*Technical Report*

UCAM-CL-TR-241
ISSN 1476-2986

Number 241

UNIVERSITY OF CAMBRIDGE

**Computer Laboratory**

# Learning in large state spaces with an application to biped robot walking

Thomas Ulrich Vogel

December 1991

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

*https://www.cl.cam.ac.uk/*

This technical report is based on a dissertation submitted November 1991 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Wolfson College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

*https://www.cl.cam.ac.uk/techreports/*

ISSN 1476-2986

## Thanks

I am very grateful for the friendship and support from all members of the Computer Laboratory of the University of Cambridge. William Clocksin supervised my work and gave me the freedom to do what I wanted to do. Andrew Moore and Thomas Clarke read various drafts of this thesis and helped me to clarify my thoughts in numerous discussions. Barney Pell, John Bradshaw, Julia Galliers and Karl MacDorman read the thesis and provided useful advice. My office mates Prof. Mike Stanton, Mark Lomas, Rajeev Goré, Innes Ferguson and Feng Huang provided a friendly and stimulating environment.

Thanks a lot to Sylvia Draper, Martin Coen, Innes Ferguson, Jonathan Friday, Peter George, Paul Jardetzky, Mark Lomas, Cormac Sreenan, and Frank van Diggelen, who waded through my English, corrected numerous typos and battled against the German in my phrases.

Work on this thesis was supported by the German Academic Exchange Service, the British Council, Wolfson College, and the Computer Laboratory of the University of Cambridge. I am grateful to all these institutions for their support. Special thanks to Prof. Roger Needham.

Many thanks to all my friends at 1 Dean Drive. Sitting together in our garden in Cambridge and watching an apple fall from a tree made me realize that I couldn't ask for a more stimulating environment.

Ich danke meinen Eltern und meinen Schwestern für die Unterstützung, die ich während all der Zeit und überhaupt erfahren habe. Family is a great thing indeed.

This thesis is dedicated to my late grandmother, Wilhemine Jürgensmeier.

## Declaration

I hereby declare that this dissertation is the result of my own work and contains nothing which is an outcome of work done in collaboration. No part of this dissertation has already been or is currently being submitted for any degree, diploma or other qualification at any other university.

# Contents

## Abstract

Autonomous robots must be able to operate in complex, obstacle cluttered environments. To do this the robots must be able to focus on the important aspects of their environment, create basic strategies to carry out their operations, generalise these strategies and finally learn from successful experiences.

Based on simulated dynamic biped robot walking, this thesis investigates these issues. An algorithm is given which analyses the state space of the robot and orders the dimensions of the state space by their importance relative to the task of the robot. Using this analysis of its state space, the robot is able to generate a set of macros (gaits) which enable it to operate in its immediate environment. We then present a control algorithm which allows the robot to control the execution of its gaits.

Once the robot has learned to walk on an obstacle-free horizontal surface, it uses its knowledge about gaits in order to derive obstacle crossing gaits from existing gaits. A strategy based on the qualitative equivalence between two behaviours is introduced in order to derive new behavioural patterns from previous ones. This enables the robot to reason about its actions at a higher level of abstraction. This facilitates the transfer and adaptation of existing knowledge to new situations. As a result, the robot is able to derive stepping over an obstacle from stepping on a horizontal surface.

Finally the robot analyses its successful obstacle crossings in order to generate a generic obstacle crossing strategy. The concept of a virtual evaluation function is introduced in order to describe how the robot has to change its search strategy in order to search successfully for obstacle crossing behaviours. This is done by comparing how the successful obstacle crossing of the robot differs from its normal behaviour. By analysing and operationalising these differences, the robot acquires the capability to overcome previously unencountered obstacles. The robot's obstacle crossing capabilities are demonstrated by letting the robot walk across randomly generated obstacle combinations.

# Chapter 1

# Introduction

Human and animal walking have been the focus of scientific interest for more than 100 years, and while great advances have been made they are still not understood completely. One of the greatest mysteries is exactly how walking is controlled, and furthermore, how it is learned. Recent advances in robotics and artificial intelligence have made it possible to formulate computational models which describe how a robot can *learn* to perform a task such as walking, without resorting to complex mathematical models as used in control theory. In this thesis algorithms will be presented which enable an autonomous dynamic biped robot to learn how to walk and negotiate simple obstacles.

## 1.1 The Problem

The desire to build autonomous robots poses a set of challenging problems for research in artificial intelligence. Autonomous robots are, on a physical level, complex dynamic systems which are hard to describe using a mathematical model. Their autonomy is based on their ability to interact with their environment and to learn from environmental responses. They must be able to learn from payoffs without the assistance of a teacher and their learning must not lead to catastrophic failures. The interpretation of environmental data relies heavily on robust and reliable sensor information.

Clearly many of these problems are still research topics, far from immediately becoming usable in everyday industrial applications. The complexity of each of these problems also makes it impossible to consider all of them at the same time. We shall therefore concentrate on some aspects of the control of such a robot, assuming from the start that the sensor data are available and sufficiently robust. This still leaves us with many open questions.

Robot systems represent complex dynamic systems. Standard control theory is only usable if there is a mathematical model at hand from which a control strategy can be derived. This is generally only the case if the system can be linearised, its dynamics are understood, and the system parameters do not change drastically over time. There are many instances where this is not the case: unknown properties of the physical system can make it impossible to calculate a mathematical model, system characteristics can change due to wear and tear, and an unknown environment can make it necessary to develop activities that go beyond simple physical stability.

## 1.2 Learning to Control a Dynamic System

In this thesis we look at simple ways to learn to control complex dynamic systems. We consider a *simulated dynamic biped robot* in an environment full of information, both useful and irrelevant. We make the assumption that any sensor data required are directly accessible and do not have to be deduced. Indeed, we assume that there is so much information that much of it is redundant or irrelevant. The robot inhabits an environment that also contains many obstacles, and we will expect the robot to *learn* how to walk successfully in such an environment.

In order to achieve this, several problems have to be solved by the robot. First, it has to be able to focus on parameters in the state space that are immediately relevant. Irrelevant and redundant information has to be ignored. Using this reduced version of the state space the robot has to learn how to accomplish its task in simple, ordinary situations. For the biped robot this means walking on a planar surface. The result will be basic behavioural patterns or sequences of motor activities, which in the case of walking we will call *gaits*.

We assume that increasingly more complex situations can be negotiated by the robot by using modifications to solutions for less complicated situations. Thus, the robot modifies the gaits it has learned so far when it comes across an obstacle. Using these modified gaits it is then able to cross this obstacle.

Having successfully crossed an obstacle by using a modified gait the robot must analyse the gait modification to determine why it was successful. The robot does this by analysing the difference between the successful modified behaviour and the unsuccessful standard behaviour. Using this understanding of where to change the robot's behaviour and how to change it, we expect the robot to be able to cross previously unencountered obstacles.

## 1.3 Contributions

The goal of this thesis is to develop techniques for the learning of complex, incremental behaviours in a large state space. This thesis makes the following contributions to the understanding of artificial intelligence-based control of dynamic systems:

- An efficient algorithm to *structure large state spaces* is presented. Given a robot operating in a large state space the algorithm orders the parameters depending on their impact on the robot's ability to survive. The algorithm produces a hierarchy of parameters, ranging from those which always influence the performance of the robot down to those considered irrelevant to the robot. This enables a search program to consider only those parameters which are relevant to the current problem. Irrelevant parameters are detected and ignored. Additionally this technique produces a weighting of the dimensions of the state space which can directly be used to search the state space.
- A program has been written that was able to *learn to control dynamic biped walking.*
- The search for qualitative equivalence between gaits leads to a very simple and efficient set of heuristics which allows us to modify gaits systematically to create new, incremental behavioural patterns. This provides a systematic way to increase the complexity of regular behavioural patterns.
- A representational mechanism is introduced which allows the comparison of different behaviours. The result is a description of when and how a successful non-standard behaviour differed from an unsuccessful standard behaviour. Thus, it becomes possible to extract the essential differences between two similar behaviours.
- Putting all these results together a program has been developed and implemented that enabled a dynamic biped to learn to walk on an obstacle-cluttered plane.

## 1.4 Structure of the Dissertation

Chapter 2 discusses relevant previous work before Chapter 3 introduces the main ideas of this thesis: whenever the simulated dynamic biped robot (described in Chapter 4) finds itself in a new and complex environment, it has to be able to select a strategy that enables it to survive in this environment. This can be done by selecting the most important dimensions describing the environment (Chapter 5) and then searching for basic activities (walking on an obstacle free horizontal surface) which enable the robot to ensure its immediate survival based on this reduced representation of its environment (Chapter 6). These basic activities are then improved and a controller for their execution is developed (Chapter 7). Using systematic incremental modifications of these basic activities (Chapter 8) enables the robot to generate new activities which allow it to deal with more complex situations like obstacle crossings. Finally the ability to generalise and test obstacle crossing strategies (Chapter 9) allows the robot to deal with an environment of increasing complexity. As a result a generic capability to cross random obstacles can be demonstrated (Chapter 10). Chapter 11 concludes and points out future work.

# Chapter 2

# Learning Complex Behaviour: The Issues and Related Work

## 2.1 Learning Complex Behaviour: The Issues

Imagine an autonomous robot in a complex environment. The robot will be given some sort of *goal* to achieve, a *state space* describing the environment and the robot, a set of *actions* which the robot can perform, and sensor data describing the effect of these actions - the *feedback*. The robot *searches* or *plans* complex activities in order to obtain its goals. The robot *monitors* the *execution* of these plans and *adapts* to a changing environment.

This scenario is very general, and to treat it comprehensively would almost be equivalent to finding the *Holy Grail* of artificial intelligence. Therefore it will be necessary to give a more precise definition of the sort of scenario which will be dealt with in this thesis.

The following listing will take a look at the individual tasks of an autonomous robot and describe how these tasks are treated in this thesis. We will look at autonomous robots from an artificial intelligence point of view, which means that we will not be concerned with engineering aspects such as mechanics and sensor systems or communication and real time computing questions. That is, we assume that we already have a robot where such questions are solved, and we will concentrate on the development of a set of basic behaviours (also called operators) which will enable the robot to survive in an unknown environment. The robot will do this in about the same way in which we would expect an animal to learn to survive in its environment. Therefore we will also exclude higher level "intelligent" behaviour which uses a lot of domain knowledge and sophisticated reasoning and planning algorithms.

## The autonomous robot problem

- **Goal:** In this thesis we will look at robot motion planning and control, and apply the results to *dynamic biped walking*. The goal of the biped robot is to be able to "survive" in increasingly complex situations. The final goal will be to be able to cross an obstacle littered surface.

- **State Space:** We assume that the robot has an abundance of processed sensor data and environmental information available. In fact, we assume that any directly measurable environmental data which the robot might need at some point are readily available. Thus the problem is to *identify* the important parameters and to discard all other redundant or irrelevant parameters.

- **Operators:** Originally the robot is only able to send simple signals (like *on, off* or *force* to its motors. These simple signals will be the operators which allow the robot to change from one state to another. Later the robot will generate macro operators which describe entire behaviours like a step forward.

- **Search:** The robot is given a search procedure which is reasonably powerful at achieving what is expected from it. We will allow the use of *domain specific knowledge* in the search in order to guarantee the necessary power of the search algorithm. Domain specific knowledge refers to some problem specific heuristic which will allow the robot to search more efficiently. An example of domain specific knowledge for a bicycle riding robot would be to tell it to pedal. The use of domain specific knowledge appears to be justified since we do not expect the robot to detect "riding on a bicycle" as such, instead we want the robot to learn *how* to ride on a bike. (However it has to be mentioned that this domain specific knowledge is not necessarily easy to encode: for instance one could imagine learning to shift gears in a car, based on an instruction manual.)

- **Planning:** In this thesis we will look at planning from two different angles. Initially we will regard planning as *macro generation*. This will enable the robot to develop small repeatable activities. We will also include a domain specific planning module which has the main task of combining the various algorithms.

- **Plan Execution and Monitoring:** Here we will limit ourselves to the simple comparison of the current state of the robot to its goal state. If states are represented as points in some (numerical) Euclidian state space, then the comparisons won't be any more complicated than computing the Euclidian distance between two states. Thus we will ignore most of the problems of plan execution and monitoring. The only sort of additional feedback we allow is a simple *failure* message.

- **Adaptation:** In this thesis we expect problems to have an incremental nature. For example once the robot has learned how to walk in a straight line, we expect a step over an obstacle to be an adaptation of the original action, rather than a completely new activity with very little resemblance to the original action. Thus adaptation will be used as a high level search heuristic.

- **Learning:** We will investigate how the robot can learn from a successful manoeuvre and apply this to new situations. Thus the robot has to analyse the current successful behaviour and find out what made it successful. Based on this knowledge we expect the robot to be able to use a successful solution in new, previously unencountered situations.

In the next sections we will discuss how the various aspects of this task can be treated and how they have been solved in previous work. We will roughly follow the issues listed in the present section and briefly present how they have been approached by other researchers.

## 2.2 Robot Motion Control

### 2.2.1 Conventional Robotics

Robot control normally refers to the application of "conventional", i.e. mathematical models of a robot's kinematics and dynamics for its motion planning and control. Consider the manipulator in Figure 2.1. We see a two link arm operating in a plane. The end-effector of the arm is at position $(x, y)$. The robot arm is controlled by adjusting the joint angles $\theta_1$ and $\theta_2$. The problem of finding joint angles $\theta_i$ which make the position of the end-effector coincide with a specified position $(x, y)$ is called the *inverse kinematics* problem. *Inverse kinematics* specifies *where* to move the manipulator joints.

When the manipulator (or robot) moves from one position to another this movement is described by a *trajectory*. The position, velocity and acceleration of some part of the robot, normally the end-effector, describe the trajectory. Given a trajectory as well as the forces exerted at the manipulator tip, the *inverse dynamics* of the robot describes which joint torques need to be applied at which moment in time in order to move the robot along this trajectory. Thus the *inverse dynamics* describe *how* to move the manipulator joints.

Normally the robot will not stay within the precomputed trajectory. Due to various errors or external influences the robot might behave differently from the expected behaviour. A *control system* will have to be designed to keep the robot within a specified trajectory.

Finally the robot can interact with forces from its environment. When the robot lifts an egg then it should use forces different from the forces it would use to lift a car. Similar problems arise when it needs to put a peg into a hole or follow a path along a surface. This problem is called *compliant motion control.*

Conventional robotics has the advantage of using an explicit and well understood mathematical approach to control the robot. Once the robot is accurately described one only needs to deduce the relevant equations and program the robot accordingly. The disadvantages are equally obvious: the mathematics of multi-joint bodies are non-trivial, and the correctness of the formula is based on the correctness of the underlying model of the robot. A new model has to be computed if the robot model becomes inaccurate due to reasons such as wear and tear or reconfiguration.

Good introductory papers to the various aspects of robot motion planning can be found in M.Brady (et.al.): "Robot Motion: planning and control" [Bra82]. Good text-books on conventional robotics are the one by Asada and Slotine [AS86], as well as the one by Paul [Pau81].

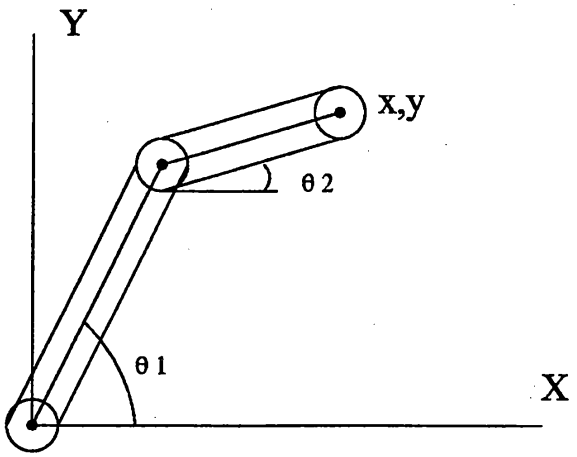



Figure 2.1: A two link manipulator

### 2.2.2 Legged Locomotion

Various successful attempts have been made to apply conventional robotics to the problem of legged locomotion. Legged locomotion offers the possibility of crossing a terrain where wheeled or traction driven vehicles would fail. Research into legged locomotion can roughly be divided into three different areas: Static Multilegged Walking, Dynamic Walking and Dynamics of Human and Animal Motion. For a good historical survey see Raibert's book "Legged Robots That Balance"[Rai86b].

Static multilegged walking machines are stable at any time during the execution of a gait. This is normally achieved by using at least four legs, and designing the gait in such a way that at any moment at least three legs have contact to the ground. The centre of gravity has to remain within the boundaries of the polygon described by these legs. The advantages of such a gait are its guaranteed stability, the disadvantages are the slow speed and redundancy of legs.

Research on static multilegged walking started with General Electric's walking truck. This was a four legged vehicle, with each of the legs individually controlled by a human operator. Later Robert McGhee and his group built the Automatic Suspension Vehicle (ASV), where the operator is partially replaced by a computer [BWP89]. The ASV still has a "driver", but all the driver does is to point a joy stick into the direction into which the ASV shall move. The ASV has been successfully used to cross complex terrains, and in a feasibility study Choi and Song [CS88] show fully automated obstacle crossing gaits for hexapod walking machines.

An interesting variant of a static walking machine is the Ambler project at Carnegie Mellon University [BHK+89]. Most static walking machines are built more or less like insects. Usually four or six legs emerge from a longitudinal body, and thus all legs have fixed relative positions. Unlike such designs the Ambler robot is designed in a completely symmetric way such that all legs *revolve* around a central shaft. Thus the robot looks from above like a modern office chair.

Dynamic walking has been an ongoing research topic, and one of the most remarkable results has been the work of Marc Raibert [Rai86b, RBM84, HR90, RCB86, RBC84, Rai86a, HR91] His work ranges from a one legged hopping machine to a four legged robot that can run, as well as two legged robots which are able to somersault [HR90]. Alternative work on dynamic bipeds has been carried out in Japan, with the emphasis on various control, sensor and mechanical questions. For a discussion of biped robotics see Fushuro and Sano's article [FS90]. All these approaches use conventional robotic control together with precomputed trajectories or a fixed control law. Apart from Raibert's robots none of them deals with obstacle crossings, and in Raibert's case the obstacle crossing is controlled by a human operator or via fixed control algorithms [HR91].

Human and animal motion has been studied as early as 1872 [Muy99, Muy01]. Research focusses on various aspects such as a precise description of the movements, a description of the nervous system generating and coordinating these movements as well as an understanding of the sensory input from the muscles (Pearson [Pea76], page 79). For a detailed treatment of the dynamics of human movements see Hemami's (et.al.) work [Hem85, CHH86, HZH82] as well as work by Pandy and Berme [PB88b, PB88a] and Yang et.al. [YWW90a, YWW90b].

### 2.2.3 Learning and Planning to Walk

#### Planning to Walk: Step Selection in Rough Terrain

Most of the work published on walking robots uses conventional control engineering to control the execution of pre-determined trajectories of the legs. These trajectories (or gaits) are usually computed by the programmer. Alternatively a fixed control law can be given which governs the robot's behaviour. However the crossing of rough surfaces requires that the robot replans parts of the gait in order to ensure that the robot places its feet into the available footholds, and to ensure that the robot reaches the obstacle with the correct posture in order to cross it. Thus rough terrain navigation forces conventional robotic approaches to incorporate some planning.

A typical approach for this type of planning is the work of Hodgins and Raibert [HR91]. Their goal was to enable a dynamic biped to cross a flight of stairs, leap over an obstacle or place its foot onto a desired location. Hodgins and Raibert used three different methods to adjust different parameters of the gaits. One was to change forward speed, another one was to change the running height and a third was to change the duration of the ground contact. Experimental results indicated that forward speed adjustment was the most powerful of these three methods. Hodgins and Raibert also experimented with an approach in which the biped first puts the foot into the required foot position and then tries to regain balance. This approach did not work for many consecutive steps (as required for a sequence of obstacles). Hodgins and Raibert used predetermined foot positions, and the task of the robot's controller is to ensure that the feet of the robot touch the ground at these points.

A recent system which deals with rough terrain crossing is discussed by Pal and Jayarajan[PJ91]. They used a simulated static quadruped robot over rough terrain. Using the $A^*$ algorithm to choose foot positions they programmed the quadruped to search successfully for a sequence of moves which enabled it to cross the rough terrain in the desired direction. This time no predefined trajectory has been used, the system was rather given an evaluation function which allowed it to search for foot placements which generated both a stable support of the robot and a movement forwards. Pal and Jayarajan call the resulting behaviour a "free gait", however there is no explicit notation for the gait and thus it is impossible to reason about the robot gait. Their robot searches for steps but does not learn: every new steps has to be searched in the same way.

### Connectionism and Learning to Walk

On the other hand work has been carried out which represents the robot's behaviour in a distributed, subsymbolic fashion. Learning can occur by automatically adjusting the properties of the neural circuitry which controls the robot. The best known work in this field is the work of R. Brooks[Bro86a, Bro89]. Using a *subsumption* architecture he carefully defined a hierarchy of automata, where each automata is responsible for a certain behaviour. Higher level automata can cancel (subsume) the activities of lower level behaviours. This approach enabled Brooks to build a set of small insect like robots which are able to crawl through the corridors, look for light etc. However the behaviours associated with each automata and the conditions under which they are subsumed are carefully hand-crafted by the designer. Thus there is no learning in Brook's original work. Later work was aimed at learning to calibrate the automata of the subsumption architecture:
Maes and Brooks[MB90] demonstrate how a statistical approach can be used to *learn* the conditions under which certain behaviours are active or passive.

They use a small hexapod robot with only ground contact sensors. They record whether the use or disuse of a motor command created positive feedback or not. The robot can learn (within minutes) to walk using the correlation between feedback (ground contact means failure) and the activities of the robot. Their architecture and paradigm does not include reasoning about entire sets of activities.[1]

Work similar to Brook's has been reported by Beer et. al.[BCS90]. They use biologically inspired coupled neural oscillators to generate rhythmic movements for each leg of a simulated hexapod. Again the neural circuitry has been carefully hand-tuned by the programmer and there is no higher level reasoning about the resulting behaviours. Holland and Snaith [HS91, SH91] also describe experiments with a real quadrupedal robot which uses neural nets in order to learn to control a gait which is generated by a central pattern generator. A similar approach this time applied to generating bipedal gaits from a cyclical pattern generator is reported by Toga et. al.[TYS91]. In their work coupled neural oscillators are used to control simulated dynamic biped walking. The different oscillation patterns representing different gaits were manually programmed. However Toga et. al. do not deal with the adaptation of gaits but rather the control of the gait execution. The important pragmatic difference between the 3 approaches mentioned in this paragraph and Brooks' work described above is the fact that Brooks achieves learning within minutes on a real robot using only on-board components. This, together with his subsumption architecture, accounts for the popularity and influence of his work.

Learning robot walking has also been studied using neural networks and genetic algorithms. Under the ambitious slogan of "brain building" de Garis [dG90] introduces a type of neural net where the weights of the net are determined by a genetic algorithm. Using such nets and an unspecified number of constraints which describe the desired gait he enables a pair of stick legs to learn to walk. The walking system is a pair of legs with knees which can move around the pitch axis. There is no motion around yaw or roll axis. This time learning to walk means learning to follow a specified trajectory.

### Adapting Gaits

Learning new gaits through the adjustment of existing gaits to new situations is discussed by Zheng[Zhe90]. Here a van der Pol pattern generator generates cyclical patterns, which will then be interpreted as gaits. A neural network is used to adjust the constants of the van der Pol generator such that the resulting gait fits best into the situation which is encountered by the robot.

---

[1] See also Mahadevan and Connel [MC91] who describe in their report a subsumption architecture using reinforcement learning.

Zheng's work is based on work by Bay and Hemami[BH87a] who introduced van der Pol pattern generators to simulate a neural pattern generator. This time the learning component of the robot learns to adjust existing gaits to new situations. The original gait is given by the designer.

### What Has Been Achieved

There are dynamic biped walking systems capable of obstacle crossings. However the entire behaviour is engineered by the robot designer. While various aspects of learning to walk have been treated, learning dynamic biped walking and obstacle crossing has not yet been reported.

Most existing approaches to learning walking robots start from some notion of a gait and then train the learning component to execute this gait. The discovery of the notion of a gait is either ignored (as long as the robot walks things are fine) or the gait is already given. There is no integral approach which develops gaits and then adapts them to new challenges in the environment.

## 2.3 Search and Planning

Most problem solving strategies use *search* to achieve their goals. In the typical search problem two states are given, an initial state and a goal state. There is also a set of operators, and each operator transforms one state into another. The search problem is to find a sequence of operators which lead from the initial state to the goal state.

Classic search techniques try to construct a tree of state transitions from the start state to the goal state. A state is discarded from this tree if the search program decides that it will no longer need this state. By discarding this state the search program effectively throws away some of the knowledge which it gained during the search. *Dynamic programming* is used as a search technique where all visited states are kept in memory and don't have to be reconstructed if the search program needs to visit them again at some later point. By doing this the program trades off computation against memory.

A typical example of an application of dynamic programming is Dijkstra's algorithm to find the shortest path in a network [Ber76]: once a state has been visited by the search program it labels the state with the cost of reaching it. If a state is visited again and the new path to this state is cheaper than the previously discovered path, then the state is labelled with the new, cheaper cost. The program continues searching from the cheapest state and stops when it reaches the goal state. Dijkstra's algorithm contains the main ingredients of dynamic programming: memorising states and updating their (heuristic) cost. The algorithm can use the previous results whenever it searches for another path starting from the same start state.

