Relearning robot control

Discovering muscle pairs of a humanlike robotic arm

Michiel Holtkamp
December 2009

Master Thesis
Artificial Intelligence
Department of Artificial Intelligence,
University of Groningen, The Netherlands

Internal supervisor:
Prof. dr. Lambert Schomaker (Artificial Intelligence, University of Groningen)

External supervisors:
Prof. Hiroshi Ishiguro (Intelligent Robotics Lab., Osaka University)
Prof. Yoshio Matsumoto (Biomimetics Robotics Lab., Osaka University)
Assistant Prof. Yutaka Nakamura (Intelligent Robotics Lab., Osaka University)

university of groningen

“Ars longa, vita brevis, occasio praeceps, experimentum periculosum, iudicium difficile.”

Hippocrates

Abstract

Research in humanlike robotic arms will not only provide insight in how to make more realistic looking and behaving robots, but can also provide insight into the way that humans control their muscles. This information can then be applied in rehabilitation projects or in the development of prosthetics. Most current robots have joint-based actuators, instead of more humanlike systems with artificial muscles. Controlling humanlike systems requires a different approach than those used for conventional robots because the muscles of humanlike muscle-based systems have overlapping effects, while actuators conventional joint-based systems only affect one joint angle. This redundancy in control is of interest for adaptability to new environments and to changes in the body like wear or other damage. This thesis presents a simple method to control a humanlike robotic arm that does not require a lot of information about the arm itself or about its environment. The goal of this thesis is to find pairs of muscles that can have an opposite effect for a specific task. These pairs are useful for higher level control, such as compliance. Results show that the wrist of the arm can be positioned using only the learned muscle pairs, even though the pairing is not always optimal. Furthermore, the inaccuracies due to approximations can be mitigated by using the Attractor Selection Model to handle noise in the robot arm and the environment.

Contents

Chapter 1

Introduction

The human arm contains many muscles (see Figure 1.1). Every day we use them to pick up or manipulate objects, with little to no mental effort. This is a fortunate thing; who would like to spend his or her day thinking about how to move each individual muscle?

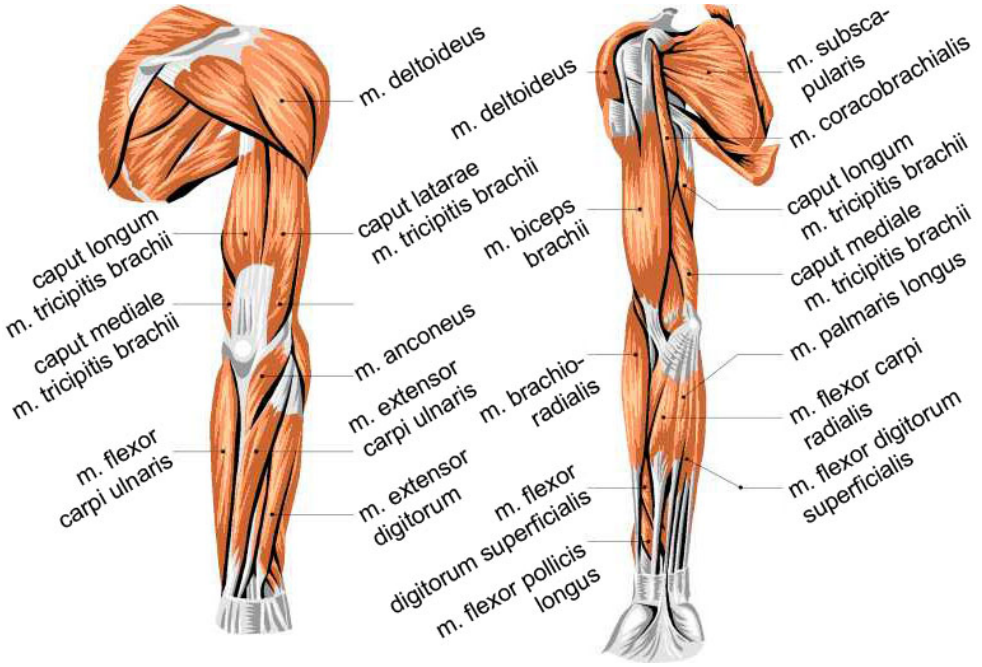


Figure 1.1: *Overview of muscles in the human arm (back/front view)*

In movies, robots typically have joint-based servomotors, emphasized by a distinct whirring sound. More advanced robots in movies look and act more human and have a humanlike muscle system (and no whirring sound). Sometimes, the latter kind is called android (literally: manlike). In the real world, the first kind of robot is most common, because the joint-based actuators are easier to control and they are very precise, especially

in controlled environments. The second kind of robot, with humanlike muscle systems, are not that common yet, because for most applications there are no advantages over joint-based actuators and the control of humanlike muscle systems is not as developed yet.

Research in humanlike robotic arms will not only provide insight in how to make more realistic looking and behaving robots, but can also provide insight into the way that humans control their muscles. This information can then be applied in rehabilitation projects or in the development of prosthetics.

A part of the research of human muscle control focuses on the so-called antagonistic muscle pairs; pairs of muscles (or groups of muscles) that have an opposite effect. This can be the rotation of a set of bones (for example the wrist) or changing the angle between two bones: extension/flexion or abduction/adduction. These muscles exist in pairs because single muscles can only exert a pulling force on bones by contracting, but can not exert a pushing force by relaxing. Figure 1.2 shows an example: the role of biceps and triceps during flexion (a) and extension (b).

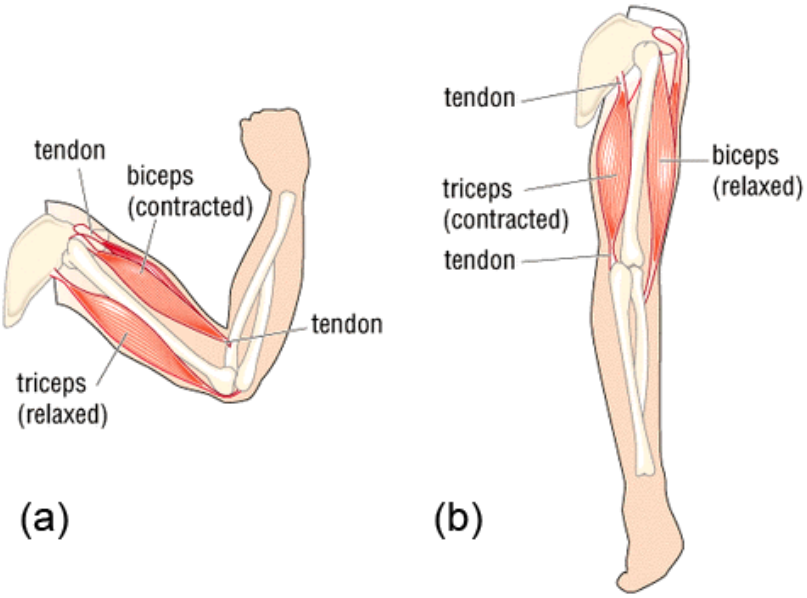


Figure 1.2: *Contracting and relaxing of biceps and triceps during flexion (a) and extension (b)*

In humanlike robots with muscle-like systems, these antagonistic muscle pairs exist, but in joint-based robots, they do not: the joint-based servomotor can both increase and decrease the angle between two bones. In joint-based systems, flexor/extensor control is done by one actuator, while in muscle-like systems this same control is done by two (groups of) actuators. How can a muscle-like system learn to use muscle pairs? The purpose of this work is to answer this question. A more specified question is presented at the end of next chapter, where more detailed background information is introduced.

Chapter 2

Background

Conventional robots with joint-based actuators are controlled by changing joint angles. To change the position of an end effector, for example a hand, the joint angles can be calculated by using Forward Kinematics. This is a method that is very predictable and very precise, especially in controlled environments and when the exact structure of the robot is known. This technique has been applied successfully in current robots and has been proven useful to learn about robot control. ASIMO (Hirose & Takenaka, 2001) is a famous example that can walk flat surfaces, slopes and take (known) stairs.[1] Other joint-based robots are capable of dancing perfomances (Riley et al., 2000; Nakazawa et al., 2002). Industrial robots with joint-based actuators are used at assembly lines for high-speed and high-precision assembly.

Muscle-based systems are more complex than joint-based systems (Hannaford & Winters, 1990). One reason is that there is no one-on-one relationship between muscles and joints; a single muscle is polyarticulate (affects multiple joints), but an effect can also be realised by different muscles. An example of a polyarticulate muscle in humans is m. biceps brachii. This well-known muscle is used to flex the elbow, but it is also used to rotate (supinate) the forearm (Gray, 1918). An example of an effect that is caused by multiple muscles is the flexion of the wrist; this is performed by m. flexor carpi radialis, m. flexor carpi ulnaris and m. palmaris longus (Gray, 1918). Another reason that muscle-based systems are more complex is that muscles have a non-linear effect on joint angles while servomotors have a linear effect.

Most muscle-based systems are operated by air pressure. The muscles are often called Pneumatic Artificial Muscles (Klute & Hannaford, 2000; Daerden & Lefeber, 2002; van der Smagt et al., 2009). Using air pressure has some disadvantages. First, a compressor is needed to pressurize the air, which is usually heavy and makes a lot of noise. Second, because air is easily compressed, the delay of control will be long if the air valves are placed far from the muscles. If air actuated muscle-based systems are so complex, why are they used so often?

[1] http://asimo.honda.com/downloads/pdf/asimo-technical-faq.pdf

2.1 Pneumatic muscle-based robotic systems advantages

Despite some problems, there are important advantages of air actuated muscle-based robotic systems over conventional joint-based robotic systems.

The first advantage is compliance. Servomotors are designed to be rigid, to stay in the desired state (e.g. joint angle) no matter what happens. Whenever the motor strays from its target by an external force, power is applied to correct this. This is desirable in controlled environments where it is guaranteed that no obstacles are in the way. However, when interacting with humans, this is not preferable. If a human is in the way, we do not want the human to get hurt or the robot to be damaged. In this situation it is better to bend than to break, which improves safety. An air operated muscle-like system is inherently compliant because air is easy to compress: applying a sudden external force to a bone structure that is controlled by muscles will move the bone structure instantly; there is no need to take action to absorb the force. Muscle-like systems are not the only systems that have this property. Some joint-based actuators are also operated by air pressure and are also used for their compliant behaviour (Matsui et al., 2005; Daerden & Lefeber, 2002).

The second advantage is adaptability. The human body changes during our lifetime: we are born small, we grow larger and when we grow older we shrink a little (Cowdry, 1979); accidents happen (large or small) that damage our body temporarily or permanently; we suffer wear and tear in joints when we grow older; when we do heavy exercise, muscle acidity occurs which affects the working of the muscle (Bangsbo et al., 1996). The environment also changes: our body responds differently in water and if there is a strong wind, we lean into the wind to prevent from falling over. To be able to control our body through all of this, we need adaptive control. In order to let robots operate in similar conditions, they need to be adaptive as well. Although current robots do not grow, they do have the problems of a changing environment and of wear and tear of the muscles or perhaps damage to a part of the body. Since a muscle-like system is redundant, other muscles can partially take over the workload of a defective muscle if its performance degrades because of wear and tear. Therefore, these types of systems degrade gracefully.

The third advantage is efficiency. A muscle-like system is more compact and has a higher power-to-weight ratio (Klute & Hannaford, 2000; Daerden & Lefeber, 2002; van der Smagt et al., 2009). This results in smaller and lighter systems.

These advantages show that in some situations muscle-based systems might be preferred over conventional joint-based systems. There are many differences between these systems, and their method of control is also very different. The question is: how can muscle-based systems be controlled? In the field of Artificial Intelligence, systems are usually biologically inspired, so to start answering this question, it is useful to take a look at how humans control their muscles.

2.2 Human muscle control

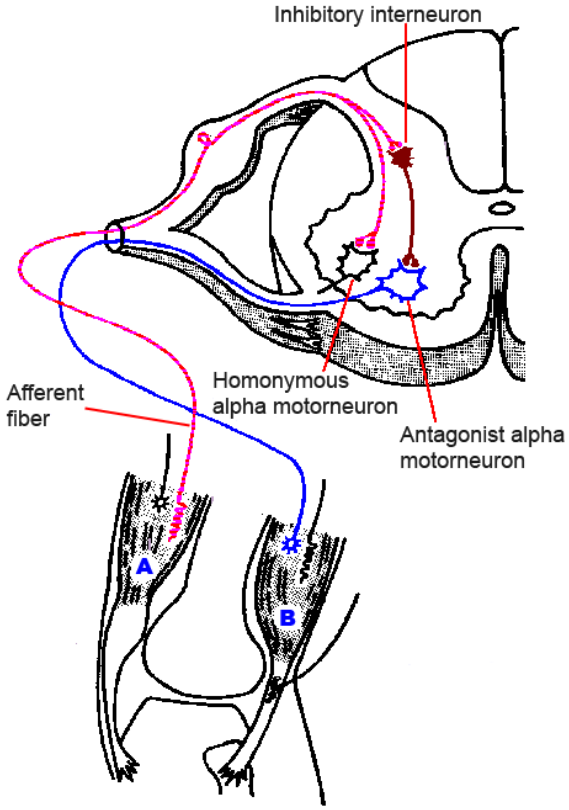


Figure 2.1: *Agonist (A), antagonist (B) muscle pair during tendon reflex shown with reflex arc in the spinal cord. When contracting the agonist muscle, the antagonist muscle relaxes because its motorneuron is inhibited by the afferent fiber of the agonist muscle. (Schadé JP, Ford DH:* Basic Neurology*. Amsterdam, Elsevier, 1965)*

Evidence is found in humans and in monkeys that suggests that spatial coordinates are used to plan movement (Morasso, 1981; Georgopoulos et al., 1982), but this is probably task-dependant (Morasso & Sanguineti, 1995).

Human control of muscles is divided into different levels. This is shown by Kots (1969), who provides evidence that the interaction between agonist and antagonist muscles in the ankle takes place in the spinal cord, while only the agonist muscle is influenced by supraspinal centers. In other words, there is horizontal (intraspinal) communication as well as vertical communication (between the spine and supraspinal centers). Further evidence is found by De Luca & Mambrito (1987), who shows that antagonistic pairs can be controlled "[...] as if they were one pool when the muscles are perfoming the same task.". An example of intraspinal communication is given by (Calancie et al., 1994), who show that more

complex movements like locomotion are generated in the spinal cord.

Another example of intraspinal communication is the well-known tendon jerk or patellar reflex. Tapping the tendon just below the patella (knee cap) will illicit a kick-movement in the leg. This simple reflex uses the extensor muscle (agonist) to kick and it relaxes the flexor muscle (antagonist) so the kick will not be inhibited. Figure 2.1 shows the afferent fiber of the agonist muscle (A) will indirectly inhibit the motorneuron of the antagonist muscle (B) via an inhibitory interneuron located in the spinal cord. This results in an automatic relaxation of the antagonist muscle when the agonist muscles contracts. Lloyd (1941) showed that this reflex can be inhibited by external signals, further evidence that different levels of control exist in humans.

Antagonistic muscles are controlled from higher levels in other ways as well, for example to control compliance, the inverse of stiffness. For different tasks, there is a different focus of control. Schomaker (1991, p. 171) describes focus of control for different tasks: for interception (e.g. catching a ball) the focus is on time of contact and position, while for object handling (e.g. picking up a raw egg) the focus is on kinematics and compliance; when handling a raw egg, one should apply enough pressure on the shell to prevent it from dropping, but not too much so that it will be crushed.

Humphrey & Reed (1983) discuss reciprocal activation and coactivation of antagonist muscles and show that separate cortical systems exists in monkeys to control this. Franklin et al. (2007) shows that compliance of the arm can be controlled for specific directions by the human Central Nervous System (CNS). Franklin et al. (2008) proposed a method which explains how the human CNS can learn stability and accuracy of movements based on antagonistic muscle pairs.

So humans control their muscles on different levels and antagonistic pairs play a large role in this. How is this used in current artificial muscle-based systems?

2.3 Previous muscle-based robotic systems

Liu & Todorov (2009) show a virtual model with seven Degrees Of Freedom (DOF) and 14 muscles, which form seven antagonistic muscle pairs. This model has different levels of control; a lower level that uses joint angles and velocities and a higher level that uses muscle activation. The arm is capable of performing a reaching task, with constraints on orientation of the palm of the hand. The higher level represents the motor cortex, the lower level represents the spinal cord. The higher dimensionality of the lower level is expressed into a lower dimensionality of the higher level by using a Jacobian matrix.[2] This is a good example of how the different levels of the human CNS can be implemented in an artificial robotic system. However, this system uses a lot of information of the physical structure (e.g. length of body parts) that may be unavailable in a real system.

[2]The Jacobian matrix is explained in more detail in a later section

Kuperstein (1988) presents a virtual model of an arm that learns to grasp objects by correlating arm muscle control signals to retinal maps from a binocular view. The arm muscles are paired as antagonistic pairs. Other than the pairing of muscles, the model has no a priori knowledge of the mechanical system and it is designed to "[...] maintain adaptation when unforeseen changes are made in the mechanical system or with partial damage to the model [...]" (Kuperstein, 1988). This virtual model is based on the "circular reaction" hypothesis about the development of the human child (Baldwin, 1894; Piaget, 1952). The circular reaction is the cycle of: a muscle action causes a sensory stimulus, which causes a certain brain-state, which in turn causes a slightly different muscle action, etc. The *primary* circular reaction is a hypothesis that behaviour in newborns is mainly directed to produce effects on the own body instead of on the environment, in order to learn from self-movement. A method commonly used for learning from self-movement is called 'motor babbling', which looks a lot like the circular reaction hypothesis.

