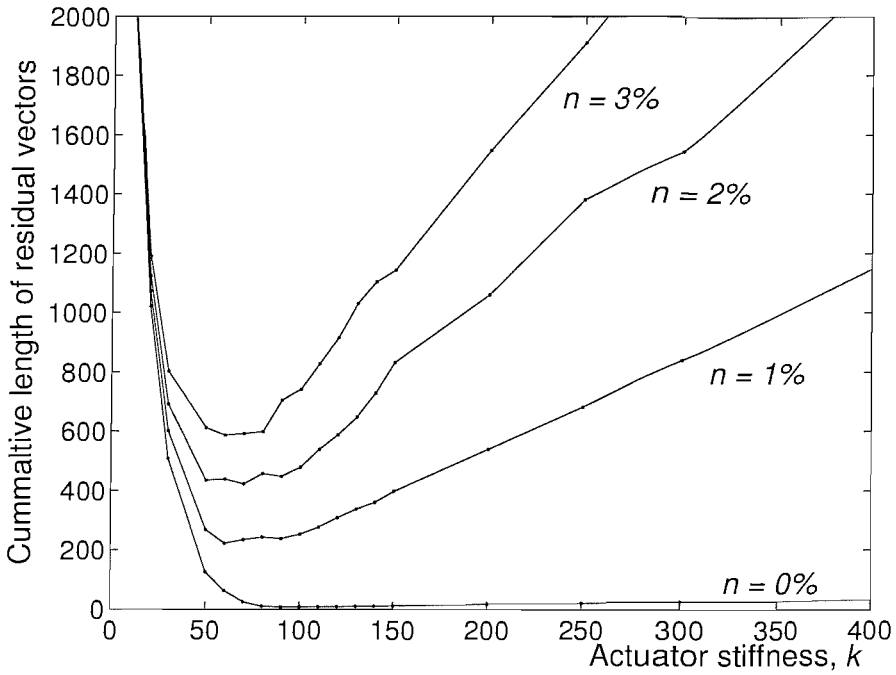


FIGURE 4.4: Sensitivity to actuator demand noise, n, of the arm models, configured with a range of actuator stiffnesses, k. The same stiffness is used for all actuators in the arm model. Due to the nature of the model the stiffness units are non-dimensional.

4.5 Arm Model Retuning

During early attempts at training the MLP field controller, structured errors were detected in the generated field that were many orders of magnitude greater than the error between the target actuator activations and those generated by the MLP. To determine how sensitive the actuator activations were to errors, a set of trained actuator activations was disturbed by various amounts of noise, and the error measured in the resultant field. The noise was a uniform disturbance of the actuator activations by a percentage of mean activation levels. Repeating this process while altering several of the arm model's parameters showed that in the region above approximately 100 stiffness units, the sensitivity to actuator noise was roughly proportional to actuator stiffness. Figure 4.4 shows the results for a range of stiffnesses. The proportional nature of the noise sensitivity has been confirmed over a larger range, but these have been excluded from the graph for clarity.

From this graph it becomes clear that when there is no additional noise, actuator stiffness does not have a significant effect on overall error. In contrast, when actuator noise is present, there is a strong correlation between stiffness and overall error. When k is approximately equal to 100, the overall error is lowest. If k falls much below this value, the arm becomes too weak to generate the require responses, and above it the arm becomes increasingly sensitive to actuator input noise.

4.6 Training Single Field Controllers

Once the optimal activation levels had been ascertained for each location, a standard backpropagation method (Haykin, 1998) was used to train the single field controller's MLP, using a proprioceptive encoding of the location as the input vector and the optimal activation level as a desired output vector. The location encoding was performed in three different ways, namely;

1. actuator lengths (4 vector);
2. arm joint angles (2 vector);
3. actuator lengths and arm joint angles (6 vector).

A separate MLP was trained for each combination of target point and proprioceptive configuration. The performance of each MLP was measured by calculating the absolute error between the learnt actuator activations and those generated by the MLP.

To remove the possibility that the MLP was merely learning sampling errors in the training data, ten-fold cross-validation was used. In this process, the training data is randomly divided into ten separate subsets. The network is trained ten times, each time using a different subset to evaluate the performance. The subset used to evaluate the performance is omitted from the data used to train the MLP, thus ensuring that the MLP was not trained using the evaluation data. The overall performance of the MLP is measured as the mean error across all ten evaluations.

4.7 Results

To compare the effectiveness of the proprioceptive configurations, the score across all 40 target points was averaged and plotted against network capacity (Fig. 4.5). As expected, the mean error decreases as the number of hidden nodes increases. The rate at which the results improve also declines, appearing to asymptote to a value of approximately 2.6. Given enough hidden nodes, an MLP is able to store any arbitrary smooth mapping. We might therefore expect the errors to tend towards zero as the number of hidden nodes is increased. Unfortunately, even though it is theoretically possible for the MLP to store a 'perfect' mapping, the learning process is constrained by the finite amount of information available from the training data. It is likely that lower final scores would be achieved if more training data was used.

The objective of the MLP was to map proprioceptive information into appropriate actuator lengths. The errors presented here therefore show the difference between the previously learnt actuator activations and those generated by the MLP, rather than the

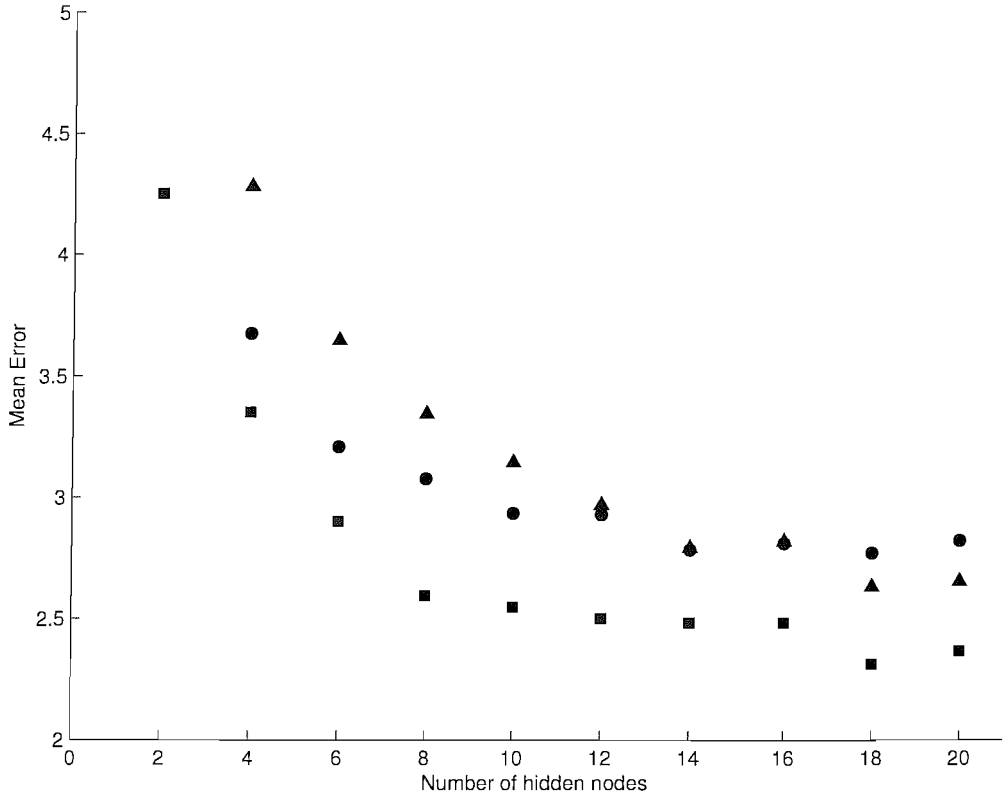


FIGURE 4.5: Comparison of three proprioceptive options; joint-angles (circles), actuator lengths (triangles) and both (squares). The results shown are the mean error for data collected from ten-fold cross validation of 150 test poses and 40 target points. The error is expressed as a percentage of the full actuator length range. There is a average standard deviation of 0.76 on each point.

difference between the ideal field vectors and those that would be generated if the MLP's actuator activations were applied to the arm.

Although using all six inputs (i.e., both joint and length values) gave a comparatively better result, it is not a dramatic difference. Due to the limited size of the training data, for each MLP the scores of each of the ten folds were quite varied, having mean standard deviations of 1.68% for joint angles, 1.84% for actuator lengths and 1.58% for both. When interpreted in this light the relative scores presented in Figure 4.5 seem less compelling.

A further constraint on the performance of the system is the amount of noise in the training data. As the data had been generated by an earlier learning process, a certain amount of training noise was unavoidable. If the initial learning process was replaced by an analytical analysis of the arm it would be possible to remove this noise. As this was a preliminary study, with a highly simplified arm model, this analysis was not performed as other factors would still have limited the strength of any resulting conclusions.

4.8 Conclusion

This chapter presented an initial static modelling exercise that showed that relatively simple neural networks are capable of generating the distinctive activation patterns typical of convergent force field control. No compelling difference between the proprioceptive configurations tested was demonstrated, although there is qualified support for the idea that more information is better than less.

The following chapters extend this work by applying a similar approach to the control of a full dynamic arm simulation.

Chapter 5

Dynamic Arm Modelling

This chapter describes the arm model used in the work discussed in the subsequent chapters, detailing its dimensions and workspace. The basic structure of the arm is derived from previous work in cybernetics (Hogan, 1985; Lukashin et al., 1996a,b), which in turn is modelled on the configuration of the primate arm. A schematic of the arm is shown in Figure 5.1. The main structure of the arm is a double pendulum chain (SE, ET), pivoted at S and hinged at E. Both the pivot and the hinge are assumed to be frictionless within their working range. The model has three pairs of antagonistic muscles. The first pair, shoulder extensor and flexor, connect rigid anchors (S_e and S_f respectively) to a single point on the upper-arm, M_1. The second pair, elbow extensor and flexor, run from two attachment points (E_e and E_f) to a second point on the upper-arm, M_2. The attachment points are constrained to lie at a fixed distance from the elbow, E, and at a fixed angle to the forearm, ET. The last pair, the two joint extensor and flexor, run between the respective attachment points for the the shoulder extensor and flexor (S_e and S_f) and the elbow extensor and flexor (E_e and E_f).

5.1 Selection of Link Lengths

Table 5.1 shows the parameters used to define the arm. The values used for link lengths (l_1 and l_2) and the actuator moments (s_f, s_e, e_f and e_e) are taken from Gribble et al. (1998). All the implementations of this model will preserve the relative component lengths but not necessarily their absolute values; the absolute values given are therefore expressed in arbitrary units. For clarity all the graphs presented in this thesis will be scaled to be consistent with the absolute lengths presented in this section. It should be noted that in Gribble et al.'s work the extensor moments are assumed to be constant, and only the flexor moments vary with arm pose, whereas in the work presented here, flexors and extensors are treated identically. It can be argued that, as the extensors

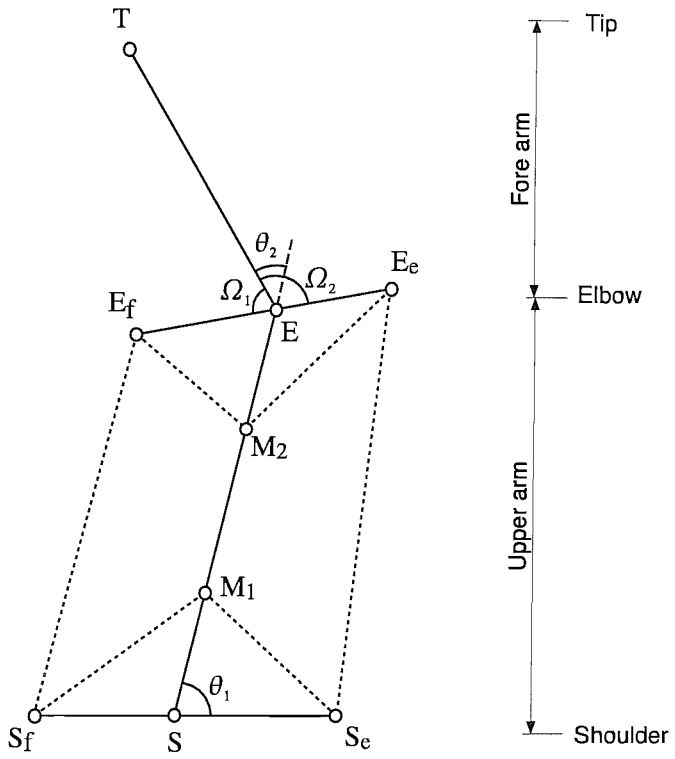


FIGURE 5.1: Six actuator arm. Solid lines show links, dotted lines show actuator locations. Angles θ_1 and θ_2 are variable, within limits. Nodes S_f, S and S_e are rigidly fixed.

wrap around the elbow joint, Gribble et al's model is a more authentic representation of the biological configuration.

Unfortunately it was not possible to implement a similar configuration in the dynamic simulation environment without significantly increasing its complexity and, therefore, development time. There have been several experimental approaches that have used disturbance of a limb, either during a movement (Gomi and Kawato, 1996) or at rest (Tsuji et al., 1994), to estimate the stiffness of the arm at the end effector. Once an effective arm controller is developed, it would then be possible to implement a similar process with this arm model. These results could then be used to compare the dynamic characteristics of the arm model and its biological counterparts. As the controller would be responsible for generating the background, or tonic, level of actuator co-activation there would be no meaningful way of testing the arm model's dynamics in isolation.

5.2 Selection of Joint Parameters

There are several parameter values that could not be directly obtained from the literature and the following paragraphs present the values used in this work and their justification. For simplicity, the limits for θ_2 were selected to ensure that there was a one to one

Label	Start	End	Length
l_1	E	S	4.6
l_2	T	E	3.4
s_f	S_f	S	0.5
s_e	S_e	S	0.4
e_f	E_f	E	0.5
e_e	E_e	E	0.2
m_1	M_1	S	1.5
m_2	M_2	S	3.0

TABLE 5.1: Table of model parameter values, using the nodes defined in Figure 5.1; lengths are given in arbitrary units, but their relative values are preserved in all implementations

mapping between joint angle and actuator length. With this in mind, the elbow moment angles (Ω_1 and Ω_2) have a significant effect on the arm's workspace. If we assume that the elbow moment angles are constrained to ensure that the line between E_f and E_e passes through E (i.e., $\Omega_1 + \Omega_2 = \pi$) then Figure 5.2 shows the arm's workspace for different values of Ω_2 .

It should be noted that the workspace in Figure 5.2a is not symmetrical because the flexors and extensors have different moment arms, and this is taken into consideration when calculating the joint limits. It is clear that for this arm, using $\Omega_2 = 180°$ gives the largest workspace. However, it does this at the expense of maximum forward (y-axis) reach. In order to maximise the workspace area and ensure that full straightening of the arm was possible, Ω_2 was further constrained so that when the arm was fully extended, E_e lay on a line between S_e and E (Fig 5.3a).

From these constraints we can calculate appropriate values for Ω_1 and Ω_2.

$$\Omega_2 = \pi - \tan^{-1}\left(\frac{s_e}{l_1}\right) \tag{5.1}$$

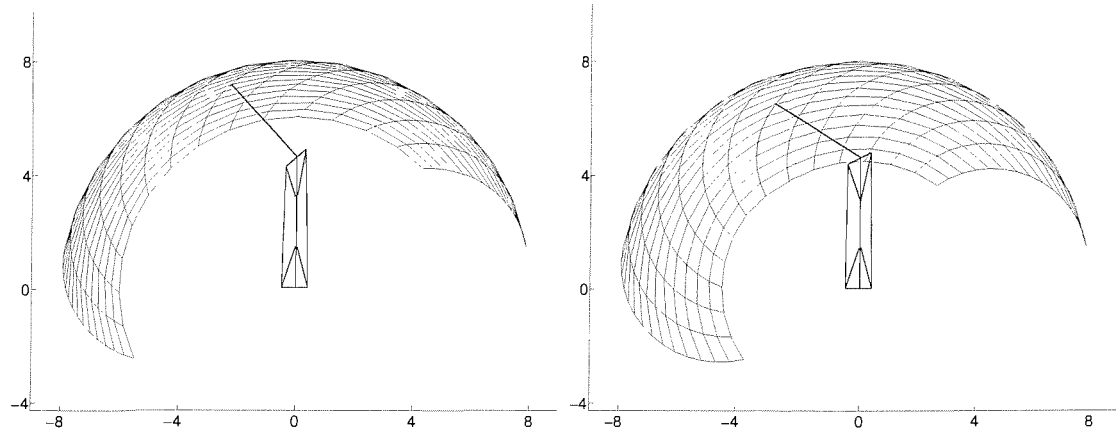
$$= 175.0° \tag{5.2}$$

$$\Omega_1 = \pi - \Omega_2 \tag{5.3}$$

$$= 5.0° \tag{5.4}$$

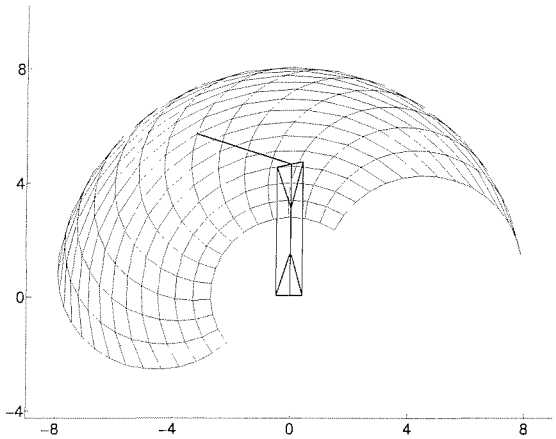
To complete the structural definition of the arm we need to specify the limits for the joints, θ_1 and θ_2. To preserve a large workspace, but minimise problems associated with extreme positions, joint one was arbitrarily constrained such that $-80° < \theta_1 < 80°$. From our selection of Ω_1 and Ω_2 it follows that joint two cannot bend past vertical in the clockwise direction, which in turn implies that:

$$\theta_{2_{\max}} = 0 \tag{5.5}$$

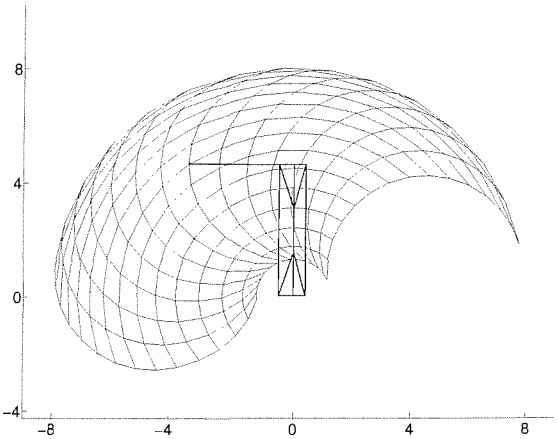


(a) $\Omega_2 = 90°$

(b) $\Omega_2 = 120°$



(c) $\Omega_2 = 150°$



(d) $\Omega_2 = 180°$

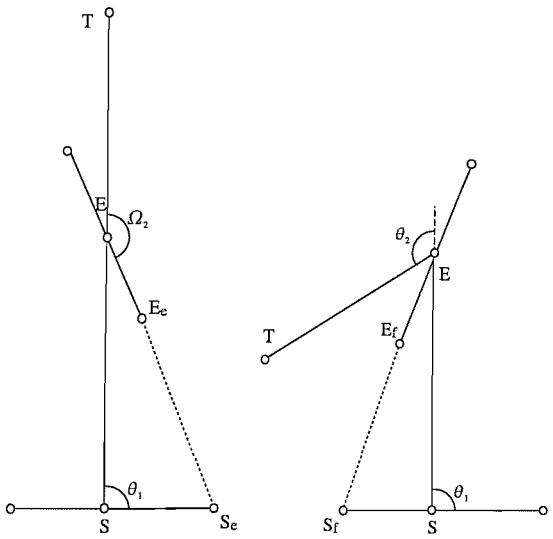
FIGURE 5.2: The workspace of the six actuator arm shown in the xy plane, for a range of Ω_2 values, where $\Omega_2 + \Omega_1 = \pi$.

The lower limit of θ_2 was chosen such that E_f never passed the line between S_f and E (Fig 5.3b). If we assume clockwise rotation to be positive, then we can show:

$$\theta_{2_{\min}} = -\Omega_2 + \tan^{-1}\left(\frac{s_f}{l_1}\right) \tag{5.6}$$

$$= -168.8° \tag{5.7}$$

The angle parameters are summarised in Table 5.2. To give the reader an indication of how the arm structure changes over its workspace, Figure 5.4 shows the arm, plotted to scale, in a set of extreme poses. It is clear that in some of these poses the actuators have very small moments of attachment and as such will have limited ability to generate torques.



(a) fully extended (b) fully flexed

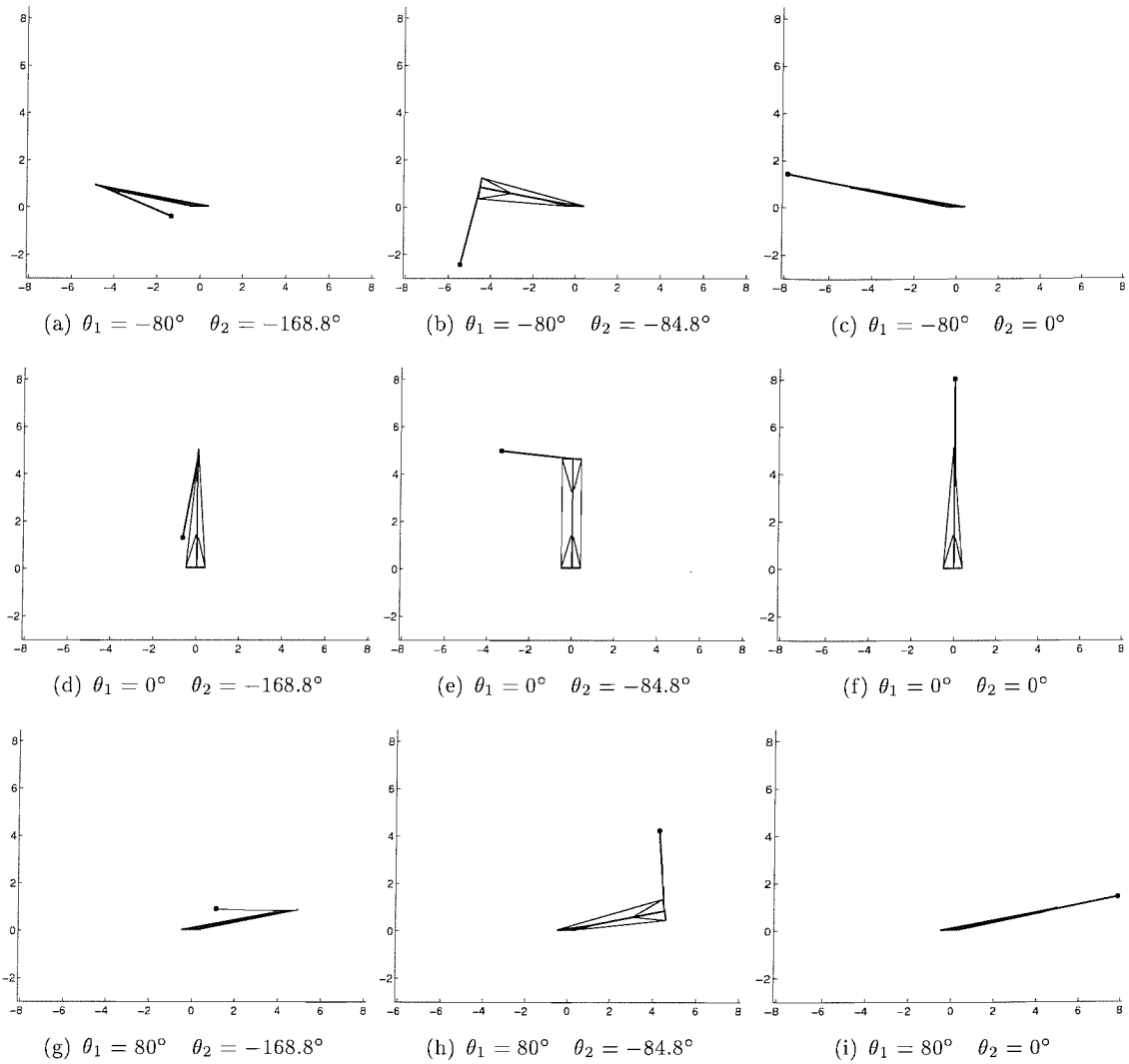
FIGURE 5.3: (a) Arm at full extension, as used to determine values for Ω_1 and Ω_2. (b) arm with joint two flexed, as used to determine maximum value for θ_2.

Label	Value	
Ω_1	5.0°	
Ω_2	175.0°	
	min	max
θ_1	−80°	80°
θ_2	−168.8°	0°

TABLE 5.2: Summary of the angle parameters used in the arm model, as shown in Fig. 5.1. Values are given in degrees, clockwise is assumed positive, with zero degrees being vertical (i.e., aligned with the y-axis).

5.3 Redundancy

The arm model, as it is presented, contains only limited redundancy. That implies that, while there is a one to one mapping between tip location, T, and the joint angles, θ_1 and θ_2, there will be multiple ways to excite the actuators to achieve the same force at the tip. This clearly simplifies the control task – but is it a reasonable thing to do? Primate arms have more actuators per joint and have more degrees of freedom in some of the joints. This naturally leads to systems that are capable of achieving the same end effect with a range of pose configurations. Contemporary robotic designers minimise the number of actuators used, as this reduces weight, cost and complexity, while increasing robustness. With this in mind, when analysing arms with a view to long-term physical implementation, one should prefer simpler systems with low degrees of freedom and few actuators. Control of redundant manipulators should only be considered when the extra control complexity incurred is balanced by appropriate increases in task performance.



(a) $\theta_1 = -80°$ $\theta_2 = -168.8°$ (b) $\theta_1 = -80°$ $\theta_2 = -84.8°$ (c) $\theta_1 = -80°$ $\theta_2 = 0°$

(d) $\theta_1 = 0°$ $\theta_2 = -168.8°$ (e) $\theta_1 = 0°$ $\theta_2 = -84.8°$ (f) $\theta_1 = 0°$ $\theta_2 = 0°$

(g) $\theta_1 = 80°$ $\theta_2 = -168.8°$ (h) $\theta_1 = 80°$ $\theta_2 = -84.8°$ (i) $\theta_1 = 80°$ $\theta_2 = 0°$

FIGURE 5.4: Arm in set of extreme poses. θ_1 set to $-80°$, $0°$ or $+80°$ and θ_2 set to $-168.8°$, $-84.8°$ or $0°$. In the most extreme poses (a, c, g and i) the actuators have relatively small attachment moments, which may lead to performance limitations in these regions. The tip of the arm is marked with a filled circle.

5.4 Implementation

A static model of the arm has been implemented using MATLAB scripts. This model has been used to generate several of the figures in this chapter and is used in later chapters to generate training and testing data. A second, dynamic, model has been implemented using a custom simulation package based on the Vortex Simulation Libaries (2002), as discussed in Appendix A.

The physical model required several other parameters to be defined (e.g., masses, friction, actuator models etc.), the selection of which is discussed in the following chapter. Figure 5.5 shows a screen shot taken from the dynamic simulation. There are two physical components (upper arm and forearm links) and several massless components

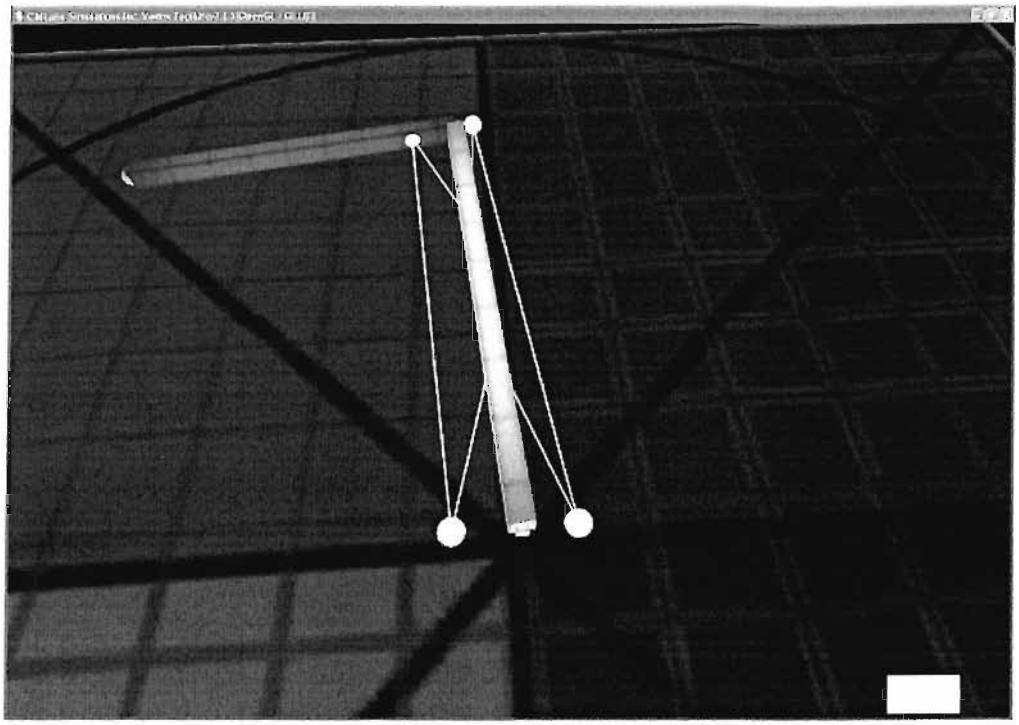


FIGURE 5.5: Screen shot taken from dynamic arm simulation environment. The prominent balls at the front of the picture are the two shoulder attachment points. Between them is the shoulder point where the upper arm is pivoted. The upper arm is shown in light grey and is aligned vertically with respect to the window. The forearm can be seen lying at 90° to the upper arm, pointing to the left of the window. Thin lines, representing the actuators, can be seen attached between the components.