In simple cases exhaustive enumeration of all possible operator sequences can be an easy but efficient way to implement a search strategy. A typical algorithm in this class is the $A^*$ algorithm [Ric83, Kor85a] which uses a heuristic to guide its exploration of the search space. $A^*$ is guaranteed to find the optimal solution provided its heuristic is optimistic (it always assumes that states are closer to the goal than they actually are). However many problems exist for which such a heuristic is of little use. A well known example of such a problem is Rubik's Cube[Kor85b].

In many such cases *search with macros* provides a solution. A macro is a combination of several operators into one. The goal of macro creation is to build macro operators which enhance the power of the search algorithm with respect to a certain goal. Korf [Kor85b] introduces macro operators which solve *non serializable* subproblems. Assume the original goal $G$ can be decomposed into a set of subgoals $S_1, .., S_n$ such that $G$ is satisfied if $S_1 \wedge S_2 \wedge .., S_n$ are each satisfied. The subgoals $S_1, .., S_n$ are serializable if it is possible to solve subgoal $S_i$ without violating any subgoal $S_j$ for $j < i$. Which means that an ordering

of the subgoals must exist for which it is possible to find a solution for each subgoal in this order without violating previously satisfied subgoals. Rubik's Cube is a typical example of a non serializable set of subproblems. Once one side of the cube is in one colour the attempt to have the neighbouring side in another colour will almost invariably destroy this property on the first side. The main idea then is to use macro operators which will reestablish the original properties of a subgoal after their execution. These macro operators are found by using search techniques such as iterative deepening or bidirectional search.

Güvenir and Ernst [GE90] introduce an algorithm called *RWM* (refinement with macros). In a first step *RWM* identifies which operators influence which subgoals. Accordingly the operators are labelled relevant and irrelevant (or safe) with respect to a subgoal. *RWM* then tries to solve first the subgoal over which most operators are safe. Using only this set of safe operators, the search then continues with the remaining subgoals. Macro operators are introduced in order to widen the choice of operators. Refined macro operators are built from combining two relevant operators or an irrelevant operator followed by a relevant operator.

Apart from making the search algorithm more powerful (enabling it to find solutions which simple search techniques like hillclimbing wouldn't find), macros can also be used in order to make the search more efficient. The complexity of any search algorithm is determined by the depth of the search (the necessary number of operator applications) and the branching factor (the number of operators available). Introducing macro operators reduces the depth of the search but increases the branching factor. Thus any program that generates a reasonable number of macro operators has to contain some filter mechanism in order to limit the number of macros.

Iba [Iba89] used a peak to peak heuristic to find macro operators for the peg in hole puzzle. In his case macro operators do not have the task of achieving something that was previously unachievable. Macro operators are rather used in order to compile knowledge that has been gained during the search process. Similarly, the authors of SOAR use macros called *chunks* in order to store the results of previous reasoning in order to speed up future performance [TNR90, LRN86].

## 2.4 The State Space: Classification, Reduction and Pattern Recognition

The state space of the robot is the set of all parameters describing the robot and its environment. It can be interpreted as a multidimensional space where each dimension has binary or real-valued values. The individual parameters are the dimensions of the state space. A state of the robot is described by a single value for each parameter of the state space. The dimensions of the state space can also be called features, and accordingly the state space can be called the feature space.

The idea behind classifying the state space is the following: originally the program knows how the robot behaves in a set of situations. The robot now encounters a new situation which is similar to some of the situations encountered before. Provided the function describing the robot's behaviour is reasonably smooth one could try to predict the robot's behaviour in this new situation by drawing conclusions from its previous behaviour in similar situations.

If we further assume that the robot's behaviour can be classified into some classes of behaviour, then the program could find out how to describe the states which produce a given class of behaviour. This description can be called a *pattern*. The process of *pattern recognition* finds this pattern. Assume the patterns are described in such a way that it is easy to determine for a given state whether it fits into this pattern. Then there is an easy way to determine the behaviour of the robot.

The examples which are used to build the original classification of the robot's behaviour are called the training data. Based on this classification of the training data one can predict the robot's behaviour in a previously unencountered situation. To do this one takes the data describing the new situation and computes how it is classified according to the classification gained from the training data. This classification is then equivalent to the robot's predicted behaviour. The entire process of deriving a classification from a set of training data is often referred to as *inductive learning*.

A simple classification of the state space would be one that divides the state space between stable and failure (illegal) positions. Another classification would be one that reports for each state whether the application of a certain command would lead to failure or not. Alternatively, we could create a set of classifications by describing how long it would take until the robot reaches a failure state if we continuously applied the same command.

If the state space is sufficiently large then many dimensions would be either irrelevant to the behaviour of the robot, or redundant in the sense that other

dimensions are sufficient in order to predict the behaviour of the robot.

When one tries to build an inductive learning system which learns to predict a robot's behaviour, then there are several questions which have to be answered: (1) How will the behaviour of the robot be described and classified. (2) How can dimensions which are irrelevant and redundant for this classification be removed from the state space. (3) How can the patterns describing each class of behaviour be detected and described, and how can this description be used to compute the membership of a state to a given class.

### 2.4.1 Classification

Various approaches have been used in the literature to classify or divide the state space. Basically two different classifications are used:

1. A **behavioural classification** provides a mapping

   $$\textbf{state} \times \textbf{action} \rightarrow \textbf{behaviour}$$

   which classifies the state-action pairs of the robot in terms of the behaviour. Depending on the author a behaviour can be the next state, a sequence of actions or states, the time until failure etc. In general the term behaviour refers to some description of what the robot is expected to do if a certain action is taken.

2. A **control classification** provides a mapping inverse to a behavioural classification, namely

   $$\textbf{state} \times \textbf{behaviour} \rightarrow \textbf{action}$$

   which can be used to determine the action to be taken if the robot is in a certain state and a certain behaviour is required.

The robot is represented as a Cartesian product of the state of the robot and the command or action applied to it, which is then mapped into the behaviour of the robot, which in turn is usually the next state of the robot. The behaviour can be classified by putting all state-action pairs which produce the same behaviour into the same class. The future behaviour of the robot can then be predicted by matching the current state-action pair with the behavioural classes.

Such a simple classification has the distinct disadvantage that it requires one to take the full scope of the robot's behaviour into account before the state-action pairs can be classified according to the behaviour which they produce.

This is usually not seen as a disadvantage since most importantly the intention is to have a classification that predicts the robot's behaviour as accurately as possible.

Alternatively in some cases the task of the robot is extremely simple. Its only task is to avoid a failure state, and the success of the robot is measured in the mean time between failures. In this case behaviours can be classified in very much the same way, ie. by using the expected time until failure as classification. This classification is usually used in cart-pole systems (Figure 2.2), since in such systems the time until failure is directly equivalent to the quality of the behaviour [MC68, BAS83].

### 2.4.2 Reduction

Once a classification of the state space has been obtained, it becomes possible to ask whether there are dimensions of the state space which are irrelevant to this classification. Two approaches to the reduction of irrelevant and redundant dimensions are possible:

1. **Transformation Techniques:** Such techniques create new dimensions of the state space and discard the old ones. The number of new dimensions has to be less than or equal to the number of dimensions of the old state space representation. In the terminology of linear algebra this corresponds to the transformation of the base coordinates of a vector space into a new set of base coordinates. The new set of base coordinates is chosen in such a way that a subset of its dimensions is sufficient to identify the classification of each behaviour.

   A standard algorithm for this problem is the Karhounen-Loève expansion [TG74]. The (linear) coordinate transformation is computed by choosing the eigenvectors of the autocorrelation matrix of the members of each class. Since this doesn't yet decrease the number of dimensions, the eigenvector with the largest eigenvalue is chosen first, and more eigenvectors are included until the new state space allows a distinction between all different classes. Another approach has been suggested by E. Saund [Sau89]. He describes a neural net based algorithm for (non-linear) dimension transformation which works for low dimensional ($n < 4$) state spaces.

2. **Elimination Techniques** do not attempt a coordinate transformation, but rather look at the dimensions of the state space individually and test whether each dimension is needed in order to distinguish between different classes. Two well known and commonly used techniques are decision tree methods and genetic algorithms.

(a) Decision tree methods, most notably PLS1 [Ren83, RC90] and ID3 [Qui83] and its successors [Qui86, CN89, Mic89] provide a constructive approach to dimension reduction: dimensions are chosen based on their *information content* until all classes can be correctly distinguished. All remaining dimensions are discarded.

(b) Genetic Algorithms (also called classifier systems) [Wil87b, KGV83, DeJ87, BF88, Gre86, HHNT86, BGH87, Wil87c, Mau84, Gol85, Dav85] reduce dimensions by searching for increasingly general classification rules. It combines correct classifications by using evolutionary inspired operations like "cross-over" and "mutation" in order to create new classification rules. More general classification rules (called classifiers) have a higher chance of being used to create new classifiers.

### 2.4.3 Pattern Recognition and Prediction Techniques

The future behaviour of the robot can be predicted based on a state space representation and a classification of the robot's behaviours. This is done by looking at the current state of the robot, and the action applied to it, and retrieve the behaviour under which this state-action pair is classified. If all possible state-action-behaviour triplets are stored away, then this is a simple retrieval problem. Obviously in many cases the behaviour of the robot is too complex to store all such triplets. As a result only a limited (but possibly still large) number of such example state-action-behaviour triplets are stored. It will then be the task of a pattern recognition or prediction module to derive the actual behaviour from the stored examples.

Various techniques to do this are discussed in the literature. Some of them have been applied to robotic tasks, in which case the program either had to learn to control an inverted pendulum (Fig 2.2) or to control a multi-link robot arm. Both decision tree techniques and genetic algorithms can be used for pattern recognition and prediction purposes. One simply uses the resulting decision tree or classifier [DS90]. Other popular approaches are the following:[2]

A crude way to predict the robot's behaviour is to quantise the state space into a fixed number of **hyperrectangles**. The prediction is then based on the classification computed for the whole hyperrectangle. Thus only a limited number of classifications has to be computed. Among the earliest publications is D. Michie's BOXES [MC68] program which learns to control an inverted pendulum by learning which actions to take in which part of a coarsely divided state space. In each partition of the state space the program has the choice between two actions. For each of these actions the program records how many

[2]For a comparison of these techniques see Moore's thesis [Moo90].

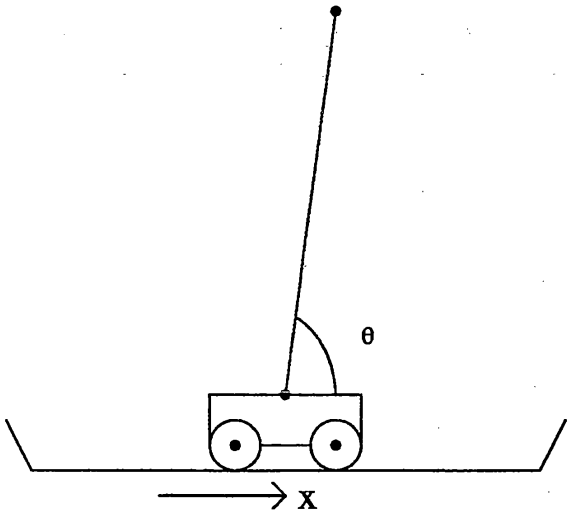



Figure 2.2: The inverted pendulum or "cart-pole" system. The cart-pole system is described using the position $X$ and velocity $X'$ of the cart as well as the inclination $\theta$ of the pole and its first derivative $\theta'$. The system crashes when either $X$ or $\theta$ exceed predefined boundaries.

seconds later the system will fail. By converging on longer and longer mean time between failures after each action the system finally learns to balance a pole. M. Raibert's work [Rai78a, Rai78b] uses a technique called *parametrisation* to split up the state space into reasonably large subspaces with linearised dynamics. The program then successfully learned the inverse dynamics of a 6 degree of freedom robot manipulator. Alternatively the hyperrectangles can be dynamically created and adjusted, as discussed by Salzberg [Sal88] in his thesis.

In 1975 J. Albus [Alb79, Alb75b, Alb75a] introduced **CMAC**, a program that uses a tabular look-up method (based on a hashing function) to learn non-linear functions such as the behaviour of a robot manipulator. CMAC took a multidimensional input function (the state space of the robot) and mapped it into a one-dimensional control function. CMAC was able to store the control action for a given state over a number of addresses in the look-up table. These addresses were distributed in such a way that neighbouring states of the robot *shared* addresses of the resulting control function. This way CMAC was able to take advantage of the continuity of the control function in order to generalise successfully. CMAC has been applied to robot control by Miller et al. [MGK87]. They used a simulated two link robot arm, and CMAC was able to learn to execute predefined trajectories within at most 21 attempts.

**Nearest neighbour** algorithms take a given state action-pair and try to find its nearest classified neighbour in the state space. A typical measurement of the distance could be the Euclidian distance between the two pairs. The

current state action-pair will then be classified in the same way as its nearest classified neighbour. Alternatively the **Q nearest neighbours** can be taken and the classification be derived by taking the mean value or a vote. Andrew Moore [Moo90] describes in his thesis how a *k-d*-tree based approach can be successfully used to learn to perform various robotic tasks like throwing a ball into a basket.

**Connectionist** algorithms represent class membership by a weighting of each dimension. A single **perceptron** [MP88] computes the weighted sum of all features. Based on a threshold the neuron then decides whether these features are below or above a certain hyperplane. A set (or layer) of perceptrons can be trained to adjust the feature weighting such that the resulting hyperplane can be correctly adjusted in order to differentiate between two linearly separable classes of behaviour. Using **multi-layer neural nets** allows us to combine various such perceptron layers. As a result the net can then distinguish classes which are not linearly separable by a set of perceptrons[RM86]. **Temporal differencing** methods [Sut88] are a connectionist approach which allow a classification mechanism to learn from its own predictions. During the training phase the program predicts for each training element its classification. The feature weighting is then updated according to the correctness of the prediction.

There exists a wide variety of connectionist algorithms, the most popular ones being Kohonen's self-organising map [Koh87, Koh82, Koh88], Grossberg's Adaptive Resonance Theory [CG87], counterpropagation networks [HN87], Bidirectional Associative Memory [Kos87], Hopfield Nets [HT85] and Boltzmann Machines[HS86]. [3]

Kawato et al. [KUIS88, KIMS87, MKSS88] use a neural net type algorithm to learn to control a 3 degree of freedom robot manipulator. Kawato et al. were interested in learning the inverse dynamics of the robot. They assumed that the inverse dynamics of the robot is the linear combination of 26 "conveniently chosen" ([KIMS87],p174) trigonometric terms. Subsequently they trained the network to choose the proper weighting of these trigonometric terms.

Another example is Barto, Sutton and Anderson's [BAS83] work (see also [BS81, BAS82]). They describe a program that learns to control an inverted pendulum based on a neural net approach similar to temporal differencing. They partitioned the state space *ex ante* in the same way that Michie did it 15 years earlier. For an overview of "Connectionist Learning for Control" see A.G. Barto [Bar89b, MSW90].

[3] It would be beyond the scope of this thesis to discuss these algorithms in detail. A good collection of the original papers can be found in Anderson and Rosenfield (eds) "Neurocomputing"[AR88]. For a gentle introduction to neural nets the see Wasserman's book on "Neural Computing"[Was89].

**Interpolation methods** compute the weighted average based on previous observations. The weights are computed as the distance of the query point from the previous observations. **Regression** techniques take a number of sample data and try to describe a polynomial function which best describes these data.

J. Koza [Koz90] describes an approach to learning control by using **genetic programming** as the learning element. This is a technique in which various code fragments are recombined in order to generate a control program. The code fragments are LISP expressions which are then spliced together to build an executable program. This technique is not unlike Kawato's where the principal components of the solutions are already known. The program's only task is to weight and arrange these components.

The greatest common denominator of all these approaches is the fact that they are trying to acquire an accurate model of the robot's state-action-behaviour model. They do not create abstract behavioural patterns which could then be used in new, related situations.

## 2.5 Adaptation

### 2.5.1 Adaptation in Existing Learning Robot Control Systems

Adaptation in robot motion control can refer to various aspects of the behaviour of the robot. In the simplest case adaptation can be seen as a control problem where the robot is kept on a certain trajectory despite a changing environment. Noise and other external disturbances as well as changing characteristics of the robot itself can be the reason for this. These issues are handled in conventional robotics by using standard control theory.

Learning systems applied to robot control adapt to a changed behaviour of the robot by updating their prediction of the robot's behaviour, or by updating the type of action which they recommend in a given situation. A. Moore [Moo90] describes in his thesis an updating mechanism for a look-up table based robot controller. Entries in the look-up table are given a time stamp, and when the actual behaviour differs from the predicted behaviour, table entries with the oldest time stamps are removed, and the new behaviour is entered into the look-up table. Thus the system predicts the robot's behaviour based on the most recent events. Neuron-like robot controllers are less exposed to this problem. They can regularly compare their predicted behaviour with the actual behaviour and update the weights in the net accordingly. Thus the network can follow the changes in the robot's behaviour relatively closely. The only problem which arises is the possibility of overtraining.

Adaptation in such learning systems is generally directed towards the adjustment of the individual state-behaviour-action triplets, ie. it corrects the robot's behaviour with respect to the individual state. It focusses on the individual states such that a given trajectory can be closely followed. However in the case of a slightly altered goal which requires a new trajectory, such systems are only able to adapt from previous behaviour in so far as the system knows about neighbouring states and can therefore predict the behaviour. The system still has to *search* in the case of every single state transition in order to execute the new trajectory. This is due to the fact that the systems discussed so far are interested in learning to predict the state space rather than a set of complex behaviours. They do not look at the entire behaviour (as a set of actions) which create the solution (eg. all actions taken in order to reach a certain state), and are therefore unable to adapt the entire behaviour as such.

### 2.5.2 Adaptation as Looking for Invariances Between Different Behaviours

If we want to adapt an existing behaviour to a new situation, then this implies that we want to *keep* some aspects of the existing behaviour. In other words we are looking for invariances between different behaviours. The question is then how these new behaviours are generated and to what extent they are invariant with respect to the original behaviour.

One way to discover invariances is to follow physiological motor control patterns and their modification[4]. Here the general idea is that the brain doesn't store individual state, command, behaviour triplets but rather entire trajectories or command sequences. The command sequences (or patterns) are executed when the brain is presented with a given stimulus. Responses to the variation in the environment are then reflected in changes of the entire command sequence. These different modifications can be seen as qualitative modifications of the same function. Viviani and Terzuolo [VT80] report that learned motor skills (composed out of several individual movements) are varied with respect to the amplitude and overall speed of the entire movement. However the time ratio between the individual movements remains constant.

Another way of looking at invariances between behaviours is by looking at the behaviour as a *qualitative function*. A. Morgan's thesis [Mor88] discusses issues of qualitative control. Here a function is considered to be equivalent to a *qualitative function* if its qualitative values and the qualitative values of its first derivative are equivalent to the qualitative values of the qualitative function. The qualitative value of a function is + if the function value is greater then some threshold $\theta$, negative if the values is smaller than $-\theta$, and 0 in all other cases. The qualitative controller is then given the task to control the qualitative value of some plant's output rather than controlling exact values of the plant's output. The representation of a behaviour as a qualitative function introduces equivalence classes between behaviours by reducing information about the individual behaviours. For that reason it is difficult to use qualitative functions directly to generate specific new behaviours.

[4]As an introduction to physiological motor control see V. Brooks and Schmidt [Bro86b, Sch82].

## 2.6 Summary

This chapter introduced the various different concepts which are needed for complex robot motion planning. The problem could be broadly divided into four different parts: state space reduction, search, adaptation and learning. For each part several algorithms and learning techniques are available.

Most solutions to robotics problems deal with a rigorous mathematical approach to robot modelling, trajectory planning, trajectory control and execution. However various learning techniques have been used successfully to learn and control robot behaviour. What all these approaches have in common is that the robot's goal was either defined relatively simply (e.g. control a cart pole system) or the robot had to execute a predefined trajectory.

None of the learning systems suggests an incremental approach to trajectory planning where previously found motion patterns can be successfully used to control the robot in new situations. This is due to the fact that in all systems discussed above the learned information enables the program to predict individual state-action-behaviour patterns, where each action is a primitive motor command, and each state is at base-level (no higher lever states have been created by generalisation). None of the approaches combines action patterns to create macros.

For this reason the explanatory power of most systems is rather limited. If a user queried the robot's behaviour the only justification the robot could give for its behaviour would be some sort of *fitness* of this behaviour with respect to some goal. This fitness might be misleading in the case of local minima. The system would be unable to see the fitness of a single action with respect to the overall trajectory. However literature concerning the aggregation of simple activities into larger activities exists, and this is discussed in the literature about macros.

Finally various approaches exist to describe invariances in behavioural patterns. However no work is known to the author which systematically generates new behavioural patterns.

# Chapter 3

# Learning Dynamic Biped Walking: A Hierarchical Approach Using Macro Operators

This chapter gives an overview of this thesis. Some shortcomings of previous work are summarised, followed by an introduction to how these problems are dealt with in the coming chapters. The aim of this chapter is to draw a picture of the whole thesis in order to allow the reader to get "the whole picture" before discussing the individual algorithms in detail. By introducing the main algorithms this chapter gives a more detailed overview of the structure of the thesis than Section 1.4.

The literature survey in the previous chapter documented the state of the art and its limitations in some areas relevant to autonomous legged robots. It is beyond the scope of this thesis to address all deficiencies and open problems of such robots. However this thesis addresses several limitations of previous work:

- **State space reduction:** Current approaches to learning robot control use a conveniently chosen representation of the state space. The problem is usually represented in such a way that no irrelevant or redundant parameters occur. This clearly is a severe limitation if we want to build robots which are able to operate in unknown, open environments.
- **Abstraction:** Existing learning robot control systems usually learn from previous experiences by acquiring a more detailed representation of the state space. This enables the system to predict more accurately the behaviour of the robot in a given state. However this does not lead to a

better understanding of the "task-level". If a robot has to learn how to throw a ball into a bucket then it will have to search for a new solution every time the bucket is moved into a new position. Previous systems do not generate higher level concepts (like "throwing the ball").

- **Generating new action patterns:** Currently, robotic learning systems have a fixed set of actions which they can execute. These actions are usually motor commands of the type "apply 100 Newton-meter to joint $\theta_1$". For this reason, these robotic learning systems have only limited abilities to generate new actions. An abstract view of a sequence of actions - *macros* - will provide a "task-level" description of behavioural patterns. By searching over the space of macro operators we will gain a powerful tool to generate new, related action patterns.

- **Learning from success:** If new successful solutions to complex problems have been found, then it is important to analyse and generalise these successful solutions. Imagine that the robot has to adapt and modify its previously used action patterns in order to negotiate an obstacle. If similar modifications prove to be successful in a set of instances, then it is important to analyse these modifications and extract a generic obstacle crossing technique from them. In order to do this, we have to be able to compare different action patterns and identify which difference was crucial with respect to the new task, and where. Previously reported systems have to search on the level of primitive actions each time a new task is solved.

The following sections describe how these issues are addressed. We see how a learning system is developed which enables a dynamic biped robot to learn to walk. Additionally this system enables this robot to cross a surface littered with random obstacles. This in itself is a result which, to the best of the author's knowledge, has not been reported yet.

## 3.1 State Space Reduction

Before the robot starts to search for the macro operators it needs to "clean up" the state space description. We assume that the robot is almost flooded with information, some of this information relevant, but most irrelevant to the task of the robot. Most importantly, we assume that the information given to the robot is *complete* in the sense that it is sufficient to describe all relevant aspects of the robot and its behaviours. It will therefore be the aim of the robot to organise its state space according to the importance of the parameters of the state space. The robot can then discard all irrelevant parameters. Using

this reduced version of the state space it can now focus on the control of the important parameters.

In this thesis we will use a hierarchy of macro operators or behaviours. We will assume that complex behaviour can be derived from simple behaviour, and thus it makes sense to look for simple and general behavioural patterns first. We will use this hierarchy of behaviours for the organisation of the state space of the robot.

The hierarchy of the state space will be organised according to the influence that each dimension of the state space has on the survival of the robot. This means that a statistic will be created which reports the extent to which changes of a parameter affected the mean time between failure of the robot. The main idea of the algorithm, the *survival time heuristic,* is as follows: the robot chooses an arbitrary behaviour, ie. a sequence of motor commands. It now picks an arbitrary start position and executes this sequence of motor commands until failure occurs. It records the time it took until failure occurred, which we will call the *survival time.* It then re-runs the same experiment with the one exception that it modifies the value of one parameter in the start position. Again the survival time is recorded. This is repeated for a set of parameter variations, and in the end, the robot records the variation of the survival time. This is then re-run for different start positions and different command sequences, and the average variation of the survival time is computed. This variation of the survival time is equivalent to the importance or weighting of the parameter. Repeating this for all parameters of the state space results in an ordering of the parameters based on their importance for the survival of the robot.

Based on these data the state space of the robot can be organised into various disjoint subspaces. First, there is the *inner state space* where a change of any parameter has an immediate impact on the survival time of the robot. Next, there is the *outer state space* where in some cases parameter changes have an immediate impact on the survival time of the robot. A typical example for a parameter in the outer state space is the height of an obstacle: as long as the robot is far away from the obstacle the obstacle height is completely irrelevant. However, the obstacle height has great impact on the robot's behaviour whenever the robot tries to cross the obstacle. Finally, there are the *irrelevant parameters* which can be altered arbitrarily without having any influence on the survival time of the robot.

The inner state space is the part of the state space which must always be controlled. The macro operators controlling the robot in the inner state space form the basic behaviour from which more elaborate behaviours can be derived. For a detailed discussion of these heuristics and their experimental validation see Chapter 5.