Non-virtual implementations of a muscle-based robotic system also exist. An example is a robot called *Airic's Arm*, created by a company called Festo.[3] Festo produces artificial muscles called Fluidic Muscles. Pressurized air is used to control the muscle, even though the name suggests that a pressurized fluid is used. These muscles are modern versions of the McKibben actuator, developed in the 1950's and 1960's (Schulte, 1961). A McKibben actuator is basically a tube that expands under pressure. Because it expands, the width increases, but the length decreases, resulting in contraction of the length of the tube. Airic's Arm has 32 Fluidic Muscles, 32 air pressure sensors and 6 length sensors.

The goal of this thesis is to find muscle pairs for a muscle-based robotic arm so a higher level of control is possible. To find pairs that have opposite effects, the effects of the muscles should be represented in some way so they can be compared.

2.4 Curse of dimensionality

Humanlike robots with muscle-like actuators have a lot of Degrees Of Freedom, so a lot of combinations are possible and this complicates control. This is referred to as the *curse of dimensionality*. Luckily, there are some factors that reduce the complexity. The reach, or workspace, of the robot arm in spatial coordinates is limited due to the structure of the bones and joints and because of redundant effects of muscles.

To illustrate the limitation due to the structure, a schematic image of the workspace of the end effector of a fictional 2-dimensional humanlike robot arm is shown in Figure 2.2. Only two joints are shown here, a shoulder joint and an elbow joint. The shaded area represents the workspace of the end effector (the wrist) in Cartesian space. If the joint angles were unrestricted, the workspace of the wrist would cover a circle centered at the shoulder. Schomaker (1991, p. 179) calls this 'Degrees of Limited Freedom' (DOLF) and says it is "[...] very likely to enhance self-organized

[3] http://www.festo.com/cms/en-us_us/5009.htm

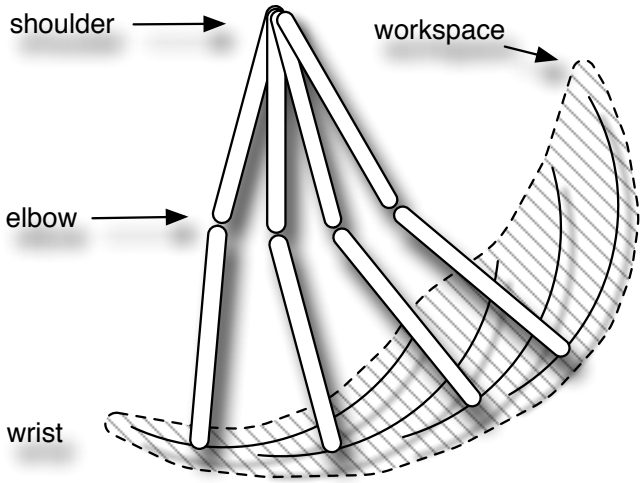


Figure 2.2: *Reach of the wrist for a hypothetical 2-dimensional arm shown as arcs for four different angles of upper arm, and the workspace of all combinations of upper/lower arm angles shown as shaded area. The workspace is constrained by the contraction range of the muscles and the limits of the joint angles.*

learning of the effector system in 'motor babbling' to a significant extent". The fictional humanlike robot arm discussed above only has two dimensions and two joints. For more dimensions and more joints, the output space expands quickly, but so do the restrictions.

The redundancy of muscle effects causes the output space to be limited as well. For example, one muscle rotates the wrist (supination, pronation), while another muscle affects the elbow joint angle. If the position of the end effector is defined off-center at the wrist, then both muscles affect the height of the end effector. Yet another muscle affects the height of the shoulder (scapular elevation), but this also causes the end effector to move upwards. This example shows three muscles that affect the same dimension. This is not surprising, as the input space is highly dimensional, while the output space is only two or three dimensional.

So the output space is limited by the DOLF and by redundancy of the effects of muscles. Is there more information that helps reduce the complexity? (Schomaker, 1991, p. 177) shows for a joint-based 2-dimensional arm that small changes in joint angle space cause small changes in spatial coordinates, so they vary smoothly. For a muscle-based system, the changes in spatial coordinates are non-linear, so the changes in spatial coordinates depend on the current position, but the overlapping effects of muscles may reduce the non-linearity. If this is true, a linear system can be used to approximate the effects of the muscles. Specifically, if a large change is subdivided into multiple smaller changes and the

state is inspected before each smaller change, the approximation may be close enough to a linear method. A very common method that describes effects in this way is a Jacobian matrix.

2.5 Jacobian matrix control

A Jacobian matrix is often used in the field of robotics to control the actuators of a robot. It denotes the rate of change in output for each change of input. Equation 2.1 shows an example for a *joint-based* actuator system. J denotes the Jacobian matrix, J^{-1} denotes the inverse, u_i is the control value of joint-based actuator i, θ_i is the current angle of joint i, $\hat{\theta}_i$ is the desired joint angle of joint i and α is a scaling factor that defines speed of change. The Jacobian matrix is used to make small changes to the current state, based on known effects of input on the output. If the Jacobian matrix is precise enough and the robot is able to move instantly, α can be 1 and the joint angle will be the desired angle after the actuator change.

$$\Delta u_i = \alpha \cdot (\hat{\theta}_i - \theta_i) J_i^{-1} \tag{2.1}$$

In equation 2.1, the inverse of a Jacobian matrix is used. The Jacobian matrix defines the rate of change of the output, given the input, but to calculate the input given a desired output, the inverse is needed. In some cases the inverse cannot be calculated because it does not exist (mathematically). For example, if input A has no effect on output B (the rate of change is zero), then what is the inverse? What is input A if output B should change? The answer can be anything as it does not matter how you change input A, so the answer might as well be zero. In these cases a pseudo inverse Jacobian matrix can be calculated with, for example, the Moore-Penrose method (Kent & Williams, 1998, p. 62).

The coordinate space of the output is usually joint angle space. For conventional joint-based actuators, this is a logical choice, but for a muscle-based system this is not necessarily the case. Muscle-like systems are polyarticular, meaning muscles can affect more than one joint, so multiple solutions may exist. Also, because actuators are not joint-based, there are no inherent joint angle sensors.

If the output is expressed in Cartesian space, the Jacobian matrix is smaller (because there are only 3 outputs for a 3-dimensional space instead of N joint angles), but also more complex, compared to a Jacobian matrix of a joint-based system expressed in joint space. The latter kind has a lot of zeroes in the matrix, because each joint-based actuator (input) only affects a single joint (output): only N of NxN elements are non-zero if there are N actuators. In Cartesian space, each actuator affects the end effector (some more than others), so there are less zeroes. Muscle-based systems have more DOF (because there are more actuators, so N is higher), this increases the complexity even further. Because of the factors described in the previous section, the complexity is reduced, but this is difficult to see in the Jacobian matrix.

2.6 Virtual muscle pairs

With a reduced complexity, the Jacobian matrix might be a useful representation for the effects of the actuators on the robot arm. If so, this representation can then be used to move the arm so muscle pairs can be learned. Because evidence shows that humans and monkeys plan actions in Cartesian space (Morasso & Sanguineti, 1995), and because endpoint stiffness of the arm is directionally tuned (Franklin et al., 2007), we are interested in muscle pairs that have opposite effects in Cartesian space.

These muscle pairs are different from the antagonistic muscle pairs because they do not affect the joint angle (or rotation of e.g. the wrist), instead they influence the end effector position. On the other hand, virtual muscle pairs also have some similarities with antagonistic muscles: both pairs are dynamic. An example of such an antagonistic muscle is the biceps, it form a pair with the triceps to change the angle of the elbow joint, but to supinate/pronate the forearm, the biceps and the supinator muscle form a pair with the pronator teres muscle and the pronator quadratus muscle. So an antagonistic muscle pair actually consists of pairs of *groups* of muscles and a single pair can form a different group with other muscles, depending on the task. This is similar with virtual muscle pairs, but the group size of a virtual muscle pair is probably larger because the virtual muscle pairs all act on the position of the end effector instead of on only one joint angle.

Virtual muscle pairs can be used for a system that has no a priori knowledge about the structure of the physical system, but only knowledge about the number of actuators, their state and the position of the end effector. Other knowledge about the arm is not needed for the operation of the arm, like length of bones or number of joints or where muscles are connected to the bone structure. Furthermore, relying on this a priori information about the structure may cause inaccuracies if the physical system changes due to use or if the knowledge is incorrect. In order to be adaptable to the environment, it is better to learn these properties. Animals have another reason to learn these properties, as their body grows and eventually shrinks during their lifetime. Robotic systems currently do not grow, but future systems might. An adaptive system can learn to cope with that. Being able to constantly relearn virtual muscle pairs is also necessary for adaptive systems, as these muscle pairs can be used, for example, to control compliance of a joint. A robotic system that is aimed to be adaptive in changing environments is developed by the Osaka University, under the Yuragi project. The Yuragi project inspired this research of a system that ignores joint space and adaptively learns to control itself.

2.7 The Yuragi principle

A natural environment is complex and changing. To enable robotic systems to work in such environments, they need to be able to cope not only with complexity, but also with fluctuation. A common approach is to try to suppress this fluctuation, but a different approach is to utilize this

fluctuation. This is biologically inspired from molecular systems that use fluctuation in the environment in order to operate (Yanagida et al., 2007). The aim of the Yuragi project at the University of Osaka is to research the use of fluctuation in natural systems for control.[4]

Fukuyori et al. (2008) used fluctuation to control multiple models of robotic arms. They show that their *Yuragi controller* can be used to perform a reaching task (move the end effector to a pre-defined position) with a 2-DOF virtual model, a 12-DOF redundant muscle-based virtual model and finally with a 26-DOF redundant muscle-based physical model. This robot arm is used in this thesis and is described in more detail in the next chapter. The Yuragi controller uses the Attractor Selection Model (ASM) (see below) to search for proper control values. The ASM is also developed at the University of Osaka.

Not only the robotic arm of the University of Osaka is used in this thesis, but also the exact implementation of the ASM, albeit in a different way. Fukuyori et al. (2008) use the ASM to control the arm, but in this thesis, it is used to improve the stability and adaptability of the proposed Jacobian matrix control. As this thesis focuses on virtual muscle pairs, more directed control is needed to decrease learning time to improve the feasibility of the research. In future systems, the ASM can play a larger role, by creating a hybrid system that incorporates principles from both the ASM and the proposed Jacobian matrix control.

2.7.1 The Attractor Selection Model

The ASM is based on bacteria that adapt to a dramatical change in the environment by altering their gene expression. The ASM uses multiple attractors that respresent different 'gene expressions'. Each attractor is a control state; in the case of the robot research above: muscle activation values. A current control state (randomly initialized) moves towards an attractor under certain conditions, and moves more randomly under different conditions by adding noise. The decision to use less or more noise is made by a so-called *activity function*. This activity function can be low or high, where low activity selects more random noise and high activity less.

For example, the activity function can be defined as low when the distance to the target is large, and while there is little movement (Fukuyori et al., 2008). If the distance to the target is low, the added noise is also low, but if for some reason the target changes, or the environment changes, the activity function becomes high. This causes the added noise to become dominant, enabling the current control state to move away from the attractor to a different one. If a different attractor (i.e. control state) results in a shorter distance to the target, the activity function becomes low again, resulting in less random noise, resulting in a convergence of the current control state to the closest attractor.

The design of the attractors is directly linked to the hardware and the activity function depends on the task to be performed. For example, if the

[4] 'yuragi' means fluctuation in Japanese

hardware has two actuators, the attractor should consist of two values. If the task is not to move the end effector close to a target, but to move the arm in circles, then the activity functions should reward circular behaviour. An adaptive version of the ASM exists as well, the AASM. This enables the system to adapt to changing noise, a changing environment or change in robot hardware (e.g. wear or other damage). AASM changes the location of the attractors in state space, depending on the previous usefulness of the attractor.

The ASM provides an abstraction by defining a goal and a method of reaching that goal without the need for detailed knowledge about the structure of the hardware or a detailed plan how to perform the task (only the goal is needed). It is a simple method, yet very versatile.

2.8 Thesis overview

With this background information, the following question can be asked:

> Can a humanlike robot with a muscle-like system learn virtual muscle pairs by using a Jacobian matrix?

Chapter 3 describes the robot arm that is used in this thesis in more detail. Before virtual muscle pairs can be learned, the effects of actuators must be studied. Chapter 4 shows the effects of the actuators on the robot arm and how the Jacobian matrix is used to represent these effects. To test if this representation is useful to control the position of the end effector, a method was developed to move the arm. This method is described in chapter 5. Chapter 6 discusses an experiment to test if the method is able to position the end effector. This experiment also tests if the accuracy can be improved by using the ASM. After it is shown that the representation is useful to move the robot arm, the representation of the effects can then be used to select muscle pairs that move the arm in a specific direction. Chapter 7 shows how this is done and also discusses an experiment to test if the muscle pairs are useful virtual muscle pairs. Finally, chapter 8 concludes this thesis.

Chapter 3

26-DOF Humanlike Robotic Arm

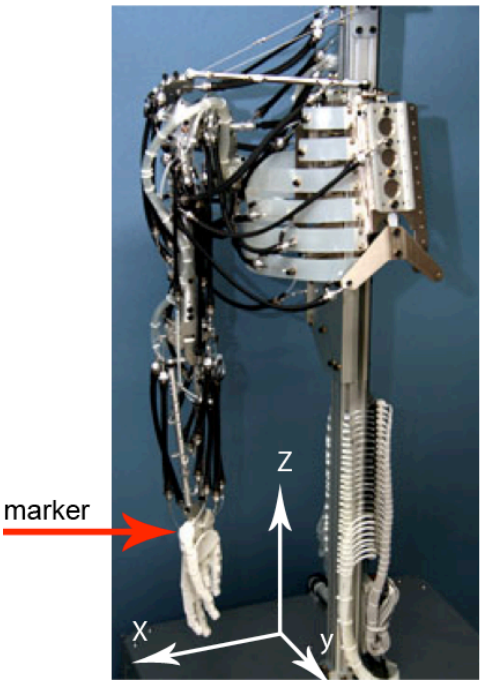


Figure 3.1: *Robot arm with marker and coordinate system of marker*

The humanlike robotic arm that is used for this master thesis is developed by the University of Osaka and by the Kokoro corporation.[1] It is part of the Yuragi project from the University of Osaka. This robotic arm is humanlike in the sense that the artificial muscles are attached like the muscles in the human arm. It has 26-DOF in the form of air-actuated muscles. The whole system consists of an air compressor, a controller, the skeletal system and the muscles. The skeletal system and muscles are shown in Figure 3.1. The red arrow indicates the position of a marker and

[1] http://www.kokoro-dreams.co.jp/english/

the X-, Y- and Z-axes indicate the coordinate space of that marker. The black tubes in the Figure are the air-actuated muscles. The air compressor (not shown in the image) delivers pressurized air to the controller, located in the base below the skeletal system. The pressure exerted on each muscle is determined by the controller. Attached to the bottom of the spine, air tubes (white) can be seen that connect the controller to the muscles via air valves and pressure sensors. The shoulder joint is a ball joint and because the shoulder is suspended by metal wires and a clavicle-like bar, the shoulder itself can move up/down and forward/backward. The lower arm consist of two bones that each have two ball joints to connect them to the upper arm and the wrist. The hand is attached to the wrist by two hinge joints to allow pitch and jaw; roll of the hand is already realised by the two bones of the lower arm. The fingers of the hand can be contracted by one muscle, but they do not have enough strength to firmly grasp an object.

3.1 Actuators

The robot arm has 26 actuators that control 32 air-actuated muscles. Each actuator has one pressure sensor. The pressure sensors are not used, except for some pilot experiments. Some of the actuators control multiple muscles, e.g. the three muscles between the upper arm and the rib cage are controlled by one air flow valve. In this thesis, the term *muscle* and *actuator* are used interchangeably and always refer to the 26 actuators, since the system only has knowledge about these.

The actuators are controlled by air pressure valves, which needs Volts as input. In the proposed model, values between [-1, 1] are used. These are converted to Volt with equation 3.1, where u is the actuator value in the model and v is the value in Volt that is sent to the robot controller.

$$v = 4.0 + 1.5 \cdot tanh(10u) \tag{3.1}$$

This equation clamps the range of the actuator between [2.5, 5.5] V, as shown in Figure 3.2. This control function was empirically established during previous research on this robot arm. Actuator control values range between [-1, 1], and is centered at zero. Actuator values below zero

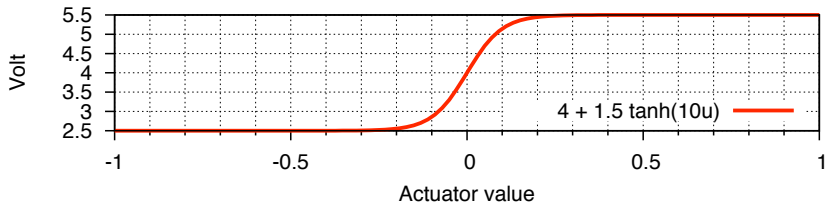


Figure 3.2: *Control function that converts actuator control value (u) to voltage. The air valve regulates pressure, its input is voltage. The control function limits the air valve control range to [2.5, 5.5] V*

result in the same pressure as a value of zero, even though the voltage of the actuator is set to a lower value. If an actuator is set to 0.3 (which is correlated to the maximum of 5.5 V, thanks to the control function), pressure values measured at the pressure sensors for each actuator differ. They can be around 2.0 or 2.3 V, depending on the actuator, while the lowest pressure value is measured at about 1.0 V. The actual pressure in the actuator ranges from 0.1 MPa (atmospheric pressure) to 0.3 MPa, which is the pressure delivered by the compressor. This maximum of 0.3 MPa is restricted by a regulator.

Changing actuator values almost instantaneously changes pressure, but it takes a while before the pressure is stabilized. Some muscles stabilize faster than others for large movements, ranging from 1 second to 5 seconds.