(actuators, attachment points and sensors). Underneath the arm is a ground plane that has been marked with a regular grid and some radial lines to assist in the qualitative assessment of arm movements.

5.5 Conclusion

This chapter has outlined a simple geometric arm model and defined its parameters. Although, due to numerical constraints, the model does not use standard metric units, the ratios of the various components, and the angular ranges of the joints have, where possible, been chosen to reflect those found in nature.

Chapter 6

Convergent Force Field Control Experimentation

This chapter details the development of an arm control technique inspired by convergent force field control. An online learning algorithm (single mutation hill climbing) is used to train an artificial neural network to control the movements of the dynamic arm simulation.

Initially, only a single field controller was trained. This controller is responsible for moving the arm towards a single fixed target, using the available proprioceptive information. Once a selection of such single networks were been trained, further testing was performed to see if was possible to combine them to create a controller that can steer the arm towards intermediate positions.

The single field controller was implemented using an MLP with a logistic hidden layer and linear output layer, as shown in Figure 6.1. Feedback signals are taken from the physical simulation. Various combinations were tested, but work presented here uses the length of each of the actuators and the rate of change of length of each of the actuators.

The six actuators are connected to the Newtonian arm model and simulated using the custom built manipulation simulation client (see Appendix A). The multi-layer perceptron is implemented using the publicly available NETLAB neural network toolbox for MATLAB.

6.1 Physical Modelling Environment

As mentioned, the Vortex Simulation Libaries (2002) were used as a foundation for the physical simulation environment. These libraries model the world using an assortment of rigid bodies. Although the libraries themselves are capable of using a wide variety

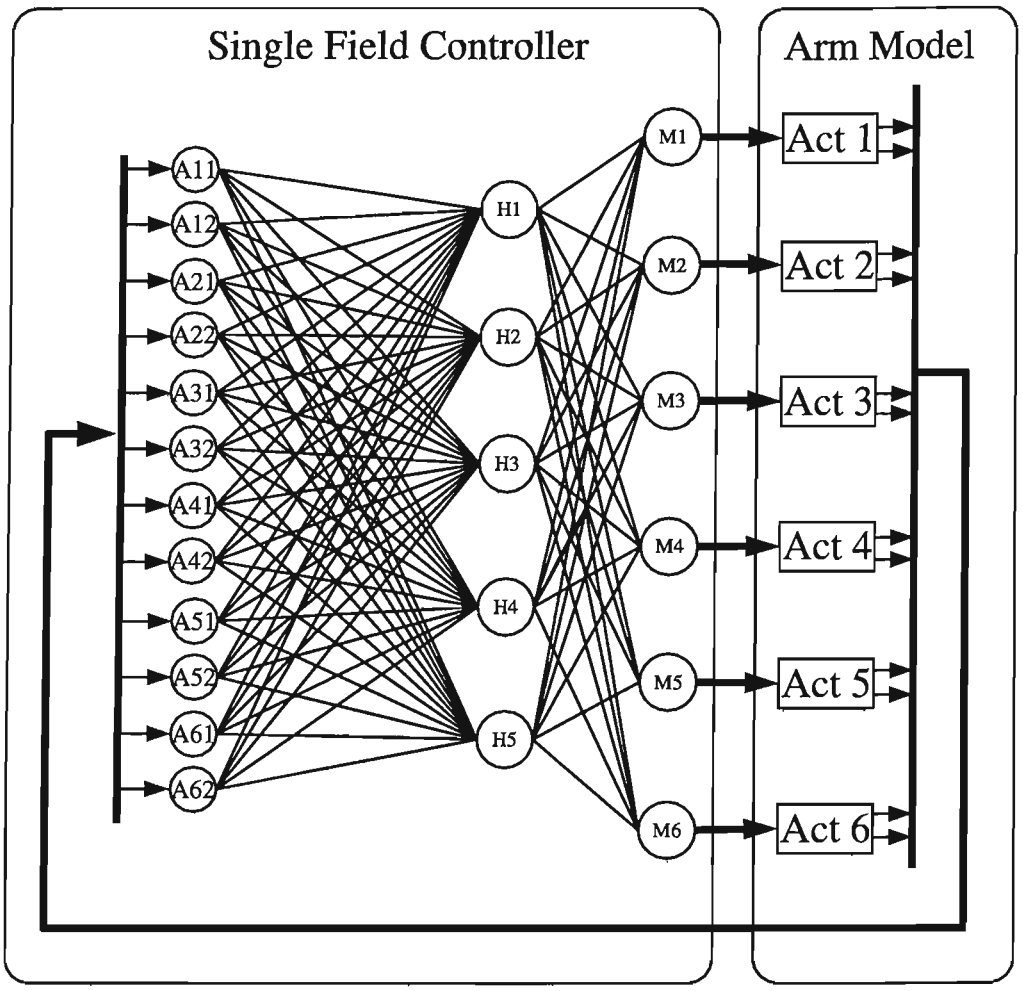


FIGURE 6.1: Network diagram for the single component controller.

of component descriptions (convex-mesh, primitive shapes and height fields), the work presented here only used primitive shapes (i.e., cuboids, spheres and cylinders), because they were the most computationally efficient and accurate.

Vortex models joints as constraints, which create the required reaction forces to resist any undesired movements. These have the potential to create relatively large forces within the simulation, which in turn can lead to instability if not managed correctly. In comparison, an analytical model of the arm movement would assume that many of the reaction forces would be perfectly balanced and as such they could be factored out of the mathematics. Model instability can be caused by stiff force-models, and this is of particular concern when adding novel actuator models to the simulation (as is done here) because they have the potential to introduce energy to the system. In contrast, Vortex's own force-models are designed such that numerical errors tend to drain energy. Care must therefore to taken to limit the amount of force introduced to the simulation. This has various implications, like the avoidance of 'whip' like structures as these can

propagate energy along the length in such a way as to result in movements of the end point that are too fast for the simulation.

Clearly a custom designed analytical model of the arm dynamics would have simpler parameters, and be capable of running much faster. So why has this route not been taken? Partly for historical reasons (the simulation environment was already substantially finished before the arm model was finalised) but also there are a few key advantages to a Vortex based approach. Firstly, Vortex is a tried and tested set of simulation tools, so there is less likelihood of fundamental errors going undetected. Secondly, Vortex (and other rigid body simulators) provide a very flexible set of tools and as such would permit future researchers to rapidly integrate and assess the arm's interaction with other environmental features (e.g., fixed or movable objects). The simulation application has been specifically designed in a strongly modular way to facilitate this rapid extension and modification.

6.2 Actuator Models

The actuators used in this simulation are modelled as a linear spring damper system. The spring has variable stiffness and creates no reaction to compression. For an actuator of length l_n, previous length l_{n-1}, rest-length r, damping coefficient d and stiffness gain k, the following equations define the force generated, f, for an input demand α_n:

$$f_{\text{damp}} = d(l_n - l_{n-1}) \tag{6.1}$$

$$s = \begin{cases} l_n - r & : \quad l_n > r \\ 0 & : \quad l_n < r \end{cases} \tag{6.2}$$

$$f_{\text{drive}} = \alpha_n k s \tag{6.3}$$

$$f = f_{\text{drive}} + f_{\text{damp}} \tag{6.4}$$

Figure 6.2 represents the components of the actuator model. For each of the six actuators we must therefore determine a suitable value for d, k and r. Rather than trying to find a justifiable value for each of these 18 parameters individually, some basic constraints were first applied. The objective of these constraints was to give each actuator roughly equivalent performance while taking into account its geometrical configuration. Firstly, the rest-length for each actuator was set be equal to its minimum length, as determined by the joint limits. This would mean that the actuators never entered their slack, zero-force producing, phase. But it does provide a sensible value that would scale appropriately between actuators. Secondly, the stiffness gain, k, for each actuator was

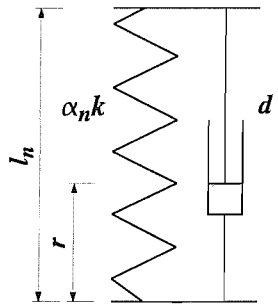


FIGURE 6.2: Components of the actuator model, showing current length l_n, rest-length r, damping coefficient d, stiffness gain k and input demand α_n. Note that the spring does not create any force if $l_n < r$.

constrained to be a common value, $\tilde{k}$, divided by the range of lengths the actuator could achieve.

$$k = \frac{\tilde{k}}{l_{\max} - l_{\min}} \tag{6.5}$$

This would mean that, for the same input signal, α, all the actuators could generate equal force at full extension. Damping was kept to a minimum as the objective was to train the controller to produce smooth movements, rather than mechanically constrain the system to enforce this effect. In a real world system, the effective dissipation of unwanted kinetic energy would present a significant technical challenge in its own right. Damping could either occur in the joints, or in the actuators themselves. Further studies are required to compare the effectiveness of these two alternatives with respect to system stability, efficiency and behavioural flexibility.

Once the system's geometry had been implemented, a manual trial and improvement process was used to determine a value for $\tilde{k}$ and the damping coefficients that allowed the controller to generate rapid accelerations of the arm without becoming unstable.

A value of $\tilde{k} = 1000$, with damping of 500 units per single joint actuator and 100 per two joint muscle, produced responsive and stable dynamics. Table 6.1 details how this breaks down into individual actuator parameters.

The dynamic arm model's underlying simulation application scales lengths and masses internally to reduce numerical integration error and therefore improve simulation stability and accuracy. This has the unfortunate side effect of making it difficult to put meaningful real-world units on the arm's parameters.

Actuator	k	r	d
Shoulder flexor	1017.2	1.0445	500
Shoulder extensor	1270.7	1.1415	500
Elbow flexor	1005.8	1.0377	500
Elbow extensor	1256.1	1.1354	500
Two joint flexor	505.0	3.6107	100
Two joint extensor	631.4	3.8077	100

TABLE 6.1: Actuator model parameters. Stiffness, lengths and damping coefficients are given in terms of simulation units.

6.3 Link Parameters

The links are modelled as solid cubiods, with constant density. The upper link is assumed to have twice the density of the lower link, in order to represent the extra mass created by the actuators. They are modelled as having a square cross sectional area of width 0.8. Appropriate polar inertias are calculated using a Vortex function.

6.4 Feedback Encoding

To reduce the potential for bias, the feedback presented to the MLP was normalised. In total twelve inputs were presented. The first six were derived from the length of the actuators, as reported by the physical simulation. These were rescaled to lie between -1 and 1 using information gained from the MATLAB model of the arm. The remaining six inputs represented the rate of change of the actuators. These were approximated in MATLAB as the difference between the previously reported and the current actuator lengths. As these inputs had a large dynamic range any linear scaling that guaranteed a -1 to 1 range would have resulted in very small signals for most conditions. Instead a hyperbolic tan function was applied to the rate signals before they were presented to the network. This ensured that, while the complete range of the signals was preserved, small movements still resulted in reasonable response of the network input signal.

6.5 Method

It was not possible to use standard techniques like backpropagation to train the MLP as there was no reasonable way to generate appropriate training data. Instead mutation hill climbing was employed. This technique involves creating a randomly initialised network and attempting to improve it incrementally. At each stage a child network is created by mutating the existing one. The relative performance of the networks is then measured according to some reasonable test of fitness. If the child network performs better, it is

adopted as the new network; if not it is rejected and the existing network reinstated. This technique should usually be tested before more advanced learning algorithms. As only a single solution is explored, rather than a population of solutions, it will often be the most efficient technique, in terms of number of evaluations of the fitness test.

As with any learning algorithm, there are a number of parameters to specify. In this case we must select the capacity of the neural network (i.e., how many hidden nodes it will have), and the mutation rate. It will also be necessary to determine sensible values for the parameters of the fitness function.

6.6 Fitness Function

The role of the fitness function is to compare the performance of two competing controllers and determine which should be retained, and which rejected. There are two conflicting demands on this function; it should be fair and it should be efficient. If the system does not reliably select the better of the two controllers the learning algorithm will either converge to an inappropriate solution, or not converge at all. With this in mind the fitness function must test both controllers thoroughly, ensuring that a representative subset of its responses is evoked. If a poor selection of responses is selected then the controllers produced may not perform well in the regions that were not covered. It is also important that the fitness function does not waste time further analysing controllers that are clearly inferior.

The fitness function presented in this chapter takes the following steps:

1. Select a range of start poses.
2. For each start pose, initialise the arm simulation and connect the controller.
3. Run the combined arm and controller for a fixed number of steps.
4. The score for each run is the mean distance between the end of the arm and the target point.
5. The score for the controller is the mean of the run scores.

Clearly a lower score indicates a better fitness. There are two parameters in the process which must be carefully chosen to strike the balance between thoroughness and processing time. Firstly, we must determine how many steps should the simulation be run for. Secondly we must decide which start poses should be used.

The simulation should be run for at least enough steps to allow an effective controller to bring the arm to rest near the target point from any start pose within the workspace.

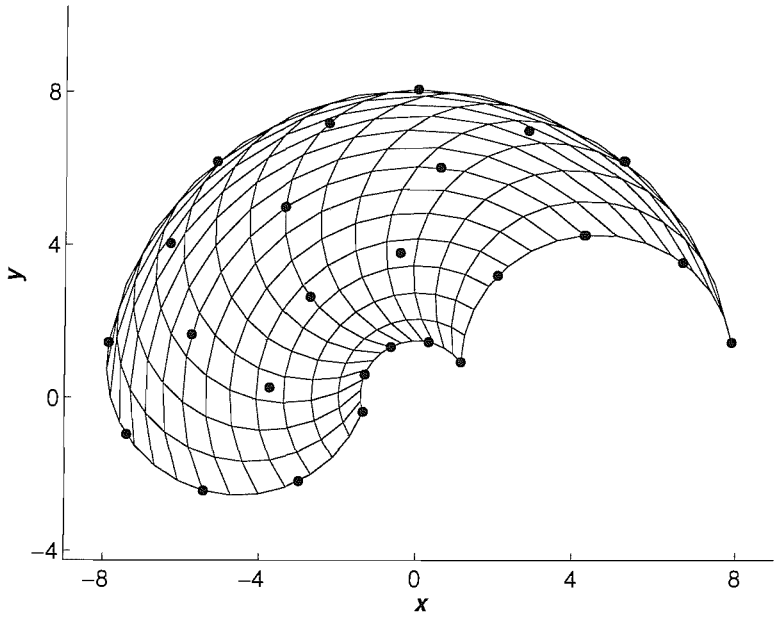


FIGURE 6.3: Initial start poses, marked with black dots, are arranged in a five by five grid with respect to joint angle.

The optimal value for this was hard to determine, but it was found that 500 steps was generally long enough.

There are two alternative approaches to the selection of start poses. The method least likely to introduce unfair biases into the result is to select the start poses at random from within the workspace. Unfortunately, to ensure fairness, this technique requires that the current controller is evaluated at every iteration (rather than just the mutation currently being evaluated), as some sets of start poses may be more likely to produce good scores than others. A second approach would be to select a fixed set of start points, and use these for all the trials. As the current controller would not then need retesting each time, this would decrease the overall computational burden at minimal cost to thoroughness. The second approach was used throughout the following work.

In order to ensure good coverage of the workspace, the start points were arranged in a grid, evenly by joint angle. In the following work a five by five grid of start poses was used throughout (Fig 6.3).

6.7 Mutation Algorithm

For mutation hill climbing to work it must be possible to express the configuration of the controller in such a way that small changes can be performed in a randomised way. Ideally these changes should lead to small alterations in the behaviour of the controller, which can then be accepted or rejected by the fitness function. The configuration of an

MLP can be fully expressed as a list of floating point values (representing the weights and biases of the network). This list could either be mutated at the bit level or at the real number level. As we know that this data encodes real numbers, it makes sense to work at the real number level, as it makes selection of mutation rate more intuitive.

The mutation algorithm used was as follows:

1. Use NETLAB's mlppak to convert the MLP weights and biases into a row vector.
2. Select a single component pseudo-randomly.
3. Disturb the component by a value chosen from a scaled normal distribution.
4. Use NETLAB's mlpupak to rebuild the network from the modified row vector.

6.8 Trial Run

To test that the system was working and capable of converging towards a target, an initial trial run was performed. A 15 hidden node MLP was randomly initialised and then trained for 500 testing cycles. A target was selected in the middle of the workspace, $(-2, 4)$, to ensure that the network performed real feedback control, and could not succeed by merely driving the arm into an extreme position.

Figure 6.4 shows the mean error for the network plotted against training cycles. Several unsuccessful runs were observed (i.e., where the training process did not converge), before this run was recorded. As the network improves, the scores of the unsuccessful tests start to fall further away from the current best score. This means small changes in the network parameters are resulting in larger changes in the fitness function score. A more advanced training technique would reduce the size of the mutation rate as the system converged. This would improve the quality of the mutations tested and therefore increase the learning rate towards the end of the trial.

To demonstrate how the errors presented in Figure 6.4 relate to actual movements of the simulated arm, Figure 6.5 presents plots of the arm movements used by the fitness function taken at five evenly spaced intervals during the training process. Each subfigure shows the results of a single evaluation of the fitness function. A separate line is plotted for each of the 25 start poses. Each run starts in one of the initial poses, arranged evenly within the workspace. Time, in simulation steps, is plotted on the vertical axis, with the workspace coordinates of the tip of the arm plotted in the horizontal plane. A vertical dashed line on each subfigure indicates the location of the target point.

The initially randomised network (Fig. 6.5a) demonstrates strong, stable oscillatory motion. The arm movement converges towards the same limit-cycle irrespective of its

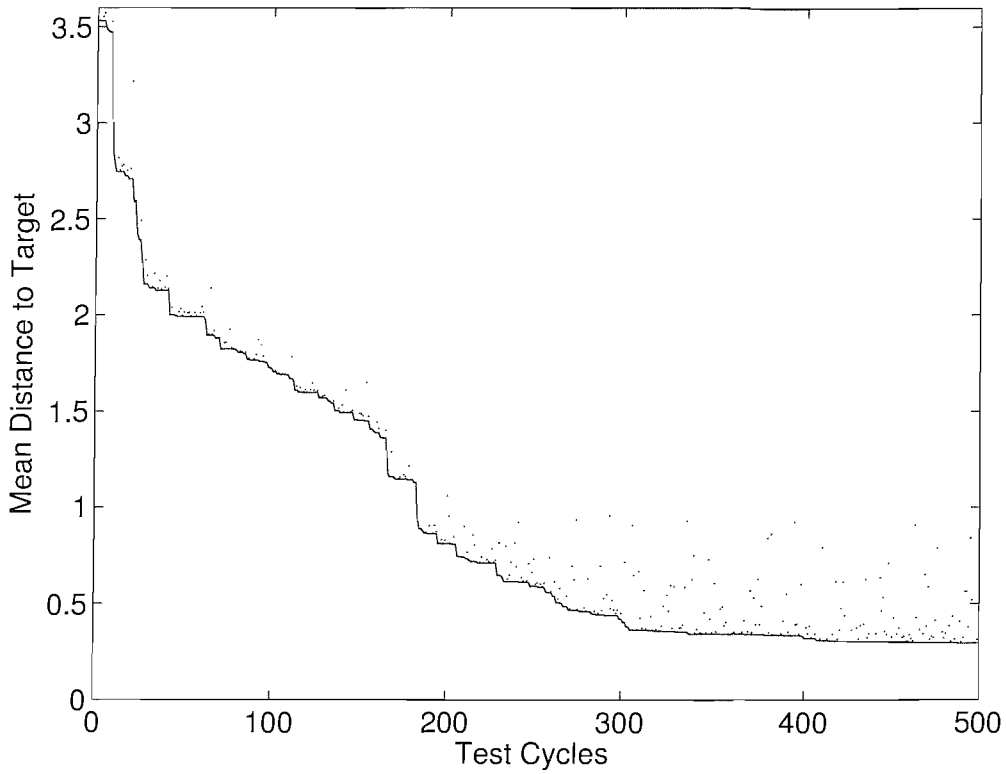
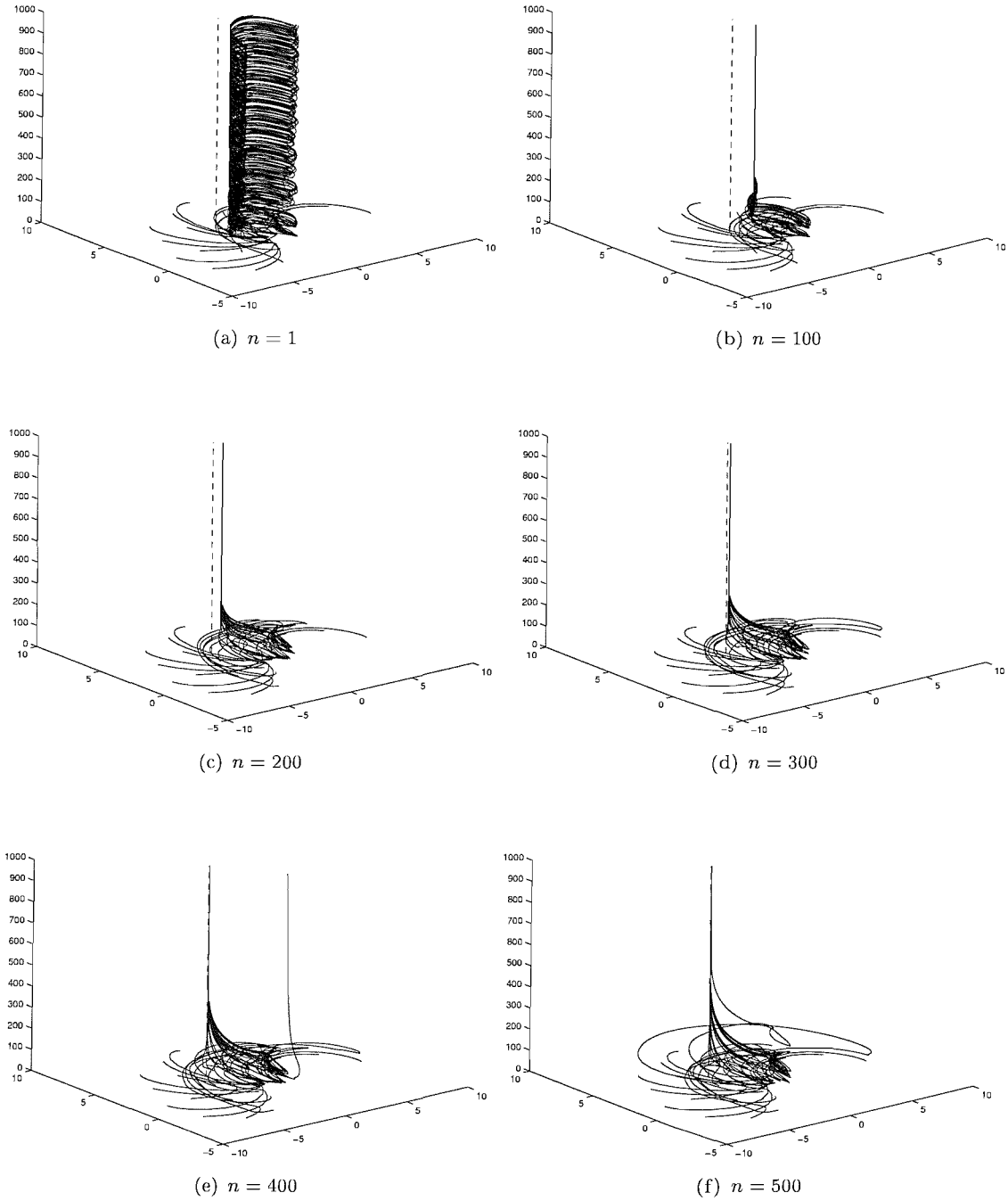


FIGURE 6.4: Trial run scores. The mean error (vertical axis) is taken to be the mean euclidean distance between the tip of the arm and the target point over 25 trials, each with 1000 simulation steps. The points marked above the line indicate the scores of unsuccessful mutations.

initial workspace position. After the first 100 training cycles (Fig 6.5(b)) the oscillatory behaviour has been replaced by a stable, single rest point, response. The next four subfigures (Fig 6.5(c-f)) show the gradual movement of the rest position towards the target. Interestingly, this is sometimes achieved at the expense of controlling all the runs. Figure 6.5(e) in particular shows a stage where a slight improvement in average distance from the target has resulted in one run heading in entirely the wrong direction. It would be possible to design the fitness function to be very sensitive to such outliers. This could be achieved by using the maximum, rather than the mean, distance from the target. Such an approach would, however, ignore any improvements to runs that were not the worst, and might therefore make the fitness landscape less smooth.

To demonstrate the interaction between the arm model and the network controller, Figure 6.6 plots the network inputs and outputs for a single run, taken from a fully trained controller. The run started with the tip of the arm at $(4, 4)$ and moved towards the target at $(4, -2)$. The top two plots show the two sets of six signals used as network inputs; normalised actuator lengths and hyperbolic tan encoded actuator length velocities. The third graph, network demand, shows the output of the network after it has been shifted to be positive. This was required because the actuator models required a scalar input. The fourth graph, actuator forces, shows the force developed by the



(a) $n = 1$ (b) $n = 100$

(c) $n = 200$ (d) $n = 300$

(e) $n = 400$ (f) $n = 500$

FIGURE 6.5: Trial run demonstrating convergence of the arm response towards the target point. The vertical axis represents time, expressed as simulation steps, while the remaining axes represent workspace coordinates. On each figure a vertical dotted line indicates the location of the target point, which was constant throughout. The figures are all drawn from the same training process, with the number of training steps, n, labelled underneath.

actuators. This is not a direct scaling of the network demand because the force developed is also a function of the current actuator extensions. The last two figures show how joint angles and arm tip location varied throughout the run.

6.9 Selection of Mutation Rate

Having demonstrated a working system, the next stage was to select the optimal mutation rate. Figure 6.7 shows the fitness function scores of seven randomly initialised networks, as trained at four different mutation rates. On average setting $m = 0.001$ produced the greatest improvement, but the results are far from conclusive. There are two ways of interpreting the results: either the system is not very sensitive to mutation rate or the mutation rates tested are all a long way from ideal. Although we can not be certain that the second of these is not true, as a reasonable range was covered (three orders of magnitude), we have some confidence that $m = 0.001$ is a reasonable value to use.

6.10 Selection of Network Capacity

A much longer experiment was then performed to compare the effectiveness of various network sizes. Five network configurations were tested with 5, 10, 15, 20 and 25 hidden nodes, respectively. For each configuration, six networks were randomly initialised and trained for 1000 cycles. The mean distance between the tip of the arm and the target, $(-2, 4)$, is plotted against training cycles in Figure 6.8.

For nearly all the network configurations there are several runs where no significant improvements are found; the training process gets stuck in local minima. In several cases, while there is some improvement, the hill climbing process fails to do better than would be achieved by random reinitialisation.

That said, it would appear that the networks with 20 hidden nodes seemed to train more reliably than any other of the tested capacities. At this stage it is not possible to determine whether this was solely a feature of the training process, or of the system being trained.

So, from the data presented, we can conclude that although mutation hill climbing is capable of producing networks that move the arms towards the desired target, it is not capable of doing so reliably. Further modifications to the training technique, like selective random restart, would be required to improve reliability.