## 3.2 Macro Operators in Biped Motion Planning

Macro operators as defined in the literature serve two different purposes: they allow for serialisability of subgoals [Kor85b, GE90] and they are used for the speed up of the search process by chunking together a successful sequence of actions into a macro [Iba89, LRN86]. In this thesis we will view macro operators as a means of speeding up the search process.

Imagine the situation where a biped robot has to learn to walk. In terms of the state space description this could mean that we expect the robot to change its position along some axis, which could be called the $X$ axis. We can then search for a sequence of motor commands - a macro - which results in such a displacement of the robot along the $X$ axis.

Unfortunately not all these macro movements will lend themselves to a repeated execution. In this thesis we will view a macro as a sequence of primitive actions and a set of preconditions. The preconditions have to be satisfied if the macro is to be applicable. A macro operator is called *linear*, if for any $n$ the macro can be repeated $n$ times. Thus we are looking for *linear macros*.

Linear macros do not affect their preconditions. Therefore at the end of the macro execution all preconditions have to be reestablished to their original values. Since the preconditions describe the value of some parameters of the state space of the robot, these parameters have to change either in a cyclical manner during the gait execution or they must not change at all. If the preconditions of the macro require parameter values to be within a certain interval, then the parameters may change only within the boundaries of this interval.

Applying macro operators to biped locomotion leads to the discovery of the gait. A gait is a cyclical movement of the legs. Now, "all" the program has to do is identify such gaits and describe them. The precondition of the macro will correspond to the start position of the biped at the beginning of the gait, and the operator sequence will correspond to the sequence of commands given to the motors.

Thus, we have identified the trajectory planning of the robot as the search for gaits. Executing this search for gaits within the inner state space will produce the basic behaviour from which we will derive more complicated obstacle crossing gaits for the outer state space. Chapter 6 will describe this search for basic gaits in detail. Chapter 7 will show how a gait can be linearised and how a simple controller can be developed to control the gait execution.

## 3.3 Exploring the Outer State Space

In this thesis it is claimed that complex robotic motions can be derived incrementally by adapting simple behavioural patterns to new, more complex situations. This adaptation is based on the interpretation of a gait as a macro. By creating new macros from old successful macros we hope to preserve the generic parts of the old macro that made it successful, while at the same time, we hope to add new features to it that will make it successful in a new situation. Therefore, the main aspect to modifying macros is to identify *invariances* which have to be preserved. All other features will then represent the search space over which new macros can be generated.

In this thesis macros represent gaits. A gait in its ideal form is a periodic function of some parameters. By looking at various ways in which to describe these functions, we can generate an abstract description of a function which would allow us to define invariances. We will abstract functions by defining equivalence classes between functions.

The definition of equivalence classes can be guided by the parameters describing periodic functions: *frequency, amplitude* and *phase-shift*. Using equivalence classes which strongly resemble a modification in one of these parameters we generate modifications of the original gait which are still similar to the original gait but powerful enough to achieve obstacle crossings.

In order to describe another way to define new functions which preserve some of the important properties of an original function, let us look at the following example: two functions $f$ and $g$ are qualitatively equivalent ($f \approx_q g$) if they are defined over the same domain and for each value in the domain the qualitative values of the two functions and their first and second derivatives are equal. The qualitative value of a function is computed by mapping the function value into the set of qualitative values +,−,0. Thus, $q(x)$, the qualitative value of $x$, is defined as

$$q(x) = \begin{cases} + & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ - & \text{if } x < 0 \end{cases}$$

such that

$$f \approx_q g \Leftrightarrow \forall x : q(f(x)) = q(g(x)) \wedge q(\dot{f}(x)) = q(\dot{g}(x)) \wedge q(\ddot{f}(x)) = q(\ddot{g}(x))$$

By creating new functions (gaits) which are qualitatively equivalent to the original gait we hope to create new gaits which still preserve the general "behaviour" of the original gait while modifying it in such a way that new results can be achieved.

Searching over the space of qualitatively equivalent gaits will enable the robot to create powerful new gaits. We will call the heuristic generating these behaviours the *qualitative equivalence heuristic.* These various gait modifications will be discussed in more detail in Chapter 8. Using the qualitative equivalence heuristic we will be able to generate new behaviours which enable the biped robot to learn to cross obstacles like a wall or a step.

## 3.4 Identifying Obstacles and Representing the Obstacle Crossing Gait

Using the terminology of the outer and the inner state space, the robot is in the outer state space when its normal gait is leads to failure and it needs to modify its behaviour. If we assume that the outer state space is regular, in the sense that the same types of obstacles will appear at various places in the outer state space, then it will be meaningful to identify these obstacles and their properties.

When the robot encounters an obstacle (its normal gait fails), the qualitative equivalence heuristic will be used to find an obstacle-crossing gait. Once this gait has been found it will be used to identify which parts of the outer state space constitute the obstacle itself. This is done by again varying parameters of the outer state space one by one (through simulation), recording those whose variation resulted in a failure of the gait. These parameters constitute the obstacle. Together with the inner state space these parameters form the *obstacle space* for this type of obstacle.

At this point, the program has identified the parameters which constitute the obstacle. Using the qualitative equivalence heuristic enables the robot to develop an obstacle crossing gait. What we want to achieve next is to generalise from these successful obstacle crossings such that the robot is able to cross future obstacles without having to search for an obstacle crossing gait from scratch.

What the program has to achieve is a description of *how* and *where* the behaviour of the robot changed. This will be done by using the concept of a *virtual evaluation function.* This concept refers to the fact that during the successful execution of the obstacle crossing, the robot executed some behaviour which was different from its normal behaviour. (This is identical to the definition of an obstacle). Now the *virtual evaluation function* describes how this behaviour differs from the behaviour which the robot would normally have performed.

To illustrate the use of a virtual evaluation function imagine the biped robot has to step over a wall. The robot will discover a gait which results in lift-

ing one leg over the wall and then lowering it again. This behaviour will be compared, state by state, with the behaviour which the robot would have displayed if the obstacle didn't exist. Thus, every state in which it lifts the leg even further (in order to get it over the wall) will be marked as different. However, when the robot lowers its leg behind the wall the behaviour will not be marked as different. This is due to the fact that the robot would always try to get the foot back to the ground. The virtual evaluation function describes this difference in behaviour by describing what sort of state evaluation the robot must have used in order to "prefer" to lift the leg. This description will then enable the robot to recreate this behaviour in an individual state, rather than having to execute an entire gait.

Using a qualitative filter (looking only at qualitative differences in the behaviour), the program will be able to detect various phases of divergent behaviour. These phases of divergent behaviour constitute the obstacle crossing "technique". It will be the task of the learning component of the program to identify the parts of the state space where this divergent behaviour can be applied in order to achieve a successful obstacle crossing.

## 3.5 Experiments and Epilogue

Chapter 10 will conclude the experiments with the survey of the robot's obstacle crossing capabilities. The biped will be tested on a variety of randomly generated surfaces with different obstacles in various positions. We will use four types of obstacles: a wall, a step up, a step down, and a slope. Distributing them arbitrarily on a test surface we will then demonstrate how a simulated dynamic biped robot can cross such an obstacle path.

Chapter 11 will summarise and evaluate the results of this thesis and point towards future work.

# Chapter 4

# A Dynamic Biped Robot

## 4.1 Dynamic Walking and State Space Learning

In this thesis dynamic biped walking has been chosen as an application environment. While there are many "standard problems" (such as pole balancing and standard one-arm robot manipulators) to which machine learning algorithms could be applied, various reasons exist for choosing dynamic walking as an application domain: the size of the state space, the extendible nature of the solutions, the complexity of the problem, the ease of verbalising phenomena, and the potential practical use. Finally there is the challenge of advancing the state of the art in dynamic biped walking.

**Size:** The modelling of multilegged walking systems introduces a state space of challenging size. In order to control the biped, several parameters (degrees of freedom) have to be controlled simultaneously. For example, the original state space of the biped used in this thesis contains 10 dimensions: the angles describing the position of each leg around the roll and pitch axis as well as the position of the hip around the roll axis, and their respective velocities (see Figure 4.1). The state space can be almost arbitrarily enlarged by adding additional parameters describing the surface on which the biped is walking.

**Extendibility:** In order to walk a series of activities (adjustment of various parameters) has to take place. All these activities are composed from discrete motor commands (actions), and these activities can be modelled as a sequence of operations rather than using a discrete time "point". Walking therefore allows us the definition of a *behaviour* as a sequence of actions. Walking is at the same time complex and repetitive enough to justify the use of *macro operators*. Using these macro operators dynamic walking can display increasingly difficult types of behaviour. Starting with walking in the plane the robot can be trained to change the direction in which it is walking. It can be trained to

cross obstacles of increasing difficulty. The important point is that *transfer* of behavioural pattern from one behaviour to another is possible and meaningful: a step forward and a step over an obstacle are comparable in so far as both movements are *steps*.

**Complexity**: Dynamic walking is non-trivial. It involves balancing, trajectory planning and search. Dynamic walking can be seen as a variation of the well known pole-balancing problem. When the robot stands on one leg then it tends to fall over this leg in a similar way to the pole. The cart which controls the joint velocity of the pole corresponds to the speed with which the biped is walking. The main difference is of course the fact that in biped walking the "falling" support leg cannot be controlled until a support exchange command is executed and the other leg becomes the support leg. However, the ability to control a walking dynamic biped does not imply that the applicable algorithms can also control a pole balancing problem.

In the case of an actual (non-simulated) robot this might be further complicated by adding real-time computing constraints. However, real-time constraints will not be treated in this thesis.

**Verbalising**: Everybody is reasonably familiar with the process of biped walking. Most people know that standing on one leg involves balancing, and they are able to communicate and visualise the basic motions involved in biped walking.

**Usefulness**: Walking robots are able to move where traction driven vehicles fail. The state of the art in legged robotics does not yet allow the industrial use of legged robots. However there are two important factors which will enhance the use of walking machines outside the research labs: cheap and powerful onboard computing will make walking machines economically feasible, while at the same it will become less acceptable to have humans working in hazardous environments. For the moment the cheap supply of human labour, and the many open problems involved in the design of autonomous robots (eg. sensing and real time computing) are responsible for keeping autonomous robots wandering within the boundaries of the floors of the research laboratories.

**State of the Art**: Until recently rough terrain crossing has been limited to static walking machines. However, Hodgkins and Raibert [HR91] demonstrate a planar dynamic biped walking machine which is capable of crossing obstacles. This obstacle crossing capability is due to a (manual) controller design which enables the robot to adjust its step-length and to jump onto and over the obstacle. *Learning* dynamic biped obstacle crossing has not been reported yet.

## 4.2 The Model

In this thesis a simulated model of a dynamic biped robot will be used in order to demonstrate the power of the developed algorithms. This simulation is based on a "real" dynamic biped, Biper3 [MS84]. Biper3 is essentially a pair of stiff (kneeless) legs, linked by a hip. Biper3 walks like a person on stilts, a little bit like *Charlie Chaplin*. The robot weighs about 1 kg, and it is about 30 cm high and 7 cm wide (precise data are given in Figure 4.2). Figure 4.1 shows a view of Biper3 from the front and from the side. The view from the front is also called the view along the *roll axis*, whereas a view from the side corresponds to a view along the *pitch* axis. The biped can move along both of the axes, which means it can move forwards and sideways, but it can not turn (no moves around the *yaw* axis).

Let us assume that Biper3 stands on one leg. The leg on which the biped stands is also called the *support leg*. This other leg is freely movable around the roll and the pitch axis, it will be called the *free leg*. Biper3 is a dynamic biped in the sense that it will fall down if it doesn't keep walking: originally the biped stands on one leg (the *support leg*). Let this leg be *leg1*. The support leg, *leg1*, will lean into some direction along both the pitch and the roll axis. The robot will *fall* into this direction, until a *support exchange* command declares the other leg (the *free leg (leg2)*) as support leg. Then the robot will start to fall into the direction into which *leg2* is leaning. In order to walk properly the robot has to make sure that it brings the free leg (*leg2*) into a reasonably upright position before it falls over too much. It then executes a support exchange command (ie. the robot turns *leg2* into the support leg, *leg1* becomes the free leg) and then starts to adjust the previous support leg (*leg1*) and brings it into a reasonably upright position.

The expression "reasonably upright" has been used because it can be important that the robot doesn't start with an absolutely upright support leg. If the support leg is leaning slightly forward around the pitch axis, then the centre of gravity of the robot will shift into this direction and at the next support exchange command the robot will have executed one step into this direction.

The biped is described by 5 parameters, $\theta, \psi, \phi, \eta$, and $\zeta$ and their first derivatives. $\theta$ describes the inclination of the support leg around the roll axis, $\psi$ describes the orientation of the hip with respect to the roll axis, and $\phi$ describes the orientation of the free leg around the roll axis. $\eta$ describes the inclination of the support leg around the pitch axis, wheres a $\zeta$ describes the inclination of the free leg around the pitch axis. 3 motors exist in order to control the biped: motor $u_1$ moves the free leg around the pitch axis, and motors $u_2$ and $u_3$ move the hip and the free leg respectively around the roll axis. The support leg described by the parameters $\theta$ and $\eta$ behaves like an inverted pendulum and simply falls into the direction into which it is leaning at the moment. More

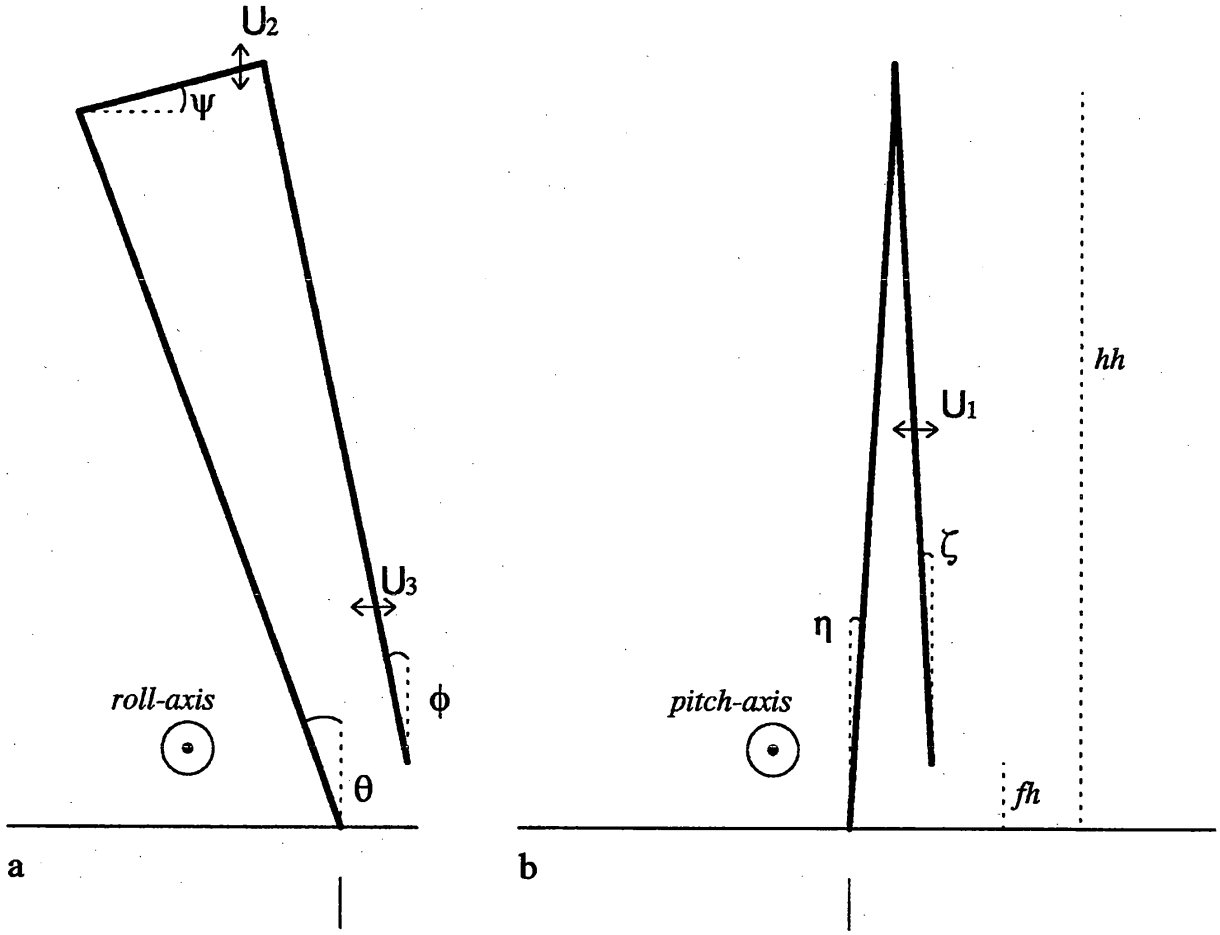



Figure 4.1: The biped. Position (a) shows the biped robot from the front (along the roll axis), position (b) shows the biped from the side (along the pitch axis). The robot's behaviour is described by the 5 angles $\theta$, $\psi$, $\phi$, $\eta$,and $\zeta$, and their respective velocities. The vertical line underneath one of the legs indicates the support leg. Motor1 moves the free leg with torque $u_1$ around the pitch axis, Motor2 moves the hip with torque $u_2$ around the roll axis, and Motor3 moves the free leg with torque $u_3$ around the roll axis. The footheight (*fh*) describes the height of the lower end of the free leg above the ground, whereas the hipheight (*hh*) describes the height of the centre of the hip above the ground.

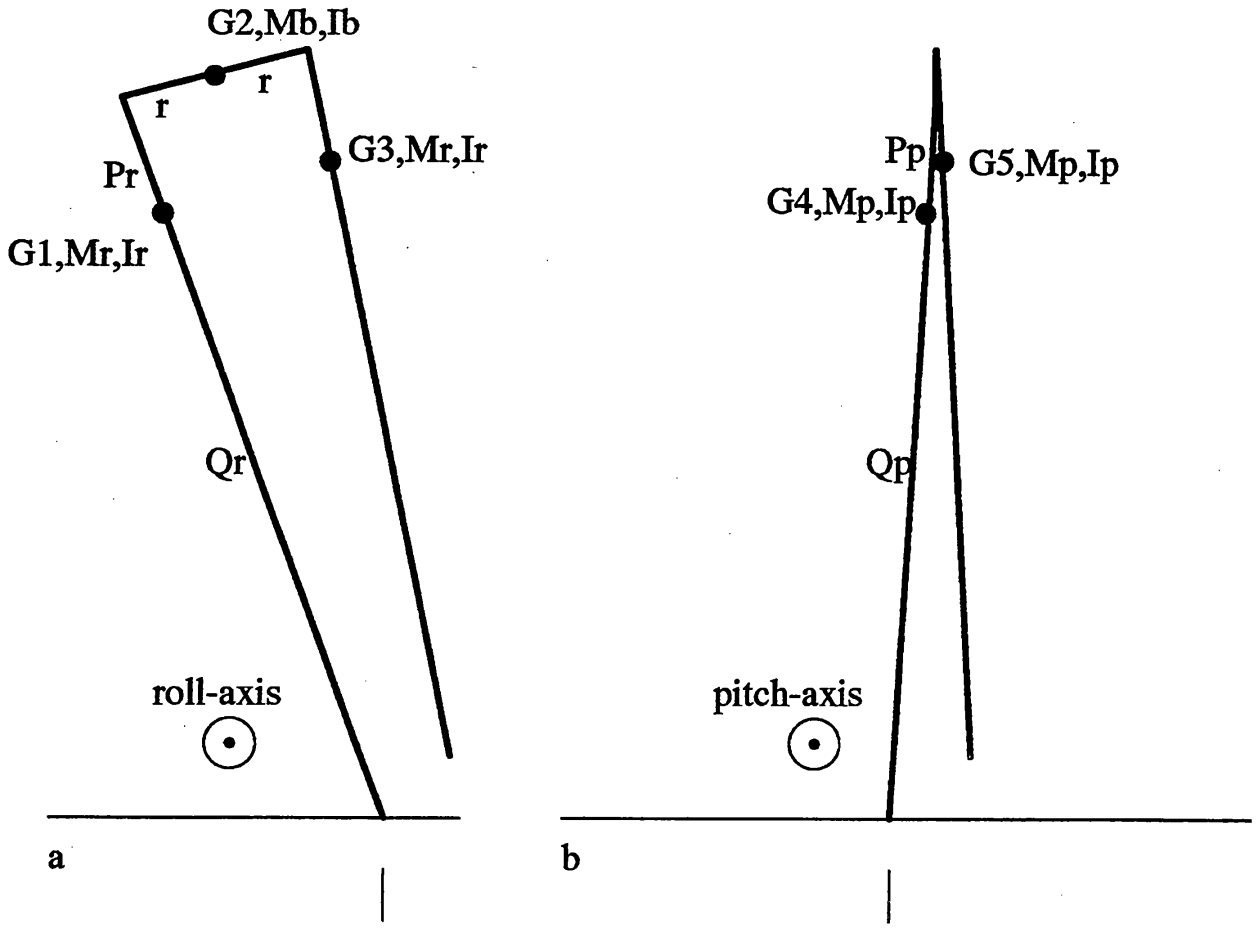



Figure 4.2: Further description of the biped model: G1,..,G5 are the centers of mass of each segment, Mr, Mb and Mp describe their respective masses, and Ir, Ib, Ip the respective moments of inertia about the centers of mass. Pr, Qr, Pp, Qp, and r describe the distance from the end of each segment to its center of mass.
Mr = 0.56 kg, Mb = 0.664 kg, Mp = 0.892 kg, Ir = $4.17 \times 10^{-3}$ kg $m^2$, Ib = $9 \times 10^{-4}$ kg$m^2$ Ip = $6.21 \times 10^{-3}$ kg $m^2$, Pr = 45 mm, Pp = 45 mm, r = 37.5 mm, Qr = 263 mm, Qp = 263 mm.

precisely: the support leg behaves with the free motion of an inverted pendulum, and it will fall into the direction into which it is leaning at the moment provided there is no large enough momentum going into the other direction left over from the movement of the leg before the last support exchange.

Miura and Shimoyama give a mathematical model for the motion of the biped [MS84]. This model is based on several assumptions[1]:

1. The motions about the roll, pitch and yaw axis are independent.
2. The motion about the yaw axis is neglected.
3. There is no frictional force at any joint.
4. The equations of motion may be linearised.
5. The foot contacts the surface at a point, where a large frictional force is produced and no slipping occurs. Thus, the point may be regarded as a universal joint which is free to move about the pitch and roll axes.

Using these assumptions Miura and Shimoyama develop the following set of equations describing the motions of the robot:

$$\ddot{\theta} = 34\,\theta \tag{4.1}$$
$$\ddot{\psi} = 201\,u_3 - 201\,u_2 - 132 \tag{4.2}$$
$$\ddot{\phi} = 50.9\,\theta - 50\,\phi + 203\,u_3 \tag{4.3}$$
$$\ddot{\eta} = 37.4\,\eta \tag{4.4}$$
$$\ddot{\zeta} = 57.7\,\eta - 56.1\,\zeta + 131\,u_1 \tag{4.5}$$

The robot is controlled by changing the control parameters $u_1, u_2, u_3$, or by executing a *support exchange* command. The support exchange command is basically a renaming of the parameters (the index $i$ indicates the new values after the support exchange, $f$ refers to the old value):

$$\theta_i := -\phi_f \tag{4.6}$$
$$\phi_i := -\theta_f \tag{4.7}$$
$$\eta_i := \zeta_f \tag{4.8}$$
$$\zeta_i := \eta_f \tag{4.9}$$

In order to justify this behaviour Miura and Shimoyama introduce a further set of assumptions, namely

[1] cited from [MS84], page 305

6. The duty cycle of the double support phase (support exchange phase) is short compared with the period of the single support phase, and the posture of each segment does not change during the support exchange phase.

7. The collision of the foot against the surface is non-elastic, and $\psi = \dot{\psi} = 0$ is equivalent to the state in which both feet are in contact with the surface.

8. The joint velocities do not change except for renaming: $\dot{\theta}_i = \dot{\phi}_i = -\dot{\theta}_f$ $\dot{\eta}_i = \dot{\zeta}_i = \dot{\eta}_f$

In the biped model used in this thesis, Miura's and Shimoyama's assumptions will be followed except for assumption 7 ($\psi = \dot{\psi} = 0$). This will be relaxed by allowing non-zero values for $\psi$ as well as for $\dot{\psi}$. Thus the biped doesn't have to be exactly upright around the roll axis when it comes to a support exchange command. In this thesis the biped was simulated by recomputing the position of the robot in time steps of 0.04 seconds.

However it will not always be assumed that the robot behaves exactly as predicted by the equations. In some experiments the algorithms will be tested by adding noise which perturbs the motor commands or the joint positions.

# Chapter 5

# A Hierarchy Based Approach to State Space Control

## 5.1 Introduction

Autonomous robots are normally expected to operate in a rich environment and hence they are represented by a rich state space. Apart from the parameters describing the robot itself, various other parameters are added in order to describe the environment. If the robot is not pre-programmed, but rather is expected to learn how to survive in such an environment, it is of paramount importance for the learning program to reduce the state space and to single out the dimensions which must be controlled when executing the individual tasks.

It is assumed that despite the richness of the environment the behaviour of the robot is simple and repetitive. As an example the task of walking in obstacle littered surroundings has been chosen. We assume that the robot has a limited repertoire of activities and we expect it to use these activity patterns and their modifications in order to survive in its environment. The behaviour of the robot will be guided by some pre-defined *goal*. This means for the walking robot that we expect it to be able to walk across a plane, but apart from this task we simply expect it to *survive* in its environment. This doesn't necessarily represent a strong restriction in the viability of the model since it can be argued that additional tasks that need to be mastered in order to guarantee survival can be learned separately. It is therefore assumed that the task of the robot can be learned *incrementally*.