The actuator values can be changed 100 times per second. The computer that sends commands to the robot controller runs ARTLinux, a variant of Linux that allows real-time (deterministic) operation.[2]

3.2 Motion capture system

The motion capture system is a setup of 5 Hawk Digital cameras, built by Motion Analysis Corporation.[3] The accompanying EVaRT capture software is used to record the position of markers on the arm 60 times a second, though the maximum is 200 times a second. The motion capture software runs on a separate Windows computer and a simple program allows access to the data over the network. This software outputs the position of the markers 100 times a second. Multiple markers are present, but for this research only one marker is used, placed on the wrist. In Figure 3.1 the marker is indicated by a red arrow and the coordinate system of the motion capture system is shown by the axes.

The axes are shown for illustration purposes, the actual origin is set to the marker when the arm is in the resting position (shown in the Figure) each time the program is initialized. This makes comparing values between experiments easier, as wheels located under the base enable the robot to be moved. To also ensure the orientation is the same each experiment, tape is used to mark the location and orientation of the robot. The Y-axis is reversed compared to the original coordinate system. This ensures that the Y-axis is in the same range as the Z- and X-axes, so the graphs can be more compact or contain more detail when they are plotted together.

[2] http://www.dh.aist.go.jp/en/research/humanoid/ART-Linux/
[3] http://www.motionanalysis.com/

Chapter 4

Methods: Jacobian matrix training

The previous chapter described the hardware of the robot arm, but not the effects of the actuators on the Cartesian position. Before the arm can be controlled successfully, these effects must be learned. After the effects are learned, the virtual muscle pairs can be deduced. How this is done is explained in later chapters. Learning the effects and the use of this information are separate phases: a training and testing phase. Therefore, the adaptability of the system is not perfect; if the environment changes a lot, the system should retrain. However, by having a separate training and testing phase, the system can be analysed better and the reproducibility is increased as well.

4.1 Jacobian matrix

As explained before, a Jacobian matrix is a representation of the effects of a system; for each input the effects on the output are stored in a matrix. The Jacobian matrix is stored as a MxN matrix, where N is the number of inputs and M the number outputs. For our system, $N = 26$ and $M = 3$ (because there are three coordinates in our Cartesian space). Equation 4.1 shows the Jacobian matrix for our system, where u_n are the actuator control values and x, y and z are the Cartesian dimension of the end effector.

$$J = \begin{bmatrix} \frac{\partial x}{\partial u_1} & \frac{\partial x}{\partial u_2} & \cdots & \frac{\partial x}{\partial u_{25}} & \frac{\partial x}{\partial u_{26}} \\ \frac{\partial y}{\partial u_1} & \frac{\partial y}{\partial u_2} & \cdots & \frac{\partial y}{\partial u_{25}} & \frac{\partial y}{\partial u_{26}} \\ \frac{\partial z}{\partial u_1} & \frac{\partial z}{\partial u_2} & \cdots & \frac{\partial z}{\partial u_{25}} & \frac{\partial z}{\partial u_{26}} \end{bmatrix} \tag{4.1}$$

A Jacobian matrix can be obtained by changing each actuator separately a little bit and recording the change in output space. This numerically estimates the Jacobian matrix instead of finding the actual functions

and partially derive them. While calculating a Jacobian matrix for our system in this way, an assumption is ignored: for a constant matrix, it is assumed that the effect of each actuator is independent on the current state of the system, which is not the case in our robot arm.

4.2 Posture dependent effects

A constant Jacobian matrix describes an effect of each input that is the same for each state of the system. In joint space, this may be true, but in Cartesian space the effects of each actuator depend on the current posture of the arm. Figure 4.1 illustrates the difference in effect depending on posture. Changing the actuator value u will move the end effector upwards (dy) and to the left (dx) in posture 1. Changing it in posture 2 with the same amount will again move it upwards (dy), but also back to the right (dx). As the effect is different for different postures, how can a Jacobian matrix still be used?

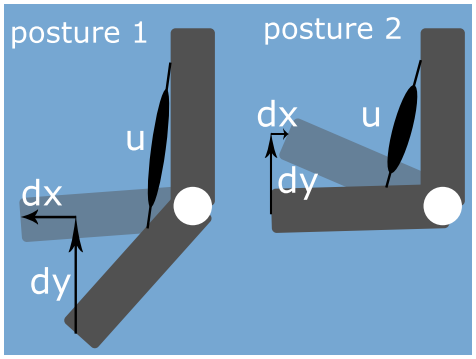


Figure 4.1: *Effect of actuator change (du) on coordinates (x, y) depends on current posture*

4.3 Multiple Jacobian matrices

As discussed in section 2.4, the workspace of the robot arm is rather limited; that is, the redundant muscles cause similar effects in Cartesian space. Because there is a lot of overlap in the output space, we expected that the average effect of multiple Jacobian matrices can be used to control the position of the end effector. So instead of using one Jacobian matrix, multiple Jacobian matrices are calculated at different postures and the average is used to create a new Jacobian matrix. To test if this average Jacobian matrix has predictive abilities, a pilot experiment was performed.

4.4 Pilot experiment: effects of actuators

The pilot experiment tested if the average displacement of each actuator is different between actuators and if their displacement is predictable. This is tested by calculating a Jacobian matrix at 20 different postures and from these 20 matrices, the mean and standard deviation of the displacement for each dimension for every actuator.

Before the Jacobian matrix is determined, it is unknown how to move the arm to a specific posture. To move the arm to 20 different postures, 20 random states were generated. Each state represents a vector of 26 actuator values, one for each actuator. Each actuator value is set to 0.0 or 0.3. The value of 0.0 represents a relaxed state, while 0.3 represents a contracted state, as shown before in Figure 3.2 on page 14. Setting actuators to each of the 20 states will result in random postures which will hopefully span enough of the workspace of the robot arm.

The two actuator values describe above are base values. An actuator with the higher value is an activated muscle, while actuators with the lower value are deactivated muscles. To activate a single muscle, the lower value is gradually increased to 0.5 and the higher value is gradually decreased to -0.2 (also a difference of 0.5). The actuator values are changed gradually to prevent damage to the arm, as the effect of changing muscles with a high speed was initially not known. An interval of 0.5 is preferred over 0.3, to easily distinguish the changing actuators in graphs, so this is mainly for analysis. For normal operation, an interval of 0.3 will suffice.

When moving between each of the 20 random postures, all muscles are changed simultaneously. The control values of each actuator are linearly interpolated over five seconds between two postures. No actuator value is changed for another five seconds, to allow the arm to stabilize, as some muscles' effect span five seconds (see section 3.1). The long stabilizing period is to make sure the effects of actuators do not overlap each other.

At each of the 20 random postures, each of the 26 muscles is moved once by changing the value of one of the actuators; if the muscle is activated, it will be deactivated, otherwise it will be activated. Other muscles are unaffected during the movement of a single muscle. For the only changing muscle, the actuator value is linearly interpolated over a period of 100 milliseconds. Five seconds later, the arm is returned to its previous posture by interpolating the actuator values back, also over a period of 100 milliseconds. Before the next actuator is moved, no actuators are changed for another five seconds, again to ensure that effects of actuators do not overlap.

During this experiment, the actuator values and the position of the marker are stored for each time step (1/100th of a second). At each time step, the current state of the training phase (changing posture, moving one muscle, moving muscle back, stabilizing) is also stored so the effect of each actuator can be calculated after the training phase.

The training phase looks like "primary circular-reaction hypothesis" (Piaget, 1952), the theory that behaviour in newborns is mainly directed to produce effects on the own body instead of on the environment, but our

training phase is more structured. It is possible to move less deterministically, but this would take a lot more time to train and the results would be more difficult to reproduce.

4.4.1 Results

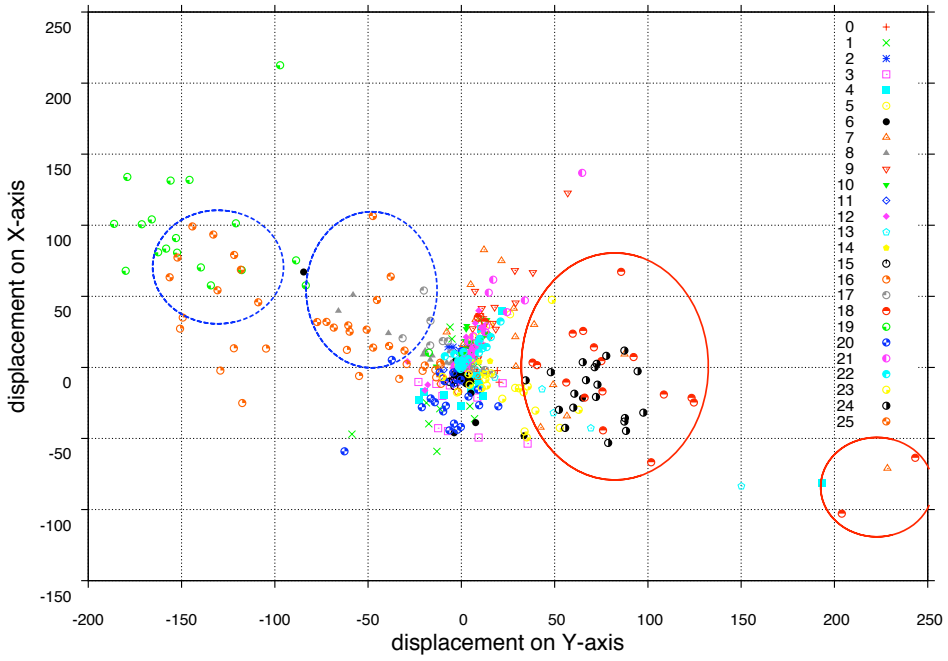
Figure 4.2 shows the change of the position of the end effector during the training phase. Each dot represents the displacement of one actuator for one posture. For each of the 20 postures, all 26 actuators are shown. Figure 4.2a shows the Y displacement on the horizontal axis and the X displacement on the vertical axis. Figure 4.2b shows the Y displacement again on the horizontal axis but the Z displacement on the vertical axis. These axes correspond to the axes drawn in figure 3.1, page 13. Looking at this data, a few observations can be made:

1. actuators 18 and 25 seem to have a multi-modal distribution,
2. other actuators seem to have a unimodal distribution,
3. if one actuator has a large displacement in the negative direction, it does not have a large displacement in the positive direction and vice versa,
4. some actuators have similar effects (e.g. 1 and 4),
5. the effects are spread out roughly along a diagonal.

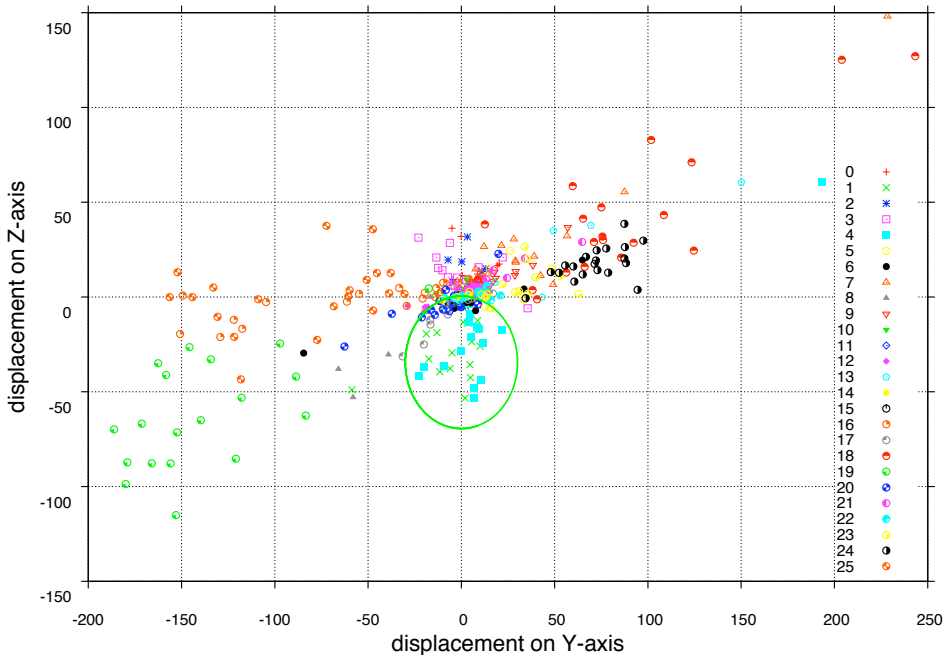
Actuators 18 and 25 are marked in Figure 4.2a with two solid and two dashed ellipses respectively. Actuators 1 and 4 are marked in Figure 4.2b with a single ellipse. Observations 2 and 3 show that the way the end effector will move when actuators are changed can be predicted, while observation 4 confirms that some muscles are indeed redundant. Even though observation 1 shows that the distribution of actuators 25 and 18 looks multi-modal, one can also argue that actuator 18 has some outliers. Observation 1 will not pose problems considering observation 3; if this data is averaged, the direction of motion will be correct even though the exact distance is not. All these observations show that there is indeed redundancy in the effect of muscles and the distribution of effects is rather limited, which confirms our expectations.

4.5 Average Jacobian matrix

The average and standard deviation of each of the 20 postures were calculated per dimension. Figure 4.3 shows the averages and standard deviations for the Y dimension ordered by average Y displacement. The horizontal axis shows the actuator number, the vertical axis shows the average displacement of the end effector in the Y dimension when the actuator is increased by 1.0. The standard deviation is shown as an error bar at the end of each average Y displacement bar. For a normally distributed population, one standard deviation is 68.2% of the set (almost



(a)



(b)

Figure 4.2: *Change in end effector position due to actuator change during the training phase. All 26 actuators are shown for 20 postures. Ellipses in Figure 4.2a mark actuators 25 (dashed line) and 18 (solid line). Their distribution seems multi-modal. A single ellipse in Figure 4.2b marks both actuators 1 and 4. Their distributions seems similar to each other.*

14 postures). Therefore, two standard deviation is also shown to show the spread of 95.4% of the set, which is 19 postures. The two standard deviation is shown as a rectangle around each error bar. The population is not perfectly normally distributed, so these percentages are not accurate, but they do give an indication of consistency.

Figure 4.3 shows that when the actuator value is increased by 1.0, actuator 11 and 15 are not very likely to move in the Y-direction. Actuator 24 however, will likely move around 70 mm. Actuator 25 on the other hand, is likely to move around -85 mm. Because the standard deviation of actuator 25 is larger than that of actuator 24, the exact displacement is less likely to be -85 mm than it is 70 mm for actuator 24. Therefore, the average is an estimator for the direction and the amount of displacement, while the standard deviation gives a rough estimate to the consistency of the displacement.

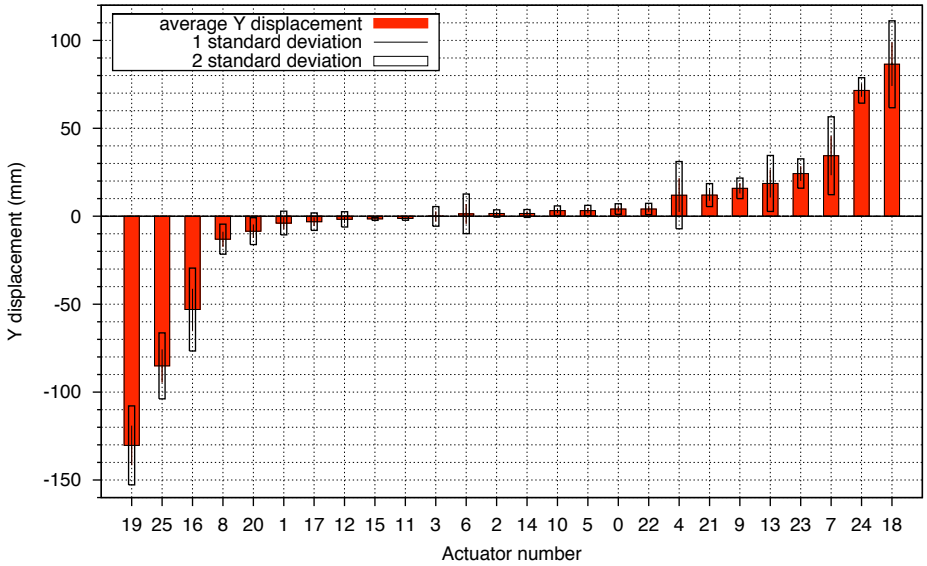


Figure 4.3: *Average and standard deviation of the Y-displacement of all 26 actuators for 20 Jacobian matrices. The horizontal axis shows the actuators ordered by their average Y-displacement. There is a lot of difference in standard deviation (consistency) of the displacement between actuators.*

4.6 Weighted Jacobian matrix

To predict the effect of each muscle, the average of each actuator is divided by its standard deviation, resulting in a weighted Jacobian matrix. This is expressed in equation 4.2, where m is the dimension (x, y or z), n is the actuator number, $\vec{J}$ is the vector of 20 Jacobian matrices and W is the resulting weighted Jacobian matrix.

$$W_{n,m} = \frac{average(\vec{J}_{n,m})}{standard\_deviation(\vec{J}_{n,m})} \tag{4.2}$$

Figure 4.4 shows the weighted Jacobian matrix for the Y dimension ordered by weight. The weighing caused the order to change slightly from the one in Figure 4.3. For example, actuator 7 has moved to the left, because its standard deviation is fairly large compared to an actuator with a similar average displacement. Actuators 3 and 6 remain somewhere in the middle, because their average displacement is around 0 and they have a large standard deviation. Actuator 24 changed places with actuator 18, because the standard deviation of actuator 24 is much smaller than that of actuator 18, even though their average displacements are similar. The standard deviation has a large influence on the final weight, maybe too much. In order to lower the effect of the standard deviation, it could be scaled down. However, scaling was not applied to avoid premature optimization and to keep the number of changeable parameters low.