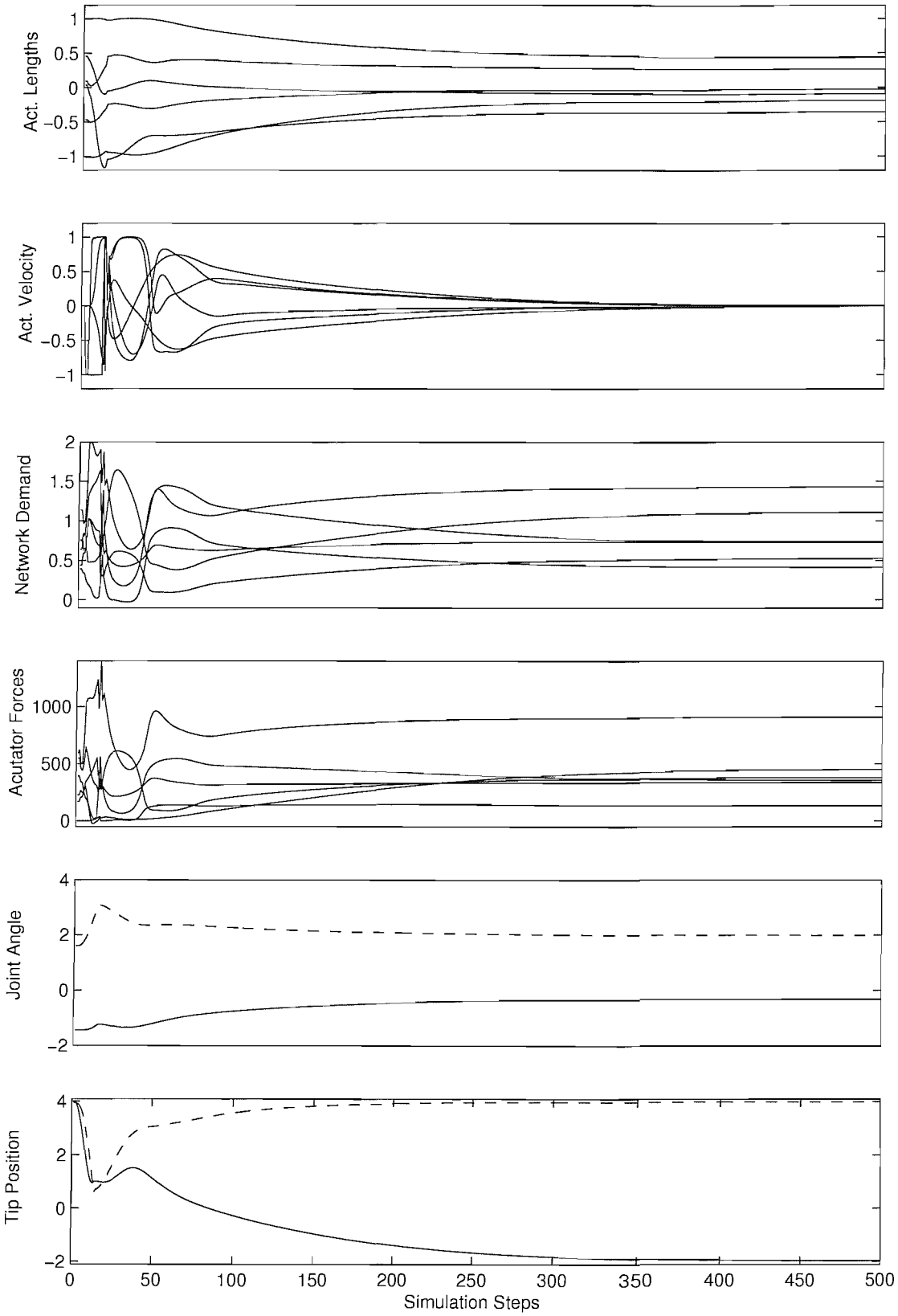
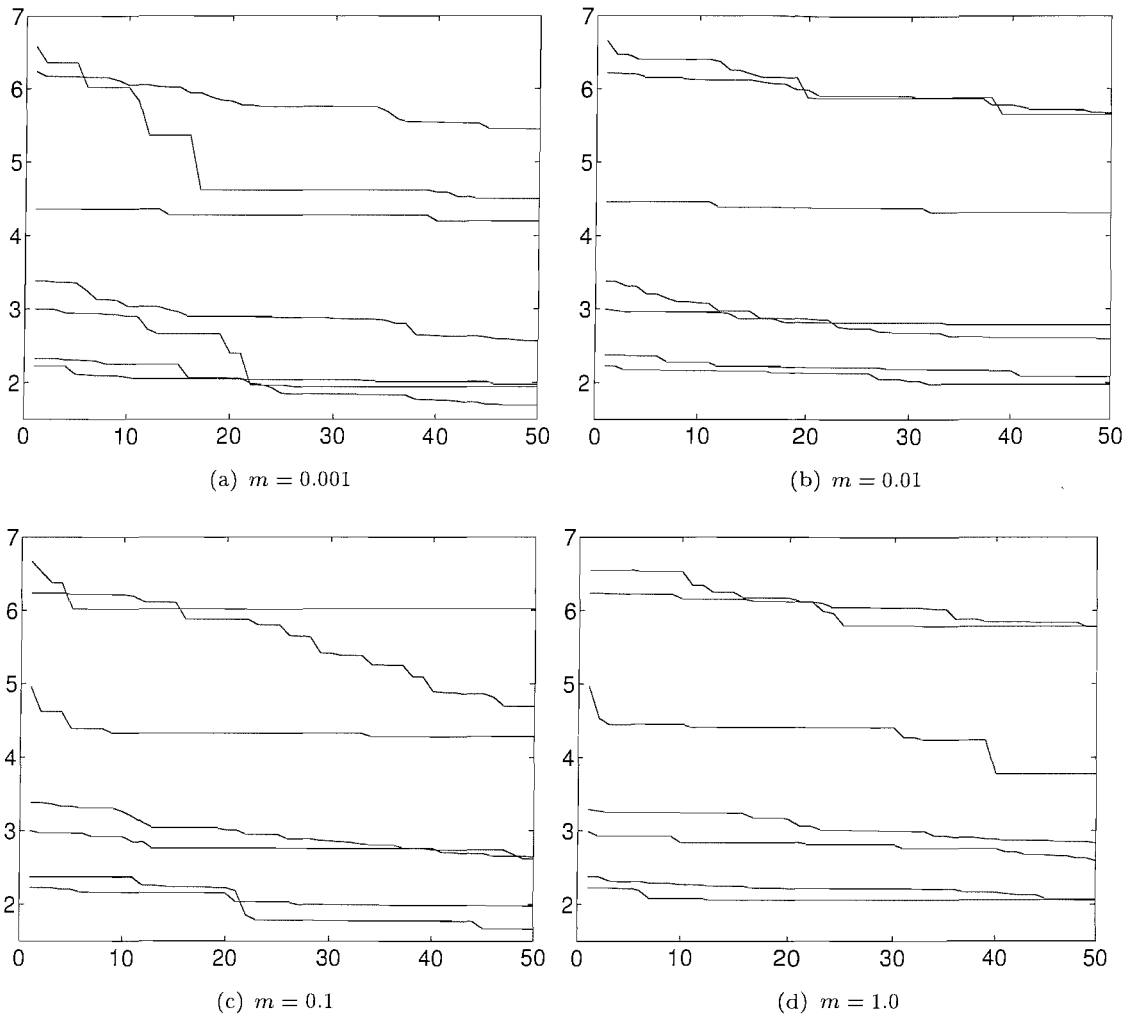


FIGURE 6.6: Network inputs (actuator length and actuator velocity) and outputs (network demand) for a single run of the fully trained controller. Actuator forces are given in simulator force units. The joint angles are in radians (solid: θ_1, dashed: θ_2). Tip position is given in workspace coordinates (solid: x, dashed: y).



(a) $m = 0.001$ (b) $m = 0.01$

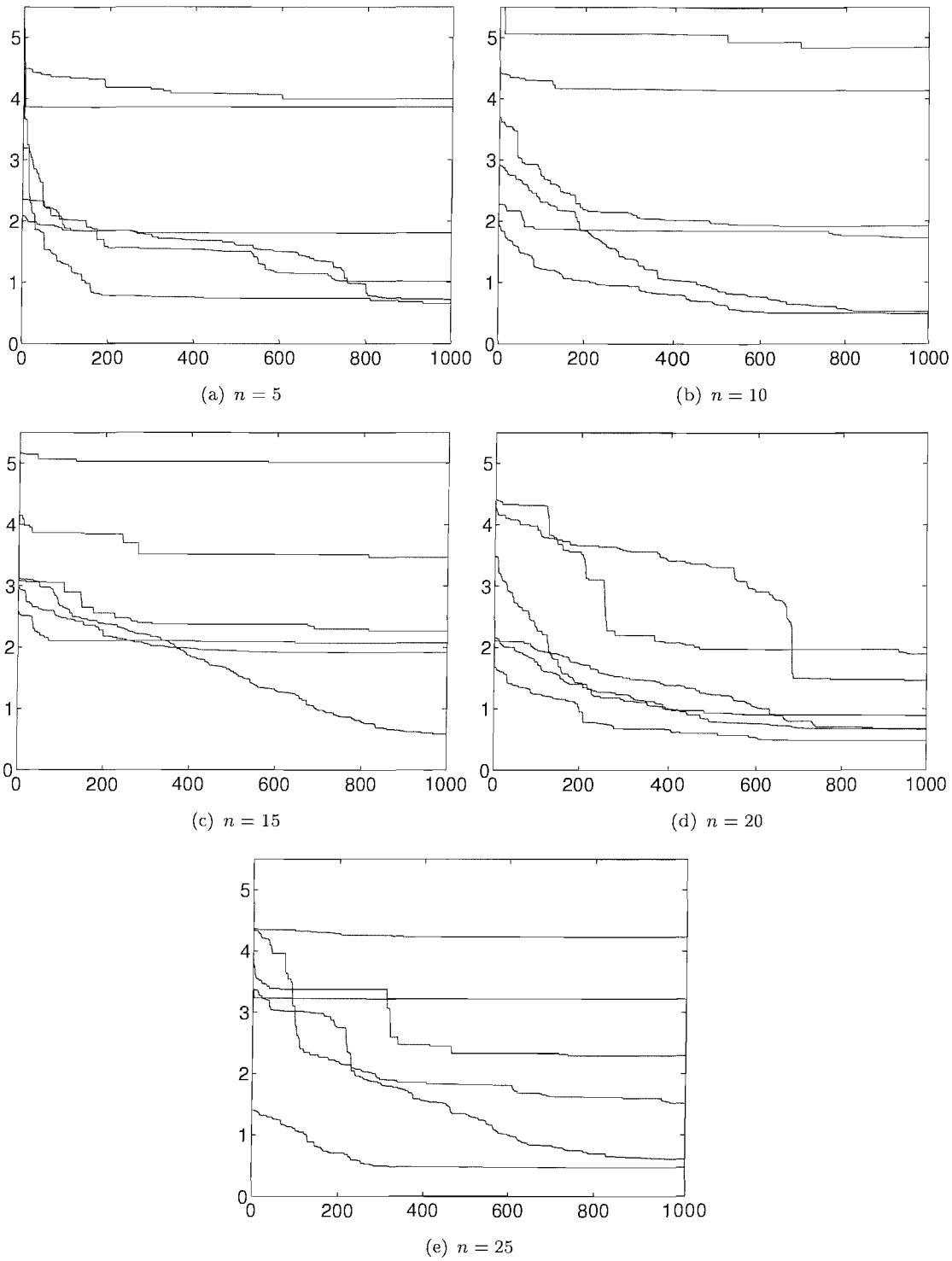
(c) $m = 0.1$ (d) $m = 1.0$

FIGURE 6.7: CFFC control with ten hidden nodes trained with a range of mutation rates, m. Each graph shows training cycles (horizontally) against mean error for 25 evenly spaced starting poses (vertically). Although $m = 0.001$ seems to do the best overall, it is clear that some initial conditions are not amenable to improvement, irrespective of the mutation rate. The initial error value is not a function of the mutation rate.

6.11 Blending Network Outputs

In order to investigate the potential for several controllers to be used in parallel, a second network was trained with a different target, $(-5, 0)$. This was less successful than the first (see Figure 6.9 for training errors), but did create a network that was capable of moving the arm approximately towards the alternative target.

Figure 6.10 shows a comparison of the fully trained output for two networks, one trained toward $(-2, 4)$ and the other toward $(-5, 0)$. The arm response is shown both in a three dimensional time plot, and from a top view of the workspace. It is clear that the networks move the arm toward different locations. The last two subfigures (e and f) show the response of the arm when it is controlled by a blending of both networks. The



(a) $n = 5$

(b) $n = 10$

(c) $n = 15$

(d) $n = 20$

(e) $n = 25$

FIGURE 6.8: Mean error (vertical axis) plotted against training cycles (horizontal axis) for five different network sizes (n is the number of hidden nodes, all other network parameters were identical throughout). Error is calculated as the mean euclidean distance between the tip of the arm and the target point, over 25 runs of 500 simulation steps. The runs used a range of starting poses which were spaced evenly with respect to joint angle within the workspace of the arm. The experiment took 225 hours to run.

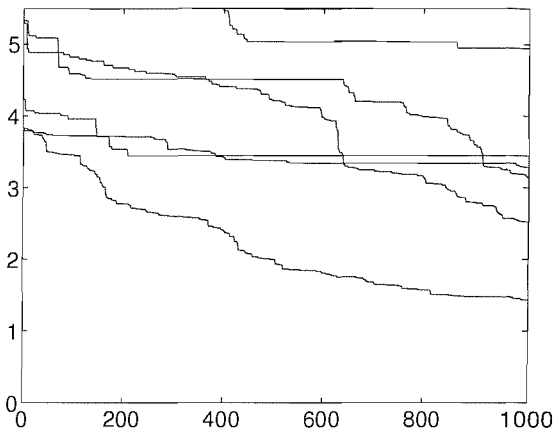


FIGURE 6.9: Training history for an alternative target point $(-5, 0)$. Horizontal axis shows training cycles, vertical axis is the mean distance between the tip of the arm and the target point.

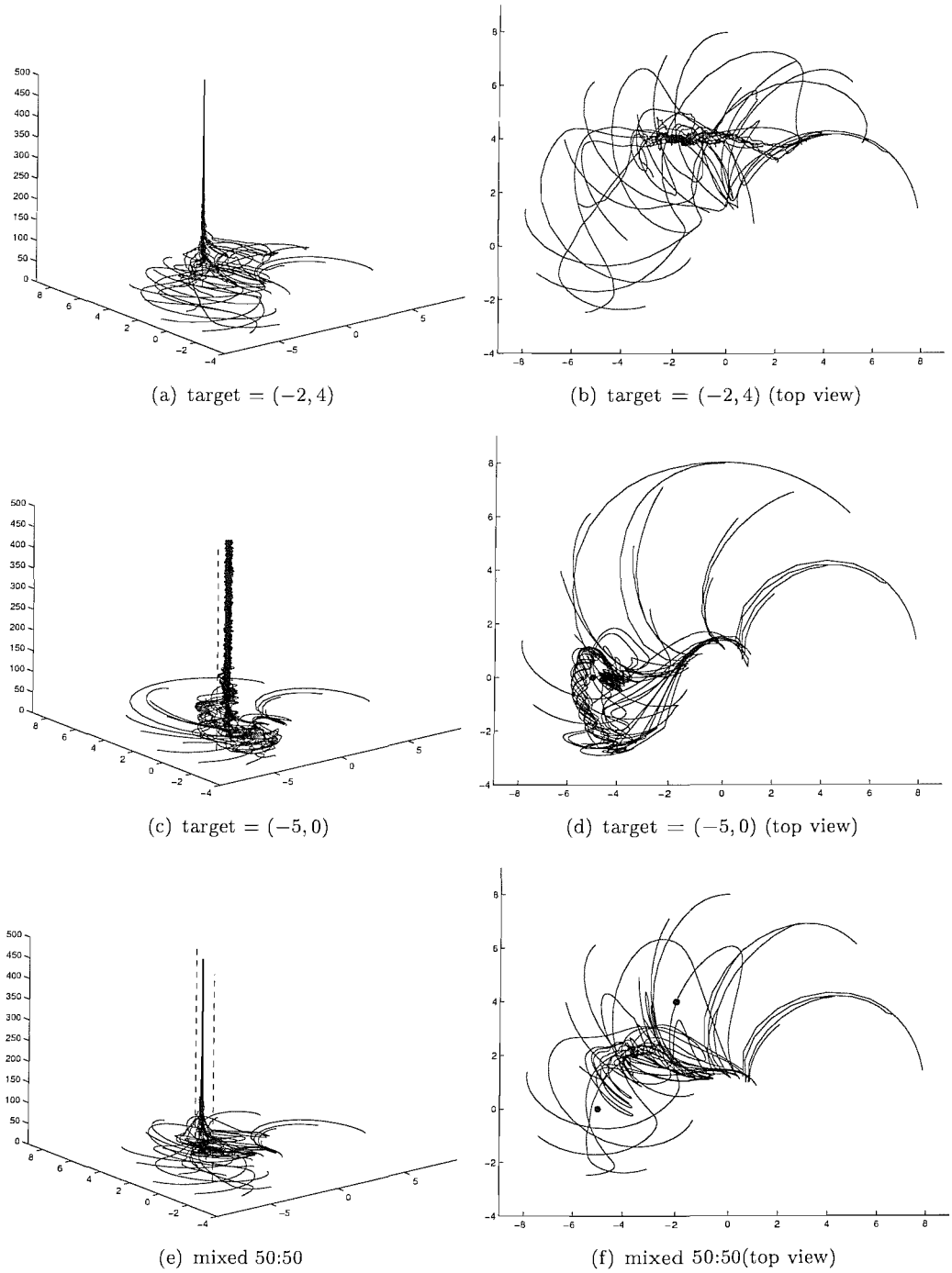
blended outputs were calculated as the mean of the two individual network outputs. The resulting blended network creates a response that moves the arm to a position roughly halfway between the two target points, which are plotted as black dots in subfigure (f).

To determine how the outputs of the networks combined, a series of weighted blendings were performed. In each blending the outputs of a single field network were multiplied by a scaling factor before being combined. The sum of the two scaling factors was always 1.0. Figure 6.11(a-i) shows a top view of nine different blended networks. The two target points are shown as black dots. As the blending ratio moves from favouring one network to favouring the other, a smooth range of intermediate responses are elicited.

Figure 6.11(j) shows the mean position of the tip of the arm for the second half of each simulation, i.e., the period when the arm is substantially at rest. These positions move smoothly from one target to the other, but not quite in a straight line, or with even step sizes. The size of the ellipse is proportional to the mean of the standard deviations of the tip position, and gives an indication of the arm's steadiness.

6.12 Discussion

This initial investigation must be considered a mixed success. Firstly, given enough training cycles, the controller was capable of learning how to move the arm towards the target from any position within the workspace. Secondly, basic mixing of the network outputs was demonstrated to create reasonable intermediate responses. This holds open the possibility that the arm could be controlled by a set of independent component controllers, whose output is mixed to create a complete range of convergent trajectories. Unfortunately the experimentation did not demonstrate the capability of generating the straight trajectories with the bell shaped velocity curves typical of human reaching.



(a) target = $(-2, 4)$

(b) target = $(-2, 4)$ (top view)

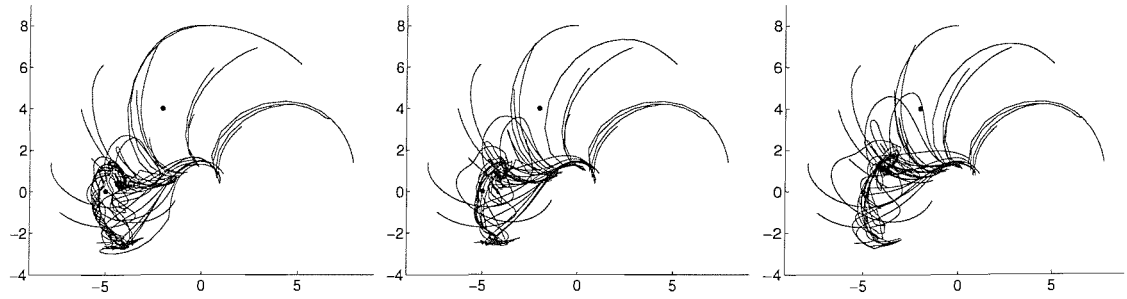
(c) target = $(-5, 0)$

(d) target = $(-5, 0)$ (top view)

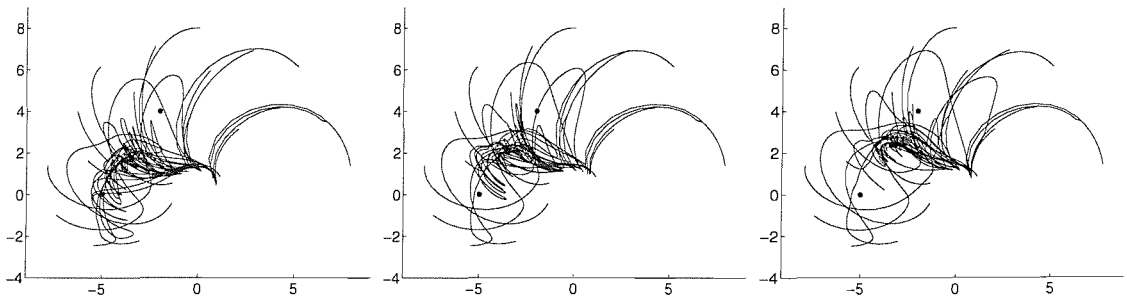
(e) mixed 50:50

(f) mixed 50:50(top view)

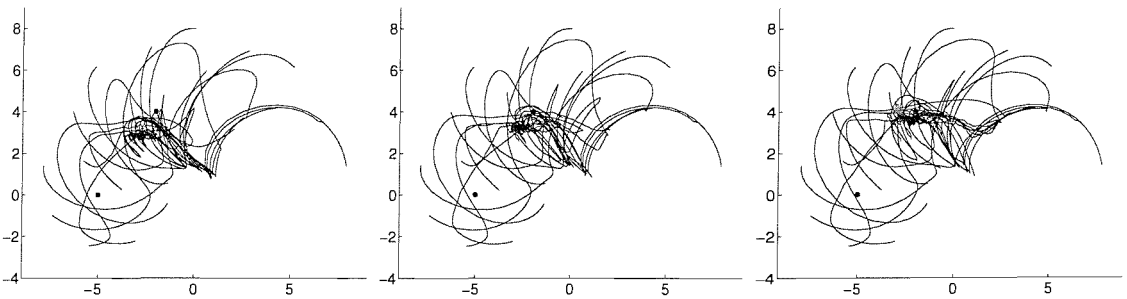
FIGURE 6.10: Comparison of the best network achieved with 20 hidden nodes for two different targets. (e-f) show the output of a blended network that used output signals from both the above networks.



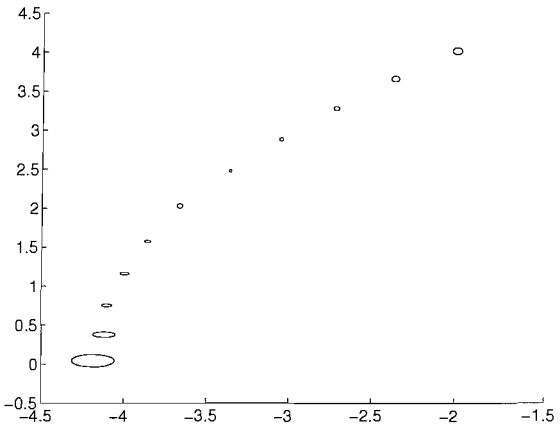
(a) Blending ratio = 10:90 (b) Blending ratio = 20:80 (c) Blending ratio = 30:70



(d) Blending ratio = 40:60 (e) Blending ratio = 50:50 (f) Blending ratio = 60:40



(g) Blending ratio = 70:30 (h) Blending ratio = 80:20 (i) Blending ratio = 90:10



(j) movement of end point

FIGURE 6.11: (a)-(i) shows the top view of a range of mixes of the controlling networks. (j) shows the mean position of the end point for the above graphs. Ellipses are centred on the mean position of the final 250 simulation steps for each of the 25 runs. The ellipse on the top right is the field one response and the ellipse on the bottom left is the field two response. The intermediate ellipses show blended responses created using an evenly spaced range of mixing ratios. Ellipse width and height are proportional to the mean of the standard deviations for each of the 25 start poses over the same range of simulation steps, plotted at three times the axis scale.

There are several potential reasons for this failure: a limitation of the model mechanics, controller capacity or of the training approach.

Classical control approaches could be used to design an optimised controller for this simplified compliant arm, and thus demonstrate whether smooth straight line trajectories are possible with the dynamic arm model. Unfortunately, the time it would take to design such a controller was not justifiable at present. A second approach could be to implement a learning system that performed mutation hill climbing on a vector that contained both the arm's mechanical parameters and the MLP's weights and biases. This would allow the technique to tune these in parallel and should demonstrate whether it is possible to control the arm in the way envisioned. This would present an interesting research challenge in itself but is, of course, a different problem to the one tackled in this work, which focused on whether the learning technique could exploit the predefined mechanical system.

Mechanical restrictions aside, it is likely that the direct control approach outlined in this chapter will not be able to produce human-like reaching motions. The following chapters outline a more complicated, novel controller that borrows from equilibrium point control, as well as convergent force field control. This controller has some limited internal state, the encoding of which is explored and analysed.

Chapter 7

Convergent Equilibrium Trajectory Control Model

This chapter outlines a novel framework that combines equilibrium trajectory control and convergent force field control (Fig 7.1). It is hoped that it will provide a useful foundation for the development of robust autonomous robotic reaching behaviours. As it is envisioned, convergent equilibrium trajectory control (CETC) could be extended to control more complicated articulated structures, including artificial hands. Each stage in this framework can be seen as a mapping that modifies the apparent dynamics of the following stages; the modification should be advantageous to the proceeding stages by reducing dimensionality, redundancy or mapping-complexity. With this in mind, the following sections describe the various stages of the model, working from the actuators back up to the top level input.

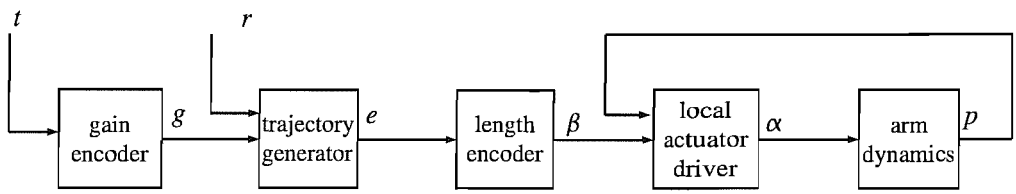


FIGURE 7.1: Proposed framework. The target pose (t), is encoded in terms of the gains of the component force-fields ($\boldsymbol{g}$). These gains are used to drive the internal representation of the equilibrium point ($\boldsymbol{e}$) towards the target defined by $\boldsymbol{g}$ at the drift-rate (r). A set of equilibrium muscle lengths (β) is derived from $\boldsymbol{e}$. The final stage combines β with proprioceptive feedback ($\boldsymbol{p}$) to generate actuator drive signals ($\boldsymbol{\alpha}$).

7.1 Local Actuator Driver

In the proposed framework the local actuator driver takes the place of the local feedback loops present in muscles (i.e., Renshaw cell, spindle, and Golgi feedback, as discussed in section 3.2). This feedback should provide the preceding stages of the controller with appropriate arm and actuator dynamics for equilibrium trajectory control. Central to the success of this approach is the assertion that constant rate movement of the arm equilibrium point will result in smooth acceleration and deceleration of the arm's end point (Gribble et al., 1998).

For a given time step, n, and actuator, i, the local actuator driver combines the required-rest length, β_{i_n} and the available proprioceptive information, p_{i_n} to calculate the optimal actuator input signal, $\alpha_{i_{n+1}}$. Although there is an independent input signal for each actuator, the feedback depends on the dynamics of the whole arm model and is therefore a function of all the actuator inputs.

$$\alpha_{i_{n+1}} = d_i(\beta_{i_n}, p_{i_n}) \tag{7.1}$$

The exact parameters of the transfer function, d_i, will depend on the properties of the individual actuator being controlled, but some commonality is likely. The separation of the transfer functions assumes that the feedback from one actuator should not influence the input of another. If this was found to be overly restrictive, the parallel mapping functions d_i could be replaced by a single mapping with access to all the actuators' feedbacks.

$$\boldsymbol{\alpha}_{n+1} = d(\boldsymbol{\beta}_n, \boldsymbol{p}_n) \tag{7.2}$$

Although this would be a more flexible mapping, it would come at the cost of more free parameters, and would therefore be harder to train. It is therefore important to decide whether this cross-feedback is required.

7.2 Length Encoder

The local actuator driver must be supplied with appropriate rest-length information, $\boldsymbol{\beta}$, so that it can re-tune the actuators to align the arm's mechanical equilibrium point with the desired equilibrium point, $\boldsymbol{e}$, demanded by the preceding stages of the controller.

$$\boldsymbol{\beta} = b(\boldsymbol{e}) \tag{7.3}$$

In conventional robotics, inverse kinematics would be used to calculate the required joint parameters, and then an abstract geometric model of the arm would be used to translate this into appropriate actuator rest-lengths. This framework aims to avoid such direct calculations for two reasons.

Firstly, we do not want to assume access to an accurate abstract geometric model of the arm because this would make the mapping unsuitable for learning. Such a system would therefore have to be preprogrammed and would have limited potential to respond to changes in the mechanics of the system – something that will become increasingly important as articulated robots become more complicated.

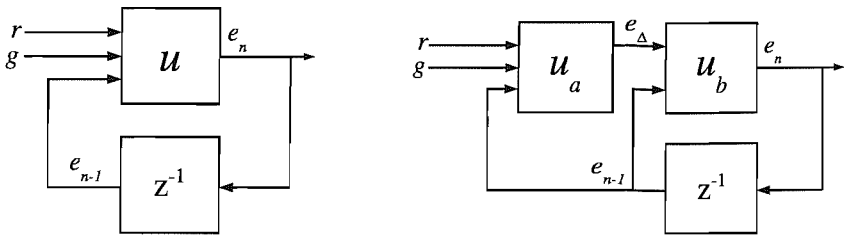
Secondly, it is important that the system is capable of developing appropriate amounts of co-activation (i.e., tightening opposing actuators) so as to increase the overall stiffness, or *tone*, of the arm. People often use more stiffness when attempting novel tasks, where they are less likely to be able to predict changes in loading and disturbance. Clearly, over-using co-activation incurs energy costs and possibly reduces responsiveness, while under using co-activation can lead to disturbances impairing task performance. This therefore implies that the length encoder mapping, b, will be to some extent dependant on the novelty of the current task.