The analysis of the state space will proceed along the following lines: first (in Section 5.3) the program analyses which parameters of the state space can be directly influenced by the robot itself. In the next step the program identifies

which of the parameters of the state space have an influence on the behaviour of the robot itself (Section 5.4). Based on these two pieces of information the robot can then construct a look-up table of how to control the parameters which it can control *and* needs to control.

## 5.2 Assumptions about the State Space

### 5.2.1 Linearly Dependent Parameters

The robot and its environment are described using a large number of parameters. Some of these parameters are dependent on other parameters. For example the height of the robot's foot is dependent on the inclination of the legs around the roll and the pitch axis, and it is impossible to alter the height of the foot without affecting the joints of the legs.

In this thesis it will be assumed that from the start the robot is given a maximal set of linearly independent parameters. Let this set of parameters be called the *base parameters*. The set of base parameters is sufficient to describe the robot and its environment. The values of all other parameters can be derived from it. Thus this set of parameters is equivalent to a base in a vector space. This thesis will not deal with the discovery of dependencies between parameters. The robot will therefore know about the existing dependencies between parameters.

The rest of this chapter will use the term parameter synonymously for base parameter unless stated otherwise.

### 5.2.2 Choosing Start Positions

It is assumed that the robot is able to put itself into any physically possible position anywhere in its environment. This is important in order to compare for example the outcome of various motor commands in the same position.

Any such position is described in terms of the set of linear independent parameters described in Section 5.2.1. If the robot wants to reach a position defined by some other set of non base parameters then it has to search for a description of this position in terms of its base parameters. For example if the robot wants to start its activities in a position where one foot is two inches above the ground, then it has to search for a combination of joint angles (the base parameters) which generates such a state.

### 5.2.3 Discussion of Assumptions

The knowledge of a set of base parameters and the ability of the robot to start its activities in an arbitrary state correspond strongly. If the robot didn't know which parameters it can choose freely then selecting different start states for various tests would be difficult and time consuming. However this ability to pick arbitrary start positions is important for many of the algorithms developed in this thesis. If the robot wasn't able to start its activities in a given state then it would have to wait until it reaches such a state, be it by coincidence or as the result of a search process. This would slow down considerably the efficiency of the discussed algorithms.

Nevertheless it must be admitted that robots operating in an open environment would not have this ability. New features in the environment will make it impossible to give the robot in advance a set of base parameters for all eventualities. The only thing one could assume is that the robot has some sort of *reset mechanism* which enables it to set itself into one out of a finite set of possible start positions. This reset position would be defined in terms of a set of a priori known parameters. If the robot wants to start its activities in a state different from its reset positions then the robot will have to *search* for a set of activities which brings it into such a state. However in this thesis this will not be the case, and it is assumed that the robot can start its activities in any combination of base parameters.

## 5.3 Identify Operator-Specific Subspaces

Originally the system presents itself as a potentially large set of different parameters or state variables. Some of these parameters will be redundant, some will be irrelevant and not all of them will be mutually dependent. That means that some parameters can be controlled independently of some other parameters. They can only be influenced using certain commands (or operators), and we will call the subspace which is influenced by an operator $op$ the *operator-specific subspace* $\mathcal{R}(op)$. Once the program discovers which parameters are influenced by which commands it might become possible to split the state space into mutually independent subspaces. The program could then control each of these subspaces independently. This would also imply a potentially dramatic increase in efficiency since several dimensions could be discarded.

### 5.3.1 Constructing $\mathcal{R}(op)$

In a first step the state space is split up into operator specific subspaces. An operator specific subspace describes a set of parameters and the control commands which influence exactly these parameters. This means that by applying one of the control commands we will only be able to witness changes in value of those parameters represented in this subspace. However, operator specific subspaces do not have to be disjoint. Splitting up the state space into operator specific subspaces will result in a shrinking of the original search space by potentially several orders of magnitude. Henceforth the parameters that can be directly influenced will be called *control parameters* whereas all other parameters will be known as *reference parameters*.

The program has to find out which commands control or influence which parameters. In order to do this the program picks a set of random reproducible states. Reproducible means that the system must be able to get back into this state without any major effort.[1] In each state all control parameters except one are kept equal while the latter gets replaced by a number of random values. By means of comparing the different resulting behavioural patterns it becomes possible to find out which parts of the system are influenced by which control parameter. The program simply looks at the sequence of states through which the system goes and observes which parameters behave differently if in a new run some control parameters are changed.

More formally the system is in a state

$$\mathcal{S}_0 = x_1, x_2, ...x_n$$

where $x_i$ are the reference parameters. The control vector

$$\mathcal{U}_0 = u_1, u_2, ...u_m$$

with $u_i$ being the control parameters is applied to the system. As a result the system will go through a set of states $\mathcal{S}_0..\mathcal{S}_i$ before the test terminates.

$$\mathcal{U}_0 \circ \mathcal{S}_0 \rightarrow \mathcal{S}_1 : x_1^1, x_2^1, ...x_n^1$$

$$\mathcal{U}_0 \circ \mathcal{S}_1 \rightarrow \mathcal{S}_2 : x_1^2, x_2^2, ...x_n^2$$

$$...$$

$$\mathcal{U}_0 \circ \mathcal{S}_{i-1} \rightarrow \mathcal{S}_i : x_1^i, x_2^i, ...x_n^i$$

[1] Such reproducible states exist because the existence of a reset mechanism is assumed which is able to set up the robot in any legal state.

In the next step the original state $\mathcal{S}_0$ is again established, but this time the control parameter $u_j$ in $\mathcal{U}_0$ is altered before the otherwise unchanged control signal $\mathcal{U}_0'$ is again applied to $\mathcal{S}_0$, producing a series of states $\mathcal{S}_i'$.

$$\mathcal{U}_0' \circ \mathcal{S}_0 \rightarrow \mathcal{S}_1' : x_1^{1'}, x_2^{1'}, ...x_n^{1'}$$

$$\mathcal{U}_0' \circ \mathcal{S}_1' \rightarrow \mathcal{S}_2' : x_1^{2'}, x_2^{2'}, ...x_n^{2'}$$

$$...$$

$$\mathcal{U}_0' \circ \mathcal{S}_{i-1}' \rightarrow \mathcal{S}_i' : x_1^{i'}, x_2^{i'}, ...x_n^{i'}$$

As a result some parameters $x_i$ in $\mathcal{S}_i'$ will have values which are different from their respective value in $\mathcal{S}_i$, and some parameters will remain unchanged in comparison to previous trials. If for a large set of start states $\mathcal{S}_0$ no parameter changes, and

$$\forall i : \mathcal{S}_i' = \mathcal{S}_i$$

then $u_j$ is irrelevant and can be discarded. Otherwise all these parameters $x_i$ that change their behaviour are collected in a set $\mathcal{R}(u_j)$. This set contains only the parameters that change their behaviour due to a change of the control parameter $u_j$.

$$\mathcal{R}(u_j) = \{x_i \mid x_i^k \in \mathcal{S}_k, x_i^{k'} \in \mathcal{S}_k', x_i^k \neq x_i^{k'}\}$$

where $x_i$ refers to the name of the reference parameter, and $x_i^k$ refers to its actual value after $k$ steps. In order to make this procedure more robust the definition can be replaced by one that requires the absolute difference between the reference parameter values to be above a certain threshold $\theta$ :

$$\mathcal{R}(u_j) = \{x_i \mid x_i^k \in \mathcal{S}_k, x_i^{k'} \in \mathcal{S}_k', abs(x_i^k - x_i^{k'}) > \theta\}$$

This test has to be repeated with different "start-states" $\mathcal{S}_0''$ in order to avoid wrong results due to performing these tests at a fixed point. By detecting the parameters which change more than a certain limit it becomes possible to find out which parameters are influenced by which control signals. The same test has then to be repeated with a number of random values for $u_j$ and a number of different "start states" $\mathcal{S}_0''$. The union of all results will then describe the set of all parameters that are influenced by the control parameter $u_j$.

This allows the state space to be split into independent subspaces, with each subspace containing mutually exclusive sets of control parameters and the set of reference parameters which are influenced by them.

For example consider a car which is described using *speed*, *orientation*, *x-position* and *y-position* as parameters. There are two command signals, *accelerate*

and *turn*. In a first test the car starts from a random position with a random control command and the next, say, 10 steps are recorded. In consecutive tests everything is identical except that the *accelerate* command is altered. The test series is then repeated with a different set of starting positions $\mathcal{S}_0''$. Now the program checks what dimensions of the state space of the car display a different behaviour. In this case this will be the parameters speed $v$, x-position $x$ and y-position $y$, while the orientation $\alpha$ remains unchanged. Similarly, *turn* changes everything except speed $v$. Accordingly the following two sets will be created :

$$\mathcal{R}(accelerate) = \{v, x, y\}$$

and

$$\mathcal{R}(turn) = \{\alpha, x, y\}.$$

### 5.3.2 Creating Operator Specific Look-up Tables

Once the operator specific subspaces $\mathcal{R}(op_i)$ have been defined, the next step is for the program to generate a simple representation of the relationship between the operator $op_i$ and the parameters in its operator specific subspace $\mathcal{R}(op_i)$.

In this thesis we will assume that a *very simple* look-up table based *bang-bang* control is sufficient to control the robot. (The details of this controller will be discussed in Chapter 7.) This section will be limited to the construction of the look-up table. This table will then be used directly for the construction of the controller in Chapter 7.

In order to construct such a look-up table the program will identify for each parameter $p$ in $\mathcal{R}(op)$ which value of $op$ suffices to ensure that the value of $p$ increases or decreases. In other words for each parameter that appears in an operator specific subspace $\mathcal{R}(op)$ two entries in a look-up table will be created. These entries will indicate which operator $op$ and force $u$ have to be used in order to increase or decrease the parameter. Access to the look-up table will be via a function *look_up*.

$$look\_up : P \times \Delta \rightarrow U$$

where P is the set of all parameters in $\mathcal{R}(op_i)$ for some operator $op_i$, and

$$\Delta = \{+, -\}.$$

$\Delta$ indicates whether the value of $P$ should increase $(+)$ or decrease $(-)$. The output $U$ is the specific value which has to be chosen for the operator $op_i$ in order to achieve this change $\Delta$ of the parameter $P$.

This look-up table will be constructed by experimenting with different values for the operator $op_i$. The problem is to find a choice of $U$ such that the parameters will always change their values according to the look-up table description, while avoiding having forces which are so powerful that the system "overshoots".

Imagine the program tries to create the look-up table for some parameter $P$. Assume that $P$ is part of the operator specific subspaces of $op_1..op_k$. Further assume that $op_1..op_k$ are ordered by the number of parameters in their operator specific subspaces. Let $op_1..op_j$ be the operators with the smallest operator specific subspace containing $P$. Now pick the operator $op_c$ $i \leq c \leq k$ with the strongest impact on parameter $P$, which is the operator where a given change of force leads to the largest change in the values of $P$. Store one value $u_{\text{up}}$ for $op_c$ which in most states leads to an increase of the value of $P$, and store another value $u_{\text{down}}$ of $op_c$ which leads to a decrease in the value of $P$ in as many states as possible. The look-up table has then the following two new entries:

$$look\_up : P+ \rightarrow u_{\text{up}}$$
$$look\_up : P- \rightarrow u_{\text{down}}$$

The operator $op_c$ was chosen because it is "safe"[GE90] over the largest number of parameters[2]. At the same time it is the most powerful operator with this quality, and this should ensure that even in a critical situation that enough "power" is available in order to control the robot.

### 5.3.3 Limitations

There are many systems which can not be controlled by such a look-up table. Imagine a cart-pole system as it is depicted in Figure 5.1. Here the cart can be pushed to the left or to the right, and this action will influence four parameters: the position $X$ and velocity $X'$ of the cart as well as the inclination $\theta$ of the pole and its first derivative $\theta'$.

Clearly it is possible to choose forces which are so strong that they will always guarantee that some parameter changes in a certain way. Yet this might be too crude in order to ensure that the entire system doesn't overshoot drastically. If we want to control a cart-pole system based on such a simple look-up table controller we will need a more elaborate representation of the cart-pole system.

As an example let us look only at the inclination of the pole $\theta$ and its first derivative $\theta'$. The goal is to control the pole in such a way that it remains

[2]Applying this operator only affects a minimal number of parameters.

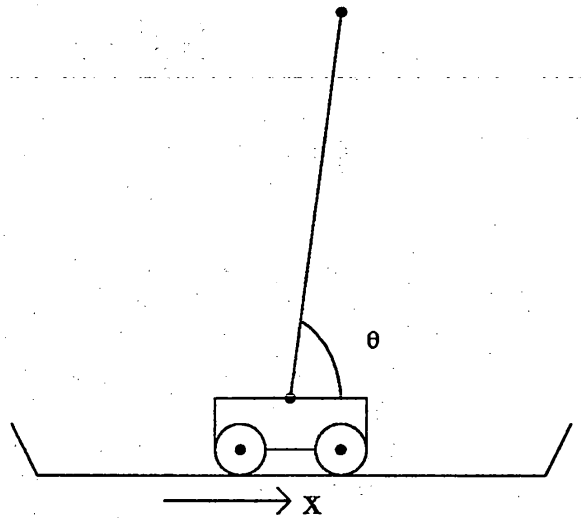



Figure 5.1: The inverted pendulum or "cart-pole" system. The cart-pole system is described using the position $X$ and velocity $X'$ of the cart as well as the inclination $\theta$ of the pole and its first derivative $\theta'$. The system crashes when either $X$ or $\theta$ exceed predefined boundaries.

upright with minimal angular velocity $\theta'$. It is possible to build a look-up table controller by introducing a variable $\Gamma$:

$$\Gamma = \theta * \theta'^{-1}$$

$\Gamma$ would then be equivalent to a very simple proportionate controller. The aim would be to keep $\Gamma$ equal to some constant value $C$, and depending on whether $\Gamma$ is greater or smaller than this constant $C$, a set force in one or the other direction is applied. Obviously it would be possible to introduce variables which correspond to more complex controllers, eg. a PID controller. But it is unreasonable to assume that such variables are present in the original description of the state space.[3]

Instead we will limit our attention to systems where a simple look-up table based controller of the type described above will suffice. In the later part of this thesis it will be shown that this is sufficiently general to achieve a robust control of biped walking. It is not within the scope of this thesis to investigate more general control strategies.

[3]Tony Morgan [Mor88] discusses this concept of control under the name of *qualitative control* in his thesis. He discovers the same limitations in as far as the plant has to be described by some variable $\Gamma$ which represents the "correct" control value in order for qualitative control to work in non trivial circumstances.

## 5.4 Hierarchical Organisation of the State Space

Imagine one has to operate an unknown device like a bicycle. Some domain-knowledge will be given (one has to sit on it and pedal, holding the handlebar). The application of the previous section leads to the understanding that pedalling produces speed, and turning the handlebar changes the direction. Yet we don't know how to use a bicycle. What we will do is to play around with these various commands and see what happens. We will realize that the distance to the next building is rather unimportant at the moment, but that keeping the balance is, for obvious reasons, extremely crucial. Thus, using the state space terminology we will separate the state space into a hierarchy of subspaces. The most important subspace will be the one immediately responsible for keeping the balance. Later on we will be concerned with questions like how to cycle around the block and other things. Thus we introduce a hierarchy of subspaces based on the *importance* of their control.

To emphasize the necessity for a hierarchical organisation of the state space let us consider the actual application. Since the biped is unable to control the position of the support-leg, it will fall in the direction in which the support-leg is pointing. Thus controlling a dynamic biped requires us to find start positions from which the non-controllable legs of the robot simply fall into the proper position within the time that it takes to adjust the controllable parameters of the system. If we search for such a position without reducing the search space, then we have to search 24 dimensions. If we have 4 possible values for each parameter we end up with $2.8 * 10^{14}$ possible start positions. If the environment becomes richer this problem increases exponentially despite the fact that the original basic problem, how to walk, remains the same.

### 5.4.1 Filtering out Dimensions on the Basis of Local Survival

In order to identify the above mentioned "important" subspaces, we have to find out which parameters may not be altered by even small amounts without changing the immediate behaviour. To detect this, we change the parameters one by one and observe the result.

Assume again that the state space $\mathcal{S}$ is represented as a $n$-ary tuple with dimensions $d_1..d_n$. Thus a state of the system $\mathcal{S}_0$ is described as

$$\mathcal{S}_0 = x_1, x_2, ...x_n,$$

and we create a (random) set of start states, $\mathcal{S}$:

$$\mathcal{S} = \mathcal{S}_0, \mathcal{S}_1...\mathcal{S}_p.$$

A control vector $\mathcal{U}_0$ has the form

$$\mathcal{U}_0 = u_1, u_2, ...u_m$$

and we create a sequence of (random) control vectors, a *command sequence* $\mathcal{C}_i$:

$$\mathcal{C}_i = \mathcal{U}_{i,0}, \mathcal{U}_{i,1}, ...\mathcal{U}_{i,k}.$$

The length $k$ of the command sequence should be constant, and $k$ should be chosen large enough so that a sequence of $k$ random commands almost inevitably leads to failure. In fact we create a whole set of command sequences $\mathcal{C}$:

$$\mathcal{C} = \{\mathcal{C}_0, \mathcal{C}_1, ...\mathcal{C}_l\}.$$

A set of *test-runs* or *tests* $\mathcal{T}$ is the Cartesian product of the set of start states $\mathcal{S}$ and the set of command sequences $\mathcal{C}$

$$\mathcal{T} = \mathcal{S} \times \mathcal{C}.$$

The test $\mathcal{T}_{i,j}$ describes what happens when starting from a start state $\mathcal{S}_i$ the command sequence $\mathcal{C}_j$ is applied. This means that starting in state $\mathcal{S}_i$ we consecutively apply the control vectors of $\mathcal{C}_j$. The system will go through a set of states $\mathcal{S}^0_{i,j}..\mathcal{S}^g_{i,j}$ before the test terminates. (In this case state $\mathcal{S}^0_{i,j}$ is identical with state $\mathcal{S}_i$. The indexing is extended to be consistent throughout the example).

$$\mathcal{U}_{j,0} \circ \mathcal{S}^0_{i,j} \rightarrow \mathcal{S}^1_{i,j}$$

$$\mathcal{U}_{j,1} \circ \mathcal{S}^1_{i,j} \rightarrow \mathcal{S}^2_{i,j}$$

...

$$\mathcal{U}_{j,g} \circ \mathcal{S}^g_{i,j} \rightarrow \perp(Crash)$$

or, if $g = k$ and $\mathcal{S}^{g+1}_{i,j}$ is a proper state of the system,

$$\mathcal{U}_{j,g} \circ \mathcal{S}^g_{i,j} \rightarrow \mathcal{S}^{g+1}_{i,j}$$

which should be very unlikely if $k$ is chosen large enough and the system is sufficiently unstable.

Finally either the system will crash after $g$ steps before all commands in the command sequence could be executed ($g \leq k$) or it will end in a proper state $\mathcal{S}^{k+1}_{i,j}$. If we choose $k$ reasonably large then we can assume that the system will crash after state $\mathcal{S}^g_{i,j}$ due to the dynamic nature of its behaviour and the

randomness of the commands. We will call $g$, the number of states the system went through, the *survival time* of the test $\mathcal{T}_{i,j}$.

In the second phase of the experiment we will run the test-runs again, this time modifying the start states $\mathcal{S}_i$ slightly. To be more precise each test $\mathcal{T}_{i,j}$ will be altered in the following way: for each dimension $p$ of the state space try to alter its original values in the start state $\mathcal{S}_i$ resulting in a modified state $\mathcal{S}_i^p$.

$$\mathcal{S}_i = (x_0, x_1, .., x_p, .., x_n)$$

$$\mathcal{S}_i^p = (x_0, x_1, .., \Delta(x_p), .., x_n)$$

The amount $\Delta$ by which a parameter is modified is defined as a fraction of the range of the parameter. If during an average random test a parameter changes its value from, say, 0 to 10, then $\Delta$ would be chosen as a random value from this interval. This is necessary in order to ensure that the changes $\Delta$ correspond to the impact of some motor commands. If a parameter was changed by some abnormally large number then the program would measure the sensitivity to changes which in practice would not occur.

For all these new states $\mathcal{S}_i^p$ the tests $\mathcal{T}_{i^p,j}$ are run, which means repeat all the previous tests, using the new start state, but keeping the command sequences the same. The resulting behaviour will be

$$\mathcal{U}_{j,0} \circ \mathcal{S}_{i^p,j}^0 \rightarrow \mathcal{S}_{i^p,j}^1$$

$$\mathcal{U}_{j,1} \circ \mathcal{S}_{i^p,j}^1 \rightarrow \mathcal{S}_{i^p,j}^2$$

...

$$\mathcal{U}_{j,f} \circ \mathcal{S}_{i^p,j}^f \rightarrow \perp(Crash)$$

If the survival time $f$ of the test $\mathcal{T}_{i^p,j}$ is equal to the survival time $g$ of $\mathcal{T}_{i,j}$ then the dimension $p$ modified in test $\mathcal{T}_{i^p,j}$ does not appear to have too much influence on the behaviour of the system. If this is the case for a large number of tests then one can assume that this dimension is redundant[4]. If $f$ and $g$ differ

[4]Redundant here is with respect to the task to survive in an *average* environment. If the dimension has nothing to do with the stability of the robot but is nevertheless needed for some high level behaviour, then it clearly shouldn't be considered to be redundant. Just assume the biped robot had a parameter indicating whether it was moving towards some goal. This parameter would be very helpful for some high level search behaviour but wouldn't have any influence on the biped's stability.

considerably, then the dimension $p$ modified in test $T_{iP,j}$ seems crucial for the immediate survival of the system. Therefore it is suggested to order the dimensions of the state space depending on their average impact on the survival time. Thus we can order the dimensions of the state space by the *average deviation of their survival times.* We expect to find in the most important group those parameters that are crucial for the local control of the system. In the example of the biped robot we expect these values to be the ones that describe the biped itself.

### 5.4.2 Discussion of the *Survival-Time*-Heuristic

The underlying assumption for the above mentioned heuristic is that there are *large* subspaces in the state space where it is possible to control the system *as if some dimensions of the state space didn't exist.* In this case a random position will on average be located in one of these large subspaces, and the survival time heuristic will identify the independence from the values of some other dimension outside this subspace. Despite the fact that the program doesn't know yet how to control the system it can assume that in most situations certain dimensions can be disregarded. In order to facilitate the search for a control of the system it is therefore useful to store elements of such subspaces. Later test runs should start in such "corners" of the state space rather than in one where the situation becomes complicated.

Since all dimensions can be tested individually the program is able to find the inner state space in linear time with respect to the number of dimensions of the state space. Experiments at the end of this chapter will show that in fact about five tests are sufficient to detect the inner state space for the biped robot.

One other approach which tries to tackle the same problem is the *genetic algorithm* paradigm. One could interpret a random command sequence and the state space in which it is applied as a *classifier* in a classifier system. Over time this classifier would gain a certain fitness (if it does something that is rewarded with a high score in some given evaluation function), and cross-over would preferably happen with other successful classifiers which have a high number of "don't care" entries. Such an entry would indicate that the corresponding dimension of the state space is considered to be irrelevant. Thus over time one would hope to filter out the less relevant dimensions of the state space.

The main difference between a search using genetic algorithms and the approach discussed in Section 5.4.1 is the fact that one wouldn't have to wait for the mutation and cross-over commands to generate a suitable classifier which contains the "don't care" entries at the right place. Instead such classifiers are generated systematically. Since we know what we're looking for we might

as well do this straight away rather than hoping that some search gets us there sometime. We also don't have to try various cross-over combinations of successful classifiers since we know that we can simply "add" the redundant combinations. The cross-over operator in the genetic search paradigm creates a random split in two classifiers and then recombines them. In this terminology we simply weighted the *alleles*[5] and made sure that the crossover is not at random but works like a hillclimbing function. We can do this because we don't search for arbitrarily complicated hyperplanes in the state space but rather test and eliminate single features individually.

## 5.5 The Application to Biped Walking

In this section we will apply the above presented algorithms to biped walking. Using a simulated model of a biped robot we will develop the operator specific subspaces for this robot. Following this we will test the survival-time heuristic in order to identify relevant dimensions and to weigh them according to their importance for the survival of the robot. For the parameters which will be identified as members of the inner state space a look-up table for their control will be created.

### 5.5.1 Representing the Biped and its Environment

The effects of the state space reduction techniques will be demonstrated using the previously discussed dynamic biped robot in an "enriched" environment. In the actual application we will use a state space of 24 dimensions:

- The biped robot itself will be characterised using 12 parameters: $\theta, \phi, \psi, \eta$ and $\zeta$ and their first derivatives describe the angles of the joints of the robot and their velocity. *Footheight* and *Hipheight* describe the height of the foot and the hip. Note that the last two parameters are redundant and can be derived from the angles of the joints. They will therefore be omitted from the search for operator specific subspaces as well as the inner state space.
- The environment will be a plane in which four different obstacles can appear: (1) a wall, (2) a ditch, (3) a step and (4) a slope. The parameters $d_1, d_2, d_3$ and $d_4$ describe the distance between the robot and the corresponding obstacle. Heights $h_1, h_2, h_3$ and $h_4$ and widths $w_1, w_2, w_3$ and $w_4$ describe the height and the width of the obstacle. The "height" of the slope, $h_4$, will be the actual slope of the surface.

[5] *allel*: a substring in a classifier, typically identified with a "concept" in the state space.