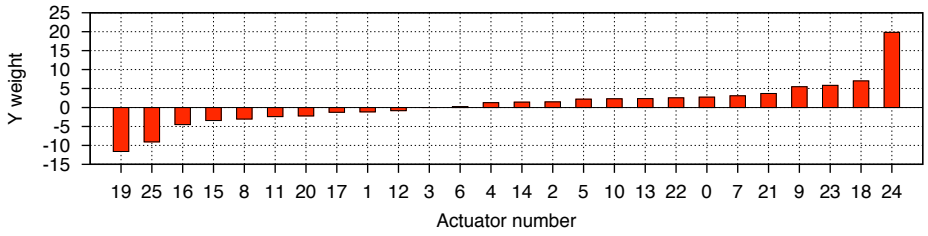


Figure 4.4: *Weighted Jacobian matrix for Y dimension, result of dividing the average displacement by the standard deviation, from Figure 4.3. The horizontal axis shows the actuators ordered by weight. Note that the order of muscles is changed compared to Figure 4.3, which will yield a higher predictability of movement.*

4.7 Discussion

The method described above shows that the weighted Jacobian matrix can be used to predict movement of the end effector when actuators are changed. Before designing and testing the control method, there is something that needs to be considered. The weighted Jacobian matrix is an approximation because of the following factors:

1. robot arm workspace coverage,
2. averaged Jacobian matrices,
3. combined effect of actuators,
4. actuator and sensor noise,
5. dynamic environment.

(1) During the training phase the robot arm is controlled randomly to learn the effects of the actuator changes. Because of this, the workspace of the robot arm remains unknown during this phase, so it is not guaranteed to cover the whole workspace of the robot arm during the training phase. (2) The average is an approximation by definition. As explained in section 4.2 the effect of an actuator is different for different postures. (3) The Jacobian matrix is made up of *partial* derivatives, which means that it describes the effect of the end effector when *only one* actuator is changed. The effect of changing multiple actuators at once is initially unknown. (4) Of course there is always noise in actuators and sensors, so every sensor reading and every actuator setting is approximate (5) Last but not least, the environment can change after training, maybe even during training. As the robot is embodied, it is part of the environment, so changes to the robot like wear and tear will also change the effect of actuators.

Because of these approximations, the system might get stuck in a local minimum. When testing the control method later in this thesis, we should be aware of these limitations. The next chapter describes the general control method using the weighted Jacobian matrix.

Chapter 5

Methods: Jacobian matrix control

The previous chapter discussed the effects of the actuators and proposed to represent these changes in a weighted Jacobian matrix. This chapter discusses the control of the robot arm using the weighted Jacobian matrix. There are some issues that need to be solved to complete the control method. First, a conventional approach of using a Jacobian matrix is to invert it, as explained in section 2.5. Second, the weighted Jacobian matrix has three separate vectors of weights, one for each dimension. They need to be combined somehow before they can be used to control the actuators. Finally, actuator values should stay within a predefined range so some mechanism should be put in place to take care of that. These issues are discussed in this chapter.

5.1 Inverse matrix

In section 2.5 the conventional approach for using a Jacobian matrix is explained. An inverse Jacobian matrix is used in equation 2.1, because a Jacobian matrix defines the change of output for each change of input while the inverse Jacobian matrix defines the change of input needed to get a change of output.

In our situation with a muscle-based system and a Cartesian output space, each muscle affects the position of the end effector, even if it is only a little bit. This means that if a conventional approach is used, muscles that change the position of the end effector only a little bit are changed a lot (because a lot of change in input is needed to make the change in output), while muscles that change the end effector a lot are changed only a little bit. This results in a lot of energy being used, as all muscles are used. The most ineffective muscles are activated most, so this is not preferable at all.

To use the most effective muscles more than less effective muscles, the normal Jacobian matrix is used instead of the inverse. With this, the most effective muscles are changed the most. However, by using the normal

Jacobian matrix, we lose the advantage of being able to calculate the precise effect needed for the movement; we only know which muscles have the most effect, but we do not know how much to change them exactly. We have to make small changes until we are close enough to the target. Also, the units of the inverse Jacobian matrix (input space) are different from the normal Jacobian matrix (output space). This can be corrected by a converting factor.

5.2 Combining dimensions

The weighted Jacobian matrix has three dimensions for each actuator; X, Y and Z. However, the control method moves the end effector by changing the actuator values, so it needs one value for each actuator instead of three. The combining of these three dimensions is different for each situation: if the arm needs to move in the direction of the X-axis, the X dimension of the weighted Jacobian matrix is the most important one. So before the separate dimensions of the weighted Jacobian matrix are combined, they are multiplied by their respective dimension of the distance to the target. After that, the dimension can be combined together by addition or multiplication. All distance values are in millimetres and most weights are above one, so most factors would be much larger than one. Using multiplication would therefore result in high actuator changes, so addition is used to combine the three dimensions into one.

5.3 Jacobian control method

Equation 5.1 shows how the Jacobian matrix is used to move the robot arm. The parameter α is a scaling factor to control the speed of change, m is the dimension (x, y or z), p is the current position of the end effector, $\hat{p}$ is the target position and $W_{m,n}$ is the weighted Jacobian matrix from equation 4.2.

$$\Delta u_n = \alpha \cdot \sum_{m=1}^{3} \left(|\hat{p}_m - p_m| \cdot W_{m,n} \right), \quad 0 < \alpha \leq 1 \tag{5.1}$$

The parameter α is set to 10^{-6} in all experiments. This parameter is so small because there are three reasons to slow down the movement:

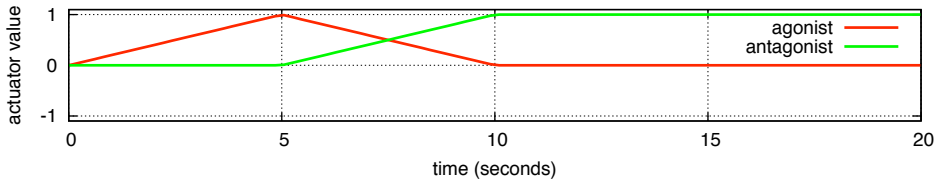
1. Cartesian range $|\hat{p}_m - p_m|$ differs from actuator range,
2. maximum safe movement speed of robot arm is still unknown,
3. combined effect of changing multiple actuators at once is unknown.

(1) The Cartesian coordinate range of the end effector has a range of about 350 (mm) while the actuator range is [-1, 1], so this takes care of the conversion factor discussed above. (2) Because the safe movement speed is unknown, the changes to the actuator values should be gradual. A small α spreads out changes over more time. (3) The combined effect

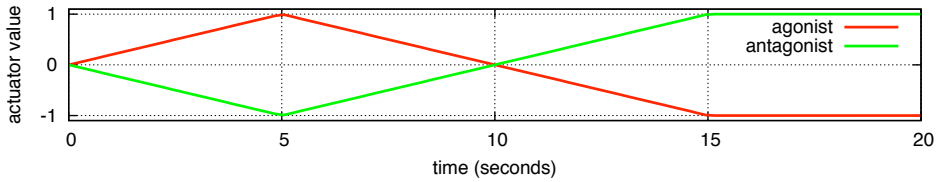
of changing multiple actuators is unknown, as discussed in the previous chapter. Because the magnitudes of slow-down needed for the second and third reason are unknown at this stage, a high α was chosen, to be on the safe side. Therefore, the value of this parameter is sub-optimal and causes the arm to move much more slowly than it is actually capable of.

5.4 Negative actuator values

Setting an actuator value to zero results in the lowest pressure for that actuator. Values lower than zero do not seem to result in lower pressure values. It is not known if negative values result in differences in other behaviour. For example, it is possible that when an external force is applied to the muscle (for example by gravity or by other muscles), it will stretch more if actuator values are negative, but this was not tested.



(a) Actuator values are *not* allowed to be negative.



(b) Actuator values are allowed to be negative

Figure 5.1: *Moving back and forth: effect of negative actuator values on two muscles that have a negative correlation. Not allowing negative values (a) results in not being able to go back to an all-zero state. Allowing negative values (b) results in longer response time.*

Negative actuator values are allowed in our method, but not because the muscles might behave differently for negative values. There is another side-effect to using sub-zero values. This is illustrated in Figure 5.1. Figure 5.1a shows actuator values that are not allowed to be negative (case 1) and Figure 5.1b shows actuator values that are allowed to be negative (case 2). Both images show the actuator values for an agonistic muscle and an antagonistic muscle. In both cases, the agonistic muscle is activated while the antagonistic muscle is deactivated. This symmetrical movement is an inherent effect of Jacobian control, because the muscles have a negative correlation: to move the end effector towards a position, some muscles are increased while others are decreased. The sign (and magnitude) is defined by the muscle weight from the weighted Jacobian

matrix. Of course this is not perfectly symmetrical in a real system, but to illustrate the problem, only two muscles are considered and the magnitudes of their weights are assumed to be the same, while their sign is different.

For both cases, all muscle values are initially set to zero. Then the agonist muscle is activated and the antagonist muscle deactivated. After 5 seconds, the process is reversed for 10 seconds, then the two values are left unchanged. The values are clamped to [0, 1] in the first case and [-1, 1] in the second case. The times to move are arbitrary and serve only to illustrate the problem.

In the first case, it is impossible to go back to a state where all muscle values are zero once they change from zero. This is because negative values are clamped to 0, while positive values are clamped to 1 and because the muscles change symmetrically. After 5 seconds, the average actuator value is 0.5 in case 1 and will always be. Of course the algorithm can be changed so that muscles do not change symmetrically, or a constriction can be added that delay the activation of some muscles, but this would make the control method much more complex.

The second case already has a delay. Because the antagonist value is decreased below zero, it will take some time before it is positive again. In our simplified example, this is the exact moment the agonist muscles is dropping below zero. The average actuator value is always 0 in this case. The control method is simple compared to the first, while also having the possibility of going back to the all-zero state.

Of course there are disadvantages to the second method: it takes more time to move the arm and tension on the bone structure can not be increased by activating both muscles at the same time, to increase stability. Both disadvantages are not problematic, because they address issues that are outside the scope of this thesis. However, if more speed is necessary, the α parameter can be increased.

The preferred range of the actuator values is [-0.5, 0.5]. Even though the values outside the range [0.0, 0.3] have no effect on the pressure, the extra range acts as a time buffer. Also, the extra positive range makes it easy to see if the control method can reach the desired position; values outside the range [-0.3, 0.3] mean that the control method is struggling to get to the right position. Values outside the preferred range [-0.5, 0.5] can also occur, but this should not happen often as it adds more delay.

5.5 Actuator value divergence

If a target can not be reached because it is physically out of range or because the control method is stuck in a local minimum, the actuator values will increase outside the preferred range without having any effect on the position of the end effector. This divergence was observed in preliminary experiments. The actuator values easily reached above 1.0 and below -1.0. An example of this is a reaching task experiment (target: 50, 120, 400) displayed in Figure 5.2.[1]

[1]The reaching task is discussed in detail in chapter 6.

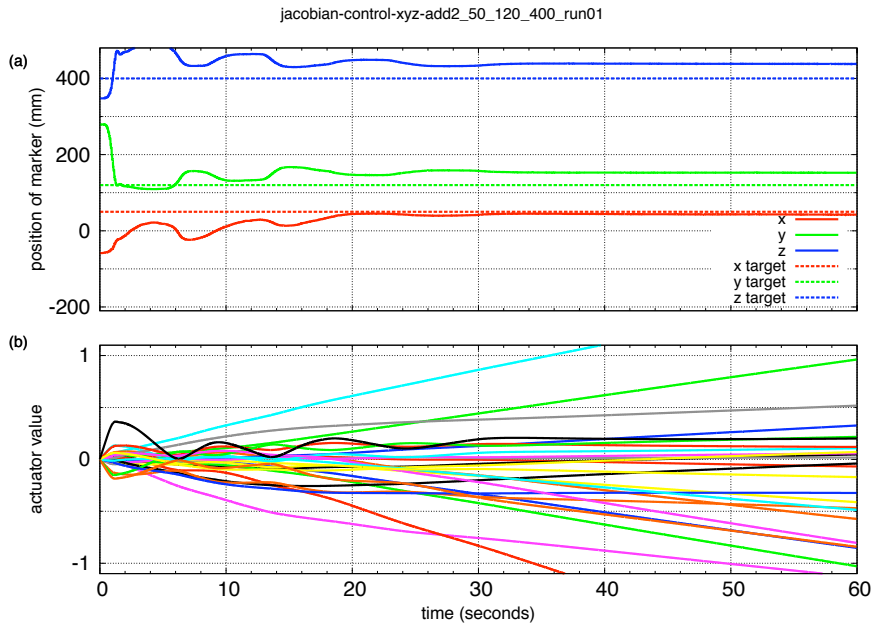


Figure 5.2: *Extreme case: strongly diverging actuator values. Figure 5.2a shows the position of the marker (not changing much after 30 seconds). Figure 5.2b shows the actuator control values (strongly diverging, even when the position does not change). The control function limits the pressure to a safe range, so large actuator values will not affect the position. The system has no knowledge about this and tries to move the arm closer by increasing actuator values.*

The horizontal axis shows the time in seconds, for a total of 60. The Y-axis of Figure 5.2b shows the actuator values, the Y-axis of the Figure 5.2a shows the position of the end effector in solid lines and the target values in dotted lines. In this extreme case the actuator values diverged strongly from zero while the end effector position stabilizes, between 20 and 60 seconds. The actuator values diverge because the control method increases actuator values in order to move the end effector closer to the target, but it is unaware that values outside the range [0, 0.3] have no effect on the position of the end effector.

The control function that converts actuator values to voltage, limits the actual pressure range, so the pressures did not reach critical values. However, allowing actuator values outside the preferred range results in long delays, as discussed in section 5.4.

5.5.1 Decay

The actuator values should be decayed each time-step to avoid them growing outside the preferred range; or, if they grew outside that range, shrink them back to the preferred range. The decay is performed on the previous values, before Δu is added. There are several choices for the kind

of decay: linear decay, which decays the values with a constant factor and exponential decay, which decays stronger when values are larger.

A disadvantage of linear decay is that it decreases values not only outside the preferred range, but also inside. Decreasing all values inside the preferred range will cause a relaxing of all the muscles so the arm will change position under the influence of gravity. This is detected and corrected by the system, but because of the low α parameter, the correction is slow. Preliminary experiments showed that the arm will oscillate up and down due to the slow correction.

A disadvantage of exponential decay is that it changes the ratio between actuator values. Changing the ratio between actuator values causes the robot arm to change posture, this is what the control method should do.

To illustrate the differences between linear and exponential decay, a comparison of different decay functions is shown in Figure 5.3. The horizontal axis shows the current actuator value, with a grid for values inside the preferred range and no grid for values outside the preferred range. The vertical axis shows the amount of decay. Two functions are plotted: linear decay with one parameter and exponential decay with two parameters.

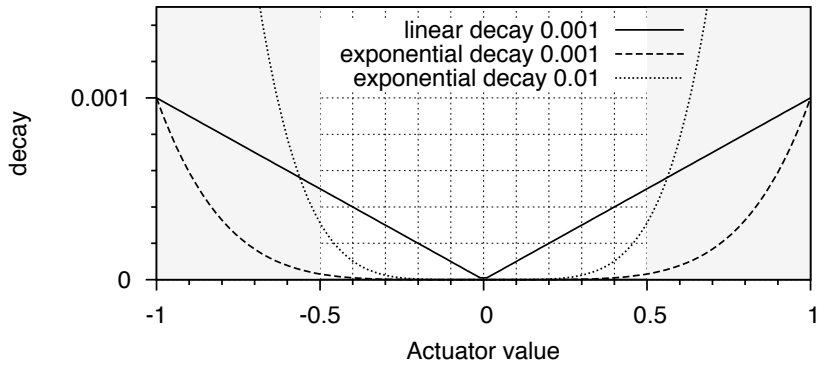


Figure 5.3: *Comparison of linear decay and exponential decay. Decay is used to keep the actuator value in the preferred range of [-0.5, 0.5]. Linear decay also decreases values within the preferred range, while exponential decay decreases values within the preferred range less strongly than outside.*

The linear decay function is $x' = \lambda x$, the exponential decay function is $x' = x - \lambda x^5$. For linear decay, a λ of 0.001 was chosen (for motivation, see Appendix A), but exponential decay with the same value will affect values outside the preferred range less heavily than linear decay. Therefore, the exponential decay function for both $\lambda = 0.001$ and $\lambda = 0.01$ is shown.

5.5.2 Pilot experiment: decay

As both kinds of decay have disadvantages, a pilot experiment was performed to determine the final decay function. The best decay function should have a low error rate and a consistent result.

Three decay functions were tested:

- exponential decay as described above (with $\lambda = 0.01$),
- linear decay with suppression, and
- conditional linear decay.

Linear decay with suppression makes a selection of muscles, based on their weight order, while suppresses the rest.[2] Selected actuators are *not* decayed, while the other actuators are decayed with the linear method. This method strongly changes proportions between actuator values, so it is called suppression. Linear decay with suppression is based on the idea that only a number of muscles is needed to make a movement, while the rest can relax. The exact number of muscles needed is difficult to determine, but from Figure 4.3 and 4.4, the number of muscles was chosen to be six muscles from the left (negative displacement) and six from the right (positive displacement). Maybe three from each side are good enough for one dimension, but there are three dimensions and the sequence of actuators can be in a different order for each dimension. Because there is probably overlap between the muscle orders of three dimensions, six muscles are probably sufficient and it will not be necessary to use 9 muscles.