7.3 Trajectory Generator

For simple point to point reaching the trajectory generator is responsible for moving the equilibrium point, $\boldsymbol{e}$, at a constant rate towards the target point and ensuring that it comes to rest immediately on arrival (i.e., no overshoot or oscillation). This sort of motion is possible with the equilibrium point because it is purely an internal state of the controller, and therefore has no inertia. The equilibrium point discussed here is the *ideal equilibrium point* which is assumed to be always identical to the equilibrium point achieved by the arm itself. In any real implementation there will be some mechanically induced constraints (both in terms of lag and range) that must be accounted for, but for simplicity this has not been included in the current model. If we consider time step n, the trajectory generator is responsible for calculating the next equilibrium point, $\boldsymbol{e}_{n+1}$, from the current equilibrium point, $\boldsymbol{e}_n$, the current field gain mixture vector, $\boldsymbol{g}$, and the drift-rate, r.

$$\boldsymbol{e}_{n+1} = u(\boldsymbol{e}_n, \boldsymbol{g}, r) \tag{7.4}$$

As we are currently only considering point to point reaching, it is tempting to assume that u could be replaced by a reasonably simple direct algebraic expression. However it is important to remember that we are developing a general framework that will be required to produce more complicated behaviours in future, potentially including cyclic



(a) Direct (b) Indirect

FIGURE 7.2: Two alternative trajectory generators. (a) a single mapping creates the next equilibrium point encoding directly from its previous value. (b) calculates an intermediate representation, $\boldsymbol{e}_\Delta$, of the step to be taken.

motion. We should, therefore, be careful to balance simplicity against flexibility and not rush to constrain the system prematurely.

The direct mapping, u, requires that the network provides an output which is nearly the same as one of the inputs ($\boldsymbol{e}_n$) but that has been changed by a small amount. To maintain a high enough refresh rate for the arm, the change required is in the region of 10–100 times smaller than the magnitude of $\boldsymbol{e}_n$. It may therefore be advantageous to split the trajectory generator into two parts: a movement direction generator, u_a, which calculates the required change, $\boldsymbol{e}_\Delta$, and an integrator, u_b which updates the value of $\boldsymbol{e}$ accordingly. So for a given instant in time, n, we have:

$$\boldsymbol{e}_{n_\Delta} = u_\mathrm{a}(\boldsymbol{e}_n, \boldsymbol{e}, r) \tag{7.5}$$

$$\boldsymbol{e}_{n+1} = u_\mathrm{b}(\boldsymbol{e}_n, \boldsymbol{e}_{n_\Delta}) \tag{7.6}$$

Figure 7.2 contrasts direct and indirect trajectory generation. Indirect updating would allow u_b to be replaced with direct vector summation if $\boldsymbol{e}_{n_\Delta}$ is encoded in the same way as $\boldsymbol{e}_n$, but allows different encodings to be employed if required.

7.4 Gain Encoder

The gain encoder in Figure 7.1 is responsible for generating a mixture of gains, $\boldsymbol{g}$, such that the resulting movement of the arm meets the demands of the top-level controller, expressed as $\boldsymbol{t}$.

$$\boldsymbol{g} = f(\boldsymbol{t}) \tag{7.7}$$

The gain vector can be seen as a simple type of population encoding of target position. Population encodings have been observed within the motor cortex of monkeys performing reaching tasks and are thought to store the desired direction prior to movement (Georgopoulos et al., 1988, 1992; Georgopoulos, 1995).

There are two, fairly distinct ways this could be achieved. The first option is for the top-level controller to specify an end point in some convenient coordinate system, like head-centred polar, which the gain encoder can map directly to a gain encoding. The problem with this approach is that all tasks then have to be converted into a series of point-to-point reaching movements. This would require the top-level controller to perform complex task decomposition, which may not always be desirable.

The second approach is to encode $\boldsymbol{t}$ in a more application orientated *action-parameter* space, where different locations represent different useful responses, e.g., *move-arm-in-to-protect-body*, *move-arm-firmly-outward-to-repel*, *move-arm-left-to-pin-object*, etc. It would be significantly more challenging to develop a method for training the second approach, but it may lead to a system that is of more value to behaviour based robotics developers.

7.5 Control Signal Encoding

The framework described above outlined the various mappings required to build the controller. It did not, however, define how the control signals were to be encoded. In a system where all the parameters are accurately known, precise selection of encoding is not essential as long as the complete system is fully defined, as any information not directly communicated can be inferred. In the framework proposed, each stage only has access to the signals directly communicated to it, and must gain all other information by learning (either online or offline). It is therefore important that signals are encoded such that learning the appropriate mapping is as easy as possible. This section outlines some of the potential encoding options for the equilibrium point, $\boldsymbol{e}$.

Cartesian : The most direct way to encode would be in some Cartesian frame, that was fixed with respect to the body frame. It would be necessary to define an origin, a rotation and a scale factor. For this work it is assumed that there is no rotation of the Cartesian frame, and it is scaled such that the entire workspace lies within ± 1 on each axis.

Shoulder Centred Polar : An alternative, and equally abstract, way of representing the equilibrium point is in shoulder centred polar coordinates. As the origin is already defined, we need only define any rotation with respect to the workspace, and any scaling. Zero degrees was taken to lie on the line between the shoulder and a point in the middle of the workspace, and all angles and distances were rescaled to lie between ± 1.

Joint Angle Encoding : Encoding the equilibrium point using joint angles may make the calculation of appropriate β easier, but would make moving at a constant rate in body space significantly harder. This would be pose encoding, rather than point encoding, which would make a difference if one was controlling a redundant arm, as it would avoid the need to build a length encoder that could select the ideal solution from the possible alternatives.

Relaxed β Encoding : An alternative pose encoding would use the actuator rest-lengths. As already mentioned, the length controller should manage the amount of co-activation in the arm, but if one considers the case where there is no co-activation, there is a one-to-one mapping from valid configurations of $\boldsymbol{e}$ to β. This may allow $\boldsymbol{e}$ to be represented as a set of muscle rest lengths. However, this is likely to make learning the trajectory generator harder. If this is possible though, it would represent the most elegant solution as it means there would be no need for a separate length encoder stage, significantly simplifying the proposed framework.

Chapter 8 explores the trajectory generator stage, and demonstrates an initial implementation. Chapter 9 briefly presents an implementation of the the length encoder stage and compares its results with those obtained for the trajectory generator.

Chapter 8

Trajectory Generation

The trajectory generator is responsible for moving an internal representation of the equilibrium point towards a target specified by the gain vector (see Fig. 7.1). This chapter details work which first defines the behaviour of an ideal trajectory generator and then implements it using an MLP.

8.1 Definition of Ideal Trajectories

This section outlines a procedure that was used to create an ideal trajectory generator. The objective here is to calculate a small, fixed length step, in a direction specified by the gain vector. The movement should be along the shortest path between the current equilibrium point and the target point that does not pass outside the workspace.

8.1.1 Direction of movement

As the purpose of the ideal trajectory generator is to generate training and testing data for the MLP implementation, the procedure outlined is not necessarily suited for deployment in a real controller. In particular it uses an abstract geometric model of the arm extensively, something that we would not wish to rely on in the final implementation.

The input to the trajectory generator, $\boldsymbol{g}$, serves a very similar role to the gain vector used in CFFC; i.e., to control how several component force fields are mixed. In this case the force fields control the movement of the internal equilibrium point, $\boldsymbol{e}$, rather than the end point of the arm. Each of the component fields moves $\boldsymbol{e}$ towards a different

location within the workspace. As we are only interested in the *relative* component field gains, the incoming gain vector is first normalised:

$$\bar{\boldsymbol{g}} = \frac{\boldsymbol{g}}{|\boldsymbol{g}|_1} \tag{8.1}$$

The L^1 norm (the sum of all the components) is used in the normalisation, rather than the more common L^2 norm (the length of the vector in Euclidean space). This means that the sum of the components of $\bar{\boldsymbol{g}}$ is always constant.

The normalisation does introduce some redundancy into the input space, such that several values of $\boldsymbol{g}$ result in the same value of $\bar{\boldsymbol{g}}$. This redundancy might assist in the coordination of higher level behaviours, effectively allowing the controller to respond to the relative balance of demands between two competing higher level behaviours, rather than responding to their actual levels.

When all the components of $\boldsymbol{g}$ are zero, Equation 8.1 does not have a finite answer. In such cases the components of $\bar{\boldsymbol{g}}$ are all set to $1/m$, where m is the number of elements in $\boldsymbol{g}$. This ensures that the behaviour of $\bar{\boldsymbol{g}}$ does not contain any discontinuities as $\boldsymbol{g}$ approaches zero.

Each of the component fields must move the equilibrium point to a different workspace location, and combinations of component fields should move towards stable intermediate locations. The most direct way to achieve this would be to use a weighted sum of unit vectors pointing towards several different carefully selected *target locations* $(\boldsymbol{t}_1, \boldsymbol{t}_2 \ldots \boldsymbol{t}_c)$. This could then be scaled by the rate at which the equilibrium point should move, the *drift rate*, r_n, and used to update the equilibrium point, $\boldsymbol{e}$. So for time step, n, gain vector $\bar{\boldsymbol{g}}_n$, and equilibrium point $\boldsymbol{e}_n$ we can calculate the next equilibrium point, $\boldsymbol{e}_{n+1}$:

$$\boldsymbol{e}_{n+1} = \boldsymbol{e}_n + r_n \sum_{f=1}^{c} \bar{g}_{n_f} \frac{\boldsymbol{t}_f - \boldsymbol{e}_n}{|\boldsymbol{t}_f - \boldsymbol{e}_n|_2} \tag{8.2}$$

where c is the number of component fields, $\boldsymbol{t}_f$ is the target location for field f and $\bar{g}_{n_f}$ is the fth component of $\bar{\boldsymbol{g}}_n$.

This approach is unfortunately flawed as, although it works fine when only one component is active (i.e., $\bar{\boldsymbol{g}}_n$ has only one non zero component), it does not produce good results when several components are blended. In order to preserve proper blending the weighted sum must be applied after the rescaling, thus:

$$\Delta_n = \sum_{f=1}^{c} \bar{g}_{n_f} (\boldsymbol{t}_f - \boldsymbol{e}_n) \tag{8.3}$$

$$\boldsymbol{e}_{n+1} = \boldsymbol{e}_n + r_n \frac{\Delta_n}{|\Delta_n|_2} \tag{8.4}$$

component	x	y
1	7.80	1.65
2	1.20	1.60
3	−0.95	−1.65
4	−7.90	0.40
5	−2.00	7.50

TABLE 8.1: Locations of the five component targets that the trajectory generator uses, as shown in Figure 8.1.

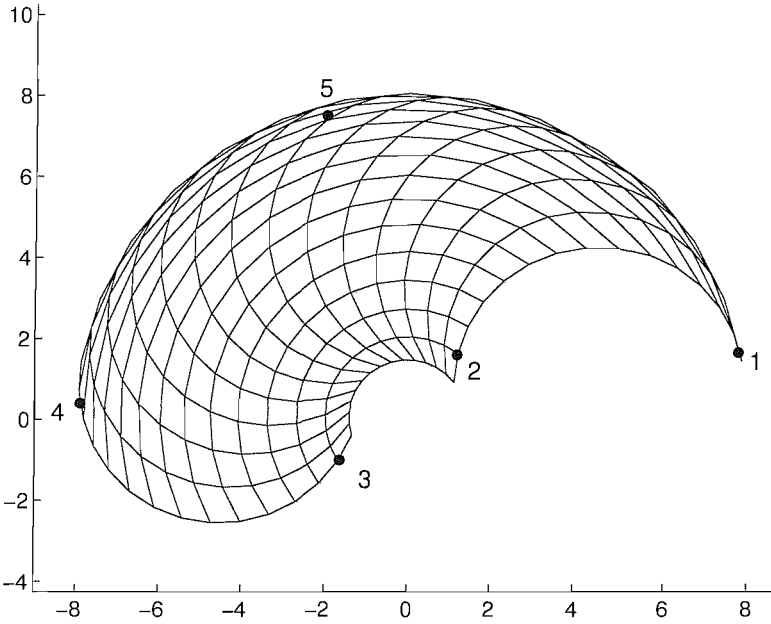


FIGURE 8.1: Workspace of the arm model with the five component targets that the trajectory generator will use.

Using this formulation, it is not possible to specify an intermediate target location that lies outside the perimeter of a region bounded by the defined component target locations. The component target locations should therefore be selected toward the edge of the arm's workspace. For the work presented in the following sections, a trajectory generator with five component fields was modelled with minima positioned as defined in Table 8.1 and shown in Figure 8.1. Unless otherwise stated, the drift rate, r_n, was kept equal to 0.1 throughout.

8.1.2 Ensuring reachability

For most situations, Equations 8.3 and 8.4 produce sensible updates for $\boldsymbol{e}$. Unfortunately, this breaks down towards the edge of the workspace. In these situations, it is quite possible for the update generated to attempt to move $\boldsymbol{e}$ to a location that the arm cannot reach. There are two distinct ways in which this can happen; either it is

physically beyond the reach of the arm at maximum extension or it would require the joint angles to move outside their permitted range. These cases will be dealt with in turn.

The main difficulty with points beyond the reach of the arm is that there is no method to determine appropriate joint angles (they simply do not exist). As later stages require that the joint angles are known, such movements need to be adjusted so that they lie within the range of the arm. The most direct approach would be simply to move the point towards the shoulder until it was within the reach of the arm. This would unfortunately have the side effect of reducing the distance between the old $\boldsymbol{e}$ and the new one. Although this will not cause a problem in most cases, there is the potential for this to create a dead end, such that $\boldsymbol{e}$ comes towards the edge of the workspace and stops moving. This is unacceptable as it means that $\boldsymbol{e}$ will never reach the target location.

A procedure was therefore used to ensure that the updated $\boldsymbol{e}$ lay inside the reach of the arm while maintaining the same step length. This was achieved by finding the intersection(s) of a circle, c_{step}, centred on $\boldsymbol{e}_n$, with radius equal to drift rate r_{step}, and a circle, c_{arm}, centred on the shoulder with radius equal to maximum reach of the arm, r_{arm}. There are four types of solution to such a system:

1. No real points of intersection.
2. One point of intersection, i.e., the centres are $r_{\text{step}} + r_{\text{arm}}$ apart.
3. Two points of intersection.
4. Infinite points, i.e., the circles have identical radii and centres.

We can assume the initial equilibrium point, $\boldsymbol{e}_n$ is reachable, and therefore inside c_{arm}. If we only attempt to adjust the position of the updated point, $\boldsymbol{e}_{n+1}$, when it is outside (not on, or inside) c_{arm} then we need not consider answers of type one or two. For any reasonable configuration we can assume that $r_{\text{step}} \ll r_{\text{arm}}$, so we need not consider answers of type four.

We therefore need only consider solutions of type three, where there are two points of intersection. We therefore must consider which of the two solutions, $\boldsymbol{e}'_{n+1}$ and $\boldsymbol{e}''_{n+1}$, to use as a replacement for the original $\boldsymbol{e}_{n+1}$ (Fig. 8.2). To maintain as much of the sense of the update as possible, the best choice is the solution nearest to $\boldsymbol{e}_{n+1}$.

8.1.3 Target substitution

The arm workspace is not convex. Specifically there are two regions within the maximum boundary that cannot be reached due to joint limits, as shown in Figure 8.3.

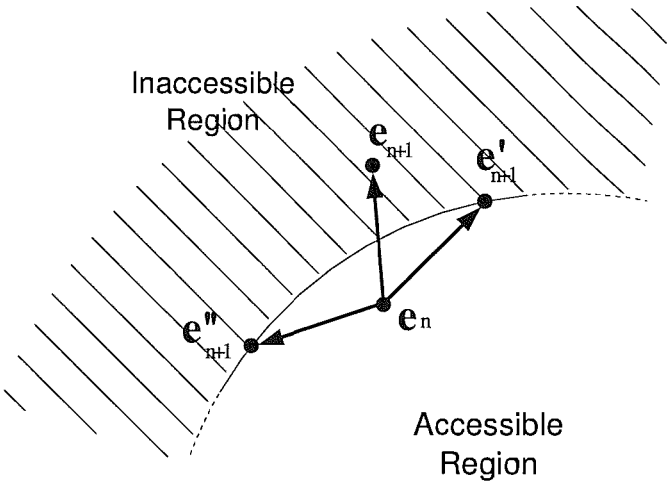


FIGURE 8.2: Updates are modified to ensure they are reachable. When e_{n+1} is not reachable, two points e'_{n+1} and e''_{n+1} are calculated that are just reachable and the same distance from e_n as e_{n+1}. The closest of these two to e_{n+1} is then used as a substitute point.

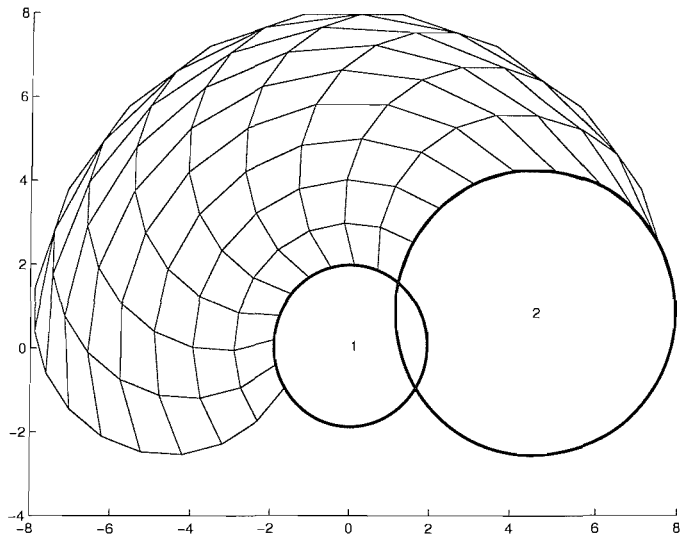
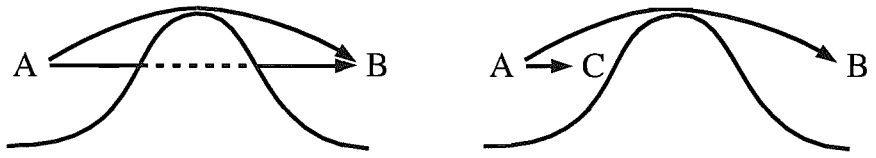


FIGURE 8.3: There are two regions, labelled one and two are within the overall perimeter of the workspace, but that cannot be reached due to joint constraints.

Their boundaries are circular and can be calculated directly from the arm configuration parameters.

This leads to situations where the shortest path between two points within the workspace is not necessarily straight (Fig. 8.4). So what are the implications of this for a trajectory generator that takes any point in the workspace and moves it slightly nearer a target



(a) (b)

FIGURE 8.4: (a) the ideal trajectory when it is not possible to move directly from point A to B because the path passes through a region that is outside the workspace of the arm. (b) the ideal solution needs planning, i.e., even though B and C are in the same direction, the ideal initial movements are not identical.

location? It means a small movement along the shortest path within the workspace can not be calculated without predicting, and correcting for, boundary collisions.

We must therefore be able to predict collisions between these two circular regions and the planned trajectory. This can be solved if we consider a circle, centred on point $\boldsymbol{c}$, and a line segment running from a given point, $\boldsymbol{s}$, to a target point, $\boldsymbol{t}$. Using the parametric line equation, we can define point $\boldsymbol{m}$,

$$\boldsymbol{m} = \boldsymbol{s} + u(\boldsymbol{t} - \boldsymbol{s}) \tag{8.5}$$

We want to calculate the value of u that will make point $\boldsymbol{m}$ closest to the centre of the circle. This happens when the line between the centre of the circle and $\boldsymbol{m}$ is perpendicular to the line between $\boldsymbol{s}$ and $\boldsymbol{t}$. When these lines are perpendicular they will have a dot product of zero:

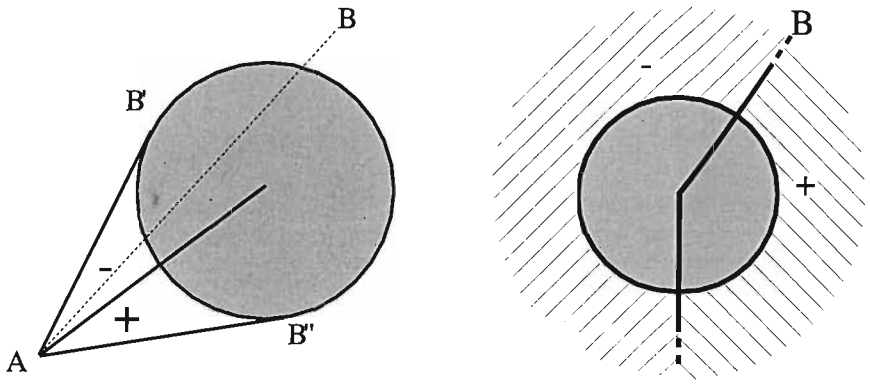
$$(\boldsymbol{c} - \boldsymbol{m}) \cdot (\boldsymbol{t} - \boldsymbol{s}) = 0 \tag{8.6}$$

Once this is substituted back into the parametric line equation, we can solve for u:

$$u = \frac{(c_x - s_x)(t_x - s_x) + (c_y - s_y)(t_y - s_y)}{(t_x - s_x)^2 + (t_y - s_y)^2} \tag{8.7}$$

If $u < 0.0$ or $u > 1.0$ then the closest point on the line to the circle does not lie between $\boldsymbol{s}$ and $\boldsymbol{t}$. In all other situations we can use Equation 8.5 to calculate the coordinates of the nearest point and then test to see if it lies within the radius of the circle.

Once we know that a trajectory would intersect with an unreachable region, we must calculate a virtual target location, $\boldsymbol{t}'$, to use as a substitute. As we are only going to take a fixed length step towards the target location, it does not matter if we vary the



(a) two alternative substitute targets for B. (b) using relative bearing to select substitute

FIGURE 8.5: The selection of which substitute target to use is dependant on the relative bearings of the start point, A, and the target location B, with respect to the centre of the unreachable region, shown here in grey. If point A is in the region marked $-$ in (b), B' should be used. Conversely, if A is in the region marked $+$ the alternative, B'', should be used.

apparent distance between the current equilibrium point and the target. The substituted target will lie on the circumference of the unreachable region, at a point where the tangent intersects with the current equilibrium point. The two solutions can be readily calculated and are depicted in Figure 8.5a.

The only remaining challenge is to select which of the two solutions to use. For the current arm model, the choice of solution depends on the relative bearings of the current equilibrium point and target location with respect to the centre of the unreachable region. When the equilibrium point has a higher bearing than that of the target location, the clockwise substitute point should be used. Otherwise, the anticlockwise substitute point should be used (Fig. 8.5b).

8.1.4 Joint deflection and clamping

Once the new target had been generated and an initial direction calculated, the next stage was to ensure that the arm's joint limits were not exceeded. This was done using a two stage process: deflecting and then clamping. The deflecting stage attempted to twist the update so that the step length remained the same, but was now within the workspace. Note that for efficiency reasons these tests were applied in the joint-angle space, so it was the length of the change in joint angles (not Cartesian space) that was maintained. Once all four joint limits (i.e., high and low on both axes) were deflected, any remaining overshoot was clamped without regard to axis interdependence.

8.1.5 Update generation summary

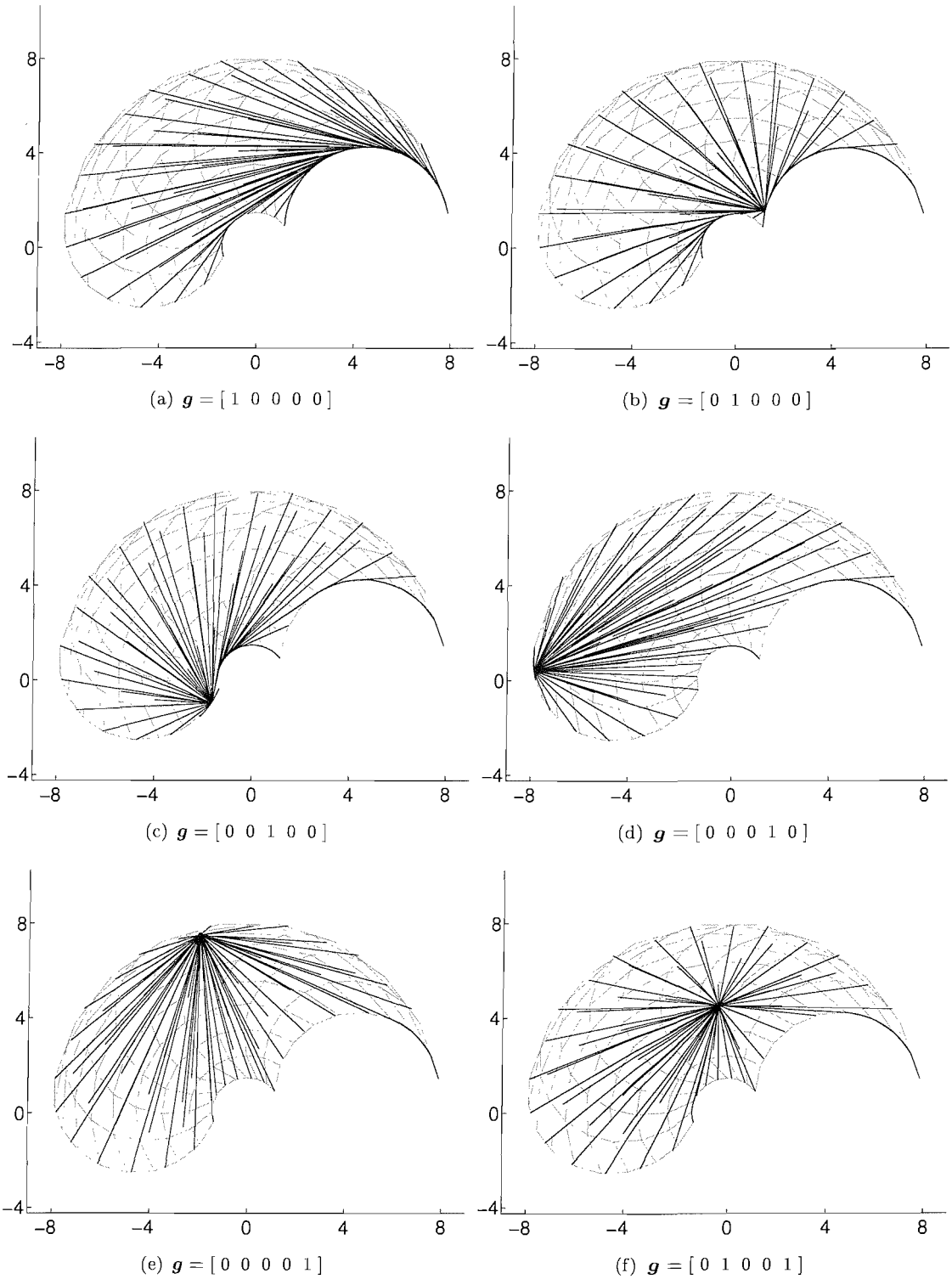
In outline we perform the following stages for every update:

1. Normalise gain vector
2. Calculate target point
3. Test for path passing outside workspace
4. Substitute target if required
5. Calculate step towards target
6. Test for step being outside reach of arm
7. Modify step to make it reachable if required
8. Calculate change in joint angles
9. Deflect step if it crosses joint limits
10. Clamp joint angles if required

This is not a simple process, and does require access to an accurate model of the arm. Figure 8.6 shows the output of this process for six different gain vectors. Figure 8.6a demonstrates target substitution effectively. Even though the final target location is at the end of a very narrow part of the workspace, the ideal trajectory generator is capable of getting there by the shortest route from anywhere within the workspace of the arm. Figure 8.6f demonstrates an intermediate response, with two component fields active.

Figure 8.7 shows the same output, but this time plotted as movements in joint-space. The horizontal axis represents a normalised value for θ_1, and the vertical axis represents a normalised value for θ_2. The same set of gain vectors are used as in Figure 8.6. Unsurprisingly, encoded as joint angles the movements appear significantly more complicated.

Figure 8.8 shows the same set of responses, but this time plotted using normalised shoulder centred polar coordinates. Even though this is an arbitrary encoding (i.e., does not require a geometric arm model for update calculation) the response appears strikingly similar to that of the joint encoding.