If the environment has several obstacles of one of the types described above then the convention will be that the parameters refer to the closest obstacle of this type.

Thus we added another 12 dimensions to the state space, of which some are clearly irrelevant: a wall always has a fixed width, the depth of a ditch is irrelevant as is the length of a slope.

### 5.5.2 Operator Specific Subspaces of the Biped Robot

The above mentioned algorithm for the detection of operator specific subspaces was tested on the biped robot. It worked as predicted and was immune against small amounts of noise provided two conditions were met: (1) the noise had to be below the filter threshold $\theta$ *and* (2) the filter threshold $\theta$ had to be small enough so that relevant differences between two states were not filtered out.

It was examined how many test runs were needed in order to detect the operator specific subspaces for each operator $u_1, u_2, u_3$ of the biped robot. Several runs were needed in order to identify the subspaces correctly due to the fact that the robot collapsed frequently after just one or two random commands and therefore only small differences could be noticed. Between 40 and 80 percent of the tests discovered the correct set of parameters in $\mathcal{R}(op)$. Thus after 10 tests there is almost a 99 % probability of having identified the complete set of parameters influenced by a given motor command.

### 5.5.3 The Inner State Space of the Biped

In a second set of experiments the state space hierarchy for the biped robot was investigated. The aim of the experiments was twofold: first to discover the parameters of the inner state space, and second to find the number of tests needed in order to identify these parameters.

The experiments were set up as follows: a series of tests was organised, and a threshold $\theta$ was defined. Each test changed all the parameters individually and recorded the change of the survival time for each changed parameter. Successive tests (ie. a test-series) were carried out. If the average change of the survival time (with respect to the already executed tests) was above the threshold $\theta$ then this parameter was included into the inner state space.

The results of these experiments may be seen in Figures 5.2 to 5.5.

Each of these figures corresponds to a different filter threshold $\theta$. In Figure 5.2 a threshold of $\theta = 0$ was used. Thus we expect all parameters which have a pos-

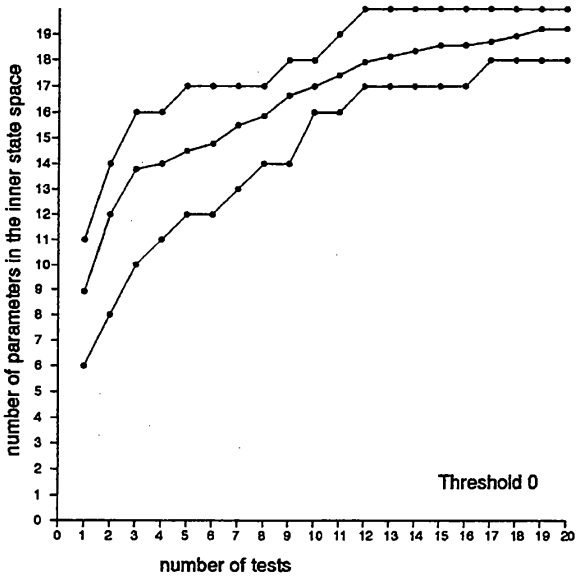



Figure 5.2: Analysis of the State Space Hierarchy, $\theta$=0. Any effect on survival time (average changes greater than threshold 0) were recorded. On average 19 parameters were identified which had an influence on the survival time of the robot. These parameters are the kinematic and dynamic parameters of the robot as well as the relevant parameters of the various obstacles.

sible influence on the robot's survival to be included in the inner state space. This was achieved after about 20 steps. As expected the number of parameters falls quickly if a threshold greater then 0 is used. It is also noticeable that only a few tests are needed to detect parameters with an average impact on the survival time which is above 1.5.

Each graph indicates how many parameters were identified as having an impact on the behaviour of the robot. A threshold $\theta$ is used to filter out those parameters which have an influence on the survival time less than $\theta$. Each graph plots the number of identified parameters against the number of tests needed in order to identify the parameters. Each graph contains 3 lines: the upper line indicates the maximum number of identified parameters, the middle line the average, and the lower line the minimum number of identified parameters. These three lines are plotted in order to show the variation between 10 different test series.

The results show clearly that the inner state space can be identified after relatively few tests, about 20 seem to suffice. The tests also reveal that the velocities of the angles can be disregarded at present, a feature which simplifies the search significantly. We expected to "loose" all the environmental features of the biped (the information about the obstacles) but also gained the insight

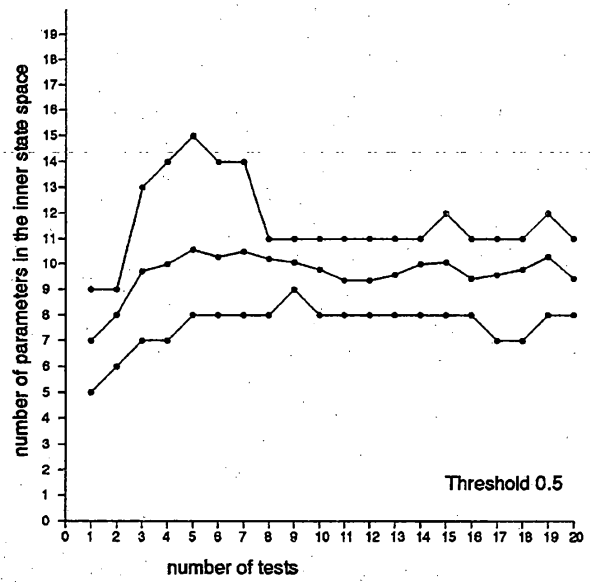



Figure 5.3: The same test as above, this time only those parameters with an average impact on the survival time greater than 0.5 are recorded. This already filters out almost half the parameters (the obstacle parameters). About 8 test runs are needed in order to establish this result. The 10 detected parameters are the dynamic and kinematic parameters of the robot.

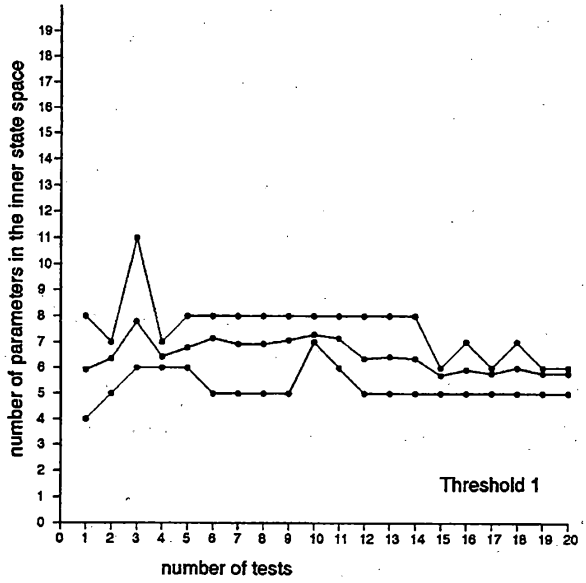



Figure 5.4: The threshold is increased to 1. About 6 parameters remain above the threshold, some first derivatives are already filtered out. The remaining parameters are the kinematic parameters of the robot and the velocity of the free leg around the pitch axis.

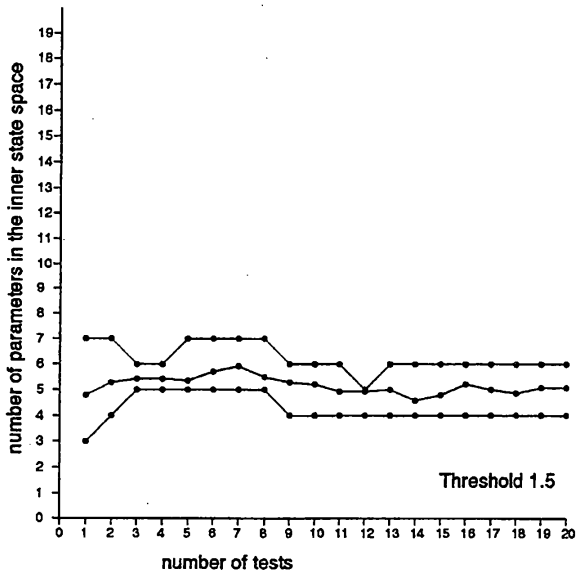



Figure 5.5: The threshold is increased to 1.5. On average only 5 parameters have an average impact on changes of the survival time above 1.5. These parameters coincide with the 5 angles describing the kinematics of the robot. The first derivatives of each angle have been filtered out.

that the first derivatives (and therefore another 5 dimensions) seem to be irrelevant for the immediate survival of the robot. This seems to be due to the fact that the encountered velocities were not high enough to bring the robot into immediate danger. The displacement of a leg (and therefore the effect on the survival time) due to a different joint position was higher than the (final) displacement of the leg due to an increased velocity.

Thus (after including linear dependent parameters) we are left with an inner state space of 7 dimensions, namely $\theta, \phi, \psi, \eta$ and $\zeta$ as well as $Footheight$ and $Hipheight$ which are linearly dependent on the first 5 parameters. We will use the average variation of the survival time $\sigma$ of each parameter as its weight in the search through the state space. The next chapter will show that this weighting produces an efficient way to search the state space, and using this weighting trajectories can be found which enable the robot to walk.

# Chapter 6

# Searching for Gaits: Planar Dynamic Walking

## 6.1 Introduction

This chapter will be concerned with the development of simple planar gaits for the robot. This means that the robot walks in a horizontal, obstacle free plane.

In order to develop these gaits the robot searches its state space for a set of connected states which represent a step into some direction. The states describe the numerical values of the parameters of the robot, and the directed connection between two states is the set of motor commands that enables the robot to change from the first state to the second state.

To carry out this search the robot uses a simple hillclimbing mechanism: based on some *start state*, a *goal state*, a *set of operators*, and an *evaluation function* (a mapping from the state space and the goal to the real numbers), the robot applies all operators individually to the start state. It then uses the evaluation function to evaluate each of the resulting states, the state with the smallest evaluation result will be interpreted as the one closest to the goal. Using this state the search then continues until it reaches a state from where on the evaluation results become worse rather than better. Once the robot reaches such a state (a local minimum) the search stops and the robot assumes it has finished its step.

Various aspects of this search will be discussed in this chapter: what are the operators; where do the start states come from; what is the goal of a step and how do we evaluate the distance of a state to the goal. Applying all this to the biped, we will demonstrate how the biped walks over a planar surface, and

how it can change the direction into which it is going.

Once the robot has successfully searched for various types of steps, the program will analyse the properties of these steps. It will extract those steps which it considers to be interesting. Interesting steps can be those which perform very regular movements or create a comparably large step forward or backward. The program will then take a more detailed look at these steps and try to refine them. As a result, the program will generate combinations of start positions and operator sequences which represent an "interesting" behaviour.

We will investigate criteria to select successful combinations of start positions and operator sequences and store these combinations of positions and operator sequences as gaits.

## 6.2 The Search

The aim of this section is to enable the robot to walk on a horizontal plane. Since in the beginning we have very little knowledge about how to do this we will originally have to *search* for postures of the biped and motor command sequences which will result in a series of steps forward. But, clearly we do not want to use this technique (search) all the time. We want the program to discover complete *gaits* which it can record and use whenever they are appropriate. A few definitions follow in order to avoid confusion over the terms used in this chapter:

**Definition 1** *The* **posture** *of a system is an instantiation of the parameters of the inner state space. Thus, the posture of the biped is a set of values for the parameters $\eta, \zeta, \theta, \psi$ and $\phi$. All other parameters are set to default values. The default values are a specific position in the plane and zero velocities for all limbs.*

**Definition 2** *The* **start position** *of a gait is a posture in which the robot can execute a support exchange command, i.e. it can shift its weight from one leg to another. Normally, this is achieved by ensuring that both feet touch the ground.*

**Definition 3** *A* **gait** *is a pair consisting of a start position and a set of motor commands. The set of motor commands starts with a support exchange command. If the system is in a state specified by the start position then applying the motor commands should get the system into another start position.*

So, when we are looking for gaits we are looking for *macro operators* which correspond to one step forward.

### 6.2.1 The Operators

The operators which make the robot change from one state to another are defined by the torque which is applied to each of the controllable joints of the robot and the time duration for which this torque is to be applied. The original specification of the robot does not constrain the amount of torque which can be applied to any controllable joint of the robot at any time, nor does it require a certain duration of such a torque application.

However practical considerations require that there is only a small set of motor commands (or torques applied to the joints of the robot) available. This will decrease the branching factor of the search. By extending the length for which a certain torque (motor command) is applied, the depth of the search becomes limited.

Thus the first task for setting up a feasible search program is to calibrate the torques and the time for which they are to be applied. When searching for steps a time slice of 0.04 seconds was chosen. In Chapter 4 the parameters for controlling the biped were introduced: $u_1$ is the torque applied to the free leg around the pitch axis, $u_2$ is the torque applied to the hip, and $u_3$ is the torque applied to the free leg around the roll axis. The motor commands were calibrated in such a way that for each of the three motors, three commands were available: one to keep the parameters controlled by this motor in about the same position, and one each to increase and decrease the value of the controlled parameters.

This resulted in a torque of 0.1 Nm for $u_1$, 0.6 Nm for $u_2$, and 0.0 Nm for $u_3$ in order to keep the parameters "neutral". In order to change the values of the controllable joints the values of $u_1, u_2$ and $u_3$ can change by $+/-0.05\ Nm$, $+/-0.15\ Nm$, and $+/-0.05\ Nm$ respectively. On average these torques were strong enough to get any joint into its required position within 0.2 seconds.

As a result, the search has a branching factor of 27 (3 commands for 3 motors), and an average depth of about 5, which (if the search space was exhaustively enumerated) would result in 14348907 different states. Clearly, the exponential nature of the search space prohibits an exhaustive search.

### 6.2.2 Creating Start Positions

Since we are looking for gaits, we have to start by looking for start positions. The behaviour of the biped is dependent on its posture. If it is leaning to one direction, then it will fall in the same direction, and the speed of the fall will depend on the angle with which it is leaning. Because the program does not know which start position to choose, it will enumerate as many start positions as economically feasible. Since the posture of the biped determines its behaviour, we would like the postures to be as different as possible. This way, we hope to detect as many different behavioural patterns as possible.

The current implementation of this search for the dynamic biped creates 392 different start positions. In order to achieve this the range of each parameter is split into a set of intervals. The start positions are then created from the Cartesian product of the intervals of each parameter.

The selected range for each parameter was as follows (units in radians):

$\theta$: -0.15..-0.08, -0.08..0.0, 0.0..0.08, 0.08..0.15
$\psi$: -0.2..-0.05, 0.05..0.2
$\phi$: -0.2..0.0
$\zeta$: -0.2..-0.1, -0.10..-0.05, -0.05..-0.01, -0.01..0.01, 0.01..0.05, 0.05..0.1, 0.10..0.2
$\eta$: -0.2..-0.1, -0.10..-0.05, -0.05..-0.01, -0.01..0.01, 0.01..0.05, 0.05..0.1, 0.10..0.2

The emphasis was to generate many different combinations of values for $\eta$ and $\zeta$. These two parameters determine how far the robot leans "forward" and thus the direction into which the robot is walking. $\theta$ was also given a larger set of intervals because of the fact that it would be necessary to synchronise the movements around the roll and the pitch axis. $\phi$ has been restricted to one interval in order to keep the size of the Cartesian product small enough.

Since the requirement for a start position is that the robot must be able to execute a support exchange command, these interval combinations were then adjusted accordingly. In this case, this meant that a simple search program tried to find a set of parameter values, such that the instantiation of each parameter was within the boundaries defined by the interval and both feet touched the ground (which is the precondition for a support exchange command).

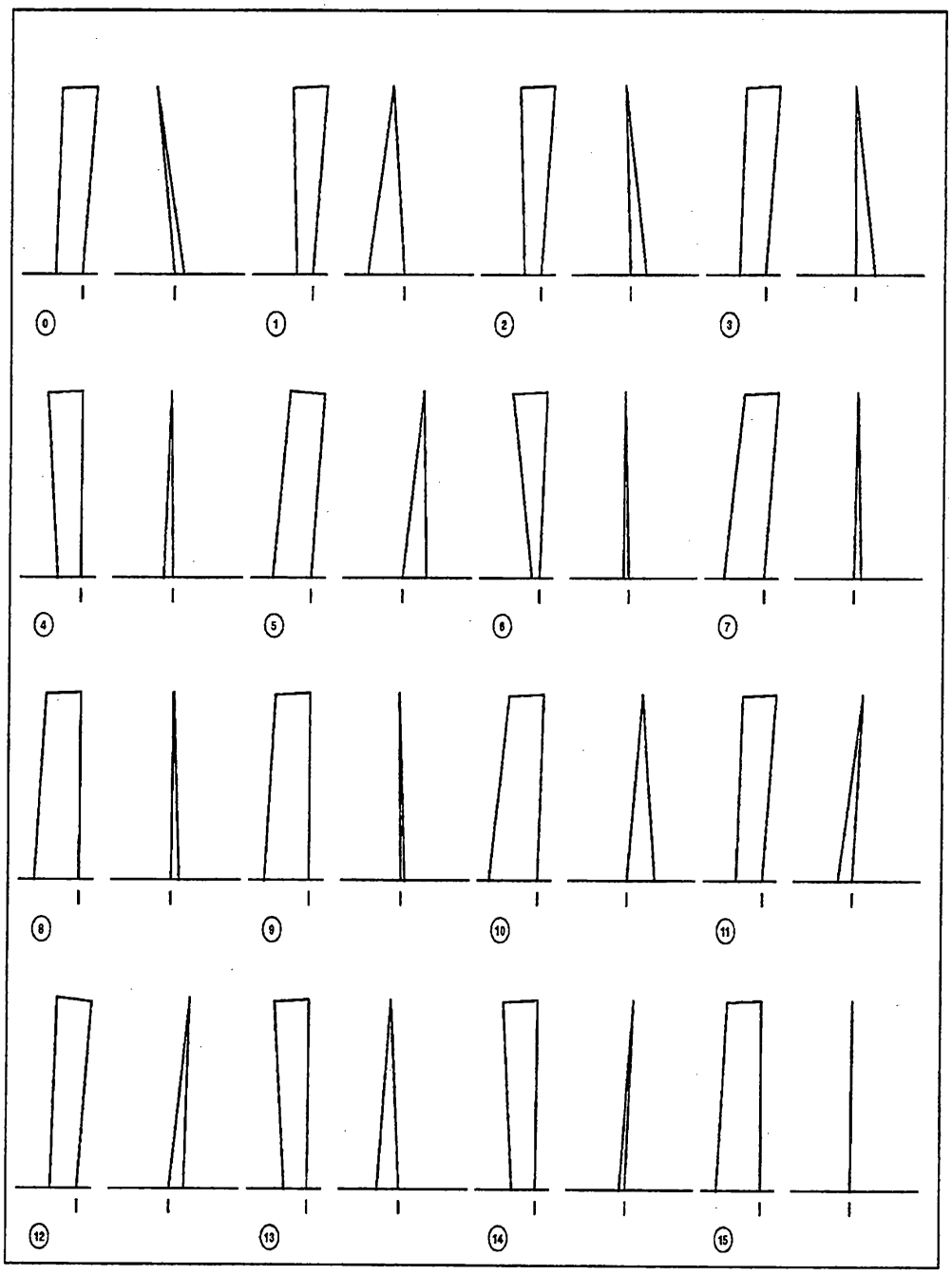



Figure 6.1: Some of the 392 start positions used during the trials.

### 6.2.3 Creating Goal Positions

By now, we have the first element of a gait, the start position. The next step is to find a set of motor commands which bring the biped into another start position (this is the definition of a gait). Clearly, there are plenty of positions in the state space where the biped can execute a support exchange, namely, all those positions where it has both feet on the ground. But, we are not interested in just some new start position. We would like to find a new start position which is as similar to the original start position as possible. This way we would end up with a repeatable behaviour: if after the application of a command sequence the system ends up in a state very similar to the one from where it started then we could re-apply the same command sequence with the same result. In other words, the behaviour of the system would be *linear*.

Thus, the ideal result of the search for gaits is to find a sequence of operators that lead to a cyclic behaviour. Starting at some state, the robot will eventually end up in a place which is identical to the start state except for a displacement on the surface. Therefore, it seems reasonable to search for gaits which are cyclical with respect to all parameters in the inner state space. The goal of the search will be the values of the parameters of the inner state space at the start position.

This would, of course, lead to a very serious problem for the hillclimbing search algorithm, since the robot will start the search in a position identical to the goal. Domain knowledge will be introduced at this point to make the hillclimbing search work: the robot starts the search with a support exchange command. Thus, all parameters will swap their values across the symmetry axes of the robot. The net effect of this command is that the robot is forced to *step*, ie. to alternate between the legs on which it is standing.

### 6.2.4 Searching for Steps and the Weighting of the State Space

The search technique is rather simple: a start position is taken and a support exchange command is applied. Then we search (using hillclimbing) for a path back to the original start position. The success of this technique depends on the complexity of the underlying search problem. If the search space is complex, then this could be the place where hand-crafted heuristics or other meta-knowledge need to be incorporated. Fortunately the search problem proves to be simple in the case of biped walking.

All that is needed is simple hillclimbing search. The goal is defined by the parameter values of the inner state space at the start position. The proximity

to the goal is evaluated as the weighted sum of the squares of the errors. An error is the numerical difference between the actual value of a parameter and its value in the goal state. The weights in the hillclimbing search are the weights discovered by looking at the average deviation of the survival time (as described in the previous chapter). All parameters of the inner state space are weighted according to their impact on the average deviation from the average survival time.

This heuristic weighting proved to be excitingly successful and enabled the search program to reliably search for gaits. In 35 % of the previously generated start positions it was able to successfully search for motor commands moving the robot several (20) steps forwards. An improvement in the search algorithm made it possible to increase the success rate to 58 %. This improvement was achieved by backtracking over the last operator only: if the robot goes through states $A \rightarrow B \rightarrow C$, and can't go any further from state $C$ (all operators lead to failure), then it tries another operator in state $B$. However, it does not backtrack further. If all transitions from $B$ lead to failure then the system stops and reports failure.

From these successful start positions a position was chosen, where the generated behaviour was regular (constantly low evaluation of the position at the end of each step) and generated an above average movement forward. This position is illustrated in Figure 6.2 together with the corresponding sequence of steps. The robot walks from right to left. The frames show the robot with a frequency of 0.04 seconds. An average step takes about 0.2 seconds, which is similar to the gait used by Miura and Shimoyama (they used a frequency of 0.25 seconds per step [MS84]).

It will be interesting to see to what extent the weighting of the state space is important for the success of the search. It might be the case that almost any simple weighting of the state space would lead to an acceptable search behaviour. In the remainder of this section various evaluation weights will be tested.

A set of evaluation weights will be tested using the same start position and then searching for a series of steps forward. Each test proceeds as follows: a fixed start position, and a set of weights for the evaluation function (the test set) are chosen. The robot searches for steps using the test set of weights for the search. Each time the robot finishes a step, a dot (the footstep) is plotted on the surface. The test is finished when either the robot walks a certain distance or it fails.

By plotting the footsteps resulting from the use of different evaluation functions next to each other it will be easy to see where the robot failed, and if it didn't fail one can compare the regularity of the steps.

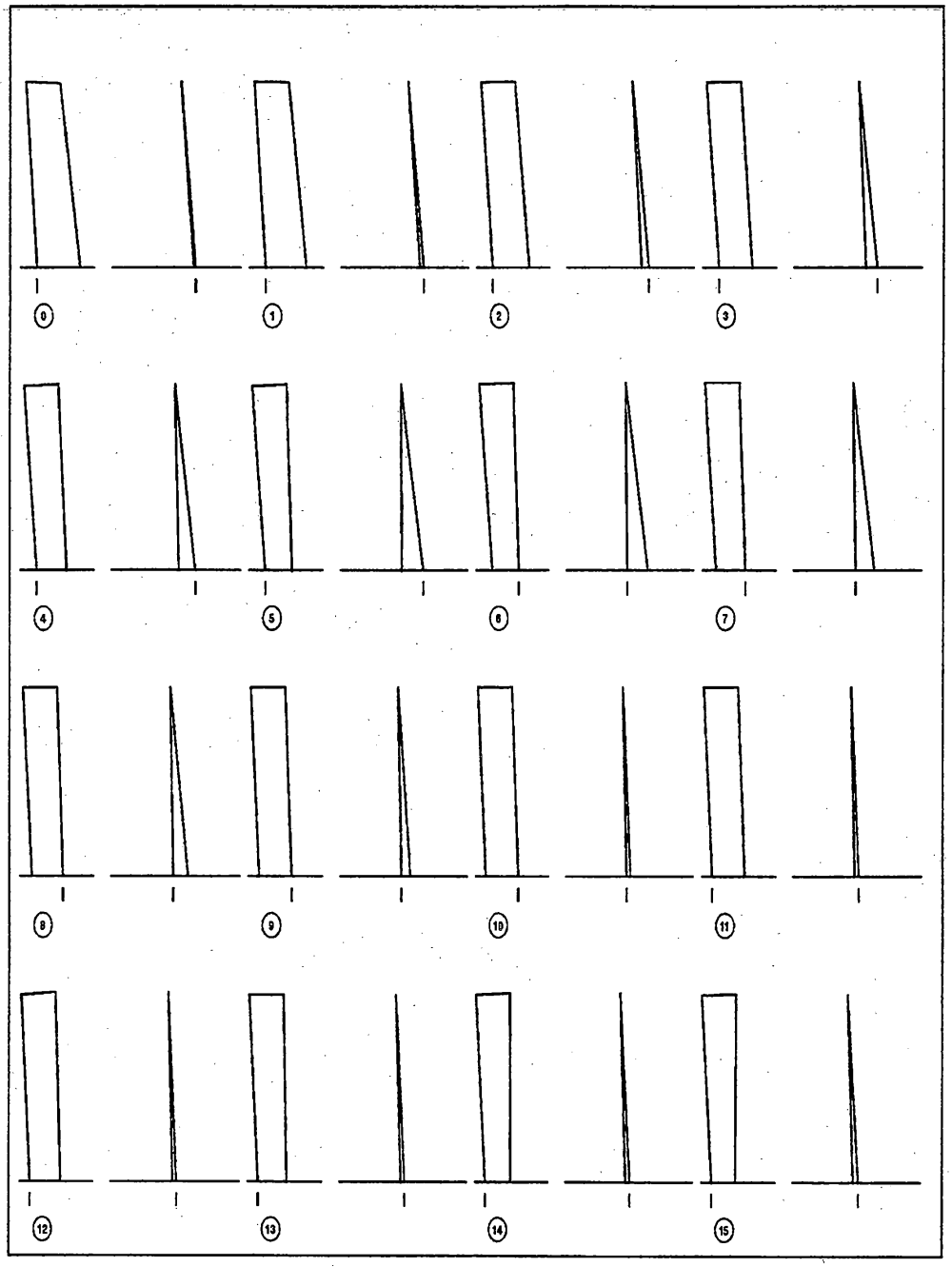



Figure 6.2: The robot walks from right to left. The small vertical line underneath the robot indicates the support leg of the robot. The robot completes a step when it changes from one foot to another. A support exchange command is carried out in positions 5, 10 and 15. Thus, the length of each step is 0.2 seconds, and the robot moves in fast short steps. On average a step is about 1.2 cm long.