The conditional linear decay control function is based on the idea that the total activation of muscles should not exceed a certain threshold, in order to preserve energy. If it does exceed this threshold, the total activation is lowered by decreasing all actuator values with the same proportion using the linear decay function. This method does not change proportions between actuator values.

For each decay function, a reaching task was performed for six targets, each five times. Per run, the actuator values were updated each time-step with equation 5.2, where $\Delta u_n(t-1)$ is the result from equation 5.1 and $f(x)$ is the decay function. Each condition was tested 60 seconds and from these 60 seconds, the last 10 seconds were used to calculate an average Mean Squared Error (MSE), where the distance to the target was used as error. This experiment is basically the experiment described in chapter 6.

$$u_n(t) = f(u_n(t-1)) + \Delta u_n(t-1) \tag{5.2}$$

[2] For a more detailed description of the selection procedure, see section 6.3

5.5.3 Results of decay experiment

The results of performance tests of these three decay functions are shown in Figure 5.4. The horizontal axis shows the targets, and for each target the three decay functions, while the vertical axis shows the average MSE over five runs. The standard deviation over five runs is shown as an error bar on top of the filled bars. The exponential function always performs best. In three cases (R3, R4 and R5) the conditional function performs similar to the exponential function. Furthermore, the exponential decay performs consistently among different targets, so this decay function was chosen as the decay function that is used in the rest of this thesis.

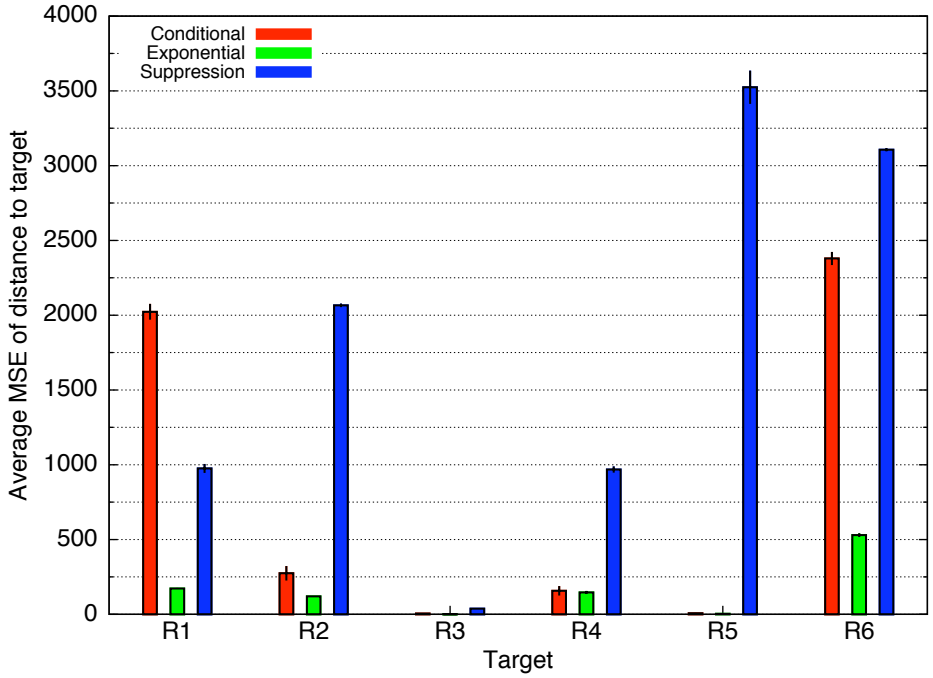


Figure 5.4: *Performance of different decay functions. Error bars show standard deviation over five runs. Conditional linear decay and linear decay with suppression always perform worse than exponential decay, which is very consistent over the targets.*

5.5.4 Final decay function

The decay function used in the two main experiments, discussed in the next two chapters, is the exponential decay with a λ of 0.01. Equation 5.3 was used to update the actuator value. This equation is the same as equation 5.2 with $f(x) = x - \lambda x^5$, the exponential decay function.

$$u_n(t) = u_n(t-1) - \lambda u_n(t-1)^5 + \Delta u_n(t-1) \tag{5.3}$$

5.6 Summary

A control method was discussed to control the robot arm with the weighted Jacobian matrix. An exponential decay was added to the control method to keep actuator values within the preferred range of [-0.5, 0.5]. Of the decay functions tested, exponential decay was the most stable and highest performing. Negative actuator values are allowed, even though they have no additional effect on the end effector, but they allow for a simpler method while still retaining the ability to return to an all-zero actuator state. In the next two chapters two experiments are discussed to test the performance of Jacobian control.

Chapter 6

Reaching Task

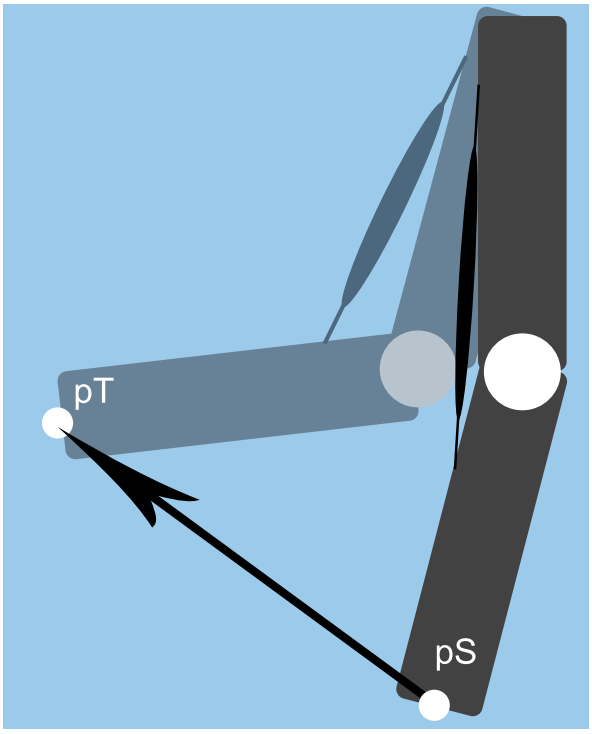


Figure 6.1: *Reaching task shown schematically. The goal of the task is to move the end effector of the robot arm from the starting position pS to the target position pT with the control method described in the previous two chapters.*

The purpose of the reaching task experiment is threefold. First, to test if the information from the weighted Jacobian matrix can be used to position the end effector of the robot arm. Second, to find out if the control method can be improved by adding noise to the control signal. Third, to test if the end effector can be positioned with a subset of muscles.

The positioning of the end effector will be successful if the the control method described in the previous two chapters can position the end effector within 10 mm from the target position for each dimension. This margin is chosen to allow for some fluctuation of the position. Two other control methods are used, both variations of the first control method. They are used to test the second and the third parts of the purpose. At first the experimental setup of all methods is described, then the control methods. After that, the results of the reaching task are presented.

6.1 Experimental Setup

The goal of the reaching task is to move the end effector towards a target in Cartesian space. This is shown schematically in Figure 6.1, where pS is the starting position (the natural resting position) and pT is the target. Five targets were randomly chosen, listed in table 6.1. Each of these targets was tested separately. For each of the targets, three methods were tested as explained below. Each combination of target and method was tested 60 seconds for five times. The pressure was taken off of all muscles between runs, for five seconds, to guarantee the return of arm to its natural resting position. In total, the reaching task consisted of 75 runs ($5 \times 3 \times 5$) of 60 seconds each.

For each time-step, the Mean Squared Error of the distance between the end effector and the moving target was calculated. The Cartesian position of the end effector at the resting position is defined as (0, 0, 0). Some targets are by definition further from the starting point and thus will have a larger MSE at the beginning compared to targets that are closer to the starting point. Therefore, the MSE was only calculated for the last 10 seconds; the position of the end effector should have been converged to the target position by then. At the end, the average of these last 10 seconds of MSE values was calculated. Because each combination is repeated five times, the average of the five averages is used to compare performance between control methods. In the text below, the average of the average MSE is referred to simply as the average MSE.

Table 6.1: *Targets of the reaching task, these are used for pT from Figure 6.1*

Target	X	Y	Z
R1	-65	170	105
R2	-85	30	15
R3	100	45	15
R4	170	0	15
R5	90	185	115

6.2 Jacasm control method

The ASM is used in the Jacasm (Jacobian + Attractor Selection Model) method to see if the normal Jacobian control has local minima and if adding noise can escape these minima. The Jacasm method is the Jacobian method combined with the implementation of the ASM which was discussed in section 2.7. The ASM adds a little bit of noise to the control of the muscles if the performance of the task is bad (see section 6.2.2 below). To use the ASM two decisions need to be made: one for the attractors and one for the activity function.

6.2.1 Attractors design

The ASM has 20 attractors, or weight vectors. Each weight vector has 26 dimensions. The n-th dimension of the vector is the weight for the n-th actuator. Fukuyori et al. (2008) use the attractors as control values, but here they are used as weights for control values. The initial state of the ASM is initialized to random values close to zero. Each of the 26 dimensions of the attractor is initialized to -1, 0 or 1 and then normalized to the sum of these numbers (can be zero, because some values can cancel each other out). This is the same algorithm as used in Fukuyori et al. (2008).

$$u'_n = u_n \cdot (1.0 + Y_n) \tag{6.1}$$

The actuator values are weighed by the output from the ASM as shown in equation 6.1, where u_n is the result from equation 5.3, the updated and decayed actuator value. The output from the ASM is represented by the variable Y. There is a small disadvantage of using this specific equation; since the values of the attractor are in the range [-1, 1], the ASM weights double the range of the actuator values. The decay is applied before the weighing, so theoretically this method could result in actuator values outside the preferred range. However, this would mean that the distance to the goal is not improving (enough), so this will trigger a selection of a different attractor and thus the weights will change. An advantage of using the ASM like this is that the control method will behave exactly the same as the Jacobian control method if the output of the ASM is all zero (recall that the beginning state of the ASM contains values close to zero), so they are compared more easily. Another advantage is that the original implementation of the ASM did not have to be altered and retested.

6.2.2 Activity function design

The activity function is designed to be low (resulting in more muscle weight change) when the the end effector is far from the target *and* if there is little movement, otherwise the activity is high. The activity function is shown in equation 6.2. Here, λ_a is the decay of the activity, A_m is the movement component and A_d is the distance component. The decay (λ_a) is set to 0.01.

$$A(t) = (1 - \lambda_a) \cdot A(t-1) + \lambda_a \cdot (100A_m + 10A_d) \tag{6.2}$$

The movement component is shown in equation 6.3, where p_m is the m-th dimension of the current position.

$$A_m = \sum_{m=1}^{3} |p(t-1)_m - p(t)_m| \tag{6.3}$$

The distance component is defined by a sigmoidal function, so there is no hard cutoff distance. The distance component is shown in equation 6.4, where I is the infliction point of the sigmoid in mm (10), M_d is the margin of the infliction point in mm (3), in other words: a rough area around the infliction point in which the transition of the sigmoid occurs.

$$A_d = \frac{1}{2}T_{hi} + \frac{1}{2}T_{hi} \cdot tanh\left(\frac{I_d - \Delta p}{M_d}\right) \tag{6.4}$$

Finally, Δp is the manhattan distance shown in equation 6.5. The ASM implementation requires the activity value to be between T_{lo} (2) and T_{hi} (100). The distance component yields a number between 100 and 0, depending on the distance. The parameters λ_a, 100, 10, I and M_d were determined by experimenting. Before the activity function is used in the ASM, it is clamped between T_{lo} and T_{hi}.

$$\Delta p = \sum_{m=1}^{3} |\hat{p}_m - p_m| \tag{6.5}$$

6.3 Jacobian selection control method

The Jacobian selection (Jacsel) control method is used to test if the order of muscle weights of the weighted Jacobian matrix is usable to determine virtual muscle pairs. In this method a selection of muscles is made every time frame and only these muscles are changed that time frame.

For each time frame, the actuator deltas are calculated as in equation 5.1. These deltas are ordered from most positive to most negative and the six most negative and the six most positive actuator changes are selected, which means that only these actuators are changed that time frame. The value of six is chosen for the same reason as in the 'linear decay with suppression' control method in section 5.5.2.

Actuators that are not selected have a Δu_n of zero, but all actuators are decayed each time-step. This means the non-selected actuators also change a little bit, but because exponential decay is used, the decay is very low when the actuator values are in their preferred range. The final muscle changes are calculated as the normal Jacobian control method, with equation 5.3.

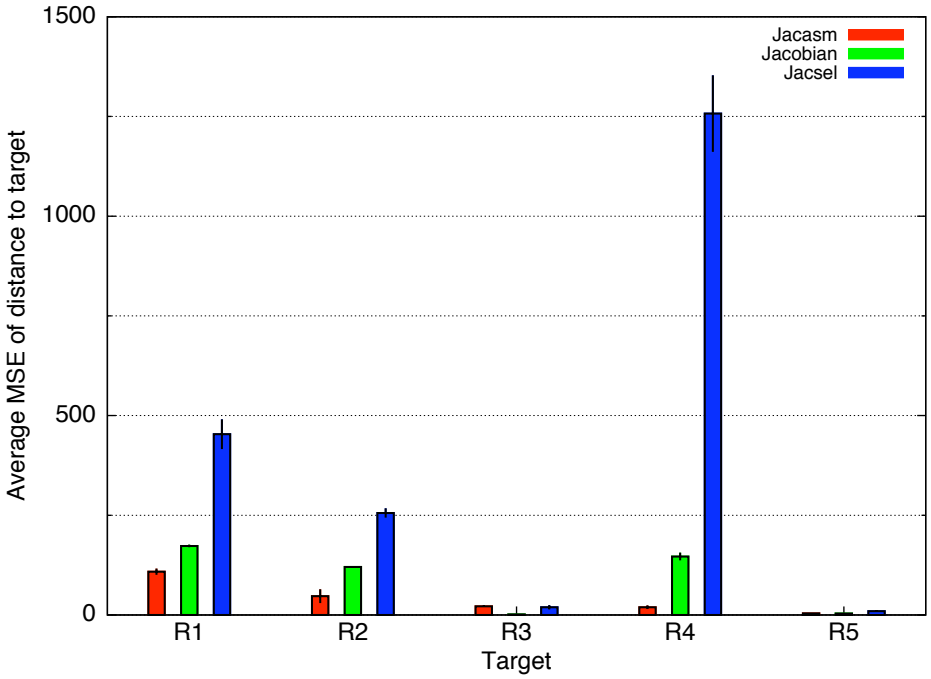


Figure 6.2: *Performance of reaching task methods. The Jacobian method outperforms the Jacsel method, while the Jacasm method outperforms the Jacobian method. An average MSE of around 150 is acceptable.*

6.4 Results

The reaching task was performed for the Jacobian, Jacasm and Jacsel control methods. Figure 6.2 shows the results. The horizontal axis shows the different targets R1 through R5, while the vertical axis shows the average MSE of the distance to the target. For each condition, the average and standard deviation of five runs is shown as error bars.

The average MSE has no clear meaning, except that low values are better than high values. This is useful when comparing results, but not for individual results. To give an idea of an acceptable value, target R4 for the Jacobian method has an average MSE of about 150, and in the final time-step, the distance between the end effector and the target is 7 mm, 6 mm and 9 mm for the X, Y and Z dimension respectively.

For most targets, the Jacsel method performs worse than the Jacobian method while the Jacasm method performs better than Jacobian. Exceptions are targets R3 and R5, where errors are already relatively low. All results are acceptable, except targets R1, R2 and R4 for the Jacsel method. The most remarkable result is the Jacsel method for target R4. The average MSE is out of proportion with respect to the rest, as the other methods for R4 are comparable to those of targets R1 and R2.

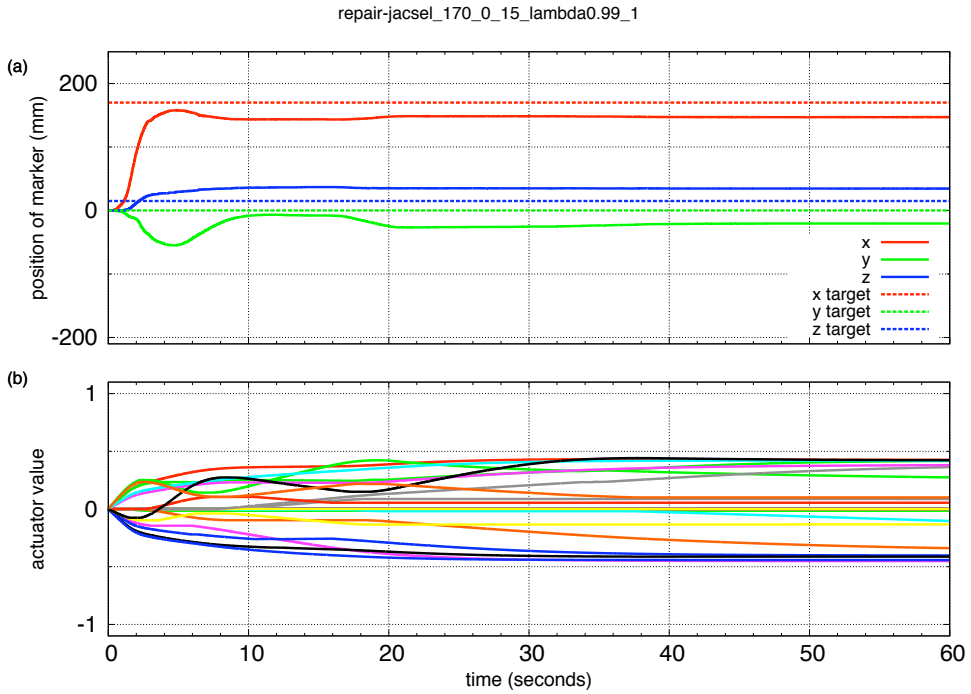


Figure 6.3: *Worst result: Jacsel method for R4 (run 1). After about 18 seconds, the end effector moves further away from the target.*

Figure 6.3 shows the Jacsel method for target R4 in more detail. Something interesting happens between 10 and 20 seconds, where the position looks stabilized at first, but changes suddenly. This was observed in each of the five runs. The sudden change is caused by changes in actuator values to get the X and Z dimensions closer to the target; they have more priority than the Y dimension at that time, because their distance to the target is higher. The distance to the target decreases for the X and Z dimensions, but not for the Y dimension, which increases until in all dimensions the distance to the target is approximately the same.