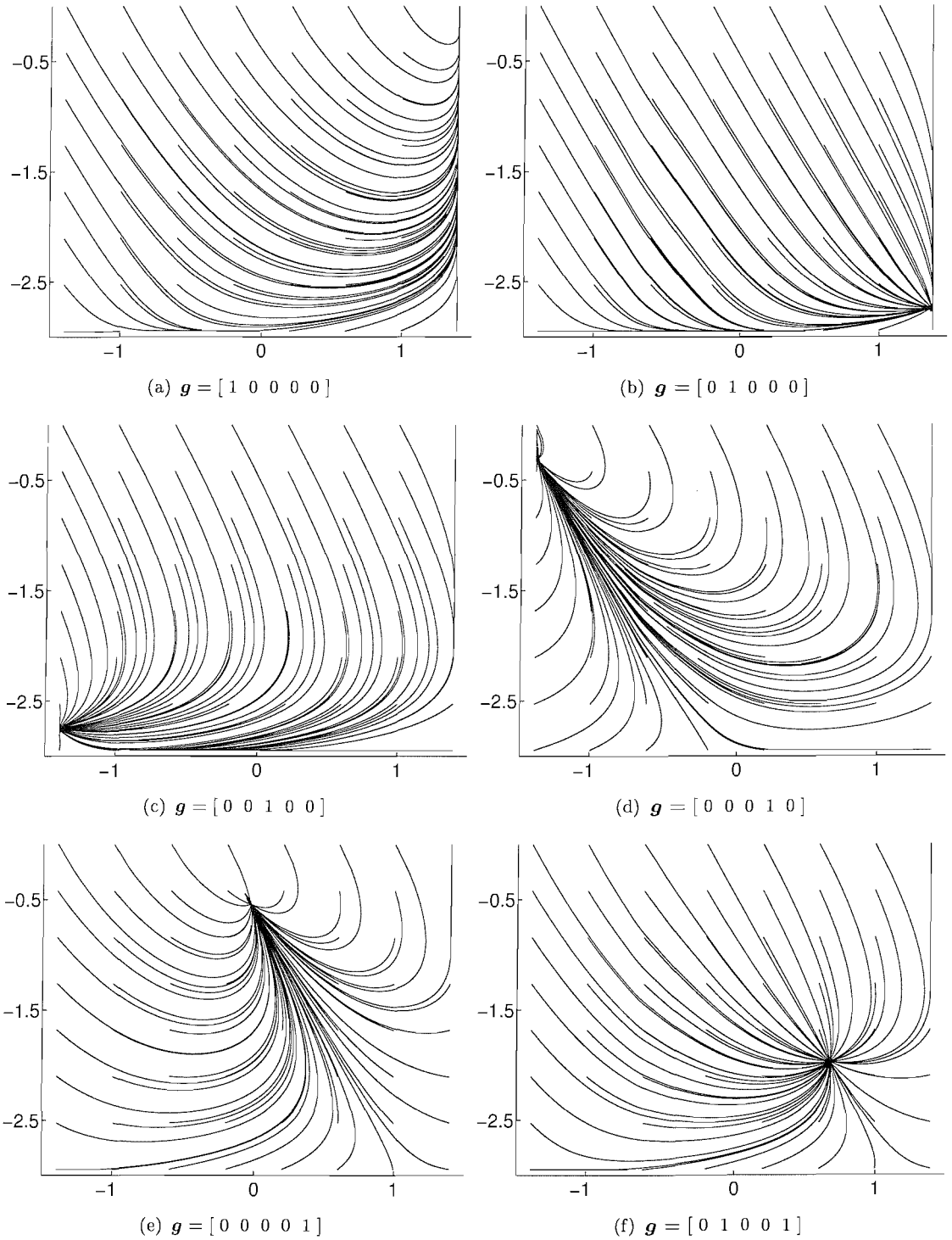


(a) $\boldsymbol{g} = [\,1\ 0\ 0\ 0\ 0\,]$ (b) $\boldsymbol{g} = [\,0\ 1\ 0\ 0\ 0\,]$

(c) $\boldsymbol{g} = [\,0\ 0\ 1\ 0\ 0\,]$ (d) $\boldsymbol{g} = [\,0\ 0\ 0\ 1\ 0\,]$

(e) $\boldsymbol{g} = [\,0\ 0\ 0\ 0\ 1\,]$ (f) $\boldsymbol{g} = [\,0\ 1\ 0\ 0\ 1\,]$

FIGURE 8.6: Output of the ideal trajectory generator plotted in workspace coordinates. The faint grid shows the workspace of the arm, with the dark lines showing the path of the equilibrium point as it converges towards a target. Under each is shown the value of the gain vector, $\boldsymbol{g}$, which was held constant for each trial.

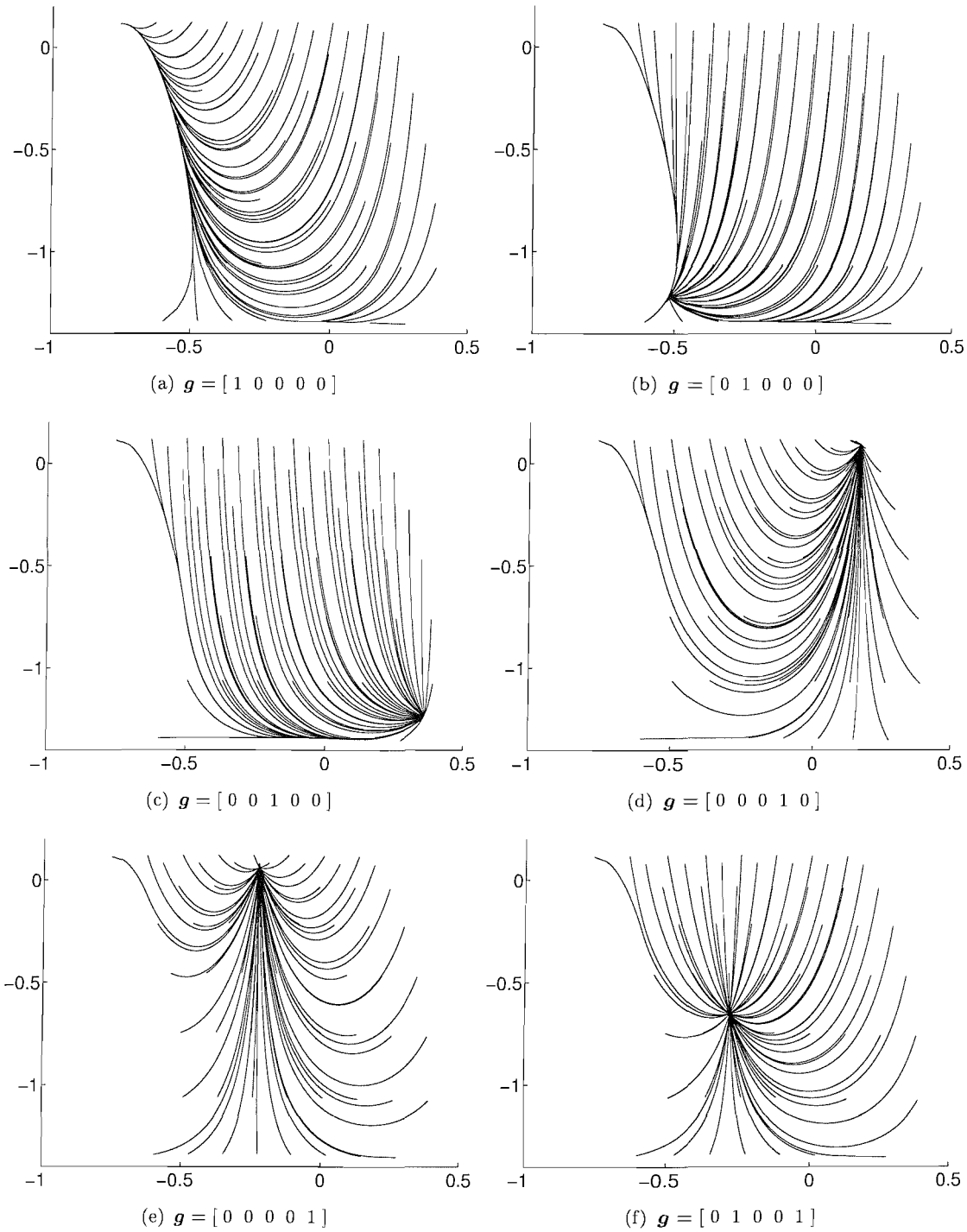


(a) $\boldsymbol{g} = [\,1\ 0\ 0\ 0\ 0\,]$ (b) $\boldsymbol{g} = [\,0\ 1\ 0\ 0\ 0\,]$

(c) $\boldsymbol{g} = [\,0\ 0\ 1\ 0\ 0\,]$ (d) $\boldsymbol{g} = [\,0\ 0\ 0\ 1\ 0\,]$

(e) $\boldsymbol{g} = [\,0\ 0\ 0\ 0\ 1\,]$ (f) $\boldsymbol{g} = [\,0\ 1\ 0\ 0\ 1\,]$

FIGURE 8.7: Graph of the same data as Figure 8.6 but plotted in joint angle space. The horizontal axis shows θ_1 and the vertical axis shows θ_2, both are plotted in radians.



(a) $\boldsymbol{g} = [\,1\ 0\ 0\ 0\ 0\,]$ (b) $\boldsymbol{g} = [\,0\ 1\ 0\ 0\ 0\,]$

(c) $\boldsymbol{g} = [\,0\ 0\ 1\ 0\ 0\,]$ (d) $\boldsymbol{g} = [\,0\ 0\ 0\ 1\ 0\,]$

(e) $\boldsymbol{g} = [\,0\ 0\ 0\ 0\ 1\,]$ (f) $\boldsymbol{g} = [\,0\ 1\ 0\ 0\ 1\,]$

FIGURE 8.8: Graph of the same data as Figure 8.6 but plotted in normalised polar coordinates. The horizontal axis shows angle and the vertical axis shows distance.

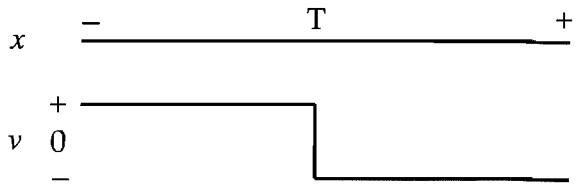


FIGURE 8.9: One-dimensional control to produce constant rate movement toward a fixed target

8.2 Trajectory Generator Implementation

Having decided on the output that the trajectory generator should produce, the next step is to determine how it should be implemented. Before we attempt this, some consideration should be given to the underlying structure of the system. One ideal of CFFC is that the resultant field should be a combination of several independent component fields. From an engineering perspective this distributed structure is likely to have significant advantages, both in required controller complexity and training time.

The most direct way of implementing this distributed structure would be to train a separate component controller for each target point. Each component controller would then be responsible for moving $\boldsymbol{e}$ toward its target at the drift rate, r. The output of these controllers could then be combined using the normalised gain vector. Unfortunately this approach will not work. To demonstrate this clearly, let us consider a trivial one-dimensional tracking problem. In this example the objective is to move $\boldsymbol{e}$ toward a target point on the line at constant rate, r. If we assume that there is only one target, then the correct movement of $\boldsymbol{e}$ is defined by a step function that changes sign as $\boldsymbol{e}$ passes the target (Fig. 8.9).

However, this will not work as a component function because when added it will lead to regions that are non-responsive. To demonstrate this let us consider the same number line, but this time with two target locations, T_1 and T_2. If we combine their independent response functions, v_1 and v_2, to create to create a response function, v_{tot}, it does not drive $\boldsymbol{e}$ to a intermediate target location halfway between T_1 and T_2. In fact it does not drive $\boldsymbol{e}$ at all in the region between T_1 and T_2 (Fig. 8.10).

The desired response can be created if the process is divided into two stages. The individual component functions should respond linearly to the distance between their target point and $\boldsymbol{e}$. These components can then be combined using the normalised gain vector. The combined output can then be rescaled to maintain constant rate movement. Using this process it is possible to create intermediate responses that move $\boldsymbol{e}$ to any point between T_1 and T_2 by varying the gain vector (Fig. 8.11). So what implications does it have for two-dimensional control of $\boldsymbol{e}$?

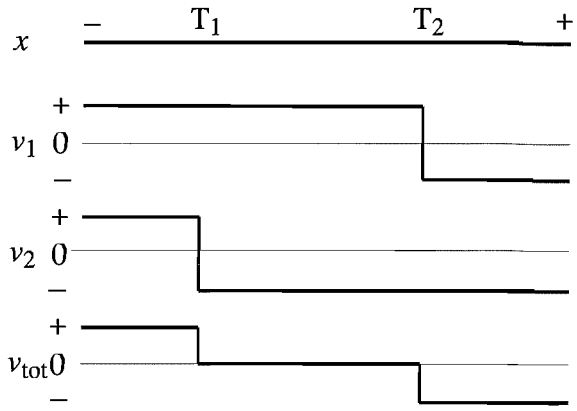


FIGURE 8.10: One-dimensional control to produce constant rate movement. Half the response of two targets, v_1 and v_2, is summed to create v_{tot}. It is clear that v_{tot} will not move e to a point on the line halfway between T_1 and T_2

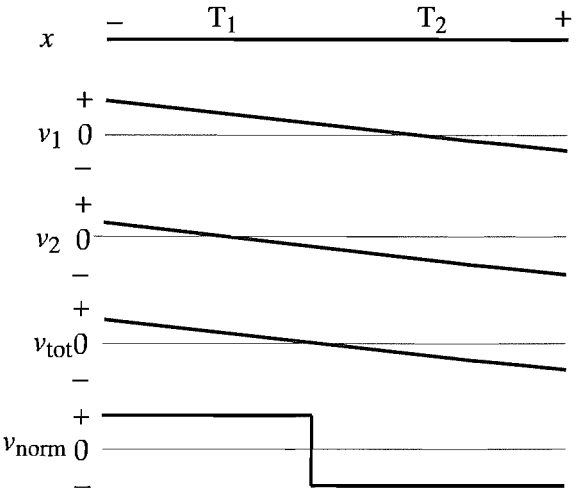


FIGURE 8.11: One-dimensional control to produce constant rate movement. The response of two targets, v_1 and v_2, is summed to create v_{tot}. In this example the result is usable, and only needs to be thresholded to create the ideal combined response function, v_{norm}.

To create a set of two-dimensional component responses that will combine to create a full range of intermediate responses, the output of the ideal trajectory generator needs to be scaled. Specifically, the relative size of the response at each location needs to be proportional to the distance from the target location. When these are combined it is unlikely that they will produce exactly the same results as those that would be generated by the ideal trajectory generator, particularly as when a target point has been substituted the route to the target is longer than the distance between the location and the target point.

Figure 8.12(a)–(e) shows the new, scaled, ideal component fields for each of the five target locations. Subfigures (f)–(h) show various intermediate responses, created by

direct blending of two of the five component fields. It should be stressed that these blends have been created by mixing the component fields, rather than re-running the ideal trajectory generator. The intermediate fields have a qualitatively similar structure to that of the component fields.

There are, however, some differences between the way this process, and the ideal trajectory generator, creates intermediate responses. Figure 8.12(h) shows the situation where responses (b) and (c) have been evenly mixed. The ideal trajectory generator would create a response that attempted to move e to a point half way between the target locations 2 and 3, which in this case would be outside the workspace of the arm. The net result is that, although the response attempts to converge towards a single point, as that point is outside the workspace, it is bound to fail. In contrast the response created by blending, as shown in (h), does converge towards a sensible point within the workspace. In this way the blending approach creates a more useful response than the 'ideal' trajectory generator.

So does this process work if the equilibrium point is encoded in a different way? Figures 8.13 and 8.14 show a similar set of graphs using polar and joint encodings respectively. In both these cases the five components are generated by calculating the scaled ideal component fields in Cartesian coordinates and then converting them into the alternative equilibrium point encodings. The intermediate fields are calculated by direct mixing of these alternative encodings of the component fields, and do not use any conversion back into Cartesian coordinates. Careful inspection of these figures will reveal that, even though the shape of these fields is more complicated than the Cartesian ones, they can still be mixed successfully to create a smooth range of intermediate fields.

8.3 Selection of Equilibrium Point Encoding

Having established that the component addition can be performed using any of the equilibrium point encoding schemes, the next stage was to implement a component field controller. A component field controller is responsible for updating the equilibrium point so that it moves towards a single target point from any location with the workspace of the arm. A 15 hidden node MLP, with a sigmoidal hidden layer activation function and a linear output layer activation function, was used. The objective was to determine if any of the encoding schemes had an advantage in terms of training time and overall performance.

To this end the following trial was performed. A set of scaled ideal field components, similar to those previously presented, was calculated for each encoding scheme. Each set contained 341 training points, spread evenly through the workspace of the arm, with respect to Cartesian coordinates. To improve accuracy, for each encoding scheme two controllers were trained for each of the five component fields. For each controller

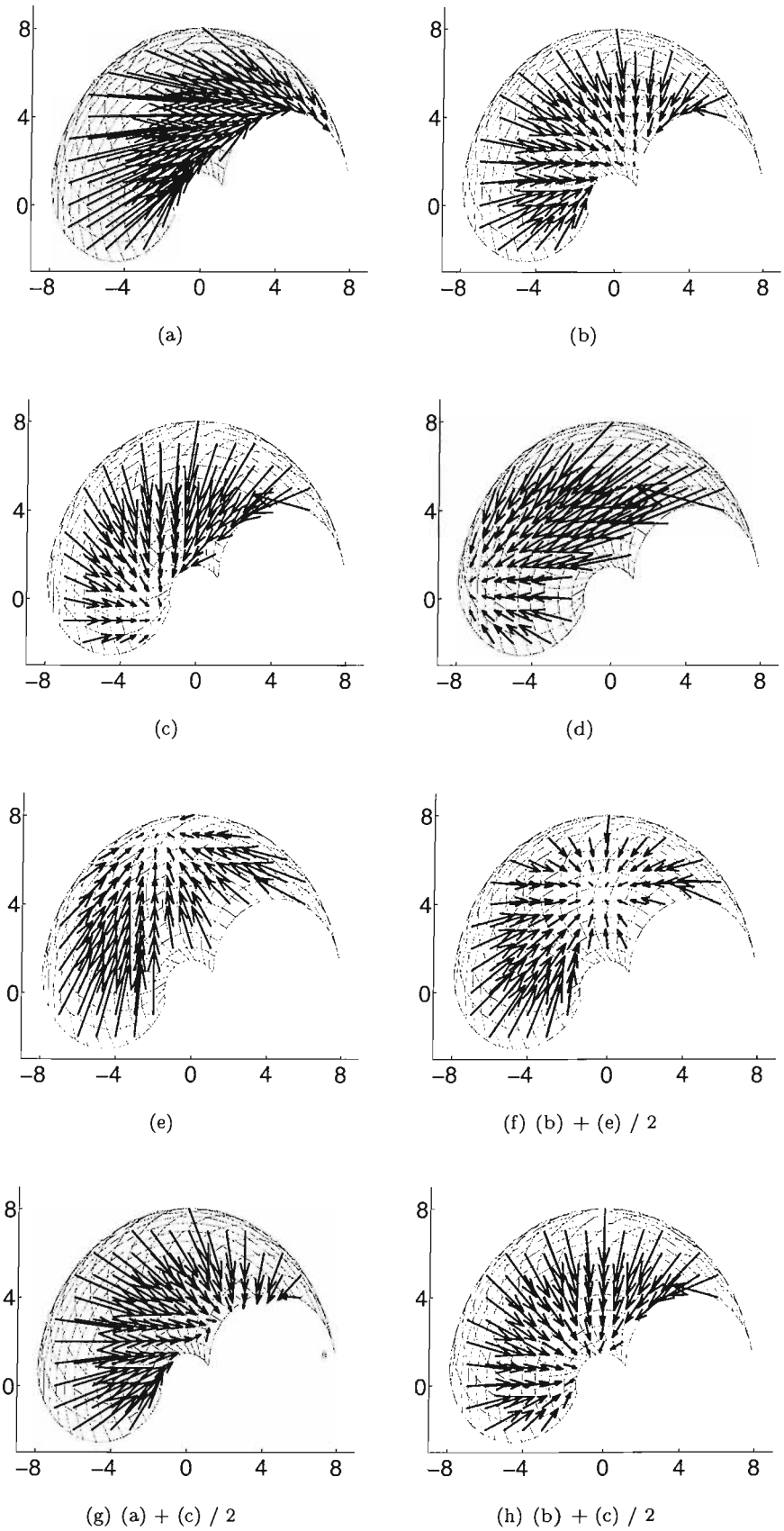


FIGURE 8.12: Ideal component fields. (a)-(e) are generated directly by scaling the output of the ideal trajectory generator, but (f)-(h) are calculated by combining two of (a)-(e) as labelled. It is clear that although the lengths of the updates diverges from the ideal, the components of this derived field have the correct orientation

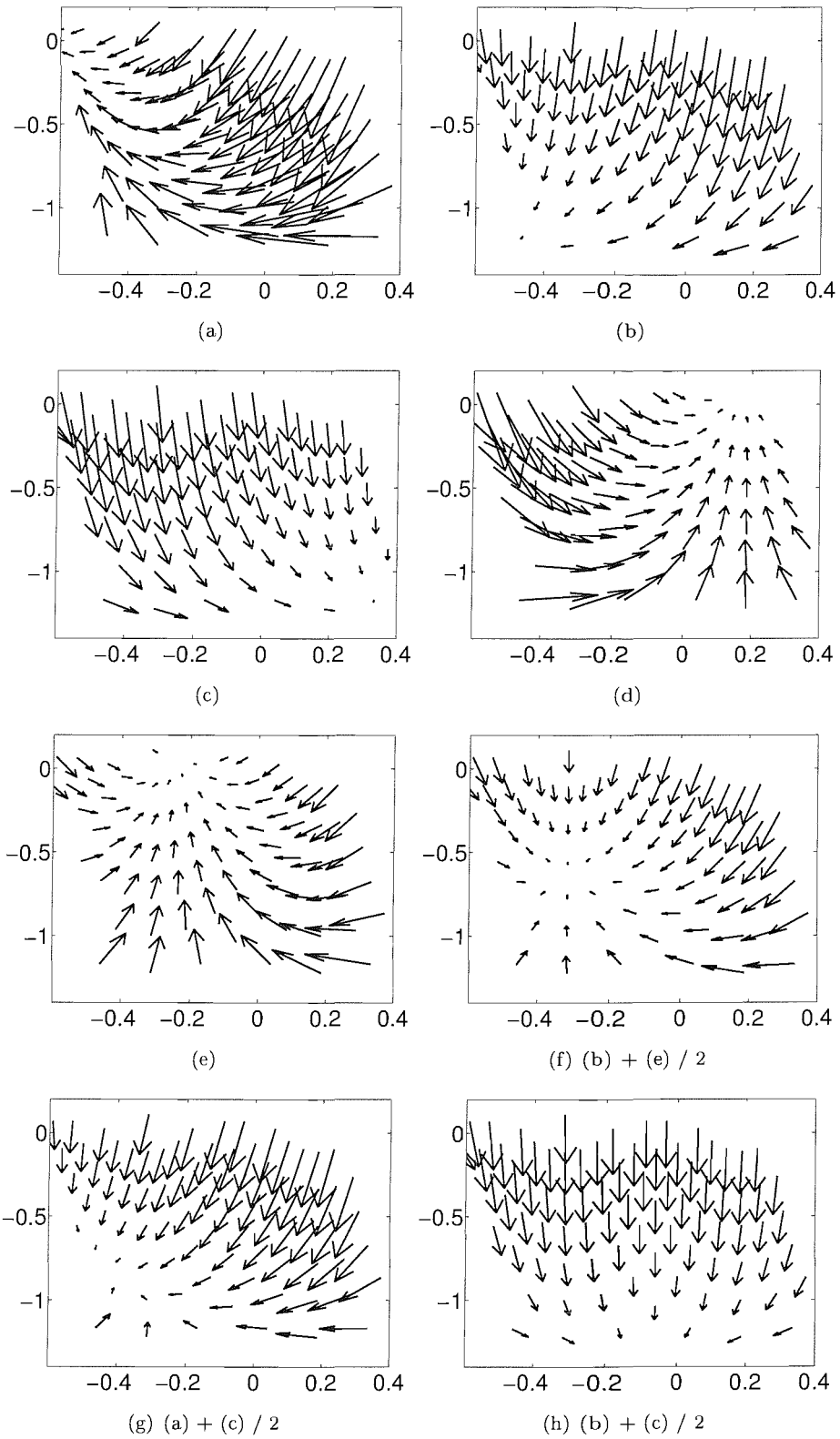


FIGURE 8.13: The ideal component fields expressed in shoulder centred polar coordinates, with combination performed after conversion. Horizontal axis shows normalised angle and vertical axis shows normalised distance.

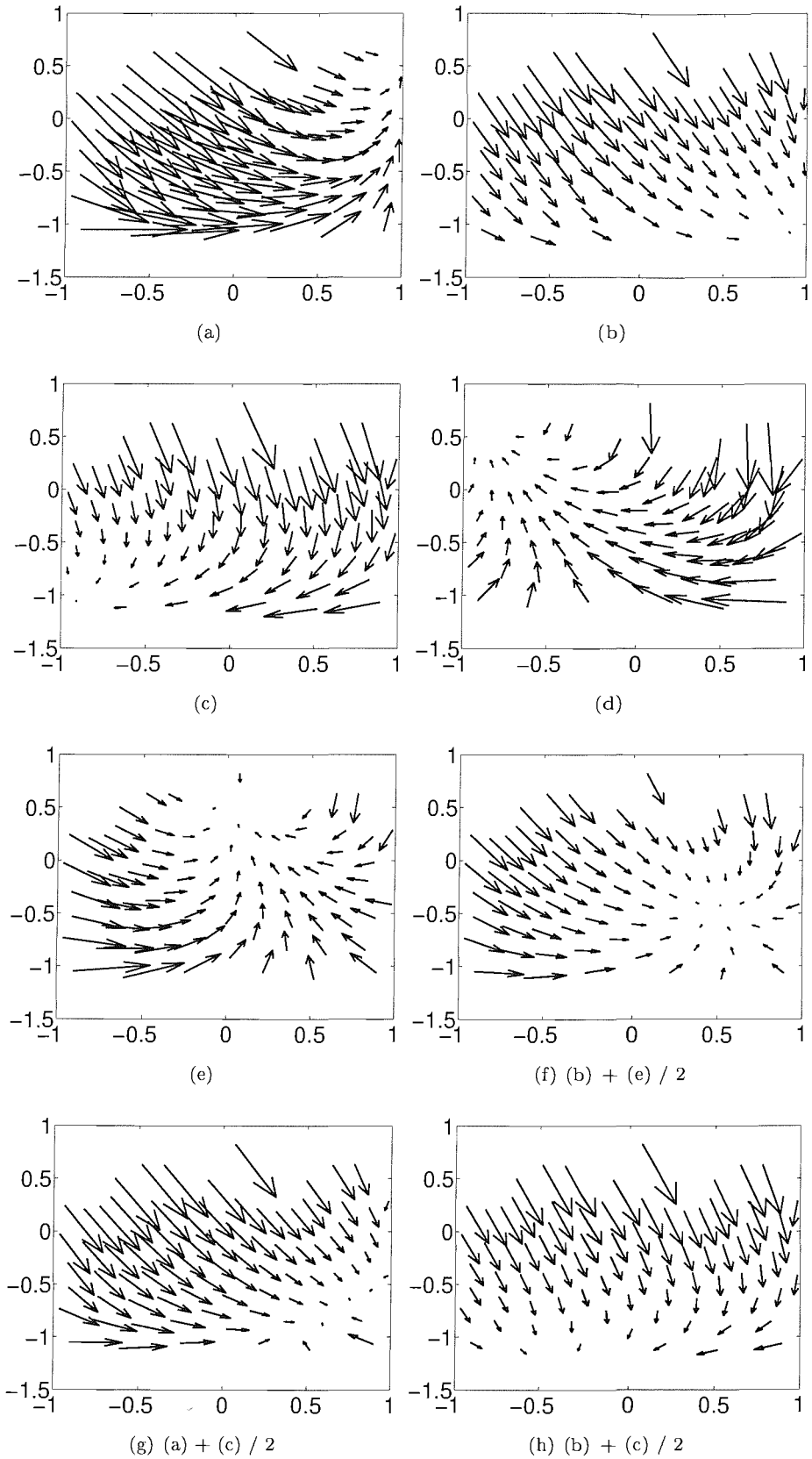


FIGURE 8.14: The ideal component fields expressed in joint angle coordinates, with combination performed after conversion. Horizontal axis shows the normalised joint one angle and the vertical axis shows the normalised joint two angle.

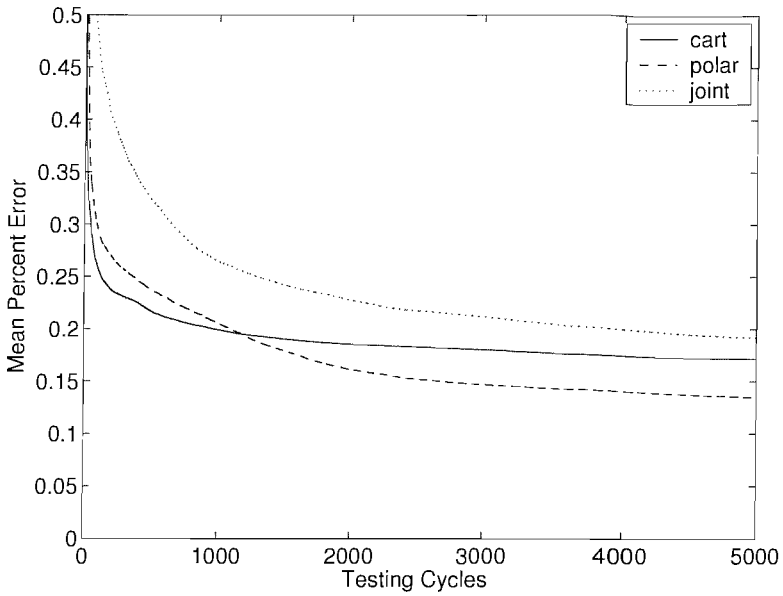


FIGURE 8.15: Mean error expressed as a percentage of mean training output value for three different encodings; Cartesian, polar and joint. Each testing cycle (x-axis) contained 20 training cycle.

a set of 500 randomly chosen test points was generated, along with appropriate ideal responses. Each controller was trained for 20 cycles of backpropagation and then had its performance assessed using its collection of test points. This process was then repeated 5000 times. The backpropagation routine was performed by the NETLAB toolbox, with the termination criteria disabled to ensure that each controller was trained for the same number of cycles. The backpropagation routine used scaled conjugate gradient optimisation.

Figure 8.15 shows the testing error for each of the encoding schemes plotted against testing cycles. The error is expressed as a percentage of the mean length of the testing output values. Figure 8.16 shows the mean of the standard deviation of the testing errors, again expressed as a percentage of the mean length of the ideal testing output.

All three equilibrium point encoding schemes were capable of creating effective trajectory generators with this capacity of MLP. That said, there are some observable differences. In the early stages (less than approximately 1000 training cycles), the Cartesian encoding outperforms the others. Despite this advantage it appears to plateau at around 0.2%, while joint and polar encodings continue to improve asymptotically.