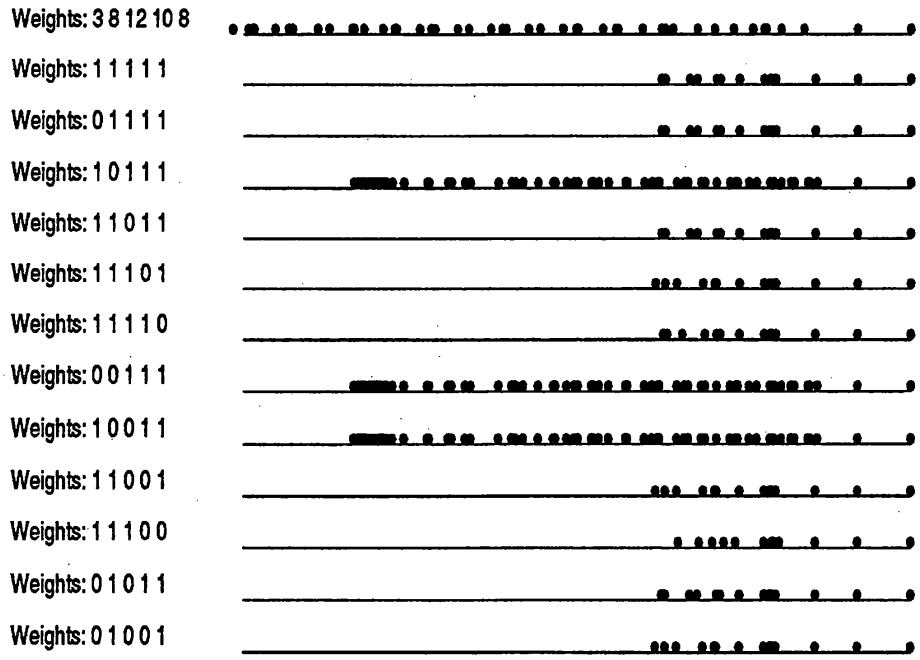



Figure 6.3: The performance of different inner state space weightings. Simple weights (1 or 0) were chosen and the robot moves from right to left. The dots indicate the position of the robot at the end of a step. The robot is successful if it reaches the left end of the line. The weights are (from left to right) the weightings of $\eta$, $\zeta$, $\theta$, $\psi$, and $\phi$ respectively. The robot is only successful when it uses the weights gained from the survival time heuristic (top row).

Various state space weightings were chosen: the weighting obtained from the sensitivity analysis of the state space (average deviation of the survival time), random weightings of the parameters in the inner state space (all other parameters weighted zero), constant weightings of individual parameters, random weightings of all parameters. These tests were then repeated for a number of start positions.

The results are shown in the Figures 6.3 to 6.5. For each of these figures one constant start position has been selected. In each row, a search for a series of steps forward starts from this start position. For this search the weighting of the state space shown to the left of the row is used. The resulting search is indicated by the sequence of dots. Each dot represents the position of the biped at the end of a step. The end of the sequence of dots either indicates failure (if this is in the middle of the line) or success (if the dots continue until the end of the line).

The results of these tests are very encouraging: the weighting of the state space, as derived from the survival time heuristic, provides the search program with an efficient and successful weighting to search the state space of

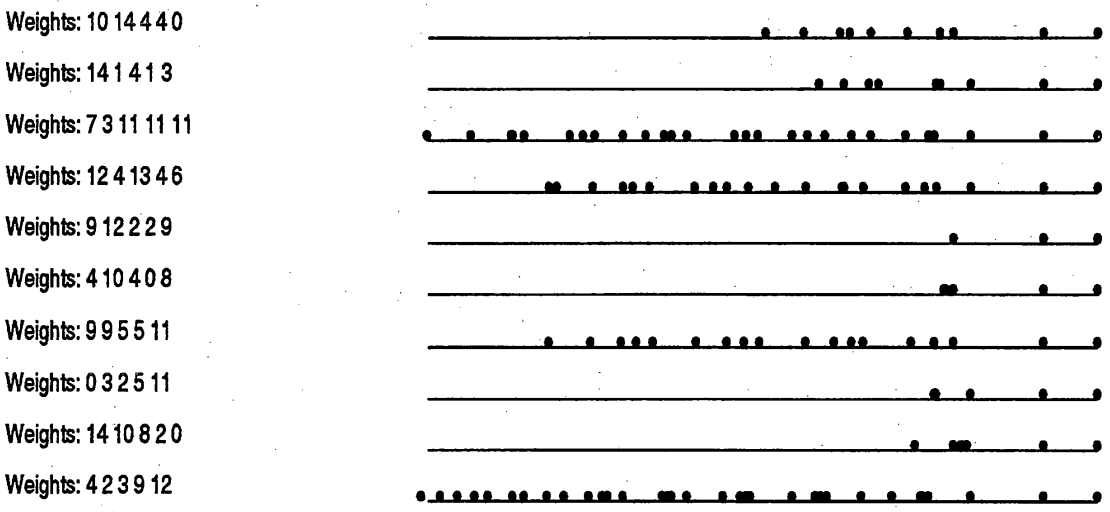



Figure 6.4: The performance of different inner state space weightings. Random weights were chosen and the robot moves from right to left. The robot is successful if it reaches the left end of the line. The weights are (from left to right) the weightings of $\eta$, $\zeta$, $\theta$, $\psi$, and $\phi$ respectively. Two inner state space weightings were successful.

the biped. Clearly, one can see that a state space which includes the dynamic model of the biped (all robot parameters and their velocities) leads to unsuccessful search. Whereas, a search based on the weighting of the kinematic model of the robot (excluding the velocities) leads to a successful search for a series of steps. At the same time Figure 6.3 shows that the search for these weights is not necessarily trivial.

The reason for the failure of a search which includes a weighting of the dynamic properties of the robot seems to be as follows: if the velocities of the limbs are as important as the position of the limb, then there is a very strong danger that the velocity of the limbs "overrides" the kinematic dimensions. As a result, the robot is less likely to end up in a position similar to the original start position, and the uncontrollable parameters (the fall of the robot over the support leg) change considerably from step to step. The performance of the robot degrades quickly since the uncontrollable parameters do not fall into the correct position. This is reflected in the survival time analysis which shows the high sensitivity of the robot to a change of its kinematic parameters.

Randomly chosen normal distributed evaluation weights of the inner state space were tested for failure (e.g. the biped crashes into the ground) during the first 200 steps. 27 weightings (out of a total of 50) lead to failure, and 11 weightings lead to a behaviour comparable with the behaviour produced by

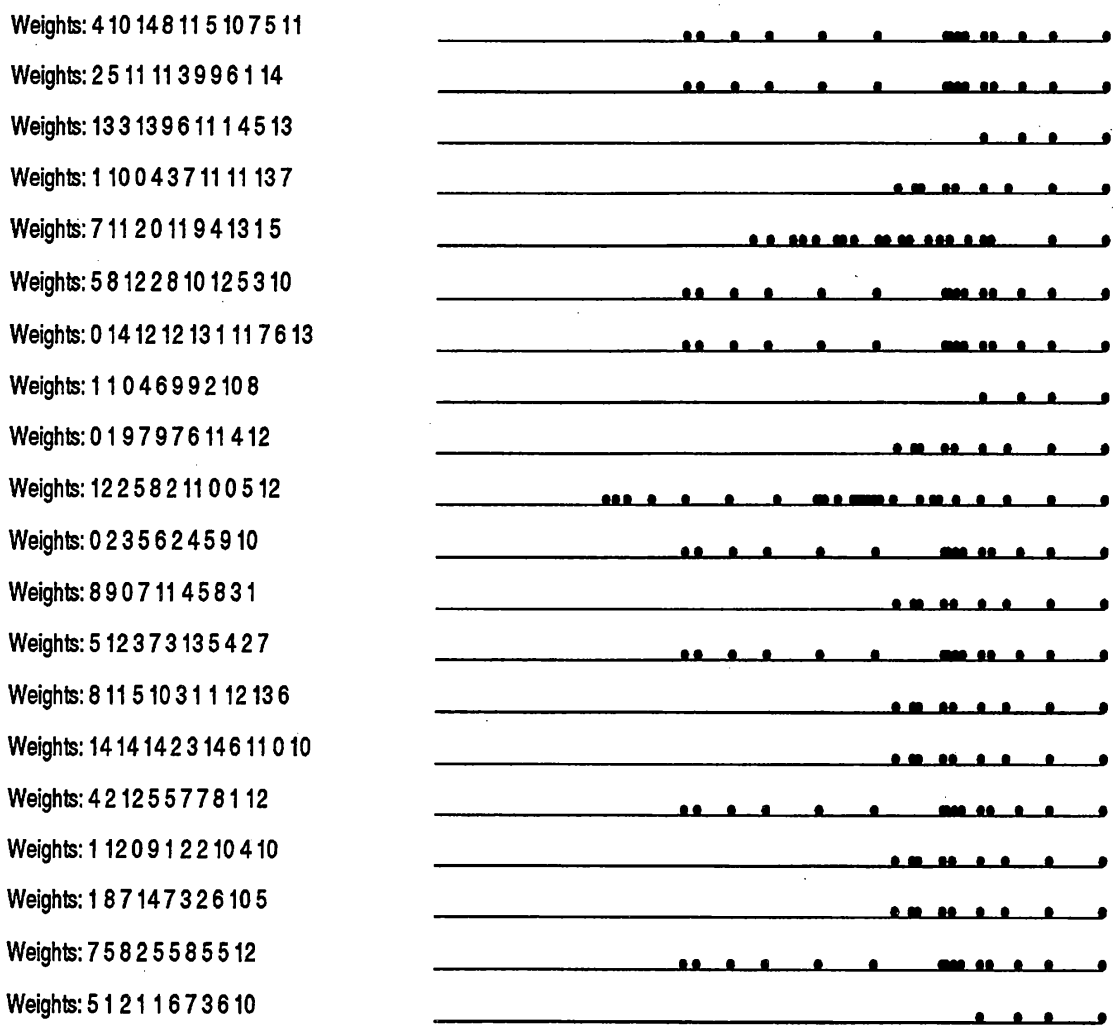



Figure 6.5: The performance of different random weightings of the robot's dynamic state space (the parameters describing the robot and their first derivatives). The weights are (from left to right) the weightings of $\eta$,$\eta'$, $\zeta$, $\zeta'$, $\theta$, $\theta'$, $\psi$, $\psi'$, and $\phi$ $\phi'$ respectively.
Random weights were chosen and the robot moves from right to left. The robot is successful if it reaches the left end of the line. None of the weightings was able to lead to a successful search for new gaits.

the original weighting derived from the survival time heuristic. Five of these successful weightings are just minor modifications of the original weighting (the relative size of the weights remained preserved).

This analysis shows that the weighting of the inner state space gained from the survival time analysis is not unique in terms of enabling the system to search successfully. But choosing these weights produces a considerable speed-up in terms of finding the proper weighting of the evaluation function in comparison to using random weightings of the inner state space.

## 6.3 Discovering Gaits

At this point, the start and goal states for the search have been defined, the weighting of the state space used in the evaluation function is known, and the operators leading from one state to another have been defined. Since the start position determines both start state and goal state of the search, and since the weighting of the state space and the fixed set of operators results in a deterministic search procedure, the search for steps is only dependent on the choice of the start position.

Testing 392 different start positions, the program started to search for successful steps. For each start position the robot searched for 200 consecutive steps and recorded the properties of these steps (distance covered and evaluation of the posture of the robot at the end of each step). All start positions from which the robot failed before completing 200 steps were deleted. The resulting start positions and the associated steps were sorted using a combination of the regularity of the movements and the resulting displacement forwards or backwards.

Using these properties several start positions were chosen: a start position from which the robot finds a regular gait forward and a start position from which the robot finds a fast (but not necessarily regular) gait forward. Similar start positions were found for backward movements. The program also stored the start position from where the most regular gaits were found.

Figures 6.6 to 6.12 show how the biped walks if it tries to execute the search for steps with these positions as goal positions. Additionally, Figures 6.9 to 6.12 shows that the search is powerful enough to turn the robot and make it change its direction. Some of these steps will form the basis for the generation of macro operators (gaits), which will allow the execution of a step without the need for search.

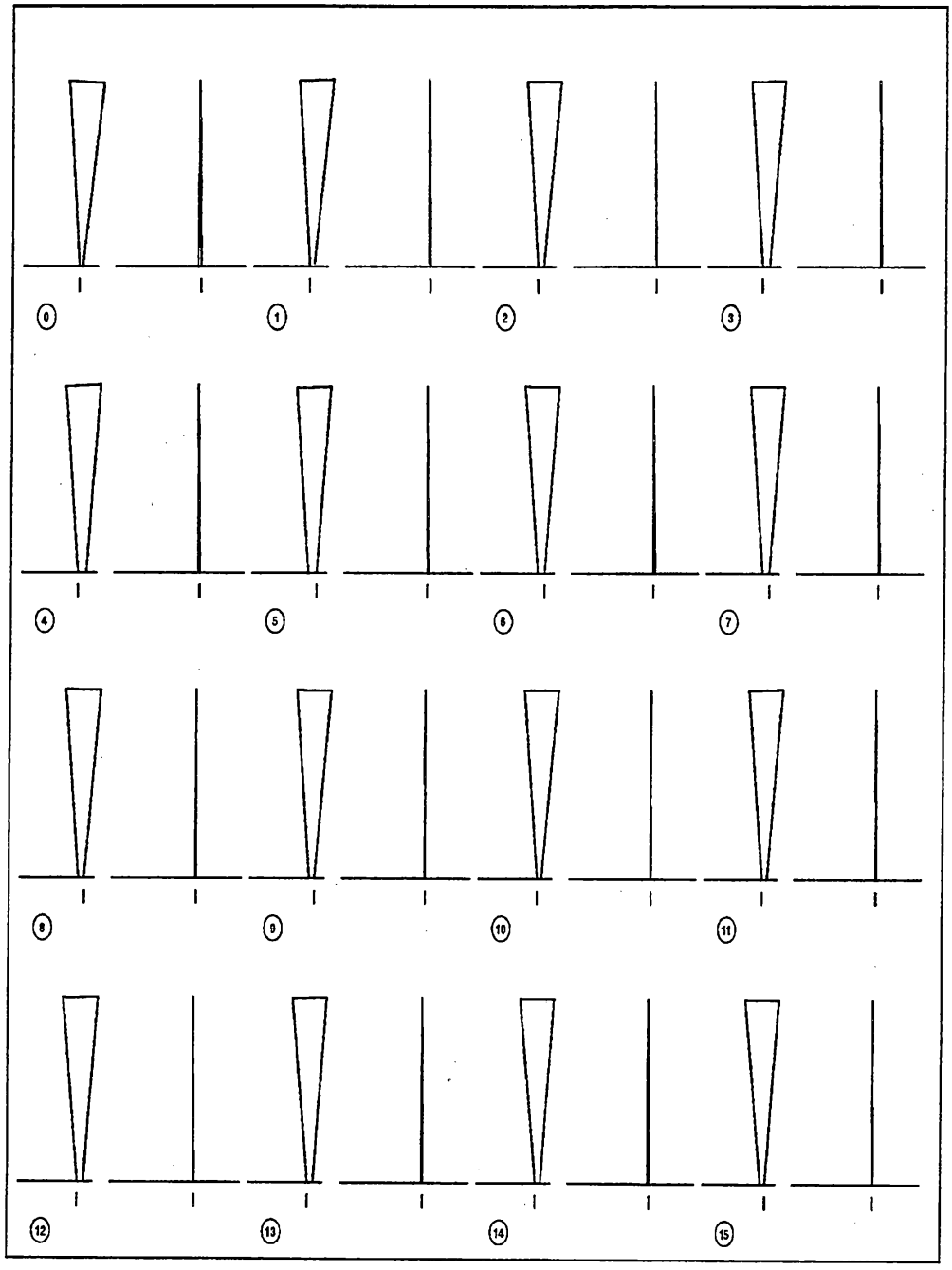



Figure 6.6: The start position selected for the most regular gait. The robot is very well balanced and there is very little movement between the frames.

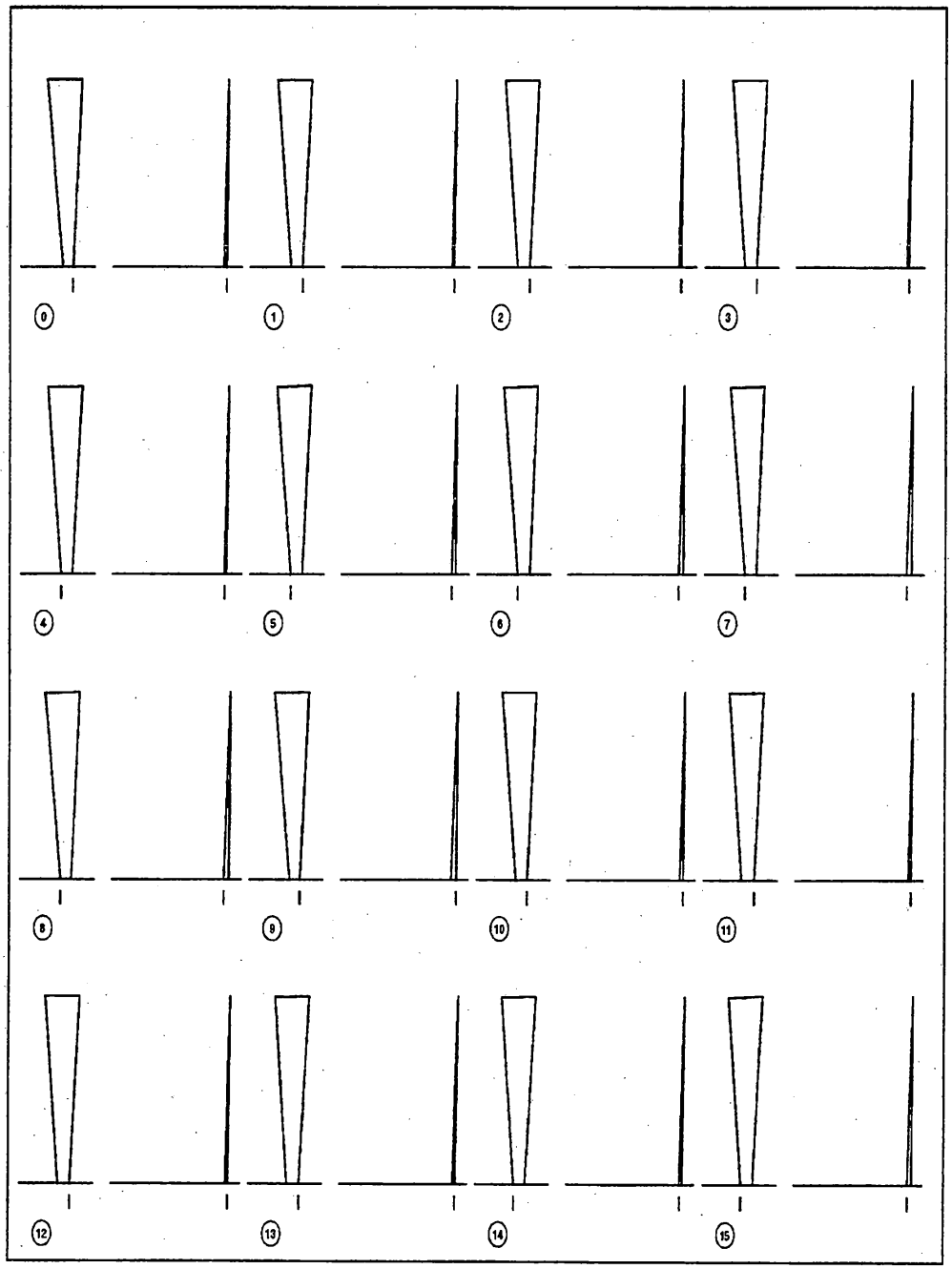



Figure 6.7: The start position selected for the combination of both a regular gait and the displacement to the right

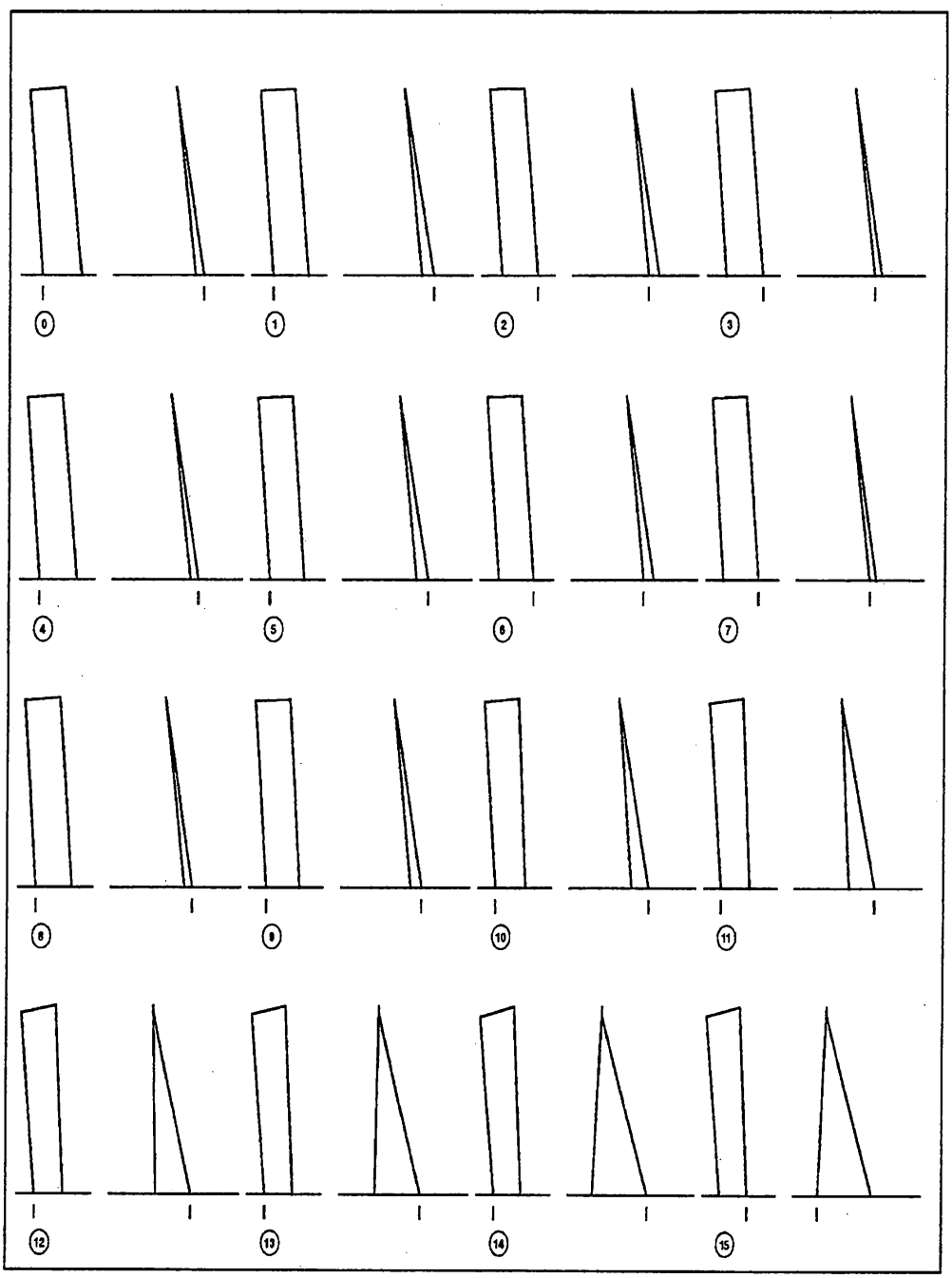



Figure 6.8: The start position selected for the fast displacement to the left. One can recognise the irregularity of the gait around the roll axis, where originally the free leg points outwards but is being pulled constantly inwards. During the last step the robot swings the free leg far too much forward around the pitch axis. However, by doing so the robot achieves a large move to the left.