The distance to the target increased because the wrong muscles were changed. Normally all muscles would change, but the Jacsel method only changes the 6 most positive and the 6 most negative muscles at each time-step. This suggests that either the the muscle weight order is suboptimal or that more muscles are needed to satisfy distance minimization for 3 dimensions simultaneously.

For comparison, a run of the Jacobian method for the same target is shown in Figure 6.4. The actuator values are smaller (they are almost within the range [-0.3, 0.3]), and so is the distance to the target.

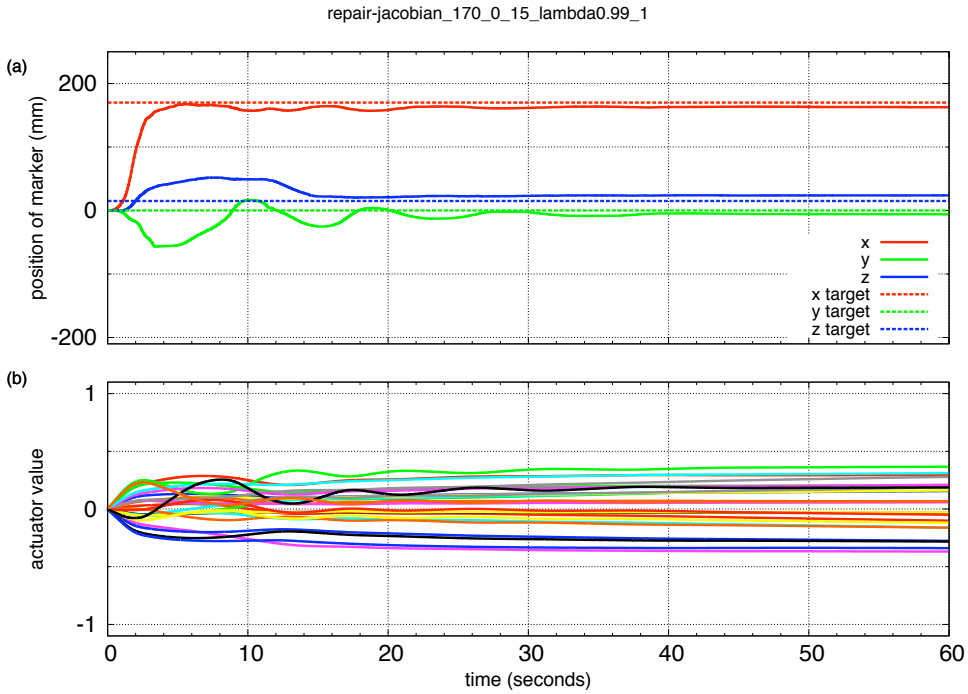


Figure 6.4: *An acceptable result: Jacobian method for R4 (run 1). The end effector stabilizes to a distance close to the target.*

6.5 Discussion

At the beginning of this chapter some purposes for this experiment were defined. We can conclude that we can use the information from the Jacobian matrix to control the robot arm (first purpose) as most targets are quite easily reached. We may also conclude that adding noise to the control improves performance (second purpose) as the Jacasm method improves the performance in almost every condition. The third purpose has not been fulfilled. The Jacsel method can control the arm, but not as good as the Jacobian or Jacasm method. Either the muscle weight order is sub-optimal or more muscles are needed. It is also possible that decay for non-selected muscles caused a lower performance for the Jacsel method. From this we cannot conclude that we can use the muscle weight order to find muscle pairs. In the next chapter, a different way of testing is used.

Chapter 7

Moving Task

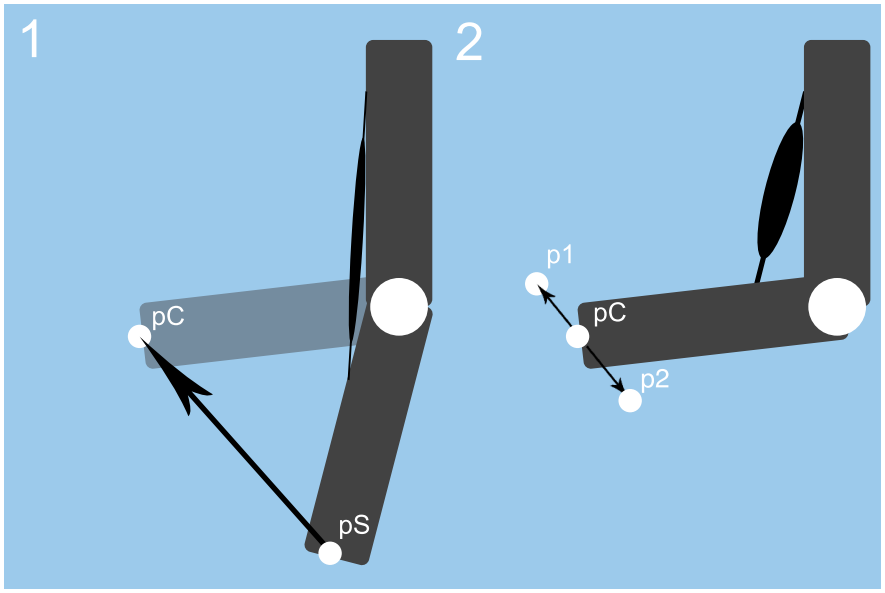


Figure 7.1: *The moving task consist of 2 phases. Phase 1 is similar to the reaching task, bringing the end effector from pS to pC (center point). Phase 2 is to move the arm towards a moving target that moves between p1 and p2. Points p1 and p2 are located to the left and to the right of pC. The movement is realised with only a selection of muscles.*

The previous chapter showed that the weighted Jacobian control can be used to successfully position the end effector of the robot arm. The purpose of this chapter is to show that virtual muscle pairs can be found by using the muscle weight ordering from the weighted Jacobian matrix. The moving task is used to test this. The goal of the *reaching* task was to move the end effector towards a static target, but the goal of the *moving* task is to follow a moving target. The target moves in such a way that a pair of muscle groups with an opposite effect can be used to follow it.

As the ordering of the muscle weight is probably not perfect for each situation (see section 4.7), different sizes of muscle groups are tested. The expectation is that as much as six muscles are needed to successfully track the moving target (see also 5.5.2), but maybe less.

7.1 Experimental Setup

Figure 7.1 schematically shows the moving task in two phases. The first phase of the task (1) is similar to the reaching task; to move from pS, the natural resting position, to pC, the center position. The Jacobian control method discussed in previous chapters is used for the first phase. After 60 seconds (the same time as the reaching task), the robot arm should have been converged to the target position (pC). In the second phase of the task (2), the target position is dynamic; it moves between p1 and p2 with a slow sinusoidal (T = 30 seconds). The amplitude of the sinusoidal is 50 mm.

Between the two phases, a selection of muscles is made, this selection does not change during the second phase. The selection process is similar to the one used in the Jacsel control method, discussed in section 6.3: N muscles are selected to move the arm to p1 and N muscles are selected to move to p2, where N = 2, 3 or 6. A difference with the Jacsel method is that non-selected muscles are not changed at all. These muscles are not even decayed, because they will have to remain in the same position for the duration of the second phase (60 seconds).

In the results section, muscle pairs are discussed in the form of 'muscle pair P'. These are not virtual muscle pairs, but pairs of muscles determined by the order of their weights. For example, muscle pair 1 consists of the muscle which has the highest weight to move towards p1 and the muscle which has the highest weight to move towards p2, muscle pair 2 consists of the second highest muscle weights. When N = 2, muscle pairs 1 and 2 are used and when N = 6, muscle pairs 1 to 6 are used.

The goal of the second phase of the moving task is to move the end effector towards the moving target by using *only* the N selected muscle pairs. The targets p1 and p2 are chosen so that the movement is only in the horizontal plane, thus assuring that the effect due to gravity is similar in both muscle groups. The non-selected muscles are fixed to the value they had at the end of the first phase. To move the end effector to p1, the 'p1' muscles are increased in actuator value and the actuator values of the 'p2' muscles are decreased. To move the end effector to p2, the procedure is reversed.

The decision to move to p1 or p2 is made by comparing the angles between three vectors: current position to moving target ($\vec{v_c}$), current position to p1 ($\vec{v_1}$) and current position to p2 ($\vec{v_2}$). Two angles are calculated, between $\vec{v_c}$ and $\vec{v_1}$ (θ_1) and between $\vec{v_c}$ and $\vec{v_2}$ (θ_2). If θ_1 is smaller than θ_2, p1 is chosen, otherwise p2. The decision to move to p1 or p2 is designed like this so it can be applied for any direction, instead of only for a horizontal one.

For each time-step, the MSE of the distance between the end effector and the moving target is calculated. The average MSE is calculated only for the last 60 seconds because that is the period where the arm is moving sideways by using the selection of muscles. The average MSE is calculated over the distance in the X dimension only, the distance in the Y and Z dimensions is assumed to be zero. The reason the Y and Z dimensions are ignored in the calculation is that they are also ignored during the selection of muscles.

Different targets for pC were tested, as shown in table 7.1. The first target is equal to pS, the starting position for each run. In total the moving task consisted of 60 runs (3 values of $N \times 4$ targets $\times$ 5 runs) of 120 seconds each. As in the reaching task, the muscles are relaxed for five seconds between runs to allow the arm to return to the natural resting position.

Table 7.1: *Center points of the moving task in mm. These are the different values for pC (see Figure 7.1). To calculate the values for p1 and p2, respectively add or subtract 50 to/from the X-coordinate.*

Target	X	Y	Z
M1	0	0	0
M2	0	30	20
M3	0	130	65
M4	0	150	95

7.2 Results

The results of the moving task are shown in Figure 7.2. The horizontal axis shows the center points of the moving targets, according to table 7.1. The different colored bars show the values for N = 2, 3 or 6. The vertical axis shows the average MSE of phase two of the task averaged over five runs. The black error bars at the top of each bar show the standard deviation over five runs. The horizontal line at 1250 shows the average MSE if the end effector would be fixed at the center point, compared with doing nothing at all.

Target M1 is a difficult target for the robot arm, because the center point is at the origin, defined there because it is the natural resting position of the arm at the beginning and also the lowest position the end effector can reach. Moving left or right will automatically also move up, because at M1 the end effector is at its lowest point. Also, because the arm needs to be fully stretched to keep the end effector at the lowest point, less muscles can be used to perform the movement to the left and right. This may explain why target M1 is the only target where N = 6 has a higher average MSE than N = 2 or N = 3, as this is an extreme position at the edge of the end effector workspace, so the ordering of muscles is most likely sub-optimal.

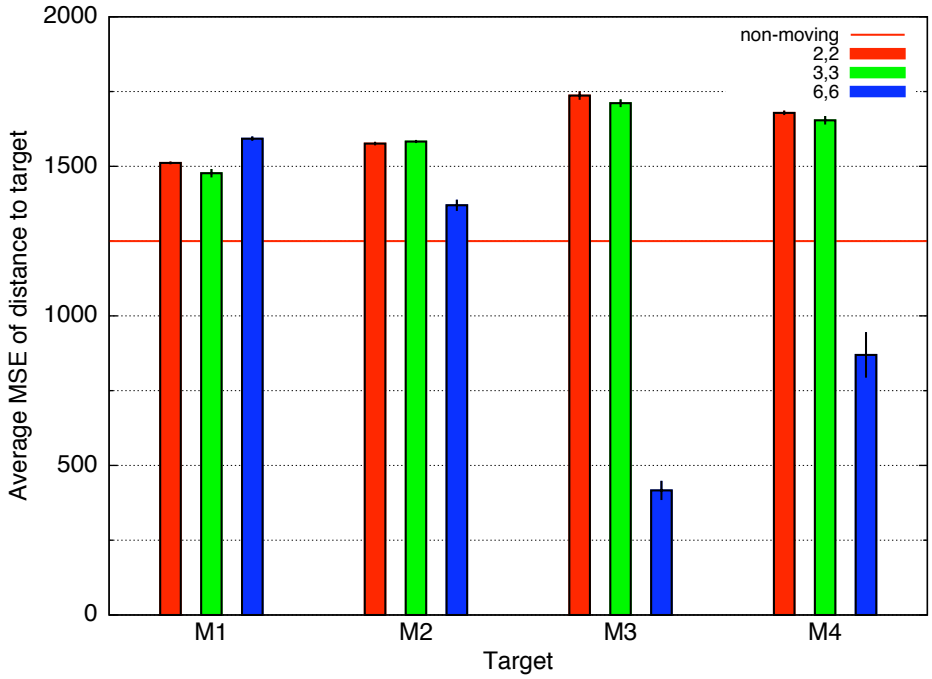


Figure 7.2: *Average MSE of the distance to the moving target, using only 2+2, 3+3 or 6+6 muscles to move. The 'non-moving' line at 1250 represents the average MSE when the end effector remains at pC during the second phase. All but two conditions perform worse than this.*

The most obvious observation is that only two conditions resulted in an average MSE below the horizontal line: target M3, N = 6 and target M4, N = 6. It looks like the performance of the tracking movement increases when more muscles are used. Runs for target M3 are shown below for N = 2, 3 and 6 to further explore what is going on. The images have a similar layout as in section 6.1, with the difference that an extra 60 seconds is shown for the moving task and the image is scaled accordingly.

7.2.1 Target M3

Target M3, N = 2 is shown in Figure 7.3. The position of the end effector (solid lines) is delayed with respect to the target (dashed lines) it was supposed to track. But what causes this delay? First of all, the effect of the two actuators is not that large; even though the actuator values changed almost linearly from 60 seconds through somewhat after 70, the position only changed roughly between 62 and 68 seconds. Maybe this happened because these two actuators were not strong enough to cause more movement, but it can also be that other (opposite) muscles slowed the movement if the first phase ended in a stiff posture; a lot of muscles were activated and they could have been balancing each other out, resulting in a lower net effect. Secondly, because the end effector

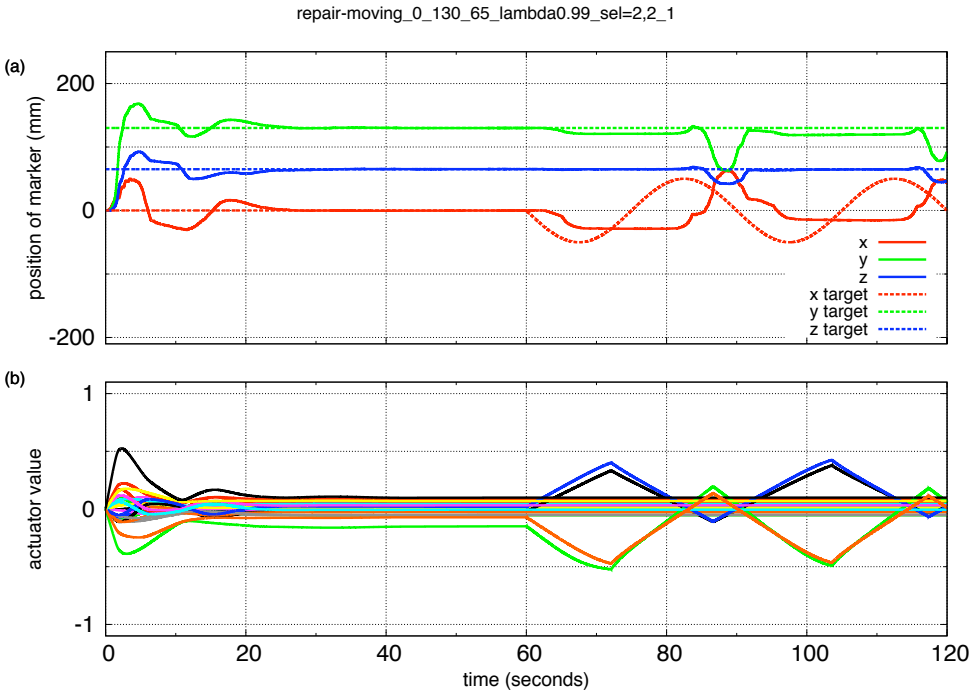


Figure 7.3: *Run 1 of target M3, N=2. Phase 2 starts after 60 seconds, where only 2 + 2 muscles are changed (b). The position of the end effector is shown in (a), where the dotted lines represent the targets. Tracking the moving target in the X dimension is delayed and not accurate, but direction of movement is correct.*

position stopped changing while the actuator values kept increasing (and on the opposite side decreasing), a 'buffer' was created which caused a second delay when the arm tried to move back. The decay that was discussed in 5.5 was supposed to avoid this, but it was assumed that all actuators would decay, which is not the case in the moving task. The delay caused by the buffer is seen slightly after 70 seconds where the direction of the actuator values is reversed; the position of the end effector only changed after the actuator values dropped below a certain value, after around 80 seconds.

There is also a pause in the X coordinate movement around 83 seconds, right at the time where selected muscles all reach zero. This suggests that both sides of the virtual muscle pair cause a positive X movement: the first one by relaxing and the second one by contracting. Because the first one causes movement by relaxing, there must be other muscles that cause an opposite effect and they can not be the muscles from the other side of the pair, because they are relaxed (below zero). This means that N = 2 is probably too low.

The actuators in the other direction caused more displacement so the desired position was reached with smaller actuator values, but the position overshot which caused another delay. Because for each direction the kind

(and magnitude) of delay is different, the shape of the movement of the end effector position is asymmetric. Using an extra actuator on each side, the graph for target M3, N = 3 shows similar results, as shown in Figure 7.4. The extra actuators hardly had any added effect.