Although we cannot be certain as to the cause of this difference, it is possible to speculate. For the greater part of the workspace, the Cartesian encoding is merely required to produce a response linearly proportional to the distance from a fixed point. In contrast the other two encodings have more complicated, curved, trajectories to generate. This may help explain the Cartesian encodings' initial strong performance. So why does

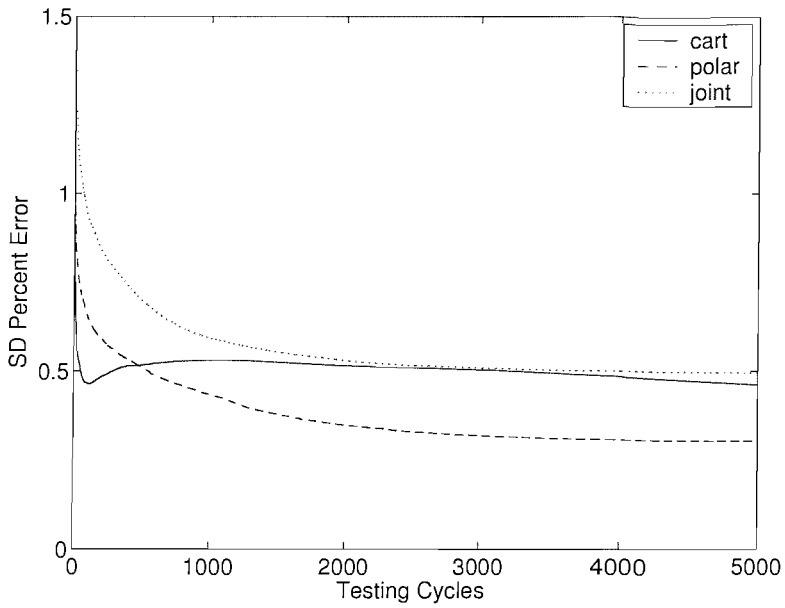


FIGURE 8.16: Mean standard deviation (SD) error expressed as a percentage of mean training output value.

it fall behind? There are several regions of the workspace where target substitution has a strong effect. These regions may be easier to detect in polar centred shoulder or joint angle encoded spaces. If this was the case, this would allow these alternative encodings to respond better to the region specific mapping variations. This hypothesis is supported by Figure 8.16, which shows the standard deviation of the error. This gives us a indication of how the error was spread throughout the workspace. The standard deviation for Cartesian encoding decreases rapidly toward a minimum value at around 100 testing cycles. After this point it worsens. In contrast the standard deviation of the joint and polar encodings improve monotonically. The standard deviation gives some indication of how consistently the trajectory generators performed throughout across the workspace; the lower the standard division, the more consistent the performance. It is possible that the Cartesian encoding system initially learns the correct response for the majority of the workspace (hence the initial rapid reduction in the SD) but is not good at extending this towards the peripheries.

To reduce simulation time, the work presented in the remainder of this chapter was restricted to a single encoding scheme. Polar encoding was selected as it appears to be the most effective in terms of mean percentage error.

8.4 Comparison of Network Capacities

The next stage was to determine the relationship between a controller's capacity and its ability to learn the component fields. To this end, seven MLPs were trained with

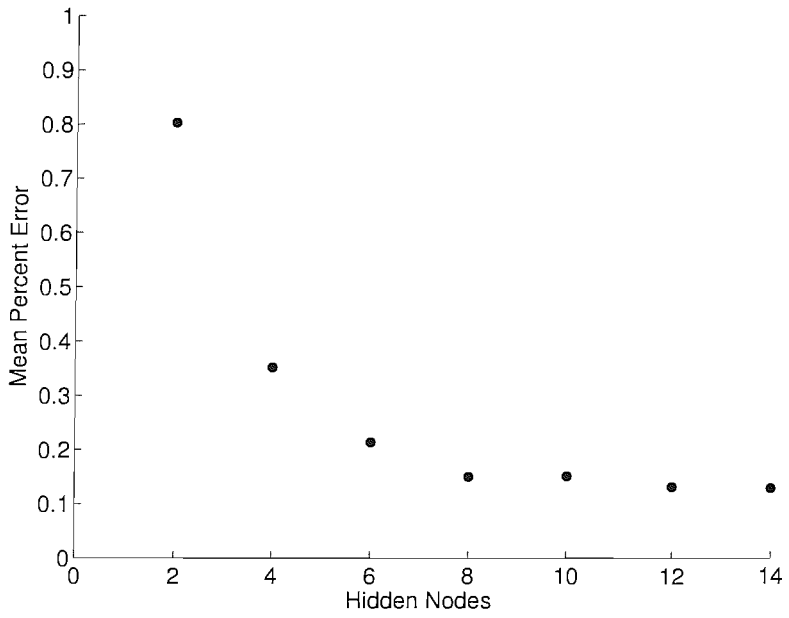
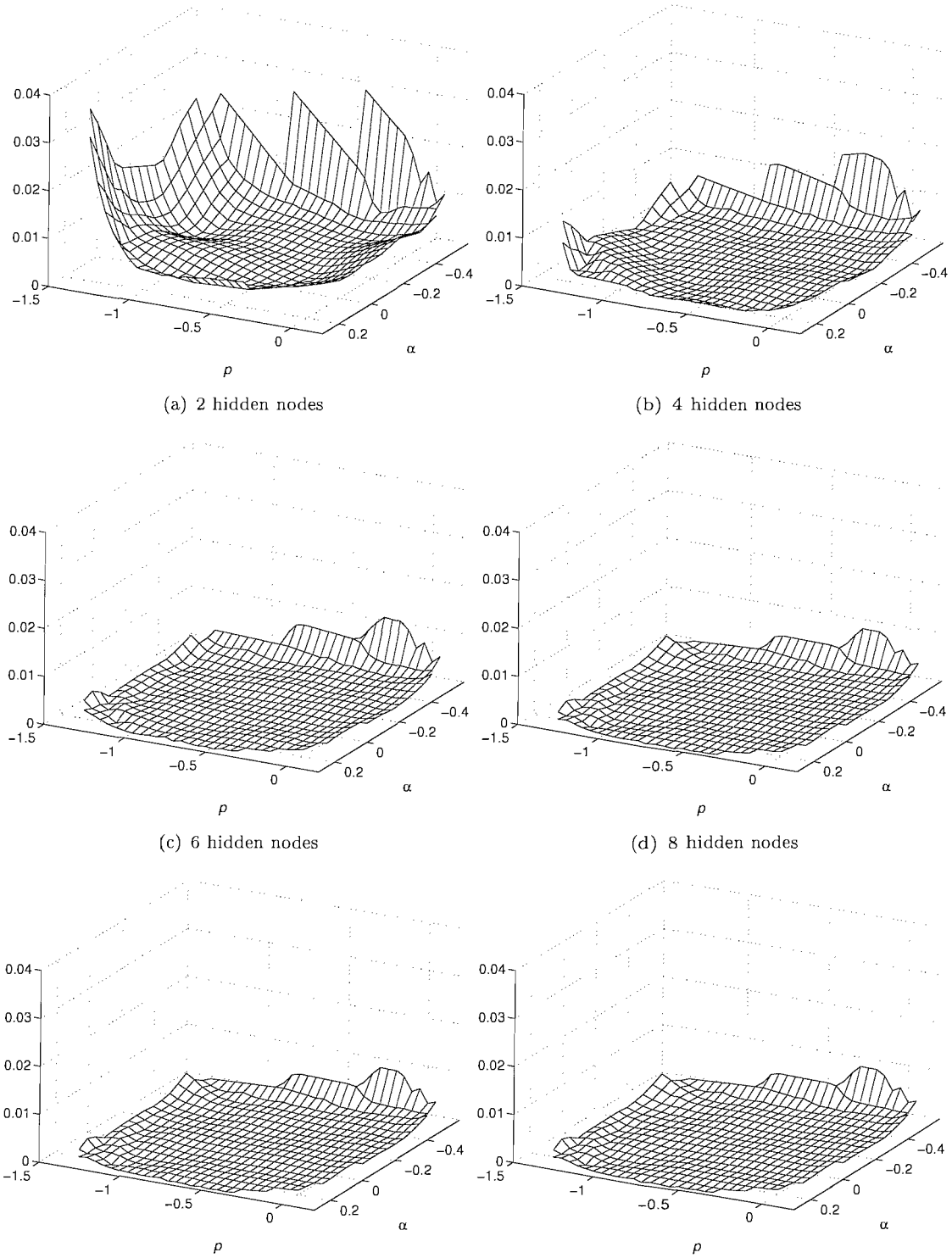


FIGURE 8.17: Trajectory generator, with polar encoding. Mean error expressed as a percentage of mean training output value for a range of MLPs with different hidden nodes counts, after 5000 training cycles.

different numbers of hidden nodes (2, 4, 6, 8, 10, 12 and 14). The same training process was used as for the comparison of equilibrium encoding schemes. Figure 8.17 shows the mean testing error after 5000 training cycles, plotted against the number of hidden nodes.

As expected, performance improves as the number of hidden nodes increases. However, the rate of performance increase drops off significantly after 8 hidden nodes. That said, all the MLPs tested produced very accurate results (less than 0.8% error) and it may be that only two hidden nodes are needed to create effective component controllers. This means that there would be only 12 parameters to train per component controller.

Figure 8.18 shows how the error varies across the workspace. The error is calculated as the mean length of the vector error between the ideal update and the update generated by the MLP, across all five component fields. Linear interpolation was used to reformat this data into a mesh for plotting. The two hidden node MLP, shown in subfigure (a), scores relatively well in the middle of the workspace, but its performance rapidly deteriorates towards the edges. The eight hidden node MLP, shown in subfigure (d), performs consistently throughout the workspace.



(a) 2 hidden nodes

(b) 4 hidden nodes

(c) 6 hidden nodes

(d) 8 hidden nodes

(e) 10 hidden nodes

(f) 12 hidden nodes

FIGURE 8.18: Error distribution throughout the workspace of the arm for six different MLP controllers that used shoulder centred polar coordinates. The α axis shows normalised angle, the p axis shows normalised radius and the vertical axis shows the length of the vector error between the ideal update and the one generated by the MLP. Linear interpolation was used to reformat these data into a mesh for plotting. The same axis scales are used for all the subfigures.

8.5 Model Free Update Normalisation

The distributed trajectory generator does not so far perform any normalisation of the generated update. For Cartesian encoding, this process is trivial and requires only access to the incremental update (i.e., the difference between the previous $\boldsymbol{e}$ and the next one to be generated). As this is exactly the output of the trajectory generator, this would be easy to implement.

There is no direct way of normalising an update expressed in polar coordinates. To achieve the appropriate rescaling, access to the previous $\boldsymbol{e}$ is required, and a method of converting between the polar coordinates and their Cartesian equivalent. The following procedure could then be used.

Firstly, calculate the equilibrium point, $\tilde{\boldsymbol{e}_n}$, that would be generated if the output of the the trajectory generator, $\boldsymbol{e}_\Delta$, was directly added to the previous equilibrium point, $\boldsymbol{e}_{n-1}$:

$$\tilde{\boldsymbol{e}_n} = \boldsymbol{e}_{n-1} + \boldsymbol{e}_\Delta \tag{8.8}$$

Then use a mapping function, p, to convert both $\boldsymbol{e}_{n-1}$ and $\tilde{\boldsymbol{e}_n}$ into Cartesian coordinates, here denoted using $\boldsymbol{c}$.

$$\boldsymbol{c}_{n-1} = p(\boldsymbol{e}_{n-1}) \tag{8.9}$$

$$\tilde{\boldsymbol{c}}_n = p(\tilde{\boldsymbol{e}_n}) \tag{8.10}$$

Then calculate the difference between these two Cartesian coordinates, normalised so that it has a length equal to the drift rate, r.

$$\boldsymbol{c}_\Delta = r\frac{\tilde{\boldsymbol{c}}_n - \boldsymbol{c}_{n-1}}{|\tilde{\boldsymbol{c}}_n - \boldsymbol{c}_{n-1}|_2} \tag{8.11}$$

We can then add the normalised update, $\boldsymbol{c}_\Delta$, to the previous equilibrium encoding, $\boldsymbol{c}_{n-1}$ to calculate the new, correctly scaled encoding, $\boldsymbol{c}_n$. This can then be converted back into shoulder centred polar coordinates using the inverse of p.

$$\boldsymbol{e}_n = p^{-1}(\boldsymbol{c}_{n-1} + \boldsymbol{c}_\Delta) \tag{8.12}$$

So although this process is complicated, it does not require access to a geometric model of the arm. Naturally, this would not be possible for joint angle encoding. It may be possible to use an MLP to learn the mapping from current encoding and update, to next encoding, that would perform the appropriate scaling, but this is left for future work.

8.6 Conclusion

This chapter has presented a method for generating idealised updates that are capable of determining the increment that will move an equilibrium point towards a target location, along the shortest path that avoids unreachable areas of the workspace. The system presented also handles collisions with the workspace perimeter in a graceful manner that sought to preserve step length.

The chapter then demonstrated how a similar effect could be achieved by mixing the output of five separate component generators. An MLP implementation of the component controllers was demonstrated to produce similar results. It was found that, although all three encoding schemes tested were capable of producing reasonable results, shoulder centred polar encoding was the most effective.

For the system considered, networks of between two and eight hidden nodes were demonstrated to produce low errors, with the behaviour of the 8 hidden node network being consistent throughout the workspace. The following chapter looks at how the equilibrium points created by the trajectory generator could be converted into appropriate actuator rest lengths.

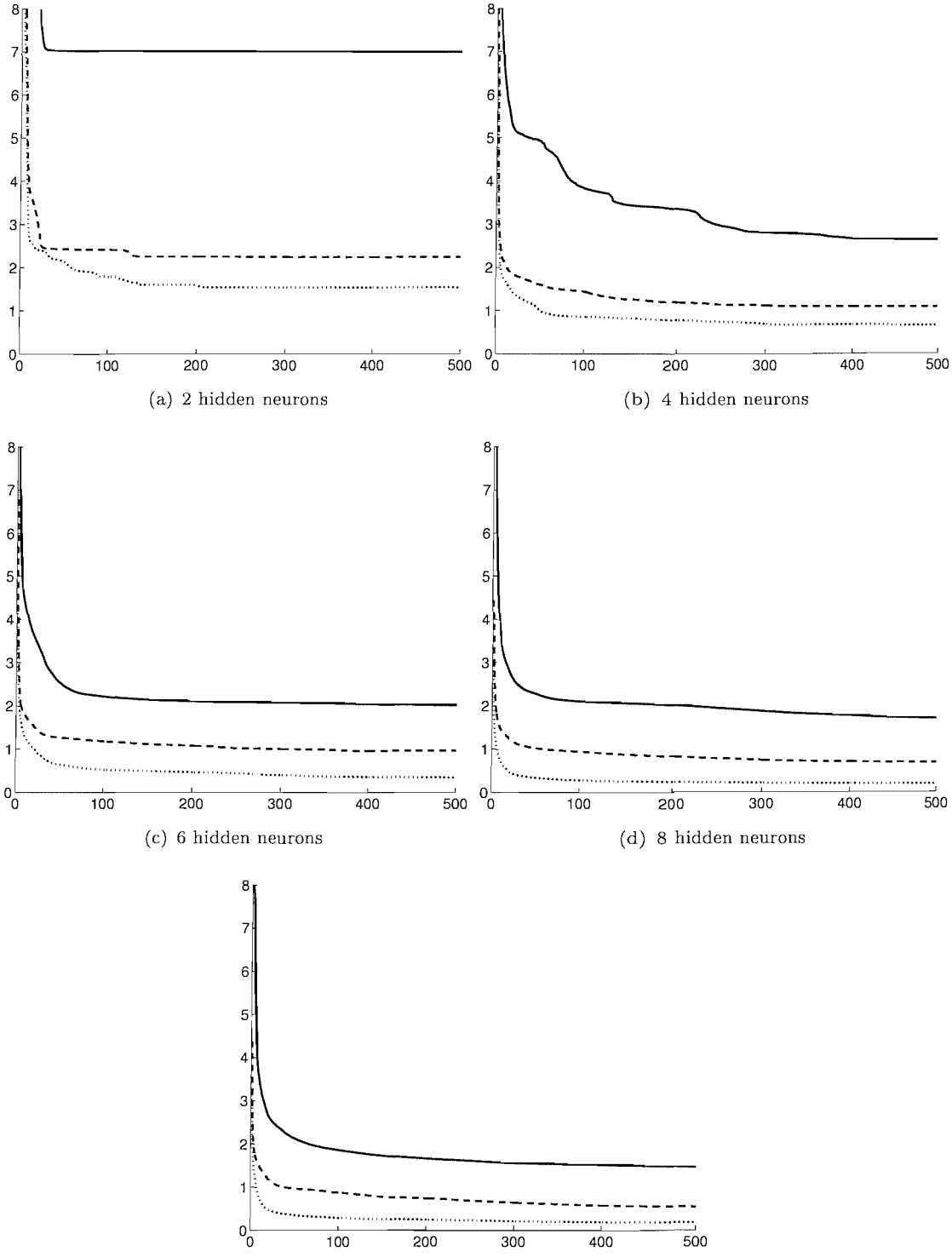
Chapter 9

Length Encoder

This chapter details an implementation of the length encoder, which aligns the mechanical equilibrium point of the arm model with the equilibrium points created by the trajectory generator by adjusting the actuator rest lengths appropriately. In this chapter we compare the three previously considered equilibrium encodings in terms of resultant error for various MLP capacities. A set of 1000 randomly selected poses was encoded using each scheme. An inverse kinematic model was used to calculate the lengths of the actuators that align the arm's equilibrium point with each of the selected poses. These lengths were taken to be the *ideal rest lengths* that the system should try to generate. This procedure assumes that the arm has no actuator co-activation.

9.1 Procedure

A number of MLPs were trained to learn the mappings between the equilibrium point encodings and the ideal rest-lengths. Each network had a sigmoidal hidden-layer activation function and a linear output-layer activation function. They were implemented using the NETLAB neural network toolbox for MATLAB. Five different network configurations were assessed, with varying numbers of hidden nodes (2, 4, 6, 8 and 10). The networks were trained for 500 cycles, with each cycle containing 10 standard backpropagation updates. A second set of 1000 randomly generated poses was used to test the performance of the MLP at the end of each cycle. The entire process was repeated ten times for each equilibrium point encoding method to reduce the effect of the random initialisation. The results of this process are shown in Figure 9.1.



(a) 2 hidden neurons

(b) 4 hidden neurons

(c) 6 hidden neurons

(d) 8 hidden neurons

(e) 10 hidden neurons

FIGURE 9.1: Comparison of multi-layer perceptron approximations of length encoder mapping function, b, using three different equilibrium point encodings: Cartesian (solid), polar (dashed) and joint (dotted). The y-axis shows the mean absolute actuator length error, expressed as a percentage of total working range, averaged over all 6 actuators, and 10 separate runs.

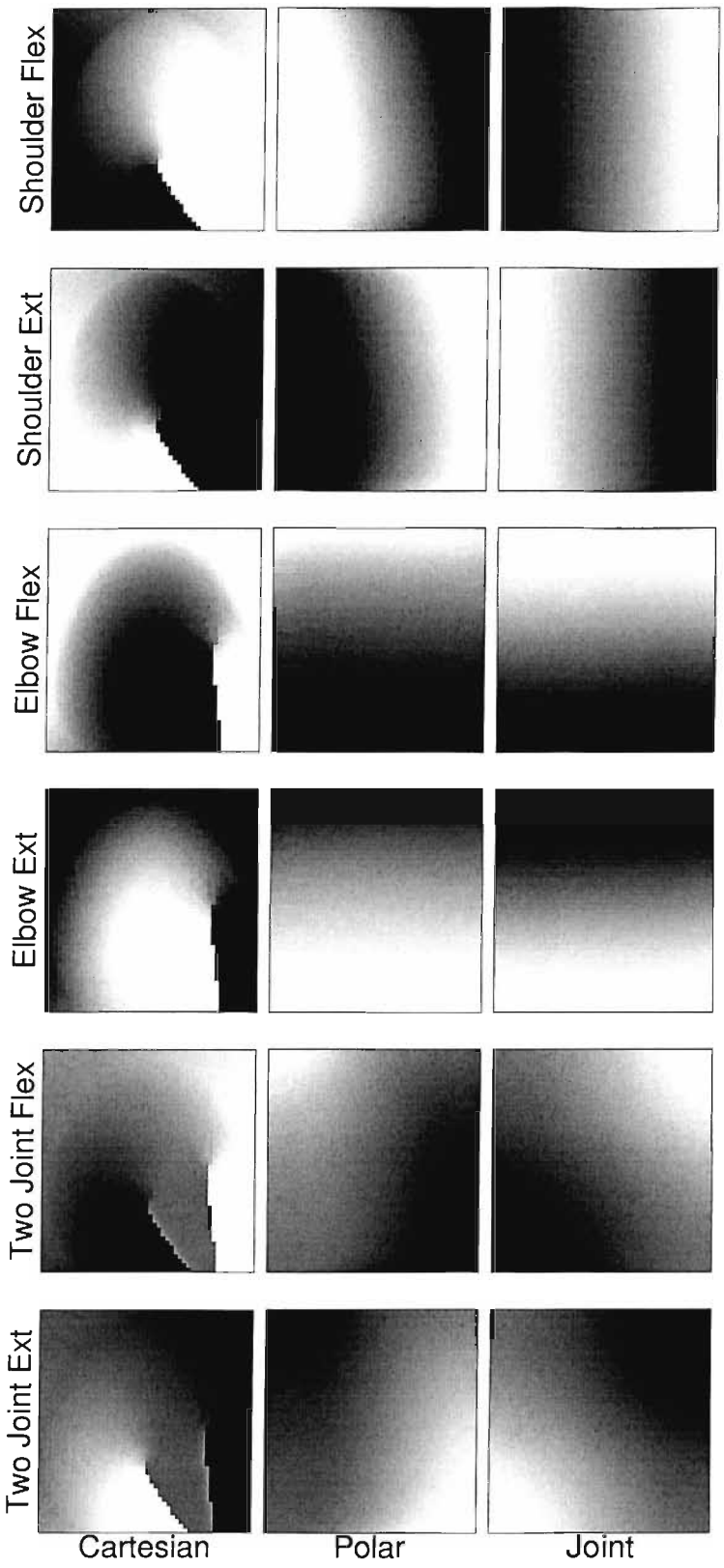


FIGURE 9.2: Normalised actuator rest length plotted as a function of normalised equilibrium point encoding. Each subfigure shows how the ideal rest length of a single actuator varies with respect to equilibrium point encoding. Each column represents a different encoding scheme and each row represents a different actuator. White represents maximum extension and black represents minimum extension. As nearest neighbour interpolation was used, actuator length values are propagated outside the arm's workspace.

9.2 Results

The Cartesian workspace encoding consistently performed worse than the other encoding schemes. This is not surprising if we consider the mappings that must be performed (Fig 9.2). Each subfigure shows how the rest-length of a single actuator varies with respect to the equilibrium point encoding. As all three encoding schemes are two dimensional, this relationship can be plotted as a surface, with grey level representing actuator rest-length and position representing the equilibrium point encoding. Eighteen subfigures are presented, one for each combination of actuator and equilibrium point encoding scheme. Actuator names are defined at the beginning of Chapter 5. The subfigures are rectangular and therefore include regions that are outside the workspace of the arm. As nearest neighbour interpolation was used to generate these figures, actuator rest-lengths are propagated into these unreachable regions.

This figure shows that the mapping from joint angles to actuator lengths is very simple and shows how the shoulder flexors and extensors can be controlled independently of joint two and the elbow flexors and extensors can be controlled independently of joint one. Even the response of the two joint flexors and extensors is contained entirely within a plane.

In comparison, the mapping between the Cartesian encoding and the actuator rest lengths is significantly more complicated, and so it is unsurprising that it is the hardest for the MLP to learn. The performance of the MLPs trained to use polar centred shoulder coordinates is reasonably close to that of the MLPs using joint coordinates. Comparison of the appropriate mappings in Figure 9.2 shows that there are strong parallels between the polar and joint mappings.

9.3 Conclusion

This chapter has demonstrated that all the encoding schemes are capable of learning an approximation to rest length encoding. That said, noticeable differences in their relative performance has been shown and justified in terms of the complexity of the mapping they must perform.

Shoulder centred polar coordinates was the most effective encoding scheme for trajectory generation, and is a close second for length encoding. It should therefore be considered a strong candidate for building future CETC implementations.

Chapter 10

Discussion and Conclusion

This thesis has explored the ideas and terminology surrounding manipulation first from a robotics perspective and then from a biological one. Two main pieces of work were undertaken. The first piece of work explores the potential of convergent force field control for simple compliant arm structures as a precursor to real engineering application. The experimentation undertaken involved using mutation hill climbing to train the dynamics of a physical compliant arm simulation coupled with a multi-layer-perceptron. The second piece of work combined two techniques inspired by neurological studies (convergent force field control and equilibrium trajectory control), to create a novel framework for compliant limb control (convergent equilibrium trajectory control). Two key components of this framework were then implemented using multi-layer-perceptrons. This chapter will review these two main pieces of work, and then close with a justification of the approach taken.

As it stands, the version of convergent force field control implemented here will only be suitable for very simple manipulating robots. It is possible with further work that it may become suitable for a broader range of applications. The following paragraphs discuss some of the limitations of the current approach and suggest some appropriate responses.

As only a limited number of target points were used, the current implementation does not conclusively demonstrate that all regions of the workspace are equally amenable to control by this approach. To rectify this a larger range of targets should be used, with a wider variety of initial conditions (specifically arm component velocities). If successful this would build confidence in the arm model, and provide a useful set of trained component field controllers for later work. The robustness of the control approach should also be assessed by introducing a variety of disturbing forces and measuring their effect on the arm's trajectory.

The learning technique used in this work was deliberately very simple. It was shown that for the arm model and the range of training parameters considered, mutation hill

climbing is not a reliable way of developing single component force field controllers, i.e., the fitness landscape is not smooth and contains significant local minima. For most of the network configurations tested, the hill climbing process produced results that were no more effective than random search. That said, the training process used did show that networks with around 20 hidden nodes were capable of controlling the simulated arm to bring it towards a predefined target pose within the workspace. They were not, however, shown to be capable of moving the arm directly towards the target, instead long looping paths were taken.

There are more advanced learning approaches, like simulated annealing and genetic-algorithm based methods, that are more tolerant to local minima, albeit at the expense of more fitness function evaluations. These training approaches should be compared in an attempt to ascertain whether the arm model and controller are capable of producing smooth point to point motions, and to develop a better understanding of the fitness landscape.

Blending of two component force field controllers was demonstrated to create a smooth range of intermediate responses, with the resultant rest location being an approximate interpolation of the two controllers' rest locations. Once a wider range of targets have been trained it should be possible to start analysing a variety of higher dimensional gain mixtures and potentially demonstrate controlled movement over the arm's entire workspace. This would then permit initial investigations into gain-space based control and behavioural composition.

The current work assumes access to fictional 'global axis' sensors, which report their location with respect to an imaginary fixed coordinate scheme. Although they are not used for controller feedback, they are extensively used for performance measurement, an essential part of any learning system. If the techniques developed in this thesis are to be implemented in hardware a viable alternative must be sought. A reasonable option would be a fixed camera, whose field of view included the whole of the arm's workspace. The location of the arm's end point, and possibly elbow, could then be detected (maybe by colour markings), and returned as a pair of iris-centred yaw and elevation values. Similar values could be obtained from the simulation environment described. Using simulated yaw and elevation angles would therefore be a sensible first step towards future camera integration.

An interesting project would involve evolving the arm parameters (relative masses, lengths, joint angles, etc.), in parallel with the controller. With this sort of approach there would be a danger that the arm parameters would become very task focused, in this case sacrificing behavioural flexibility for small gains in reaching performance. This could be combated by either increasing the thoroughness of the fitness function to include all the required dynamics or, if that proved too computationally expensive, introducing

restrictions on the permitted range of arm parameter values so as to maintain a basic level of behavioural flexibility.

The novel combination of convergent force field control and equilibrium trajectory control (convergent equilibrium trajectory control) presented here provides a tool that could be used to integrate a compliant manipulator with a neurally controlled autonomous robot, assuming that the correct actuator dynamics could be achieved artificially. The trajectory generator removes the requirement for the top level controller to have an internal representation of the arm workspace, or having to perform calculations to avoid its inaccessible regions.

The modelling and training work presented has shown that very small networks, with in the region of two to eight hidden nodes (which equates to between 12 and 42 network parameters), are capable of updating the equilibrium point smoothly and reliably. It was further shown that, for the equilibrium point encodings considered, it was possible to calculate appropriate equilibrium rest lengths using relatively small networks (in the order of six to ten hidden nodes).

When the error induced by the trajectory generation and the length encoding is taken together, for the model considered, it was shown the shoulder centred polar encoding was the most effective in terms of network capacity, and therefore computational cost. It would appear that, although at first glance shoulder centred polar encoding is just as arbitrary as Cartesian encoding, its performance and mappings are significantly closer to those of joint angle encoding. It is therefore a strong candidate for future implementation of convergent equilibrium trajectory control.

There are several ways in which the training methods used in this work, could and should be improved if pursued further. In the simulation work presented, the location of each of the component field targets was chosen arbitrarily. This is not ideal as it is likely that ideal component fields will depend on the arm's mechanical configurations and the agent's environments/tasks. Work needs to be done to develop a system that automatically aligns the component fields configuration with the arm mechanics and allows them to be updated in response to the arm's usage.

To fully evaluate the effectiveness of CETC the trajectory generator and length encoder modules developed in this thesis should be combined with an arm model capable of equilibrium trajectory control. Once the entire chain is in place, the work can move beyond offline module training and start using proprioceptive feedback to perform online learning. This will open the door to studying the controllers potential to adapt to incremental changes in the bodies dynamics.