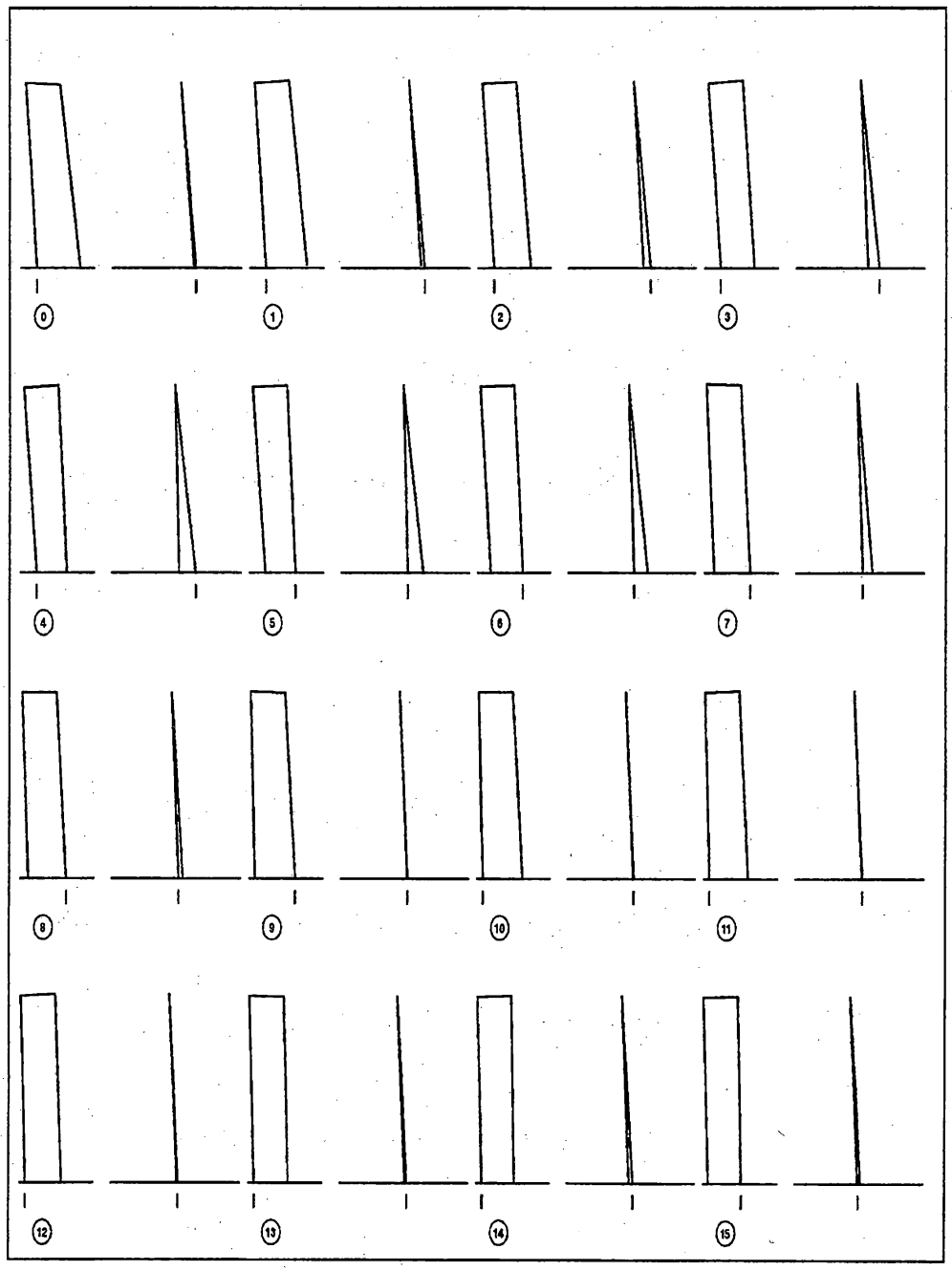



Figure 6.9: This figure and the following three figures demonstrate how the biped turns from walking to the left to walking to the right. Originally, as seen on this page, the robot walks towards the left. Notice that these first steps are the same as in Figure 6.2

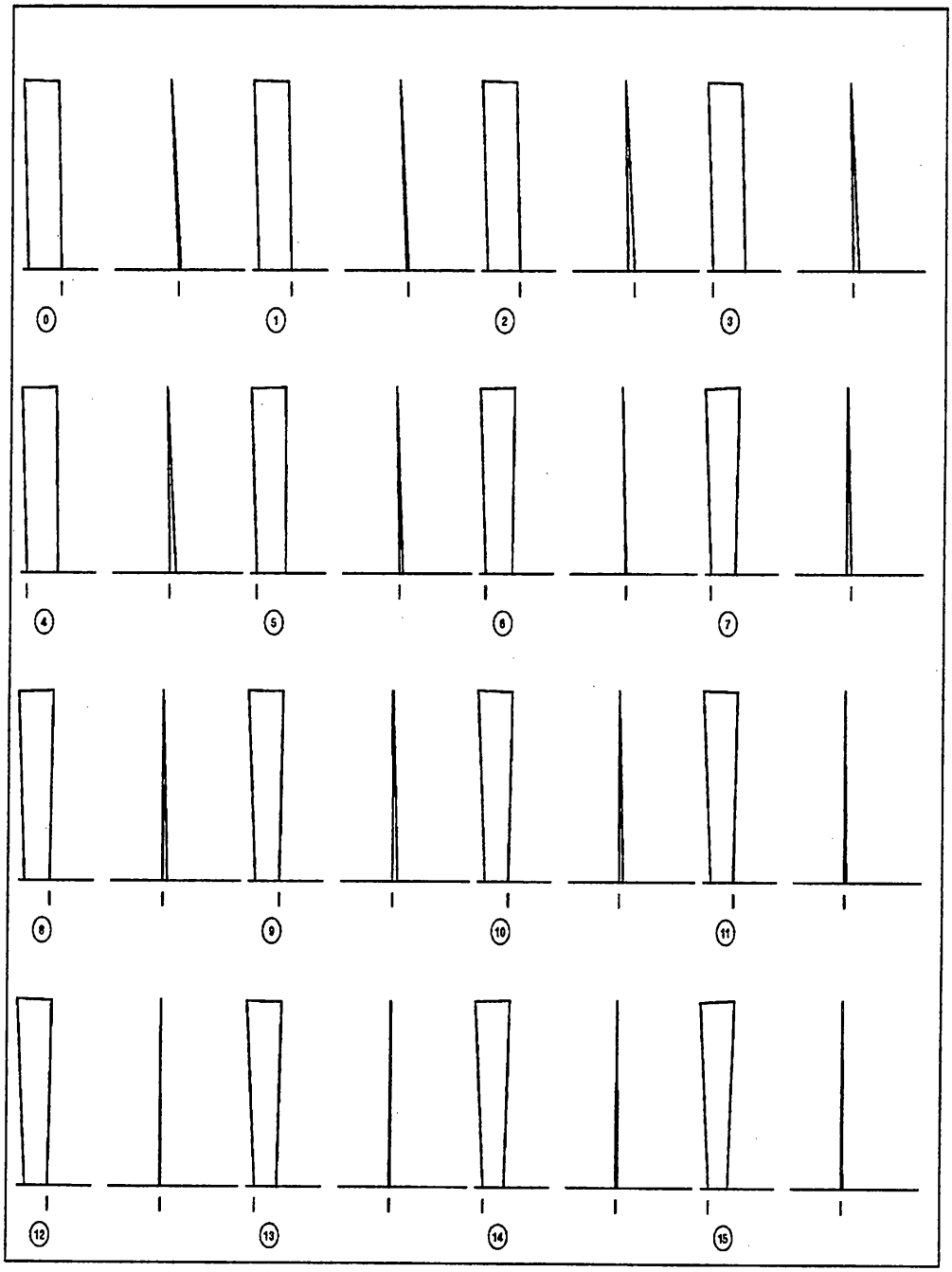



Figure 6.10: Turning continued: The robot stops walking towards the left and gets itself into an upright position.

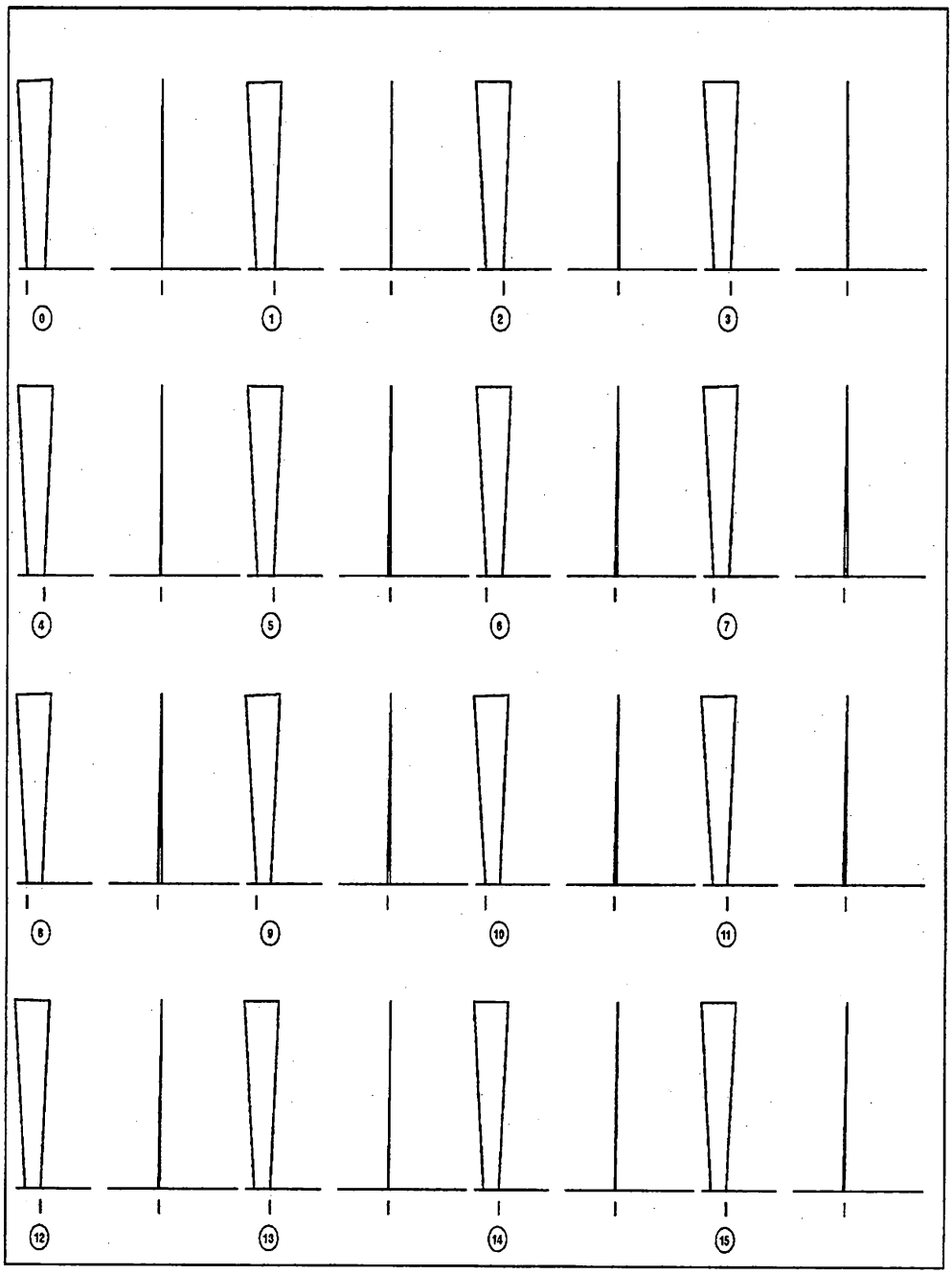



Figure 6.11: Turning continued: Form an upright position the robot starts to walk towards the right.

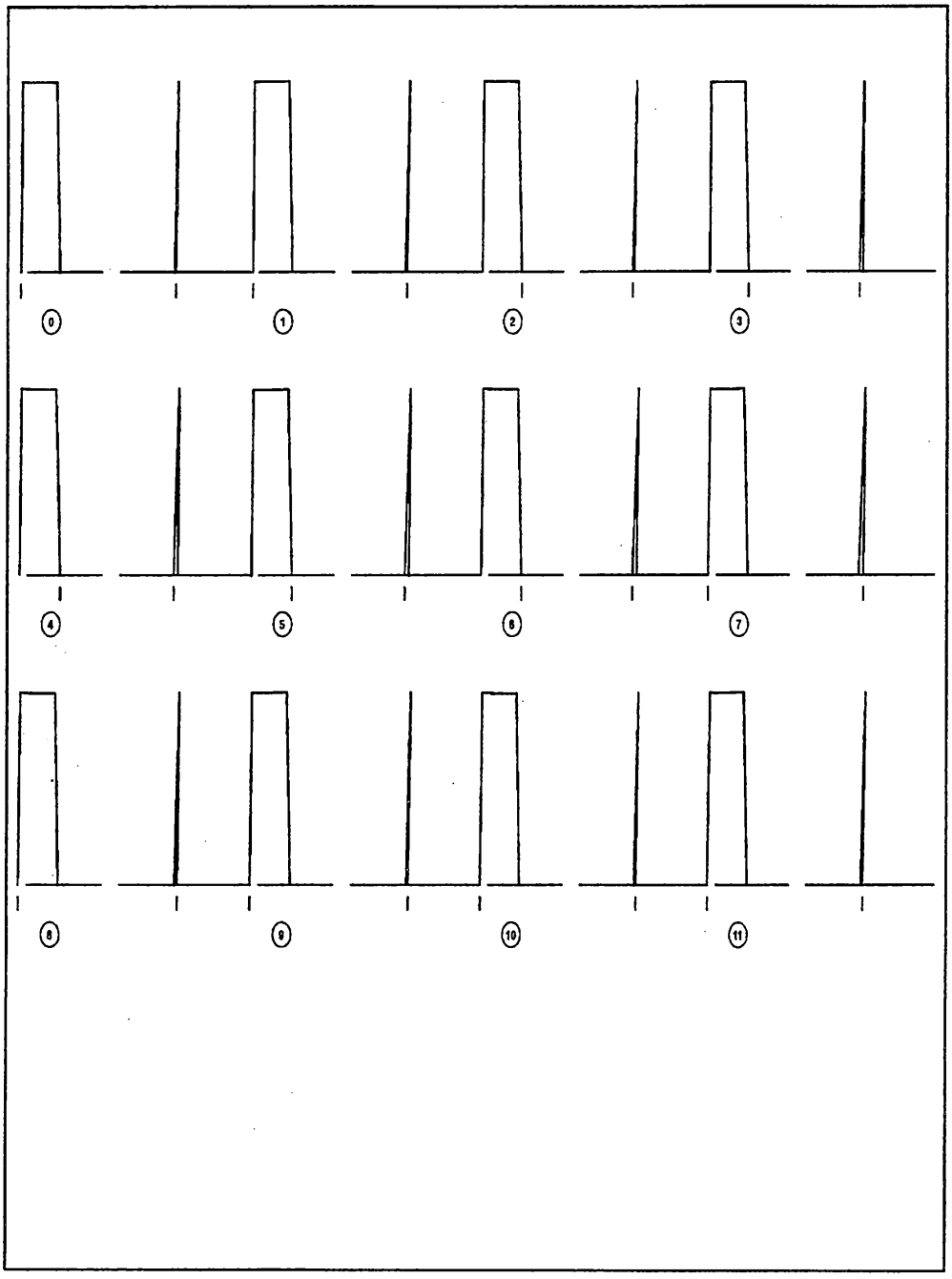



Figure 6.12: Turning continued: The robot is now walking towards the right.

### 6.3.1 Turning Steps into Gaits

The steps which were displayed in the previous section are generally not linear. This means that after one step the robot is in a position where the angles of the limbs are different from the angles in the original start position. Therefore the repetition of the same motor commands as used in the previous step does not necessarily lead to any proper step (such that at the end of the failure-free step execution both feet touch the ground). Thus, the program has again to search for the correct motor commands for the following step.

What we want to achieve in this section of this thesis is to *learn* how to walk. Learning implies that at least that some of the original effort doesn't need to be repeated in later applications. In our case, this means that the system should be able to recall what to do in order to carry out the next step. In order to be able to use previously found sequences of motor commands the robot should end up in positions similar to one in which it previously used a known motor command sequence. Obviously, if a command sequence always puts the robot back into the position from where it started (apart from a displacement along the axis along which is walking), then the behaviour would be linear and we could always use the same command sequence.

Thus, the program looked at all the steps which were carried out in the earlier test runs and identified those where the start position is very similar to the position in which the robot finished its step. If additionally this step showed a sufficient displacement along the forward axis then this step was selected. The step was then represented as a gait, ie. as a pair consisting of the start position and the sequence of motor commands which generated the step.

Now the selected gaits are modified in order to linearise them. Linearisation of a gait means that the position at the beginning and the end of the step are so similar that the same motor commands which generated the first step can be applied again and again.

The search for linearised steps introduces a shift in the way the behaviour of the robot is described. A step can be seen as the combination of a start state and a search procedure. A gait is a combination of a start position and a sequence of motor commands. In order to find linear gaits the search shifts from the search for an individual motor command to the search for an entire sequence of motor commands.

The search for linearised gaits proceeds as follows: given is a start position $S$ and a sequence of operators $u_1, u_2, ...u_n$. The result of applying an operator sequence $u_1, u_2, .., u_n$ to state $S$ will be called the result state $R$. Each operator $u_i$ is a triplet of individual motor commands $(m_{1i}, m_{2i}, m_{3i})$. Each individual motor command can be changed by some small amount $(+/- \Delta)$, or it can be

left unchanged. The search for linearised gaits looks at an individual motor command over the whole length of the operator sequence. In each operator the corresponding motor command can be increased, decreased or left unchanged. Thus if the length of the operator sequence is $n$, then there are $3^n$ different ways to change the $i$th motor command in the operator sequence. The search program tries all $3^n$ changes of the $i$th motor command and applies the resulting sequence of motor commands to the start position $S$. It picks the sequence of motor commands that results in the result state $R$ being closest to the start state $S$ (using the Euclidian distance between the parameters of the inner state space of $R$ and $S$ as a measure). This is repeated for all 3 motor commands until a local minimum is encountered. Since the average length of the operator sequence is about 5, and since the search uses strict hillclimbing, the search can be executed relatively quickly.

The result is a combination of a start position and an operator sequence (a gait), which is, for practical purposes, as linear as the coarseness of the search will allow. Thus, the resulting gait will be called the *linearised gait.*

Figures 6.13 and 6.14 show the behaviour of a linearised gait. This linearised gait can be repeated 30 times before failure occurs because the legs are spread too far around the roll axis. This is an improvement over the steps which were found by the original search for steps. Gaits extracted from these steps by simply recording the motor commands used during the execution of the step tended to be very brittle. Those simple gaits tend to fail on average after two steps only.

Nevertheless, the result shows clearly that the coarseness of the search makes it very difficult to find truly linear gaits. At some point, sooner or later, corrective action has to be taken, even in a "perfect" noise free environment.

### 6.3.2 Interleaving Search for Steps and Gait Execution

Since the execution of a linearised gait leads sooner or later to failure, it becomes important to ensure that the robot is kept from failure. It seems plausible to monitor the execution of a linearised gait until the robot reaches a start position which is so different from its original start position, that the program has to search for a step which brings it back to the original start position.

Thus an experiment was created were the program interleaved the execution of the linearised gait with the search for a step forward. Whenever the robot was close enough (i.e. within a certain tolerance $\Theta$) to the original start position of the gait it executed the command sequence associated with the linearised gait. The tolerance $\theta$ is defined as the Euclidian distance between the parameters of the inner state space in the actual position of the robot and the

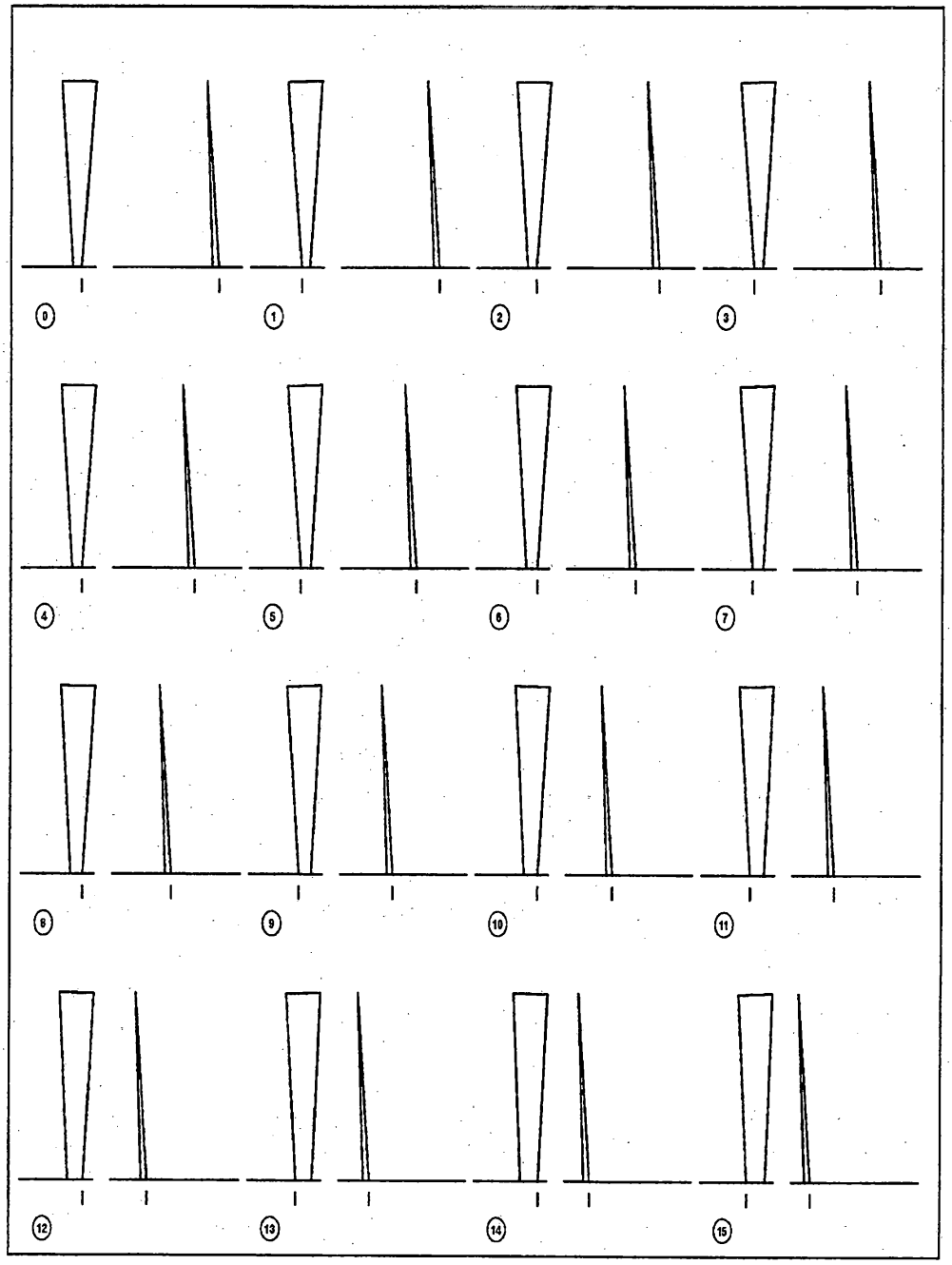



Figure 6.13: A linearised gait is executed 16 times in a row. Each picture shows the posture of the biped after the completion of a step. The regularity of the gait as well as the displacement from the right to the left are clearly visible.