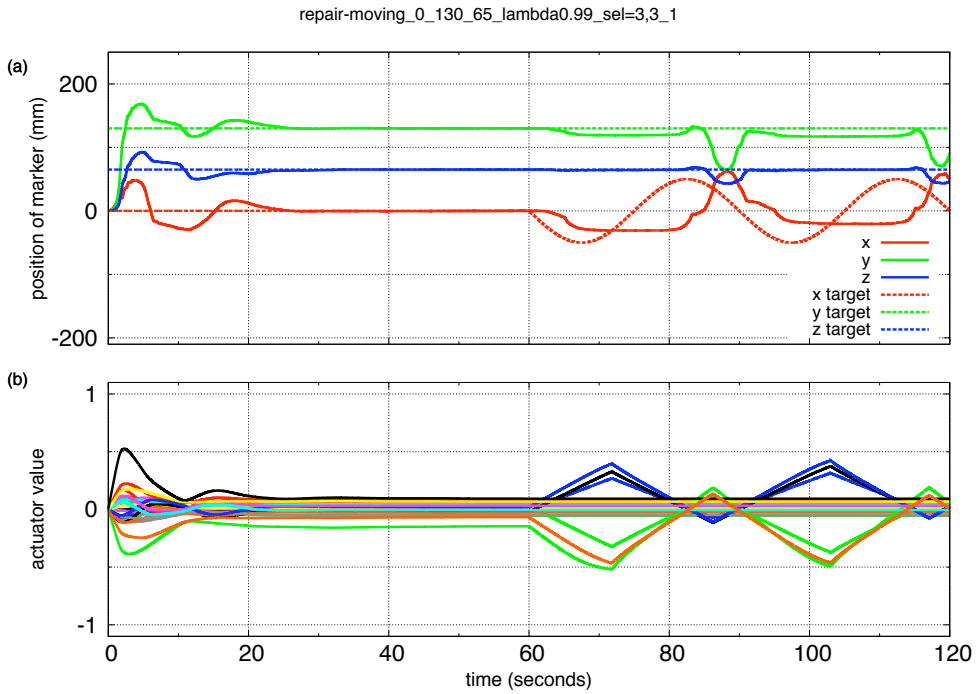


Figure 7.4: *Run 1 of target M3, N=3. Even though 1 + 1 more muscles are used, the effect on the end effector is almost identical to N=2.*

Figure 7.5 shows the graph for target M3, N = 6, using more 3 + 3 more actuators. This figure shows a little bit more overshoot, but the a much smaller delay. The actuator values were less extreme and the actuator values did not increase without also changing the end effector position as was the case with with M3, N = 3. This explains why there was a smaller amount of delay. Despite the delay, the target positions could be fully reached, on both sides (p1 and p2). So muscle pairs 4 to 6 increased the reach when used together with the first 3 muscle pairs. As the second 3 muscle pairs were not tested separately, it is impossible to tell if muscle pairs 4, 5 and 6 have more reach than muscle pairs 1, 2 and 3, because the combined effect may be larger than the separate parts added together. So from *only* looking at the delay, it can not be concluded if the order of the muscle weights is correct.

The difference between N = 2 and N = 3 is very small, while the difference between N = 3 and N = 6 is much larger. If the order of the muscle weights is assumed to be correct, muscle pairs 4, 5 and 6 should have had a smaller extra effect than that of muscle pair 3. As this was not the case, it can be concluded that the order of muscle weights is not perfect in this situation.

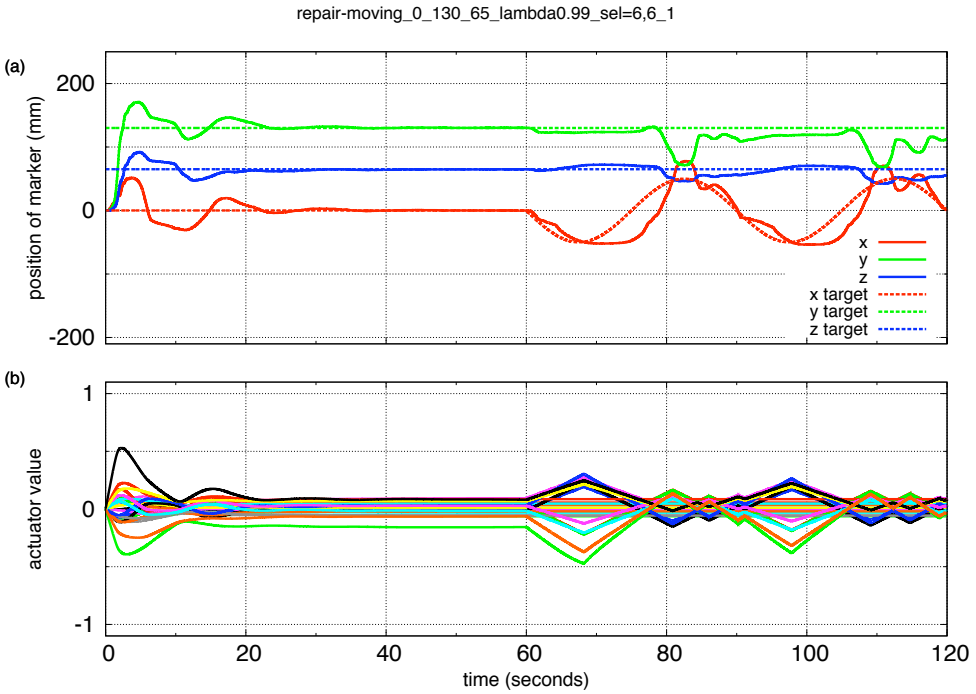


Figure 7.5: *Run 1 of target M3, N=6. Using 3 + 3 more muscles causes less delay and the end effector position is always near the target. The six added muscles have much more additional effect than the two added muscles for N=3.*

In addition to muscle weight order, there is another factor that could have caused the larger difference between N = 3 and N = 6, compared to the difference between N = 2 and N = 3. This factor is relaxation of muscles. In other words, the movement at N = 2 and 3 could have been limited by muscles that were activated at N = 2 and 3, and relaxed at N = 6. Three muscle pairs were added, so three additional single muscles were relaxing when the opposite muscles were contracted.

If this is true, this would show up in the pressure data collected during the experiment. From this data (not shown), it can be seen that of these three muscles (the relaxing muscles of muscle pairs 4 to 6), at least one of them is dropping in pressure. The rest did not change, except for small fluctuations. The muscle that dropped in pressure had a pressure value of 1.5 V at 60 seconds and drops to around 1.1 V at 62 seconds. A pressure value of 1.5 V is about mid-range, as pressure values range from around 1.0 V to around 2.2 V.

Also, the pause in the movement which appeared in N = 2 and N = 3 did not appear in N = 6, so it looks like there are little to no muscles restricting the movement. Therefore, it is likely that these muscles constricted the movement in N = 3 and allowed more movement in N = 6 because they were relaxed.

Because the weighted Jacobian matrix is an average of different postures, we know the ordering of muscles can never be perfect in every posture. The data suggests that for lower values of N, there are more muscles constraining the movement. Therefore the experiment was repeated with even higher values of N.

7.3 Using even more muscles

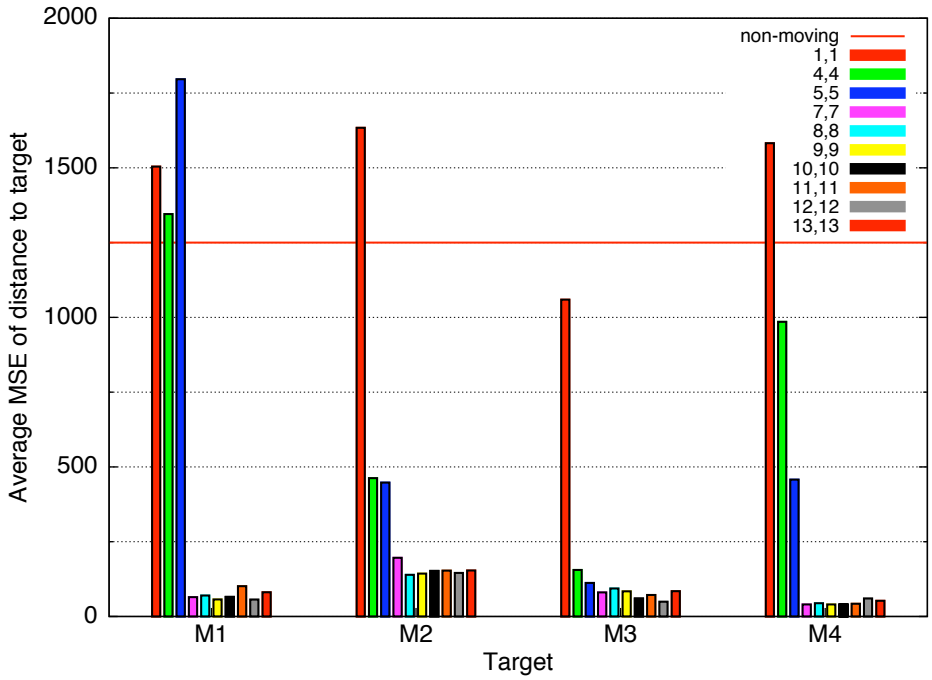


Figure 7.6: *Results for moving task for other values of N. For this experiment only, the MSE is the average of the last 60 seconds of one instead of five runs. Higher values of N (more muscles) result in better performance (except for M1, N=5), until N = 7. For N >= 7, the performance is stable and the maximum average MSE for these values of N is 200 (M2, N=7).*

To see what would happen if more muscles were used, the experiment was repeated with other values of N: 1, 4, 5, 7, 8, 9, 10, 11, 12 and 13. As this was an unscheduled experiment and each separate run took a lot of time, each condition was only tested once, instead of five times. The variations between different runs of the same conditions were small for N = 2, 3 and 6, so this was not considered a big problem. The old values 2, 3 and 6 were omitted from this last experiment because they were already tested. However, it turned out that there was a small change in the robot which caused it te react slightly different, so these conditions are not included in the final figure, because they cannot be compared directly

to the older results. Figure 7.6 shows the results of the final experiment.

The most obvious in this graph is that for all targets from N = 7 and upward, the average MSE is very low. After N = 7, the values are roughly the same. Assuming it costs more energy to use more muscle pairs, N = 7 seems to be an optimal setting for this robot arm. This is true even for target M1, which is a difficult target, as discussed earlier. Another observation is that some conditions have lower average MSE than can be expected compared to Figure 7.2: N = 4, 5 for targets M2 and M3, N = 5 for M4 and N = 1 for M3. Because the robot arm was changed between the experiments, it is unknown if these lower averages were caused by the change, by the different values for N, or both.

7.4 Discussion

It seems that a lot of muscles are needed to correctly create the horizontal movement, at least as much as 7 + 7 out of 26, which is closed to the estimated of 6 + 6. More muscles are needed probably because the task is expressed as a straight line in Cartesian space, but is more complex in joint space and thus also actuator space.

Compared to the non-moving line, the N = 2, 3 and sometimes 6 conditions perform worse. This means that the distance to the moving target is larger than it would have been, had the arm not moved at all. However, when the arm moves, the direction of the movement is correctly predicted. The performance of the prediction of direction is difficult to express in an average MSE, so this was maybe not the best way to test the performance of the muscle pairing. However, the plots of the individual runs show that the direction of movement is always correct, even if the amount of movement is not always enough.

The order of the muscle weights is not always optimal for the targets tested. This can be seen from the differences between N = 2, 3 and 6. This was expected, since our weighted Jacobian matrix is an average of different postures, so it can not be optimal for each posture. Maybe this can be mitigated by using the ASM here as well.

Relaxation of muscles that have an opposite effect could be important as well, but we did not further explore this. Whether the problem of the lower performance is caused by sub-optimal ordering of muscle weights, or by not relaxing muscles that have an opposite effect, or by the complexity of the task in actuator space, the problem is less prominent when more muscles are used.

Chapter 8

Conclusion

A weighted Jacobian matrix was successfully used to learn virtual muscle pairs for a humanlike robotic arm with a muscle-based system. The effects of each muscle was learned, weighted and stored in a weighted Jacobian matrix. The muscles were ordered by average displacement, and a selection of muscles were made. The selection of muscles was used to find pairs of groups of muscles that form virtual muscle pairs. The ordering of muscles is not optimal in every situation, this is caused by the approximation properties of the weighted Jacobian matrix. Despite imperfect ordering, the system was still able to learn virtual muscle pairs as shown by opposite movements in the moving task.

To make the desired movement in the moving task, more muscles were needed than expected. Using only two muscles for one direction and two muscles for the opposite direction already resulted in a movement in the right direction, but the range and accuracy increased as more muscles were used. An optimal value seems to be seven muscles for one direction and seven for the opposite direction. This is more than half of the total number of muscles, which is 26. The optimal value of 14 is probably this high because the moving task was expressed as a straight line in Cartesian space among one of the axes of the coordinate system, which is not the case in actuator space. This is not only because there are more DOF in actuator space, but also because the actuators directly affect joint angles which results in non-linear Cartesian changes and because the contraction of the actuators is also non-linear. Even though more muscles were needed to perform the task accurately, the high dimensionality of the input space and the non-linearity were not an obstacle to learning virtual muscle pairs with a Jacobian matrix.

Adding noise with the Attractor Selection Model improved performance in the reaching task. Even without the ASM, the positioning was already very accurate; for an average MSE of 150, the distance to the target in the last time-frame (when the position was stable for more than 10 seconds) was less than 10 mm for each dimension.

8.1 Future work

During this project, a number of ideas surfaced for future research. Some of the more interesting ideas are presented here.

8.1.1 Combine training and testing phase

The system presented in this thesis has a distinct training and testing phase. In the testing phase, the control will not adapt to, for example, a muscle break (or more likely, the cable that connects the muscle to the frame). If the Jacobian matrix is (re)learned during actual movement instead of during a separate training phase, it will be able to adapt to such circumstances.

This is impossible with the current system without making a lot of changes, because it continuously makes small changes to the actuator value. The effect of each change has a delay, so effects of changes overlap with effects of previous changes, thus masking the effect of each change.

Humans move limbs by sending commands that move the arm over a period of tens of milliseconds. Once they have given a command, they cannot cancel the command but they have to give a command with opposing effect that overlaps the first to counter the effect of the first. By moving the arm jerkily instead of continuously, there is less overlap, so it is even more easy to correlate actions with sensor input, making the system look even more like a "primary circular-reaction" (Baldwin, 1894; Piaget, 1952) system. This allows the system to learn from a changing environment and eliminates the need for a separate training phase.

8.1.2 Posture dependent Jacobian matrix

Because the physical system used in this thesis has a lot of overlap and constricted movement, one Jacobian matrix was enough to approximate the effects of each actuator on the end effector. In a more complex system, multiple Jacobian matrices can be used instead of one weighted Jacobian matrix. The Jacobian matrix that was taken from the posture that is closest to the current posture (or maybe a interpolation between the closest K postures) will probably work better because the Jacobian matrix is local instead of global.

This idea can be combined with the previous idea so the system does not have to learn every posture before it can use the information to move the arm.

Posture dependent Jacobian matrix can be further expanded to include acceleration awareness. The current model of the robot arm is not mobile, but if it was, acceleration sensors could be added to store Jacobian matrices depending on the output of the acceleration sensors. Depending on the current output of the acceleration sensors, the Jacobian matrix with the closest match can be selected. This is useful if the robot has to manipulate objects under different circumstances: while standing up, lying down or while moving.

8.1.3 Dynamic actuator values

At the moment, all actuators use the same range ([2.5, 5.5] V), while the maximum pressure of each muscle is different. We suspect that not the whole range of the muscles is used. Also, it looks like the range of the muscles differs depending on the type (length or positioning) of the muscle. Maybe the range of each muscle should be determined dynamically and scaled to the actuator values range [-1, 1]. This would use the available muscle power more efficiently.

8.1.4 Task dependent control

With virtual muscle pairs found, they can be used to control for example compliance. For different tasks the focus is on different types of control. For example, to pick up raw eggs and put them in a basket, the hand should first move towards the egg, so the focus of control is on positioning. The weighted Jacobian matrix or similar method can be used for the positioning. To actually pick up and hold the egg, the focus of control is on compliance, because the egg must not be crushed. To control the compliance, the virtual muscle pairs can be used. To put the egg in a basket, the focus of control is on positioning and navigation, because the hand must enter the basket from above (where the opening is), otherwise the hand will crush the raw egg against the basket. Compliance is also still needed in this case, to not crush the egg.

Somehow, these different factors of control should be implemented in a system that can give priority (or focus) to a specific kind of control, while not completely ignoring other factors of control, as compliance in the last step of the example. A simple selection between different kinds of controls will not suffice.

8.1.5 Looking for tension

Another way to find virtual muscle pairs, or possibly antagonistic muscle pairs is to look at the actual pressure values of the actuators. If an antagonist muscle is contracted, while other muscles are not changed, it will probably influence the pressure on other muscles, hopefully more on the agonist muscle than all the others.

Acknowledgements

This thesis was made possible by the FrontierLab@OsakaU program that enables students from outside Japan to study at the University of Osaka.[1] In no particular order, I would also like to thank some persons for helping me with this thesis. Rineke Verbrugge supplied me with a very nice recommendation letter on a short notice, while she was very busy herself. Sugahara Atsushi-san explained a lot about how to program the robot arm and showed me how to repair it when it was broken. Rick van de Zedde supplied advice on the whole research project process. My supervisor at my home university: Lambert Schomaker and of course my supervisors at the University of Osaka: Hiroshi Ishiguro, Yoshio Matsumoto and especially Yutaka Nakamura gave me advice and proved me with this great opportunity. My family and friends at home kept in touch despite the large time difference. Gertjan Haaijer, Jean-Paul van Oosten and Joost Vunderink proof-read early versions of my thesis and provided me with invaluable comments. Last but not least my friends in Japan, especially Gyuhee-san, Murakami-kun, Van-kun, Stine, Martin and Eduardo, made my experience not only academic, but fun as well. Thank you all!