Although only touched on lightly before, it is important to emphasise that in animals the forelimbs commonly play a dual role; that of environmental manipulation and locomotion. Legged robots are capable of negotiating a more varied range of environments

than their wheeled counterparts, and are therefore being actively developed for search and rescue applications. Being able to use the forelimbs in a dual role may increase the behavioural flexibility of the robot at limited extra cost in terms of weight and complexity.

If the techniques outlined in this thesis are to be suitable for integration into legged manipulating robots, the limb controllers must be capable of supporting both these activities. Locomotion presents specific challenges which should also be considered when designing a front limb controller. These include postural stability, rhythmic stability in the presence of disturbance and phase coupling between limbs. Training of these dual purpose trajectory generators presents specific challenges, foremost of which is the design of a suitable fitness function. Clearly, trying to train to an oscillating target directly would not work, as any phase delay would introduce large, and potentially misleading, errors. An alternative approach would be to define a vector field for the work space, much as was done for the ideal trajectory generator. The field would, of course, now contain a certain amount of curl and movement within it would therefore converge towards a limit cycle, rather than a point. An online learning technique could use the length of the vector difference between the current movement and that specified by the ideal field at that point. Care would have to taken to ensure that all appropriate regions of the limb's state space were explored and trained.

The long term value of the biologically inspired methods can only be demonstrated if they provide measurable advantages when compared with traditional approaches. These advantages are unlikely to be in the areas that are conventionally used to assess traditional approaches (e.g., accuracy, repeatability, and band-width) but instead focus on more nebulous qualities like unsupervised calibration, tolerance of mechanical wear and tear and robustness to changes in environmental dynamics. There is much work to be done in both developing robots that perform well on these non-traditional qualities and on the development of testing procedures to assess them. Until both are in place there will be much, justified, resistance to these novel approaches.

The length encoder developed in this thesis does not actively manage the level of co-activation of antagonistic muscles. It should be extended to include an input that would allow for varying levels of co-activation in the arm. Initial work should develop an analytical inverse dynamic model of the arm and use that to generate the ideal rest length for any given equilibrium point and level of co-activation, which could then be used to train an MLP. If this is successful an attempt should be made to replace the analytical inverse dynamics with an online learning method. This would hold the potential for the system to adapt to gradual changes in the arm's mechanics or the task dynamics.

In drawing together the work that has been presented in this thesis, two key questions must be addressed; does the area investigated warrant engineering attention, and is

the work presented a justified response? In answering these questions it is important to reconsider what we understand by evolution, the driving force behind all biological systems. Engineers often assume that it is equivalent to an incremental optimisation process. This is unfortunately a misleading assumption.

Biological evolution provides nothing more, nor less, than a continual drive to give a germ line a competitive reproductive advantage. With this in mind, every stage of the process must be justifiable in its own right, without reference to any abstract, long term goals. Powered flight is a good example of this. The morphological differences between flying creatures and their land bound counter parts are dramatic. We must therefore look at intermediate behaviours that could be improved with some, but not all of the flight adaptations. These may include passive flight, assisted jumping, or wing assisted incline running, a process where wings are used to generate increased down force and therefore increase the incline up which an animal can run (Dial, 2003). For powered flight to develop, the successful performance of these intermediate behaviours must result in competitive reproductive success. That said, development that improves the success of these behaviours may introduce changes that are detrimental to the later performance of powered flight.

We must, therefore, consider the process not as an optimisation toward a single objective but as a series of optimisations toward a range of intermediate objectives, each of which must be individually justified. What implications does this have for the morphology and behavioural responses of the resulting animals? The current state of both these characteristics will be determined by past as well as current evolutionary objectives. In point of fact, many parts of the system may well be highly suboptimal or even disadvantageous in the current context of the animal. Distinguishing currently useful from historically useful characteristics is perhaps the greatest challenge faced by engineers who look to nature for inspiration.

So what tools are available to help make this distinction? One strong clue is convergent evolution. There are some characteristics that have evolved in more than one strand of the evolutionary hierarchy. The strongest factor linking them is the fundamental physics of the environment. These physical realities must also be confronted and exploited by artificial systems, and it is through this link that we can justify seeking inspiration from natural systems. Only to the extent that it can be demonstrated that the natural and artificial systems are attempting to solve the same problem can any imitation be justified.

So are there reasonable parallels between biological reaching and artificial reaching? The answer is, predictably, yes and no. At higher levels there are some strong parallels. Foremost of these is the requirement to integrate visual and proprioceptive information into a unified, manageable, body space. Although this is effectively about internal representation, rather than real-world physics, the complexity of such systems is a direct

result of the body and the environments natural geometry. These are problems common to both artificial and biological articulated manipulators. Such encodings have been the main focus of this research and are therefore justifiable.

At implementation level, the parallels between artificial and biological systems seem somewhat weaker. There is a fundamental difference between the physical and neurological morphology; parallelism vs lumping. Natural systems typically rely on many, poor quality, independent actuators, processors and sensors. This parallelism allows the system performance to degrade gradually, allowing it to continue to work in the presence of wear and tear. In contrast engineering systems tend to rely on high quality 'lumped' actuators, processors and sensors. This reduces the system complexity and wiring requirements significantly, but unfortunately means that the system is prone to catastrophic, rather than gradual, failure.

So are techniques inspired by the extremely parallel biological approaches suited for implementation in a lumped engineering context? To some extent the worlds are not as dissimilar as the previous paragraph presented them. As mentioned in the discussion of muscles, it is possible that Renshaw cell inhibition may be capable of allowing one input signal to control the activation of an entire muscle. If this is the case, then it is an example of a natural system creating a lumped interface to a parallel system. It may be possible for an engineering system to work at this level even though the actuators are implemented using different mechanical substrates.

Equally, engineering systems are learning to cope with less than reliable hardware. A modern super computer will, for example, happily run with a reasonable percentage of its processing nodes broken or disabled. As engineering continues to explore the possibilities of micro- and nano-robotics it will have to adapt to and exploit imperfect hardware, and it is possible that imitating the parallelism prevalent in nature may be one way of achieving this. It seems likely that as biologists and engineers continue to share ideas and insight, there will be strong growth in the availability of actuators, sensors and processing units with more natural characteristics.

Of particular importance to this work is the availability of compliant actuators that are suitable for mobile robotic applications. Compliance, both in structural members and actuators, is extremely common in naturally occurring systems. Where advantageous, animals have developed stiff materials, bones and tendons being the exemplars. Yet, even for vertebrates, these stiff structures are connected with a wide range of compliant components, typically cartilage and muscle. This creates an overall system whose rich dynamics are the result of the complex interplay between compliant and rigid components. It would be unwise therefore to assume that the compliance found so universally in biological actuators is merely the result of poor raw materials. If it were, then we would eventually expect to see some classes of animals developing alternatives.

There are several potential benefits to compliant actuation, which include improved manipulation dynamics, simplified control requirements, improved energy efficiency and a reduction in the damage caused by environmental interaction. As robotic systems move away from structured environments, into more natural ones, it is likely that compliant construction and actuation will begin to confer similar advantages. There is currently significant engineering interest in the development of more compliant actuator technologies such as series-elastic actuators, electro-active polymers, deformable air-muscles and coiled shape-memory alloys. Currently none of these artificial actuators provide the required performance characteristics, but some may do better than others. An important piece of future work would be to assess the range of currently available actuators and catalogue their transfer functions, from an equilibrium trajectory control perspective. It is unlikely that this work will yield immediate results, but for those interested in bio-robotics this is probably one of the most important restricting factors. It is therefore essential that a good dialogue is fostered between the materials developers, robotic engineers and biologists.

As these technologies mature they will present novel control challenges, and it therefore seems reasonable to turn to nature for inspiration. Although this thesis does not present a complete simulation of the convergent equilibrium trajectory control, when considered alongside other modelling work (Gribble et al., 1998), it presents a framework that deserves further study. It holds the promise of a flexible and trainable system that could be used to equip autonomous robots with compliant reaching control; an essential precursor to dexterous manipulation in an uncertain environment. In this context, the work presented here will provide a useful foundation for exploring exciting new control problems and it is hoped that this will eventually result in improved manipulating robots.

Appendix A

Manipulation Simulation Client

The content presented in this appendix is largely drawn from Sunderland et al. (2004a,b). For this research, it was necessary to simulate the accurate, real-time dynamics of physical hardware (a manipulator), with a requirement for flexible and intelligent control actions. Also, to decrease development time, we wanted to minimise the amount of bespoke code by exploiting proprietary, commercial software. We therefore needed a framework that could couple together the different proprietary software components.

Figure A.1 shows a block diagram of the overall simulation environment. Two commercial packages were used — Vortex Simulation Libaries (2002) and MATLAB (1984) — together with a bespoke C++ object hierarchy that mirrors the manipulator structure, with Python (2003) used for scripting. Vortex is a very capable package for modelling physical dynamics and collisions between solid objects. It is supplied with a lightweight

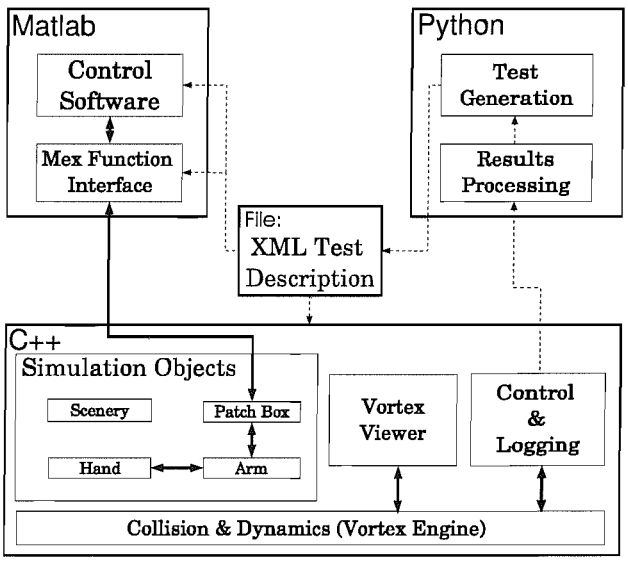


FIGURE A.1: The simulation environment combines several different software components.

OpenGL/DirectX viewer that was adequate for this work (Fig. A.2). Vortex has extensive documentation, and is easily integrated with other libraries to create powerful simulation programs. MATLAB is the industry standard package for rapid mathematical algorithm prototyping, especially for control applications.

A.1 Simulation Configuration Files

As discussed in Chapter 2, robotic control has largely been based on Denavit-Hartenberg parameters. These are highly compact and quite flexible. However, they are not a description of a 'real' robot in that they do not contain information about motor specification, physical link shape and dynamics, and sensor placement. They also have a very limited structure: basically a list of joint-link pairs, with four parameters apiece. This description has several properties that facilitate mathematical analysis, but since we are doing fully-featured physical simulation, these properties are not especially useful in this work.

A MATLAB robotic toolbox (Corke, 1996) is already available that will handle joint-link based simulations directly. However, it does not perform collision detection and is therefore unsuitable for simulation of manipulators in their environment. It was used when proving the mathematics behind some of the simulation transformations. A more versatile alternative is provided by Vortex itself, which provides a way to load and store simulation objects directly from XML files. However, Vortex's XML files do not allow any structuring or labelling information (e.g., joint names like 'wrist') to be stored and made available to other system components (like MATLAB) which use them.

There are a variety of other robotic simulation environments available. However, these are either tailored toward mobile robotics and path-finding, making them unsuitable for simulation of manipulation tasks (Gazebo, 2004; DynaMechs, 2004), or are in the early stages of development (OpenSim, 2004). Where real-time rigid body simulation is performed by these systems, the Open Dynamics Engine (ODE) (Engine, 2004) is used instead of the Vortex simulation libraries (Vortex Simulation Libaries, 2002).

At configuration time, a family of C++ objects parse an XML file using `libxml2` and store the results using standard template library container classes. Using `libxml2` reduces coding and debugging time and will benefit from future releases. At the same time, XML allows us to exploit standard tools to write, modify and verify the simulation description files, which can be used easily by all stages of the system. There may come a point where the simulation requires large amounts of binary data (height fields or vertex meshes), in which case external files should be referenced rather than included directly (in much the same way as an image in HTML).

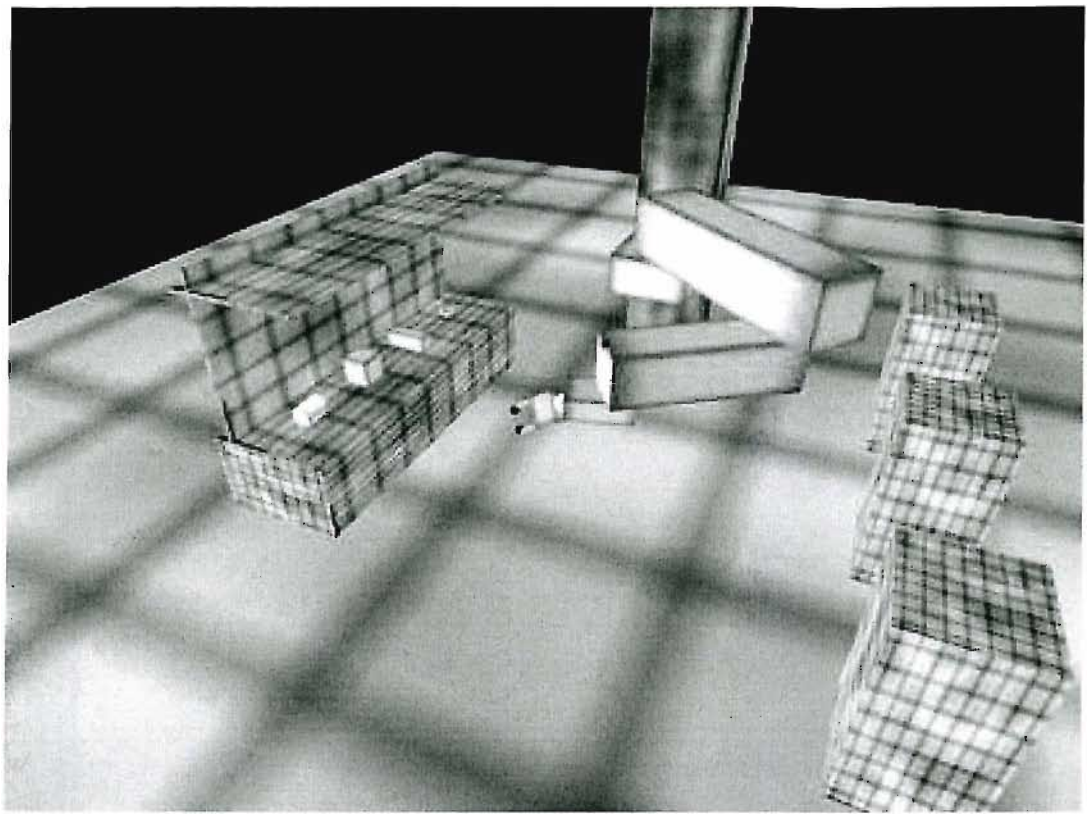


FIGURE A.2: Screen shots taken from MaSC simulation of a Selective Compliance Assembly Robot Arm (SCARA).

A.2 Example Configuration File

Configuration files used by the simulator are generated using a multi-step process. In stage one, a MATLAB script converts the contents of a struct, which defines the arm and poses parameters, into a simple XML file. The pose will be used to determine the initial location of the arm and the arm parameters will be used to build a Vortex model that aligns with the MATLAB model. This helps ensure that the two models are kept synchronised, and therefore reduces the chance of experimental error. A sample of this XML file is included here:

```
<params>
    <param name="basepos_x" value="0" />
    <param name="basepos_y" value="0.05" />
    <param name="len1" value="460" />
    <param name="len2" value="340" />
    <param name="m1" value="153.333333333333" />
    <param name="m2" value="306.666666666667" />
    <param name="j1_lo" value="-1.39626340159546" />
    <param name="j1_hi" value="1.39626340159546" />
    <param name="j2_lo" value="-2.75756004091423" />
    <param name="j2_hi" value="0.167491430954648" />
    <param name="sa_f" value="50" />
    <param name="sa_e" value="40" />
    <param name="elen_f" value="50" />
```

```
    <param name="elen_e" value="20" />
    <param name="eang_f" value="-0.275762021815104" />
    <param name="eang_e" value="2.86583063177469" />
    <param name="pose_ang_s" value="-0.121565211822889" />
    <param name="pose_ang_e" value="-2.70343593199608" />
    <param name="pose_elbow_x" value="-0.55782367281063" />
    <param name="pose_elbow_y" value="4.61605220623375" />
    <param name="pose_tip_x" value="-0.55782367281063" />
    <param name="pose_tip_y" value="4.61605220623375" />
</params>
```

Some of the included parameters are redundant and are not currently used by the following stages. They are, however, useful for testing purposes and have therefore been maintained. A second XML file is also used, which contains the extra parameters that are only required by the Vortex arm model:

```
<params>
    <param name="width_1" value="20" />
    <param name="width_2" value="20" />
    <param name="density_1" value="1" />
    <param name="density_2" value="1" />
    <param name="joint_damping" value="30000" />
    <param name="joint_stiffness" value="1000" />
    <param name="max_force" value="10000" />
    <param name="min_force" value="-1000" />
</params>
```

A python script (`genarm.py`) is then used to convert the two configuration files into a single XML document that contains the structure of the arm model, including sensors and actuators. It also generates a `patchbox` element which controls the order in which the sensors and demands are written to, and read from, the control socket.

```
<sim-world lin_unit="mm" rot_unit="rads">
    <patchbox>
        <in name="SH_FLEX"/>
        <in name="SH_EXT"/>
        <in name="EB_FLEX"/>
        <in name="EB_EXT"/>
        <in name="TJ_FLEX"/>
        <in name="TJ_EXT"/>
        <out name="ANG_1"/>
        <out name="ANG_2"/>
        <out name="ANG_1_VEL"/>
        <out name="ANG_2_VEL"/>
        <out name="SH_FLEX_LEN"/>
        <out name="SH_EXT_LEN"/>
        <out name="EB_FLEX_LEN"/>
        <out name="EB_EXT_LEN"/>
        <out name="TJ_FLEX_LEN"/>
        <out name="TJ_EXT_LEN"/>
        <out name="SH_FLEX_FRC"/>
        <out name="SH_EXT_FRC"/>
        <out name="EB_FLEX_FRC"/>
        <out name="EB_EXT_FRC"/>
        <out name="TJ_FLEX_FRC"/>
        <out name="TJ_EXT_FRC"/>
        <out name="EB_X"/>
        <out name="EB_Z"/>
        <out name="TIP_X"/>
```

```
    <out name="TIP_Z"/>
</patchbox>
<floor height="-50" size="1500"/>
<anchor fixed="true">
    <box depth="10.0" length="10.0" width="10.0"/>
    <active-spring-base linear_enc="SH_FLEX_LEN" max-force="10000" min-force="-1000"
        response_enc="SH_FLEX_FRC" sink_id="SH_FLEX">
        <attached-at>
            <position x="-50.000000" y="0.000000" z="0.000000"/>
        </attached-at>
    </active-spring-base>
    <active-spring-base linear_enc="SH_EXT_LEN" max-force="10000" min-force="-1000"
        response_enc="SH_EXT_FRC" sink_id="SH_EXT">
        <attached-at>
            <position x="40.000000" y="0.000000" z="0.000000"/>
        </attached-at>
    </active-spring-base>
    <active-spring-base linear_enc="TJ_FLEX_LEN" max-force="10000"
            min-force="-1000" response_enc="TJ_FLEX_FRC" sink_id="TJ_FLEX">
        <attached-at>
            <position x="-50.000000" y="0.000000" z="0.000000"/>
        </attached-at>
    </active-spring-base>
    <active-spring-base linear_enc="TJ_EXT_LEN" max-force="10000"
             min-force="-1000" response_enc="TJ_EXT_FRC" sink_id="TJ_EXT">
        <attached-at>
            <position x="40.000000" y="0.000000" z="0.000000"/>
        </attached-at>
    </active-spring-base>
    <joint-link axis="y" density="1.000" init_offset="-0.122" pos_enc="ANG_1" vel_enc="ANG_1_VEL">
        <upper-limit damping="30000" range="1.39626340159546" stiffness="1000"/>
        <lower-limit damping="30000" range="-1.39626340159546" stiffness="1000"/>
        <box depth="20.0" length="460.0" width="20.0"/>
        <attached-at>
            <position x="0.000000" y="0.000000" z="0.000000"/>
        </attached-at>
        <locally-attached-at>
            <position x="0.000000" y="0.000000" z="-230.000000"/>
        </locally-attached-at>
        <global-axis-sensor axis="x" src_id="EB_X">
            <attached-at>
                <position x="0.000000" y="0.000000" z="230.000000"/>
            </attached-at>
        </global-axis-sensor>
        <global-axis-sensor axis="z" src_id="EB_Z">
            <attached-at>
                <position x="0.000000" y="0.000000" z="230.000000"/>
            </attached-at>
        </global-axis-sensor>
        <active-spring-tip label="SH_FLEX">
            <attached-at>
                <position x="0.000000" y="0.000000" z="-76.666667"/>
            </attached-at>
        </active-spring-tip>
        <active-spring-tip label="SH_EXT">
            <attached-at>
                <position x="0.000000" y="0.000000" z="-76.666667"/>
            </attached-at>
        </active-spring-tip>
        <active-spring-base linear_enc="EB_FLEX_LEN" max-force="10000" min-force="-1000"
             response_enc="EB_FLEX_FRC" sink_id="EB_FLEX">
            <attached-at>
                <position x="0.000000" y="0.000000" z="76.666667"/>
            </attached-at>
        </active-spring-base>
        <active-spring-base linear_enc="EB_EXT_LEN" max-force="10000"
            min-force="-1000" response_enc="EB_EXT_FRC" sink_id="EB_EXT">
            <attached-at>
                <position x="0.000000" y="0.000000" z="76.666667"/>
            </attached-at>
        </active-spring-base>
```

```
            <joint-link axis="y" density="1.000" init_offset="-2.703" pos_enc="ANG_2" vel_enc="ANG_2_VEL">
                <upper-limit damping="30000" range="2.75756004091423" stiffness="1000"/>
                <lower-limit damping="30000" range="-0.167491430954648" stiffness="1000"/>
                <box depth="20.0" length="340.0" width="20.0"/>
                <attached-at>
                    <position x="0.000000" y="0.000000" z="230.000000"/>
                </attached-at>
                <locally-attached-at>
                    <position x="0.000000" y="0.000000" z="-170.000000"/>
                </locally-attached-at>
                <active-spring-tip label="TJ_FLEX">
                    <attached-at>
                        <position x="-13.614012" y="0.000000" z="-121.889100"/>
                    </attached-at>
                </active-spring-tip>
                <active-spring-tip label="TJ_EXT">
                    <attached-at>
                        <position x="5.445605" y="0.000000" z="-189.244360"/>
                    </attached-at>
                </active-spring-tip>
                <active-spring-tip label="EB_FLEX">
                    <attached-at>
                        <position x="-13.614012" y="0.000000" z="-121.889100"/>
                    </attached-at>
                </active-spring-tip>
                <active-spring-tip label="EB_EXT">
                    <attached-at>
                        <position x="5.445605" y="0.000000" z="-189.244360"/>
                    </attached-at>
                </active-spring-tip>
                <global-axis-sensor axis="x" src_id="TIP_X">
                    <attached-at>
                        <position x="0.000000" y="0.000000" z="170.000000"/>
                    </attached-at>
                </global-axis-sensor>
                <global-axis-sensor axis="z" src_id="TIP_Z">
                    <attached-at>
                        <position x="0.000000" y="0.000000" z="170.000000"/>
                    </attached-at>
                </global-axis-sensor>
            </joint-link>
        </joint-link>
    </anchor>
</sim-world>
```

It is clear that the final configuration file contains many parameters and complex structural information. Careful examination will reveal that the intrinsically tree-like structure of the arm model is reflected in the nesting of the XML configuration. This allows all the parameters that control the behaviour of a modelling component to be defined in a context that indicates their locations within the model. This removes the necessity for there to be a separate table of information defining how the various components interconnect.

A.3 Socket Interface

A Vortex-based client (MaSC) was developed that would execute the physical simulation while interacting with a MATLAB-driven controller, via a UNIX-socket. MATLAB

provides a direct C-interface, via late linked pre-compiled binary files (so called MEX-files). These files have access to the MATLAB workspace, and share its file descriptors. Note that the file descriptor numbers provided within MATLAB do not map directly to the operating system ones (here Linux), and so care must be taken when sharing descriptors between MEX-files and standard MATLAB M-files. Each MEX-file is loaded, run and then removed from memory, so any state information required must be read from the MATLAB workspace and then stored before termination.

The link provided by the socket contains a stop byte followed by a block of floating point values (either actuator or transducer signals, depending on direction). Although this limits the communication options available, it has the advantage of ensuring that the controller is only presented with information that it could reasonably gain from a real robot. The test configuration file provides the option to label each actuator and transducer. It also includes a PatchBox element, which contains a list of input and output labels. After loading a test configuration, the simulation environment scans through the PatchBox, looking for matches between the labels specified there and those in the rest of the file. It then presents and receives the information in the order given in the PatchBox, and forwards the information appropriately. This gives the test designer complete control over which inputs/outputs are transmitted over the socket and over the order of transmission.

A.4 Further Comments

Further notes and comments are provided directly in the code itself and are written such that the `doxygen` documentation package can automatically extract them to create a manual in either HTML or texinfo formats. This is research code and there has therefore been limited time invested in ensuring portability or providing install and maintenance scripts. That said, all submitted C++ code uses the same coding conventions and is written to be as clear and reusable as the task permitted. It is hoped that it will provide a useful research tool for future work in this field.

Appendix B

Software Catalogue

This appendix catalogues some of the software engineering effort that has gone into producing the results presented in this thesis. It is intended to give a flavour of the work, rather than a detailed breakdown, and is naturally therefore fairly terse. For more detail, the interested reader is advised to explore the source code directly.

B.1 Manipulation Simulation Client

Header	Source(s)	Notes
	main.cpp	Initialises drivers and registers parsers. Catches top level errors and reports them to cerr.
	makefile	Modified version of standard Vortex makefile.
ActiveSpring.h	ActiveSpring.cpp	Model spring actuator response function - currently bypassed to allow MATLAB to contain actuator model.
BodyTree.h	BodyTree.cpp	Defines three key base classes; Platform, Attachment and Parser. An object that is to be attached to a model must derive from Platform and/or Attachment. This allows for proper configuration and running. Any such object must also define a custom parser which registers with a static function of Platform. This parser is responsible for recognising appropriate XML definition and creating a new instance from that. In this way each object type (jointlink, sensor, actuator etc) can be defined in an entirely independent manner.
Camera.h	Camera.cpp	Defines a CameraTarget object which can be attached to a model. If such an object is included in a model the camera orientation will be continually updated to track it.
CollisionReader.h	CollisionReader.cpp	Provides access to simulation libraries list of current collisions (or near collisions). This is an essential utility class for any sensor which is designed to measure contact.
CommunicationsCenter.h	CommunicationsCenter.cpp	Convenience class that combines a PatchBox with a ScriptReader.

`Driver.h`	`Driver.cpp`	Reads command line arguments and configures and initialises the simulation libraries appropriately. Contains the main update loop as a static function.
`Error.h`	`Error.cpp`	Defines base Error class with basic functions for overloading.

`Geometry.h`	`Jacobian.cpp`
	
	`Frame.cpp`
		Utils for calculating and updating orientation using Jacobians.
		C++ wrapper for Homogeneous transforms. Makes difference between handling own memory (InternalFrame) and handling a matrix created by another class/function (ExternalFrame) explicit.

`ImagePath.h`		Defines location of required textures
`ImpulseGenerator.h`	`ImpulseGenerator.cpp`	An attachment that responds to script events by adding an impulse force to the object to which it is attached.
`JointLink.h`	`JointLink.cpp`	Creates a shape and attaches it to a parent via a joint which is optionally motorised. Contains definition for joint angle encoder and joint velocity encoder sensors.
`Limit.h`	`Limit.cpp`	Defines the mechanical properties of a limit for either an angular or linear joint. This includes limit stiffness and damping.
`LinearActuator.h`	`LinearActuator.cpp`	Contains two classes, derived from Attachment; BaseUnit and Tip. These model a linear actuator running from one to another. The BaseUnit provides all the wiring connections (for both sensing and actuation). Class must be further derived to define how the force generated will be related to the input demand signal (see ActiveSpring).
`LocalSocket.h`	`LocalSocket.cpp`	Creates a socket for communicating with the controller. Either uses a local nnix file socket or a TCP/IP socket depending on command line options.
`LogWriter.h`	`LogWriter.cpp`	A singleton class that controls log indentation and printing.
`Model.h`	`Model.cpp`	Base class for all physical objects. Contains derived classes, StaticModel and DynamicModel, which should be further derived. Contains information about Shape and initial location.
`Motor.h`	`Motor.cpp`	Contains the wiring and motor parameters for a joint, if it is motorised.
`Recordable.h`	`Recordable.cpp`	Contains a base class for those objects which wish to dump details to record.
`Recorder.h`	`Recorder.cpp`	Framework to capture entire state of simulation on a frame by frame basis - not currently in use.
`RoboSim.h`		Top level definition of RoboSim namespace — no real content.
`Script.h`	`Script.cpp`	Contains classes required to allow scripted events in simulation (Actor, Event, Parser, EventFrame, Reader). Provides extensible framework, as each event type can handle its own parsing, data storage and physical model independently.