[1] http://www.osaka-u.ac.jp/jp/international/iab/e/FrontierLab.html

Bibliography

Baldwin, J. (1894). Imitation: A chapter in the natural history of consciousness. *Mind*, *3*(9), 26–55.

Bangsbo, J., Madsen, K., Kiens, B., & Richter, E. (1996). Effect of muscle acidity on muscle metabolism and fatigue during intense exercise in man. *Journal of Physiology*, *495*(2), 58–298.

Calancie, B., Needham-Shropshire, B., Jacobs, P., Willer, K., Zych, G., & Green, B. (1994). Involuntary stepping after chronic spinal cord injury: evidence for a central rhythm generator for locomotion in man. *Brain*, *117*(5), 1143.

Cowdry, E. (1979). *Problems of ageing*. Arno Press.

Daerden, F., & Lefeber, D. (2002). Pneumatic artificial muscles: actuators for robotics and automation. *European journal of mechanical and environmental engineering*, *47*(1), 11–21.

De Luca, C., & Mambrito, B. (1987). Voluntary control of motor units in human antagonist muscles: coactivation and reciprocal activation. *Journal of neurophysiology*, *58*(3), 525.

Franklin, D., Burdet, E., Peng Tee, K., Osu, R., Chew, C., Milner, T., & Kawato, M. (2008). CNS Learns Stable, Accurate, and Efficient Movements Using a Simple Algorithm. *Journal of Neuroscience*, *28*(44), 11165.

Franklin, D., Liaw, G., Milner, T., Osu, R., Burdet, E., & Kawato, M. (2007). Endpoint Stiffness of the Arm Is Directionally Tuned to Instability in the Environment. *Journal of Neuroscience*, *27*(29), 7705.

Fukuyori, I., Nakamura, Y., Matsumoto, Y., & Ishiguro, H. (2008). Flexible Control Mechanism for Multi-DOF Robotic Arm Based on Biological Fluctuation. *Lecture Notes in Computer Science*, *5040*, 22–31.

Georgopoulos, A., Kalaska, J., Caminiti, R., & Massey, J. (1982). On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. *Journal of Neuroscience*, *2*(11), 1527.

Gray, H. (1918). *Anatomy of the human body*. Lea Febiger.

Hannaford, B., & Winters, J. (1990). Actuator properties and movement control: biological and technological models. *Multiple Muscle Systems: Biomechanics and Movement Organization*, (pp. 101–120).

Hirose, R., & Takenaka, T. (2001). Development of the humanoid robot ASIMO. *Honda R&D Technical Review*, *13*(1).

Humphrey, D., & Reed, D. (1983). Separate cortical systems for control of joint movement and joint stiffness: reciprocal activation and coactivation of antagonist muscles. *Adv Neurol*, *39*, 347–372.

Kent, A., & Williams, J. (1998). *Encyclopedia of computer science and technology*. CRC.

Klute, G., & Hannaford, B. (2000). Accounting for elastic energy storage in mckibben artificial muscle actuators. *Journal of dynamic systems, measurement, and control*, *122*(2), 386–388.

Kots, Y. (1969). Supraspinal control of spinal centers for antagonist muscles in man. *Neuroscience and Behavioral Physiology*, *3*(1), 1–6.

Kuperstein, M. (1988). Neural model of adaptive hand-eye coordination for single postures. *Science*, *239*(4845), 1308–1311.

Liu, D., & Todorov, E. (2009). Hierarchical optimal control of a 7-dof arm model. In *2009 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning*.

Lloyd, D. (1941). A direct central inhibitory action of dromically conducted impulses. *Journal of Neurophysiology*, *4*(2), 184–190.

Matsui, D., Minato, T., MacDorman, K., & Ishiguro, H. (2005). Generating natural motion in an android by mapping human motion. *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, (pp. 3301–3308).

Morasso, P. (1981). Spatial control of arm movements. *Experimental brain research*, *42*(2), 223–227.

Morasso, P., & Sanguineti, V. (1995). Self-organizing body schema for motor planning. *Journal of motor behavior*, *27*(1), 52–66.

Nakazawa, A., Nakaoka, S., Ikeuchi, K., & Yokoi, K. (2002). Imitating human dance motions through motion structure analysis. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, vol. 3.

Piaget, J. (1952). *The origins of intelligence in children (trans. M. Cook)*. New York: International Universities Press.

Riley, M., Ude, A., & Atkeson, C. (2000). Methods for motion generation and interaction with a humanoid robot: Case studies of dancing and catching. In *AAAI and CMU Workshop on Interactive Robotics and Entertainment 2000*.

Schomaker, L. (1991). Simulation and recognition of handwriting movements: A vertical approach to modeling human motor behaviour. *Doctoral Dissertation*.

Schulte, H. (1961). The characteristics of the mckibben artificial muscle. *The application of external power in prosthetics and orthotics*, (pp. 94–115).

van der Smagt, P., Grebenstein, M., Urbanek, H., Fligge, N., Strohmayr, M., Stillfried, G., Parrish, J., & Gustus, A. (2009). Robotics of human movements. *Journal of Physiology-Paris*.

Yanagida, T., Ueda, M., Murata, T., Esaki, S., & Ishii, Y. (2007). Brownian motion, fluctuation and life. *Biosystems*, *88*(3), 228–242.

Appendix A

Linear decay

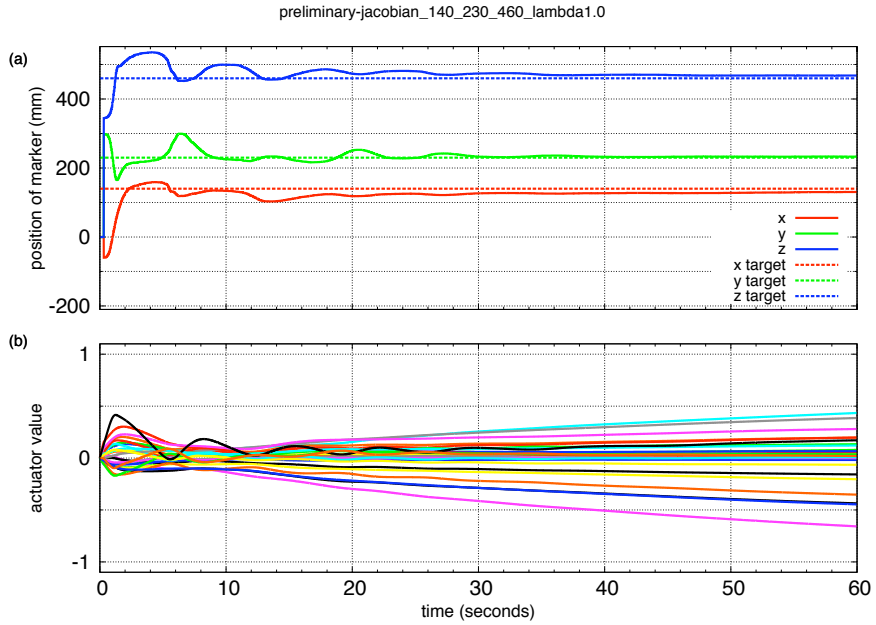


Figure A.1: *Effect of decay on actuator values, $\lambda = 0.0$ (no decay). In this situation, the actuator values (b) strongly diverge over time, while the position (a) stays stable.*

This appendix explains the decision for the parameter λ for linear decay, described in section 5.5. Different values are tested: 0.0 (no decay), 0.0001, 0.001 and 0.01. These are shown respectively in Figures A.1 through A.4[1]. All figures show the Jacobian method trying to reach a 3D Cartesian target (140, 230, 460) with the different decay parameter. The time is shown on the horizontal axis in seconds. The bottom half (b)

[1]The λ values in the title of the images are actually (1 - λ), this is because in the implementation the meaning of λ was reversed, but for this explanation the other meaning is used because it is more intuitive.

of each figure shows the actuator values, the top half (a) of each figure shows the position of the end effector in solid lines and the target values in dotted lines.

The individual actuator values are not important, so they are not listed in the legend. If there is no decay (A.1), the actuator values clearly diverge from zero, while the end effector does not come noticeably closer to the intended target. In other words, increasing the actuator values (moving away from zero) beyond some point does not help to get closer to the target, but it will make it more difficult to respond to changes, because the values are further away from zero.

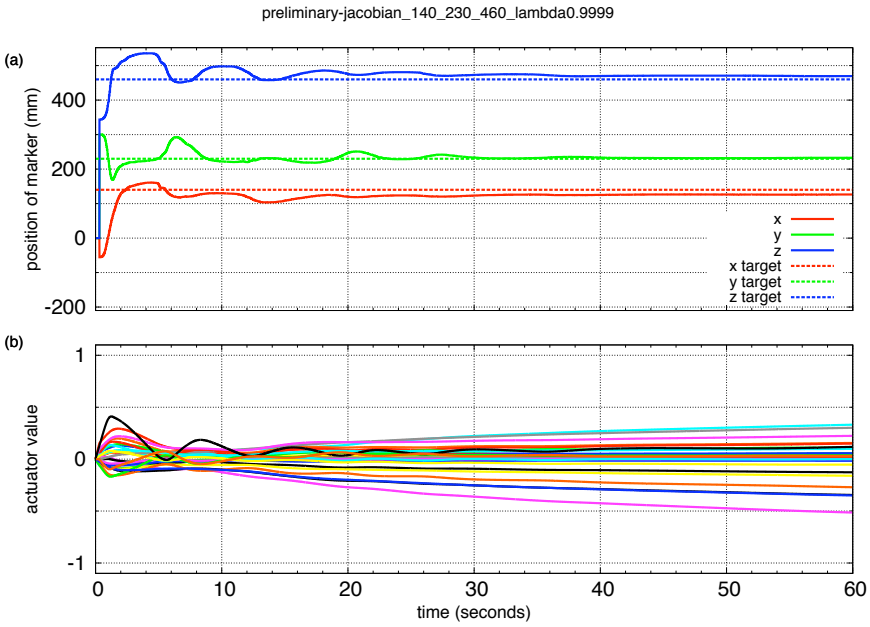


Figure A.2: *Effect of decay on actuator values, $\lambda = 0.0001$. The actuator values (b) are almost limited to the range [-0.5, 0.5], but they still diverge over time while the position (b) stays stable.*

In Figure A.2 (λ = 0.0001), the values also diverge from zero, while the end effector is almost the same distance from the target as with no delay. The actuator values are almost limited to the range [-0.5, 0.5] but because the actuator values are still diverging, it is likely that they will not be contained if the experiment was repeated for a time longer than 60 seconds.

Figure A.3 (λ = 0.001) shows that the actuator values are not that divergent, they seem to stabilize. A small oscillation is visible in both the actuator values and in the position of the end effector during this time. This oscillation seems to decrease in amplitude over time. The end effector is approaching the target position almost as close as in Figure A.1, but is slightly further away from the target.

Figure A.4 (λ = 0.01) clearly shows that the actuator values are stabilizing, but it is also clear that the end effector does not come as close to

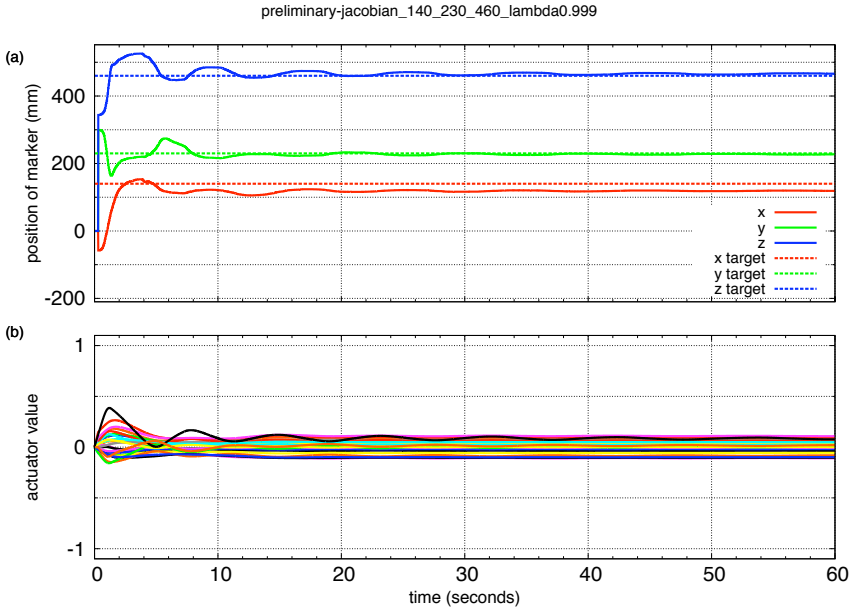


Figure A.3: *Effect of decay on actuator values, $\lambda = 0.001$. The actuator values (b) stabilize and stay well within the range [-0.3, 0.3]. The position (a) is a little but further from the target, but is still close.*

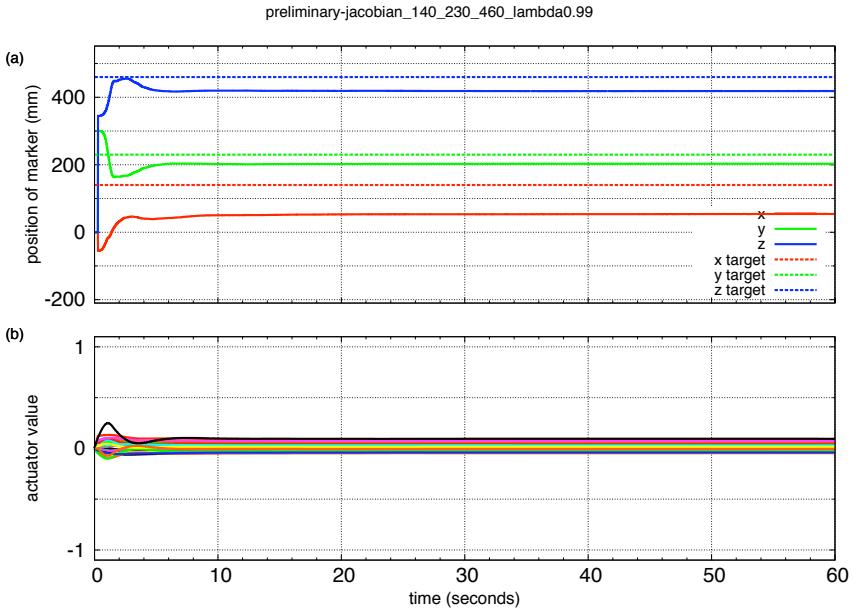


Figure A.4: *Effect of decay on actuator values, $\lambda = 0.01$. The actuator values (b) stabilize, but stay very close to zero. The position (a) also stabilizes, but is too far away from the target.*

the target as for previous values of λ.

Since the decay parameter λ is a tradeoff between target attainability and minimizing of actuator values, the value of 0.001 (Figure A.3) is chosen because the actuator values do not diverge while the end effector moves very close to the target position.

Figure A.5 shows the result of an early reaching task experiment. The vertical axis shows the average MSE of the last 10 seconds, the left bar of each target shows the parameter λ set to 0.001 and the right bar shows λ set to 0 (no decay). This shows the tradeoff between target attainability and minimizing of actuator values; the performance difference between targets is smaller when using decay.

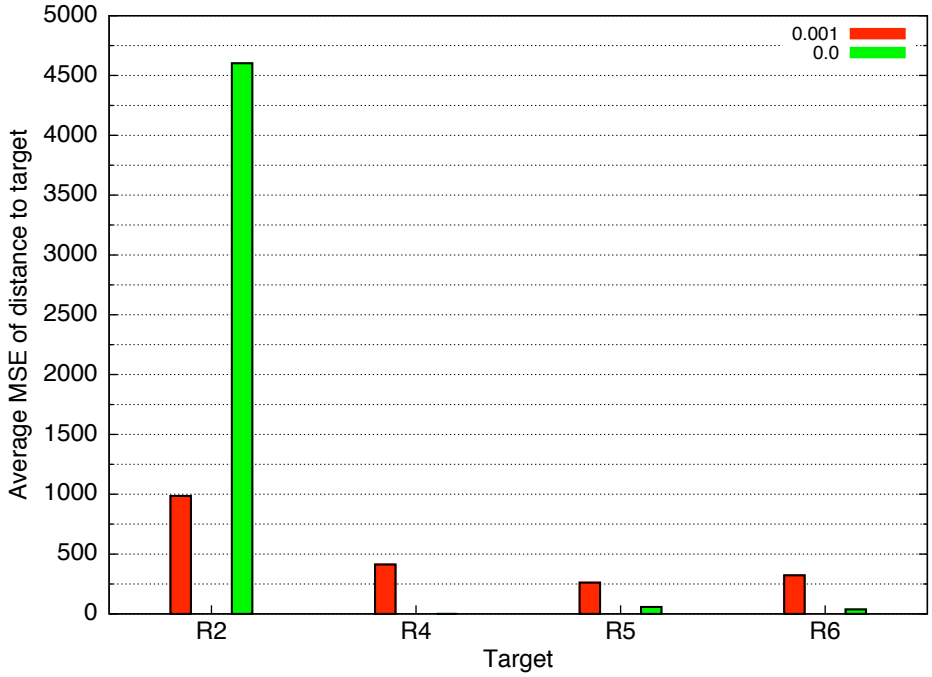


Figure A.5: *Performance of reaching task with $\lambda = 0.001$ and $\lambda = 0.0$. Using a small decay gives a more consistent result than using no decay at all.*