Sensor.h	Sensor.cpp	Defines some of the sensors that can be attached to the model, specifically; CollisionSensor, ProximitySensor, SlipSensor, Accelerometer and GlobalAxisSensor. The last of these is not physically authentic, but has proved useful.
Shape.h	Box.cpp CompositeShapeBase.cpp Cylinder.cpp Sphere.cpp Shape.cpp ShapeBase.cpp	Defines graphics, collision, and inertial models for primitive shapes. A wrapper class that may represent any shape, contains a ShapeBase. A base class for all the types of shape.
SimWorld.h	SimWorld.cpp Anchor.cpp FreeBody.cpp Floor.cpp	 Fixed body that can collide, but never move. Free body that can collide and move. Fixed plane that can collide, but never move.
VortexBase.h	VortexBase.cpp	Base class of all simulation objects. Contains static pointers to important parts of underlying simulation libraries.
Wiring.h	Wiring.cpp	Contains the classes required to correctly connect actuators and transducers to the LocalSocket, giving complete control over ordering.
XmlInterface.h	XmlLogable.cpp XmlCursor.cpp	Base class for any object which wishes to write to the log. Wraps libxml calls in C++ and provides a single location for unit conversions.

B.2 MATLAB to MaSC Link

Source	Notes
connection.h	Defines struct to contain connection details and prototypes public functions.
connection.c	Allocates buffers and contains reading/writing functions.
controller_test_main.c	Non matlab test stub for connection functions.
linkutil.h	Prototypes function to convert configuration information between MATLAB arrays and connection struct. Also includes wrapper to call MATLAB error function.
linkutil.c	Implementation of above.
launchchild.c	Stand alone mex function that allows MATLAB to fork a child and set its command line parameters. Used to launch the MaSC from within MATLAB
linkclose.c	Uses config struct information to close the connection.
linkopen.c	Attempts to open connection, return config struct on success, else failure struct.
linkupdate.c	Reads transducers values and sends actuator demands.
Makefile	Builds mex files for MATLAB to use.

Appendix C

Spring Tower Force Calculations

This appendix describes the mathematical steps required to calculate the force that would be required to balance that exerted by the tip of a fully specified spring tower (Fig. C.1). Table C.1 defines the terms used in this analysis.

J	Number of joint frames
S	Number of spring pairs $(J-1)$
$D_{1...J}$	Separation between cross-bar hinges
$\theta_{1...S}$	Rotation between each cross-bar
$\alpha_{1...S}$	Rotation between each cross-bar its vertical-link
$\boldsymbol{c}_{1...J}$	Hinge points for each cross-bar
$\boldsymbol{m}_{1...J}$	End cross-bar parameters (2 by S)
$\boldsymbol{f}_{n_{1...J}}$	The net force on the base of each frame
$\boldsymbol{f}_{ext}$	Balancing force required at the tip
$\boldsymbol{f}_s$	The force developed by an actuator starting on this frame
$\boldsymbol{f}_e$	The force developed by an actuator starting on the previous frame
t	Torque developed around the base of a frame
$\boldsymbol{f}_{ext}$	Balancing force required at the tip
$\boldsymbol{f}_{b_{1...J}}$	Force along each vertical-link

TABLE C.1: Table of symbols

We define each cross-bar as a parametric line, whose parameter, t, passes through zero at the point where its vertical-link connects it to the next cross-bar. Note that $\boldsymbol{x}$, $\boldsymbol{m}$ and $\boldsymbol{c}$ are two dimensional column vectors:

$$\boldsymbol{x}_j = p\boldsymbol{m}_j + \boldsymbol{c}_j \tag{C.1}$$

Let us define $\boldsymbol{m}_1$, $\boldsymbol{c}_1$ and α_1 so that that first cross-bar is aligned with the x-axis, passing through the origin when $p = 0$, and the first vertical-link is aligned with the y-axis. To

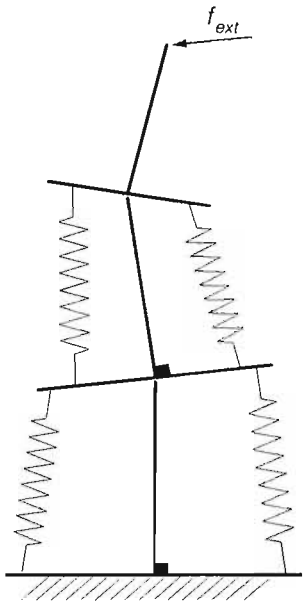


FIGURE C.1: Spring tower. The springs shown represent series-elastic actuators, (springs with a motor driven base offset, allowing them to exert a range of forces at any given length). The structure comprises 3 T-frames, but as the base frame is rigidly clamped, there are only 6 moving components (2 T-frames and 4 actuators).

define completely the structure of a J cross-bar system, we must define angles $\theta_{1\ldots J-1}$ angles and separations $D_{1\ldots J}$.

$$\boldsymbol{m}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \boldsymbol{c}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \alpha_1 = -\frac{\pi}{2} \tag{C.2a}$$

$$\boldsymbol{v}_j = \begin{bmatrix} \cos\alpha_j & \sin\alpha_j \\ -\sin\alpha_j & \cos\alpha_j \end{bmatrix} \boldsymbol{m}_j \tag{C.2b}$$

$$\boldsymbol{u}_j = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \boldsymbol{m}_j \tag{C.2c}$$

$$\boldsymbol{w}_j = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \boldsymbol{v}_j \tag{C.2d}$$

Appropriate values for $\boldsymbol{m}$ and $\boldsymbol{c}$ can then be found by the following iteration:

$$\boldsymbol{c}_{j+1} = \boldsymbol{c}_j + D_j \boldsymbol{v}_j \tag{C.3a}$$

$$\boldsymbol{m}_{j+1} = \begin{bmatrix} \cos\theta_j & \sin\theta_j \\ -\sin\theta_j & \cos\theta_j \end{bmatrix} \boldsymbol{m}_j \tag{C.3b}$$

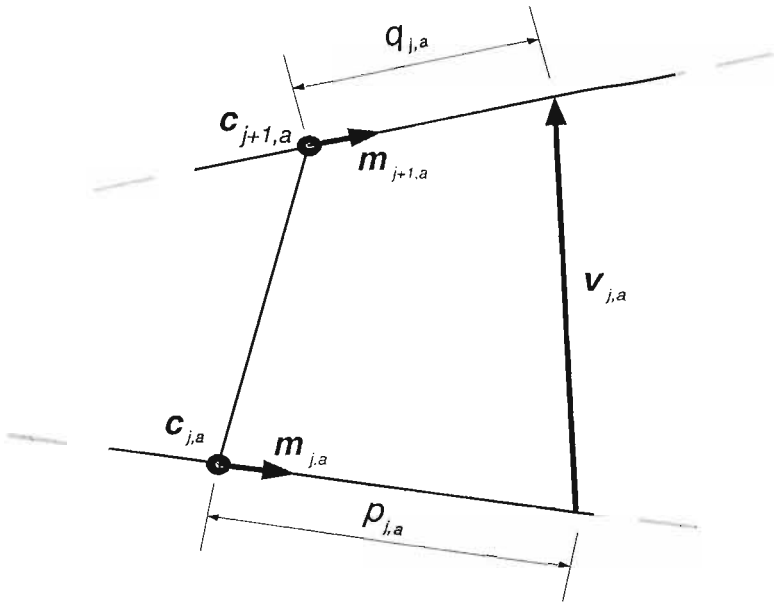


FIGURE C.2: The location of the ends of the acutator is defined using two parametric coordinates, p and q.

Since the geometry is fixed, it is possible to calculate the position of the actuators and, therefore, their current length. From this, we can calculate the forces and torques that they exert on their attachment points (in this case the cross beams). These forces ($\boldsymbol{f}_{s_{j,a}}$ and $\boldsymbol{f}_{e_{j,a}}$) and torques ($t_{s_{j,a}}$ and $t_{e_{j,a}}$) can be calculated for actuator a attached between cross-bars j and $j+1$ using following equations (Fig. C.2):

$$\boldsymbol{t}_{j,a} = (q_{j,a}\boldsymbol{m}_{j+1} + \boldsymbol{c}_{j+1}) - (p_{j,a}\boldsymbol{m}_j + \boldsymbol{c}_j) \tag{C.4a}$$

$$\boldsymbol{f}_{s_{j,a}} = \begin{cases} k_{j,a}(|\boldsymbol{t}_{j,a}| - n_{j,a})\frac{\boldsymbol{t}_{j,a}}{|\boldsymbol{t}_{j,a}|} & |\boldsymbol{t}_{j,a}| > n_{j,a} \\ 0 & |\boldsymbol{t}_{j,a}| \leq n_{j,a} \end{cases} \tag{C.4b}$$

$$\boldsymbol{f}_{e_{j,a}} = -\boldsymbol{f}_{s_{j,a}} \tag{C.4c}$$

$$t_{s_{j,a}} = p_{j,a}\boldsymbol{u}_j^T \boldsymbol{f}_{s_{j,a}} \tag{C.4d}$$

$$t_{e_{j,a}} = q_{j,a}\boldsymbol{u}_{j+1}^T \boldsymbol{f}_{e_{j,a}} \tag{C.4e}$$

Note that $n_{j,a}$, $k_{j,a}$, $p_{j,a}$ and $q_{j,a}$ are elements of arrays N, K, P and Q respectively. In this case we are considering long actuators that do not resist compression; hence, no force is generated when the displacement between the end points is less than the 'rest length' of the acutators (Eqn. C.4b).

The torque created at each cross-bar must be balanced by a moment at the end of its vertical-link. For the base of cross-bar j:

$$\begin{aligned}
\boldsymbol{f}_{spring_j} &= \boldsymbol{f}_{e_{j-1,1}} + \boldsymbol{f}_{e_{j-1,2}} + \boldsymbol{f}_{s_{j,1}} + \boldsymbol{f}_{s_{j,2}} && \text{(C.5a)}\\
\boldsymbol{f}_{twist_j} &= (t_{e_{j-2,1}} + t_{e_{j-2,2}} + t_{s_{j-1,1}} + t_{s_{j-1,2}})D_{j-1}\boldsymbol{m}_{j-1} && \text{(C.5b)}\\
\boldsymbol{f}_{strut_j} &= b_j\boldsymbol{v}_j && \text{(C.5c)}\\
\boldsymbol{f}_{ext_j} &= e_j\boldsymbol{w}_j && \text{(C.5d)}\\
\boldsymbol{f}_{net_j} &= \boldsymbol{f}_{spring_j} + \boldsymbol{f}_{twist_j} + \boldsymbol{f}_{strut_j} - \boldsymbol{f}_{strut_{j-1}} + \boldsymbol{f}_{ext_j} && \text{(C.5e)}\\
&= \boldsymbol{f}_{spring_j} + \boldsymbol{f}_{twist_j} + b_j\boldsymbol{v}_j - b_{j-1}\boldsymbol{v}_{j-1} + e_j\boldsymbol{w}_j && \text{(C.5f)}\\
&= 0
\end{aligned}$$

The ground will create a reaction force that will balance the component of $\boldsymbol{f}_{net_3}$ that is aligned with the second vertical-link, $\boldsymbol{v}_2$.

$$b_2 = \boldsymbol{v}_2^T(\boldsymbol{f}_{spring_3} + \boldsymbol{f}_{twist_3} + e_3\boldsymbol{w}_3 + b_3\boldsymbol{v}_3) \qquad \text{(C.6)}$$

If we substitute Equation. C.6 into Equation. C.5f yeilding:

$$\begin{aligned}
\boldsymbol{f}_{net_3} = {} & \boldsymbol{f}_{spring_3} + \boldsymbol{f}_{twist_3} + b_3\boldsymbol{v}_3 + e_3\boldsymbol{w}_3 - \\
& \boldsymbol{v}_2^T(\boldsymbol{f}_{spring_3} + \boldsymbol{f}_{twist_3} + b_3\boldsymbol{v}_3 + e_3\boldsymbol{w}_3)\boldsymbol{v}_2 \qquad \text{(C.7)}
\end{aligned}$$

This can be simplified by noting that $\boldsymbol{v}_j$ and $\boldsymbol{w}_j$ are perpendicular which implies:

$$\boldsymbol{w}_j^T\boldsymbol{f}\boldsymbol{w}_j = \boldsymbol{f} - \boldsymbol{v}_j^T\boldsymbol{f}\boldsymbol{v}_j \qquad \text{(C.8)}$$

This means that Equation. C.7 can be reduced as follows:

$$\boldsymbol{f}_{net_3} = \boldsymbol{w}_2^T(\boldsymbol{f}_{spring_3} + \boldsymbol{f}_{twist_3} + e_j\boldsymbol{w}_3 + b_3\boldsymbol{v}_3)\boldsymbol{w}_2 \qquad \text{(C.9)}$$

$$\begin{aligned}
\hat{\boldsymbol{f}}_{net_3} &= \boldsymbol{v}_3^T(\boldsymbol{w}_2^T(\boldsymbol{f}_{spring_3} + \boldsymbol{f}_{twist_3} + b_3\boldsymbol{v}_3 + e_j\boldsymbol{w}_3)\boldsymbol{w}_2)\boldsymbol{v}_3 && \text{(C.10)}\\
\hat{\boldsymbol{f}}_{net_3} &= \boldsymbol{w}_2^T(\boldsymbol{v}_3^T(\boldsymbol{f}_{spring_3} + \boldsymbol{f}_{twist_3})\boldsymbol{v}_3)\boldsymbol{w}_2 + b_3\boldsymbol{v}_3 + e_j\boldsymbol{w}_3)\boldsymbol{w}_2)\boldsymbol{v}_3 && \text{(C.11)}
\end{aligned}$$

We now have enough information to iterate the net force up the tower to the tip. The equations presented allow for the analysis of any static configuration, but do not include dynamic terms (inertia and damping) and would not therefore permit frame-by-frame simulation. The equations will not be extended to include these terms as they

would further complicate matters and such simulation is better performed by a generic Newtonian simulator.

References

W. Abend, E. Bizzi, and P. Morasso. Human arm trajectory formation. *Brain*, 105: 331–348, 1982.

K. Akasawa and K. Kato. Neural network model for control of muscle force based on the size principle of motor unit. *Proceedings of the IEEE*, 78(9):1531–1535, 1990.

G. E. Alexander and M. D. Crutcher. Parallel processing within motor areas of cerebral cortex and basal ganglia in the monkey. In *International Joint Conference on Neural Networks*, volume 2, pages 711–716, 1990.

AMERAH Website. Arm wrestling Match of EAP Robotic Arm against Human `http://ndeaa.jpl.nasa.gov/nasa-nde/lommas/eap/EAP-armwrestling.htm`, date accessed 13th Sept. 2005, 2005.

R. C. Arkin. *Behavior-Based Robotics.* MIT Press, Cambridge, MA, 1998.

R. Balasubramaniam and A. G. Feldman. Some robotic imitations of biological movements can be counterproductive. *Behavioural and Brain Sciences*, 24:1050–1051, 2001.

Y. Bar-Cohen. Electroactive polymers as artificial muscles: A review. *Journal of Spacecraft and Rockets*, 39(6):822–827, 2002.

M. F. Bear, B. W. Connors, and M. A. Paradiso. *Neuroscience: Exploring the Brain.* Lippincoot, Williams and Wilkins, Baltimore, MD, 2001.

G. A. Bekey, R. Tomovic, and I. Zeljkovic. Control architecture for the Belgrade/UCS Hand. In S. T. Venkataraman and T. Iberall, editors, *Dexterous Robot Hands*, pages 136–149. Springer-Verlag, 1990.

A. Bicchi. Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity. *IEEE Transactions on Robotics and Automation*, 16(6):652–662, 2000.

A. Bicchi and V. Kumar. Robotic grasping and contact: A review. In *IEEE International Conference on Robotics and Automation*, pages 348–353, San Francisco, CA, 2000.

E. Bizzi, N. Hogan, F. A. Mussa-Ivaldi, and S. Giszter. Does the nervous system use equilibrium-point control to guide single and multiple joint movements. *Behavioral and Brain Sciences*, 15:603–613, 1992.

E. Bizzi, F. Mussa-Ivaldi, and S. Giszter. Computations underlying the execution of movement: A biological perspective. *Science*, 253:287–291, 1991.

I. Boblan, R. Bannasch, H. Schwenk, F. Prietzel L. Miertsch, and A .Schulz. A human-like robot hand and arm with fluidic muscles: Biologically inspired construction and functionality. *Embodied Artificial Intelligence*, 3139:160–179, 2004.

I. Boblan, R. Bannasch, H. Schwenk, L. Miertsch, and A. Schulz. A human like robot hand and arm with fluidic muscles: Biologically inspired construction and functionality. In *Embodied Artificial Intelligence, Dagstuhl Event 03281*, pages 160–179. Springer, 2003.

R. A. Brooks. Intelligence without reason. In J. Myopoulos and R. Reiter, editors, *12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1991. Morgan Kaufmann: San Mateo, CA.

M. A. Conditt, F. Gandolfo, and F. A. Mussa-Ivaldi. The motor system does not learn the dynamics of the arm by rote memorization of past experience. *Journal of Neurophysiology.*, 78:554–560, 1997.

P. I. Corke. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24–32, 1996.

J. J. Craig. *Introduction to Robotics: Mechanics and Control.* Addison-Wesley, Reading MA, 1986.

A. d'Avella and E. Bizzi. Low dimensionality of supraspinally induced force fields. *Proceedings of the National Academy of Science of the United States of America*, 95 (13):7711–7714, 1998.

C. J. DeLuca, R. S. LeFever, M. P. McCue, and A. P. Xenakis. Control scheme governing concurrently active human motor units during voluntary contraction. *Journal of Physiology*, 329:129–142, 1982.

J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Applied Mechanics*, 77:215–221, 1955.

D. C. Dennett. Cognitive wheels: The frame problem of AI. In C. Hookway, editor, *Minds, Machines and Evolution*, pages 129–151. Cambridge University Press, 1984.

Der bionische Roboterarm of Technische Universität Darmstadt Website. `http://www.sim.informatik.tu-darmstadt.de/res/bmbf/biorob/` (Visited 29/11/05), 2005.

K. Dial. Wing-assisted incline running and the evolution of flight. *Science*, 299:402–404, 2003.

V. N. Dubey. *Sensing and Control within a Robotic End Effector.* PhD thesis, University of Southampton, UK, 1997.

DynaMechs. Multibody dynamic simulation library, `http://dynamechs.sourceforge.net/` (Visited 29/11/05), 2004.

T. Elliott and N. R. Shadbolt. Growth and repair: Instantiating a biologically-inspired model of neuronal development on the Khepera robot. *Robotics and Autonomous Systems*, 36:149–169, 2001.

Open Dynamics Engine. `http://www.ode.org/` (Visited 29/11/05), 2004.

A. G. Feldman. Functional turning of nervous system with control of movement or maintenance of a steady posture. controllable parameters of the muscle. *Biophysics*, 11:565–578, 1966.

A. G. Feldman. Once more on the equilibrium-point hypothesis (λ-model) for motor control. *Journal of Motor Behavior*, 18(1):17–54, 1986.

T. Flash. The control of hand equilibrium trajectories in multijoint arm movements. *Biological Cybernetics*, 57:257–274, 1987.

T. Flash and T. J. Sejnowski. Computational approaches to motor control. *Current Opinion in Neurobiology*, 11:655–662, 2001.

Gazebo. Client/server multi-robot simulator for outdoor environments, `http://playerstage.sourceforge.net/` (Visited 28/11/05), 2004.

A. P. Georgopoulos. Current issues in directional motor control. *Trends In Neuroscience*, 18(11), 1995.

A. P. Georgopoulos, J. Ashe, N. Smyrnis, and M. Taira. The motor cortex and the coding of force. *Science, New Series*, 256(5064):1692–1695, 1992.

A. P. Georgopoulos, R. E. Ketter, and A. B. Schwartz. Primate motor cortex and free arm movements to visual targets in three-dimensional space. II. Coding of direction of the movement by a neuronal population. *Journal of Neuroscience*, 8(8):2928–2937, August 1988.

S. F. Giszter, F. A. Mussa-Ivaldi, and E. Bizzi. Convergent force fields organized in the frog's spinal cord. *Journal of Neuroscience*, 13:467–491, 1993.

K. Goldberg, editor. *The Robot in the Garden: Telerobotics and Telepistemology in the Age of the Internet.* MIT Press, Cambridge, MA, 2000.

A. A. Goldenberg, B. Benhabib, and R. G. Fenton. A complete generalized solution to the inverse kinematics of robots. *International Journal of Robotics Research*, 1(1): 14–20, 1985.

H. Gomi and M. Kawato. Equilibrium-point control hypothesis examined by measured arm stiffness during multijoint movement. *Science, New Series*, 272(5258):117–120, 1996.

P. L. Gribble, D. J. Ostry, V. Sanguineti, and R. Laboissiere. Are complex control signals required for human arm movement? *Journal of Neurophysiology*, 79:1409–1424, 1998.

S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, Upper Saddle River, NJ, 1998.

J. A. Hoffer and S. Andreassen. Factors affecting the gain of the stretch reflex and soleus stiffness in premammillary cats. *Society of Neuroscience (abstracts)*, 4:935, 1978.

J. A. Hoffer and S. Andreassen. Regulation of soleus muscle stiffness in premammillary cats: intrinsic and reflex components. *Journal of Neuroscience*, 45:267–285, 1981.

N. Hogan. The mechanics of multi-joint posture and movement control. *Biological Cybernetics*, 53:315–331, 1985.

O. Holland. Grey Walter: The pioneer of real artificial life. In C. G. Langton and K. Shimohara, editors, *Artificial Life V*, pages 34–41. MIT Press, Cambridge, MA, 1996.

O. Holland. Exploration and high adventure: the legacy of Grey Walter. *Philosophical Transactions of the Royal Society: Mathematical, Physical and Engineering Sciences*, 361(1811):2085–2121, 2003.

J. M. Hollerbach. Computers, brains and the control of movement. *Trends in Neurosciences*, 5:189–192, 1982.

K. Hsiao, N. Mavridis, and D. Roy. Coupling perception and simulation: Steps towards conversational robots. In *International Conference on Intelligent Robots and Systems*, Las Vegas, NV, 2003.

Humanoider Muskelroboter ZAR of Technischen Universität Berlin Website. `http://www.zar-x.de/` (visited 28/11/05), 2005.

S. C. Jacobsen, J. E. Wood, D. F. Knutti, and K. B. Biggers. The Utah/MIT dexterous hand: Work in progress. *International Journal of Robotics Research*, 3(4):21–50, 1984.

J. R. Lackner and P. Dizio. Rapid adaptation to Coriolis-force perturbations of arm trajectory. *Journal of Neurophysiology*, 72(1):299–313, 1994.

M. L. Latash and A. G. Feldman. Computational ideas developed within the control theory have limited relevance to control processes in living systems. *Behavioural and Brain Sciences*, 27:409, 2004.

C. S. Lovchik and M. A. Diftler. The robonaut hand: A dexterous robot hand for space. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 907–912, Detroit, MI, May 1999.

A. V. Lukashin, B. R. Amirikian, and A. P. Georgopoulos. Neural computations underlying the exertion of force: a model. *Neuroreport*, 7:2597–2601, 1996a.

A. V. Lukashin, B. R. Amirikian, and A. P. Georgopoulos. A stimulated actuator driven by motor cortical signals. *Neuroreport*, 7:2597–2601, 1996b.

C. D. Mah and F. A. Mussa-Ivaldi. Evidence for a specific internal representation of motion-force relationships during during object manipulation. *Biological Cybernetics*, 88:60–72, 2002.

X. Markenscoff, L. Ni, and C. H. Papadimitriou. The geometry of grasping. *International Journal of Robotics Research*, 9(1):61–74, 1990.

X. Markenscoff and C. H. Papadimitriou. Optimum grip of a polygon. *International Journal of Robotics Research*, 8(2):17–29, 1989.

M. T. Mason and J. K. Salisbury. *Robot Hands and the Mechanics of Manipulation.* MIT Press, Cambridge, MA, 1985.

MATLAB. The Mathworks, `http://www.mathworks.com/` (Visited 28/11/05), 1984.

B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2:541–558, 1987.

B. Möhl. Two jointed robot arm with elastic drives and active oscillation damping. workshop. In *Bio-Mechatronic Systems, IEEE-RSJ International Conference on Intelligent Robots and Systems (IROS)*, Grenoble, 1997.

P. Morasso. Spatial control of arm movements. *Experimental Brain Research*, 42:223–227, 1981.

F. A. Mussa-Ivaldi and E. Bizzi. Motor learning through the combination of primitives. *Philosophical Transactions of the Royal Society of London, Series B–Biological Sciences*, 355(1404):1755–1769, 2000.

A. M. Okamura, N. Smaby, and M. R. Cutkosky. An overview of dexterous manipulators. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 171–180, San Francisco, CA, 2000.

OpenSim. 3D robot simulator, `http://opensimulator.sourceforge.net/` (Visited 29/11/05), 2004.

D. J. Ostry and A. G. Feldman. A critical evaluation of the force control hypothesis in motor control. *Experimental Brain Research*, 153:275–288, 2003.

K. Otsuka and X. Ren. Recent developments in the research of shape memory alloys. *Intermetallics*, 7(5):511–528, 1999.

Oxford English Dictionary. Online version, 2000. New edition: draft entry Sept. 2000, `http://dictionary.oed.com` (Visted 28/11/05).

R. P. Paul. *Robot Manipulators: Mathematics, Programming and Control.* MIT Press, Cambridge MA, 1981.

R. Pfeifer. Building fungus eaters: Design principles of autonomous agents. In *From Animals to Animats, Fourth International Conference on Simulation of Adaptive Behavior*, volume 4, pages 3–12. MIT Press/Bradford Books, Cambridge, MA, 1996.

R. Pfeifer and C. Scheier. *Understanding Intelligence.* MIT Press, Cambridge MA, 1999.

A. Polit and E. Bizzi. Processes controlling arm movements in monkeys. *Science, New Series*, 201(4362):1235–1237, 1978.

G. Pratt and M. Williamson. Series elastic actuators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-95)*, volume 1, pages 399–406, Pittsburg, PA, 1995.

Python. Documentation `http://www.python.org/doc/` (Visted 28/11/05), 2003.

F. Reuleaux. *Theoretische Kinematic. Translated as Kinematics of Machinery.* NY: Dover, 1875.

Robonaut Website. `http://robonaut.jsc.nasa.gov/robonaut.html` (Visted 28/11/05), 2005.

J. K. Salisbury. *Kinematic and force analysis of articulated hands.* PhD thesis, Department of Mechanical Engineering, Stanford University, Palo Alto, CA, 1982.

R. Shadmehr and M. A. Arbib. A mathematical analysis of the force-stiffness characteristics of muscles in control of a single joint system. *Biological Cybernetics*, 66:463–477, 1992.

R. Shadmehr and F. A. Mussa-Ivaldi. Geometric structure of the adaptive controller of the human arm, *MIT AI Memo 1437*, MIT, Cambridge, MA, 1994.

K. B. Shimoga. Robotic grasp synthesis algorithms: A survey. *International Journal of Robotics Research*, 15(3):230–266, 1996.

P. Somoff. Über Schraubengeschwindigkeiten eines festen Körpers bei Vershiedener Zahl von Stützflächen. *Zeitschrift für Mathematick und Physik*, 42:133–153, 1897.

J. A. Stevens and M. E. P. Stoykov. Using motor imagery in the rehabilitation of hemiparesis. *Archives of Physical Medicine and Rehabilitation*, 84(7):1090–1092, July 2003.

R. Sunderland, R. Damper, and R. Crowder. An approach to the simulation of robotic systems using XML-based configuration files. In *Proceedings of DETC'04, Design Engineering Technical Conferences*, pages no pagination–CD–ROM, Salt Lake City, UT., 2004a.

R. Sunderland, R. Damper, and R. Crowder. A framework for biologically-inspired control of reaching motions. In *Proceedings of 3rd International Symposium on Adaptive Motion in Animals and Machines (AMAM 2005)*, pages no pagination–CD–ROM, Ilmenau, Germany, 2005.

R. M. Sunderland, R. I. Damper, and R. M. Crowder. Flexible XML-based configuration of physical simulations. *Software Practice and Experience.*, 34(12):1149–1155, 2004b.

R. S. Sutton and A. G. Barto. *Reinforcement Learning.* Bradford Books, 1998.

T. Tsuji, K. Goto, M. Moritani, M. Kaneko, and P. Morasso. Spatial characteristics of human hand impedance in multi-joint arm movements. In *IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, volume 1, pages 423–430, 1994.

P. Tuffield and H. Elias. The shadow robot mimics human actions. *Industrial Robot*, 30 (1):56–60, 2003.

T. Uchiyama and K. Akazawa. Neural network model for muscle force control based on the size principle and inhibition by Renshaw cells. *Proceedings of the 20th Annual International Conference on the IEEE Engineering in Medicine and Biology Society*, 20(3):1430–1433, 1998.

A. L. H. Van der Meer, F. R. Van der Weel, and D. N. Lee. The functional significance of arm movements in neonates. *Science*, 267(5198):693–695, 1995.

C. P. van Schaik, R. D. Deaner, and M. Y. Merrill. The conditions for tool use in primates: implications for the evolution of material culture. *Journal of Human Evolution*, 36:719–741, 1999.

Vortex Simulation Libaries. Version 2.0.1 `http://www.cm-labs.com` (Visited 28/11/05), 2002.

W. G. Walter. An imitation of life. *Scientific American*, 184(8):42–45, 1950.

W. G. Walter. A machine that learns. *Scientific American*, 182(5):60–63, 1951.

W. G. Walter. *The Living Brain.* W. W. Norton, NY, 1963.

M. W. Wichman. The use of optical feedback in computer control of an arm, *Stanford Artifical Intelligence Memo 56*, Stanford University, Palo Alto, CA, 1967.