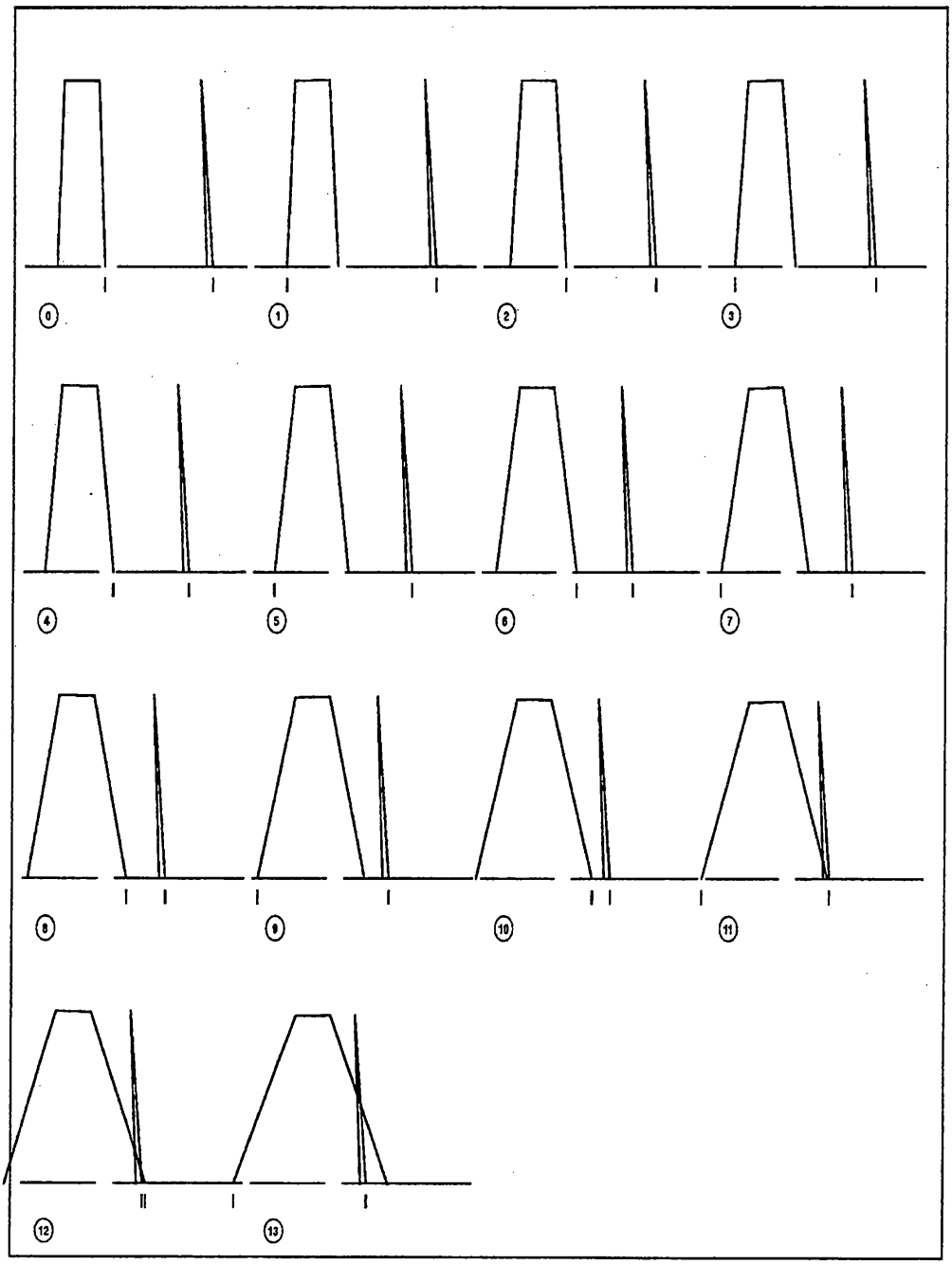



Figure 6.14: Linearised gait fails after 36 steps. Each picture shows the posture of the biped after the completion of a step. In this figure, the last 14 steps of the robot are depicted. Failure occurs because the legs of the robot are spread too far around the roll axis. The view of the pitch axis has been shifted to the right in comparison to Figure 6.13 in order to fit the two views (roll and pitch axis) into the same limited space.

corresponding parameters in the start position of the linearised gait. If the actual start position differed too much from the original position, the system switched back and *searched* for a step bringing it back close to the original start position. Figure 6.15 shows how often search had to be interleaved in order to keep the robot close to the original trajectory. The system often considered the robot close enough to the trajectory to execute the prestored command sequence and failed nevertheless. The only way to prevent this was to define the tolerance $\Theta$ so narrowly that in effect the system hardly ever executed the stored command sequence but rather kept searching. Thus, prestored command sequences are extremely brittle and help in very few cases to speed up the system.

As the previous example demonstrated, it is not very useful to simply store approximately linear movements in the hope that this would enable the robot to walk. The coarseness of the search leaves us with an error which is too large to ignore. The execution of linearised gaits has to be *controlled*.

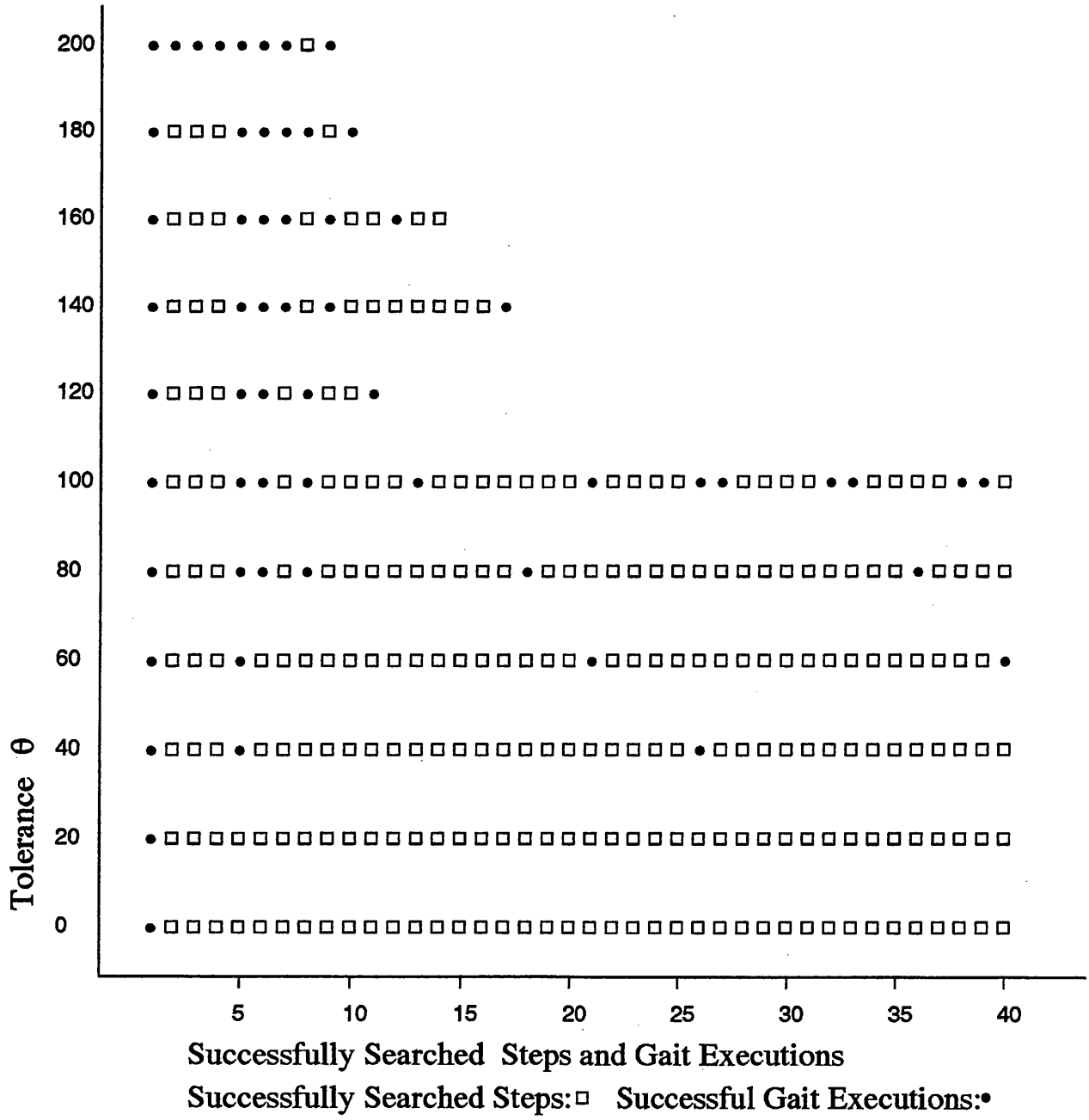



Figure 6.15: Interleaving search and gait execution. Boxes indicate search, circles indicate gait execution. The threshold (of the evaluation function) for search was increased for each trial (vertical axis). The horizontal axis indicates the time until failure. A test terminated successfully after 40 steps or gait executions.

## 6.4 Summary

This chapter shows that simple hillclimbing search enables the program to search effectively for steps. As discussed in Chapter 5 the success of the search was due to the use of a state space weighting based on the survival time heuristic. The result is a sequence of steps, which are stable in the sense that after each step, the search program is able to successfully search for yet another step.

The search also proved to be powerful enough to change between different types of steps, and thus the robot was able to change its direction.

Using a refined local search based on gaits produced by the original search for steps resulted in almost linear gaits. The robot was able to execute the same gait up to 30 times before failure. However, a control mechanism is needed to avoid repeatedly searching for steps. The following chapter will show how a simple table look-up controller can be build to allow quick and robust control of the gait execution.

# Chapter 7

# Control

The previous chapter described how the robot successfully searched for steps. These steps were then improved so that they could be executed repeatedly. The result was a *linear gait*, which is a start position and a sequence of motor commands. Ideally the robot reaches the start position of the linear gait and then keeps executing the same sequence of motor commands for as long as needed. However a problem occurs: the robot is usually unable to reach exactly the required start position of the linearised gait, and the gait itself decays with time.

This chapter deals with the control of the execution of such a gait. Two simple look-up table based controllers are introduced: an *inertial controller* which controls the joint positions of the robot, and a *positional controller* which controls the position of the hip and the free foot. These controllers are built directly from the analysis of the state space as described in Chapter 5. The performance of the controller is analysed with respect to two different phenomena: (1) since the robot is not necessarily in the proper start position for the execution of a desired gait, the controller has to be powerful enough to control the robot even if it is relatively far away from this desired start position. Alternatively the robot must be able to search for steps which bring it close enough to this start position so that the controller can take over. (2) The characteristics of the robot itself can change over time: due to wear and tear the actual behaviour of the robot can change, or due to faulty sensors the perception of the robot's behaviour can change.

## 7.1 Inertial Control

The first technique used to control the biped is called *inertial gait control*, and it is based on the control of the parameters of the inner state space of the biped.

Each time the robot tries to execute a given gait, its actual joint positions are compared with the desired joint positions. Whenever a joint position deviates by more than a certain tolerance, $\Theta$, from its desired trajectory, a corrective control command is executed.

During the earlier analysis of the state space a table was created, indicating which command influences which parameter to what extent (see Section 5.3.2 on page 45). From this a function, *look_up(parameter,effect)*, can be computed, which returns a *force* which is needed in order to change the parameter to achieve the desired effect. E.g. *look_up*$(\zeta, +)$ would return $(+, 0, 0)$, indicating that in order to increase the value of $\zeta$ the force of the first motor has to be increased. Similarly *look_up*$(\eta, +)$ would return $(0, 0, 0)$, indicating that the parameter $\eta$ is not controllable.

When the linearised gait is executed for the first time, all states through which the system goes are recorded. So there exists a list of states $S_1, S_2, .., S_n$ associated with the linear gait. Now the gait is executed repeatedly. Each time the current state $S_i'$ is compared with the corresponding original state $S_i$. If one or more parameters (of the inner state space) of $S_i'$ differ by more than a certain threshold, $\Theta$, then the action suggested by *look_up(parameter,effect)* is retrieved and the next command is modified accordingly. The *effect* is expressed as the difference between current value and goal value. In the actual implementation of the inertial controller tolerances, $\Theta$, of 0.01 ($\eta$), 0.1 ($\zeta$), 0.01 ($\theta$), 0.01 ($\psi$) 0.01 ($\phi$) have been chosen (all units in radians). The corrective force corresponded to $\pm$ 0.02 $NM$.

Thus an extremely simple tabular look-up controller has been built. It is basically a "bang-bang" controller that applies a constant corrective force independent of the size of the error. Theoretically such controllers can suffer from overshoot and make the controlled plant oscillate. The following results will show that despite this weakness a "bang-bang" inertial gait controller is completely sufficient to control the biped during gait execution.

The inertial controller will be tested first with respect to its ability to keep the biped on the trajectory described by the linear gait. Starting in the start position associated with the linear gait the corresponding sequence of motor commands is repeated 50 times. Whenever the robot deviates too far from the trajectory of the linear gait the controller becomes active. Figure 7.1 shows the resulting regular behaviour. The performance of the controller is documented as follows: for each parameter of the inner state space a "zero" line was plotted. This line indicates the desired position of the parameter during the gait execution. The gait was executed 50 times and the maximum (positive and negative deviation) and average deviation from this position was recorded and the resulting three graphs were plotted accordingly. Vertical bars through the centre line indicate the standard deviation.

In the two subsequent tests the execution of this trajectory was tested again, the difference being that this time the robot started in a position different from the original start position of the linearised gait. As Figure 7.2 shows, the controller was unable to bring the robot back to the desired trajectory and failed after executing one step. However if the robot *searches* for one step (Figure 7.3) or two steps (Figure 7.4) towards the desired start position of the linear gait, then it gets close enough to the desired trajectory so that the controller can take over.

These experiments indicate that the robot seems to be able to search for a number of steps which bring it close enough to some desired trajectory such that from then on the controller can take care of the robot. It was therefore interesting to find out how many search steps were needed in order to get the robot close enough to a certain start position. From the 392 originally generated start positions some 171 start positions were selected, based on the fact that they were leaning (around the pitch axis) into the same direction as the linear gait. Figure 7.5 shows how many of these states could be controlled after up to five search steps. It also shows the cumulative number of all states which were controllable after a certain number of search steps. The results indicate that the robot can reach a "controllable" position for the execution of a linearised gait after no more than three search steps, on average.

Since various numbers of search steps were needed in order to bring the robot close enough to the trajectory, it is interesting to see whether there was an easy way to indicate in advance when such a position was reached. Thus each time the controller was used to guide the execution of the trajectory, the evaluation function value of the start position was noted. Together with this the program recorded whether the controller was successful or not. Figure 7.6 shows the results: the histograms of states with a certain evaluation function value which have been successfully or unsuccessfully controlled. As one can see the evaluation function offers no measure of whether the search succeeded in bringing the the robot close enough to a state from where on the controller could be used. Of two different states with about the same evaluation function value one could be successfully and one unsuccessfully controlled.

Therefore the program has to monitor the performance of the controller: when a certain gait has to be executed and the robot is not in the start position for this gait, then search for steps leading to a position where the robot points into the same direction as the desired gait. Search then for another three steps towards the start position of the gait which is to be executed, and then start executing the stored gait, controlling it with the inertial controller. This technique worked in approximately 84% of all tested start positions.

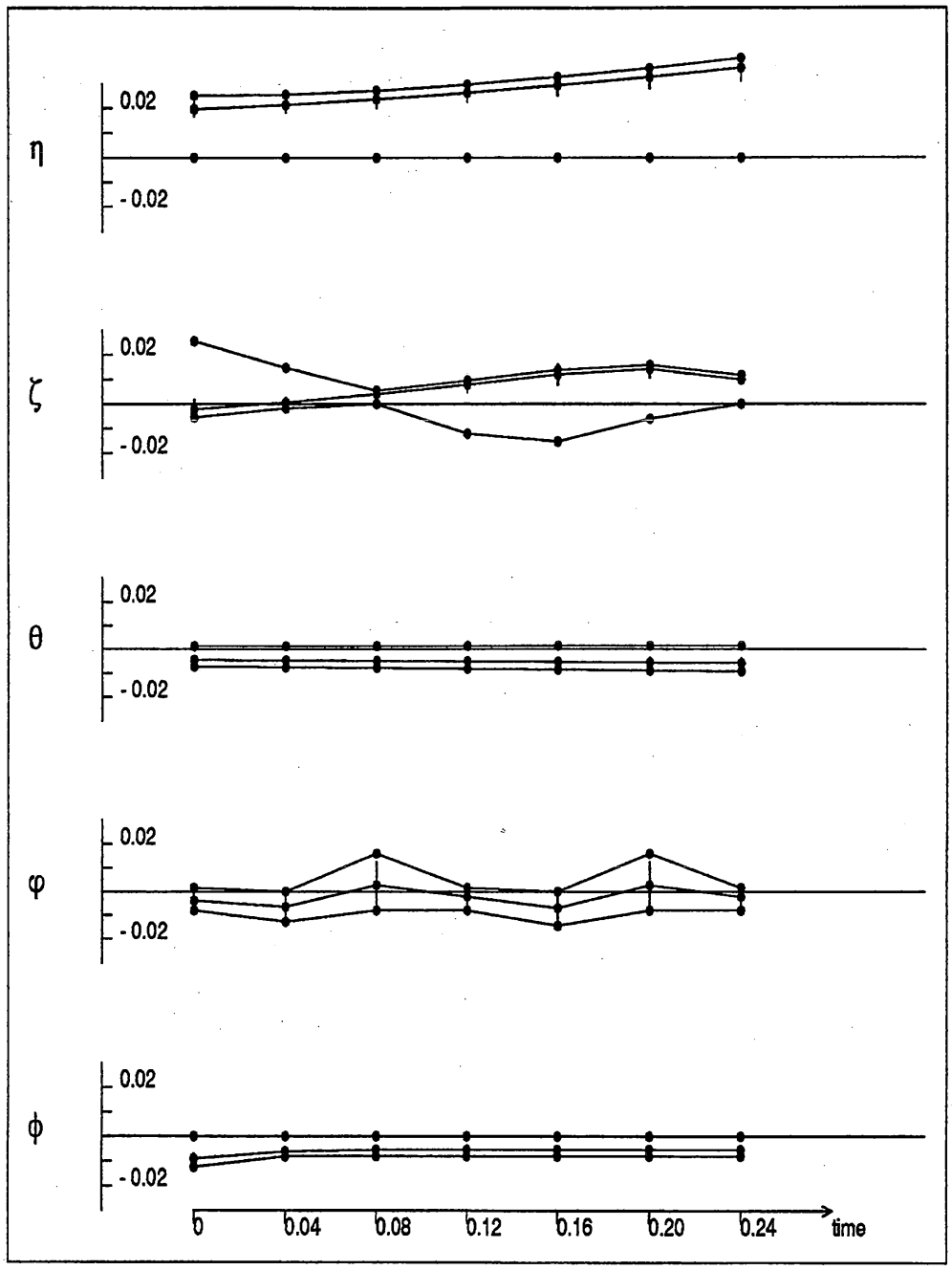



Figure 7.1: For each parameter the divergence of the actual (controlled) gait from the ideal gait is plotted in intervals of 0.04 seconds. The horizontal line indicates the ideal gait position. The uppermost and lowest curved line indicated minimum and maximum distance from the gait, the line in the middle describes the average position during gait execution. The vertical bars through the middle line describe the standard deviation. Units are in radians. This figure documents the use of the look-up table controller to control the execution of a gait. The biped starts walking in the original start position of the gait. Note that the standard deviation is almost zero for all parameters except $\psi$. This is equivalent to a rapid convergence towards the frequency response of the gait.

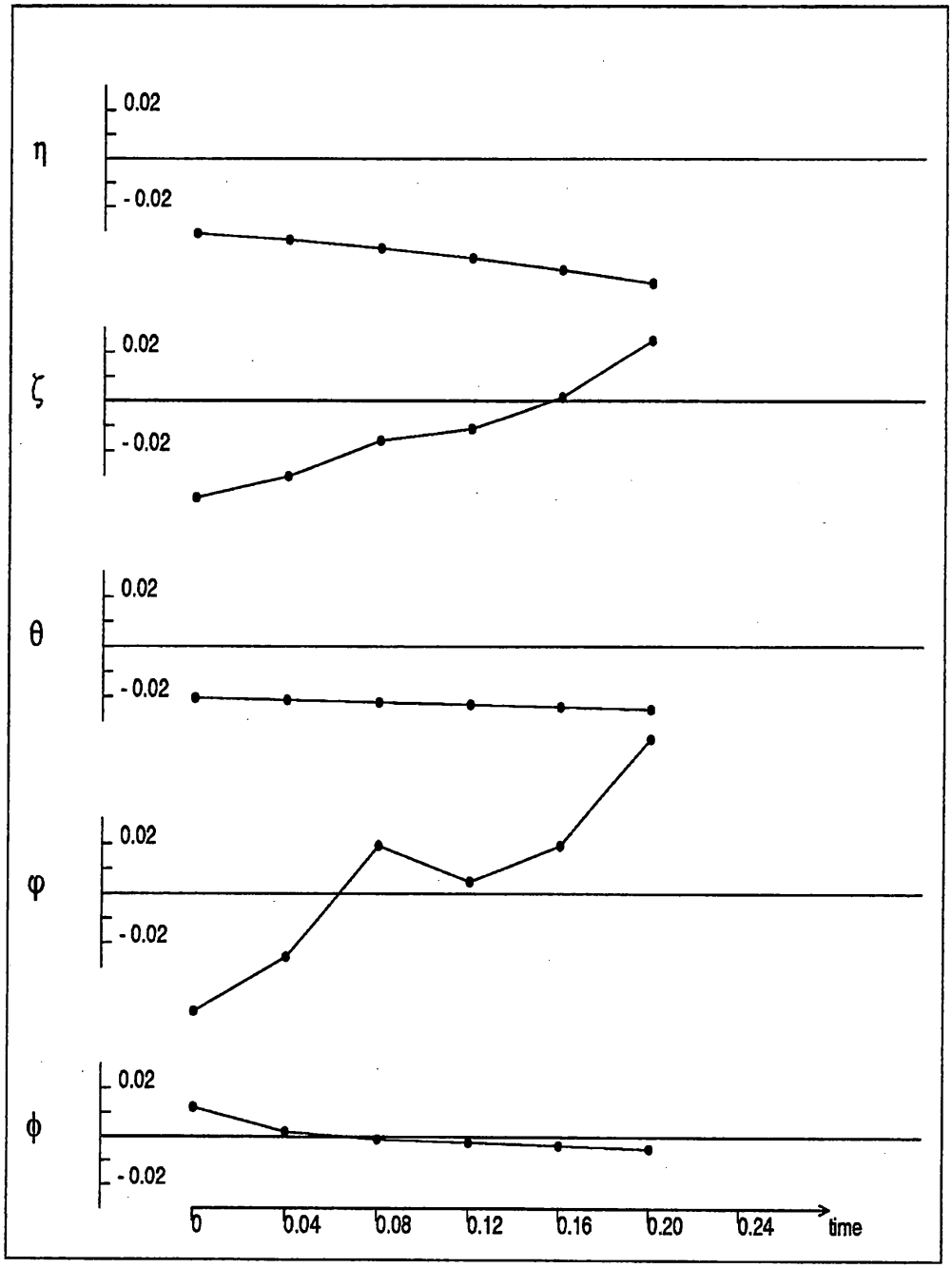



Figure 7.2: Failure of Controller due to overshoot. The biped starts in a position different from the desired start position. The biped executes only one step and then fails.

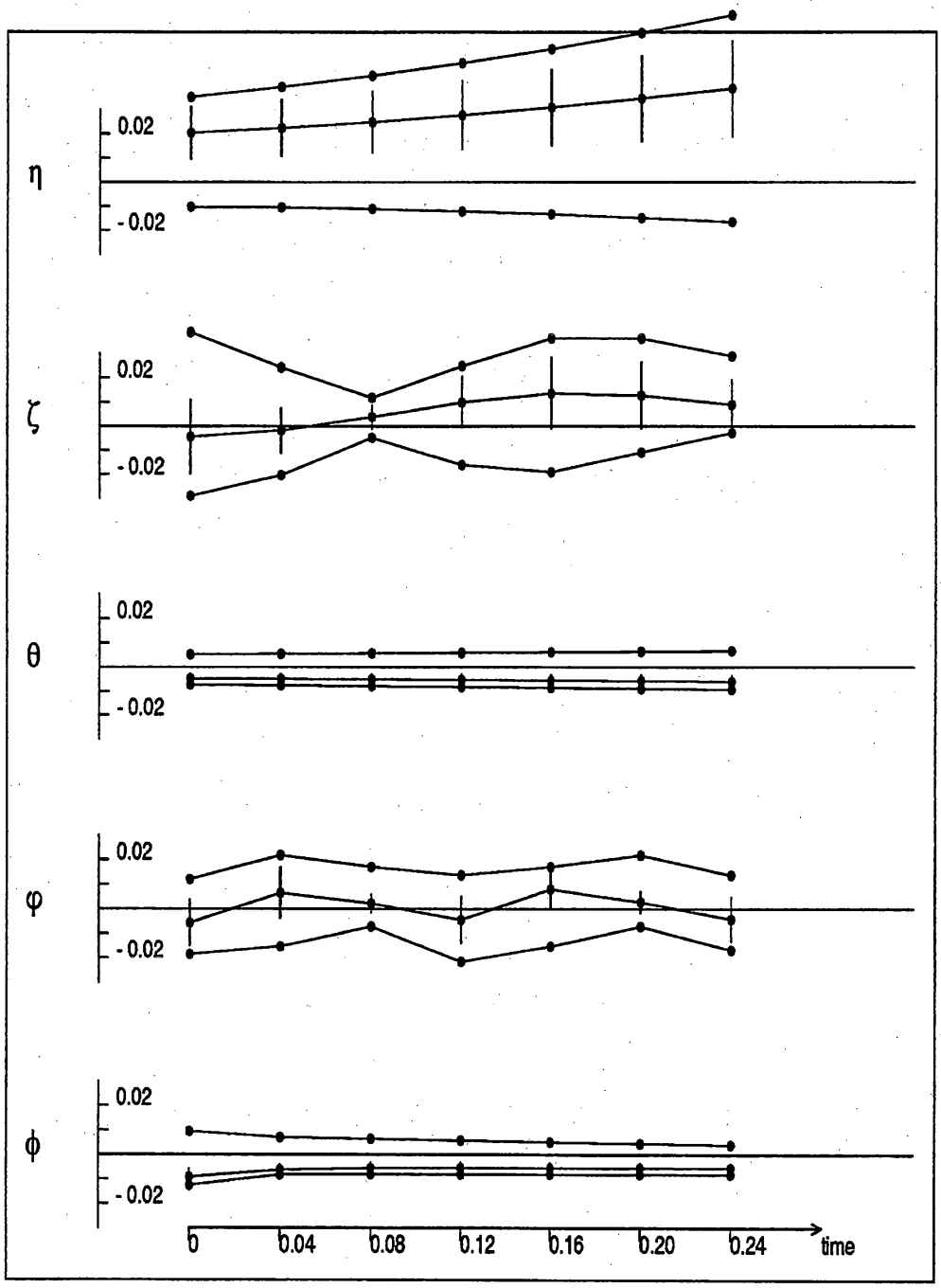



Figure 7.3: Successful use of inertial controller after one search step.

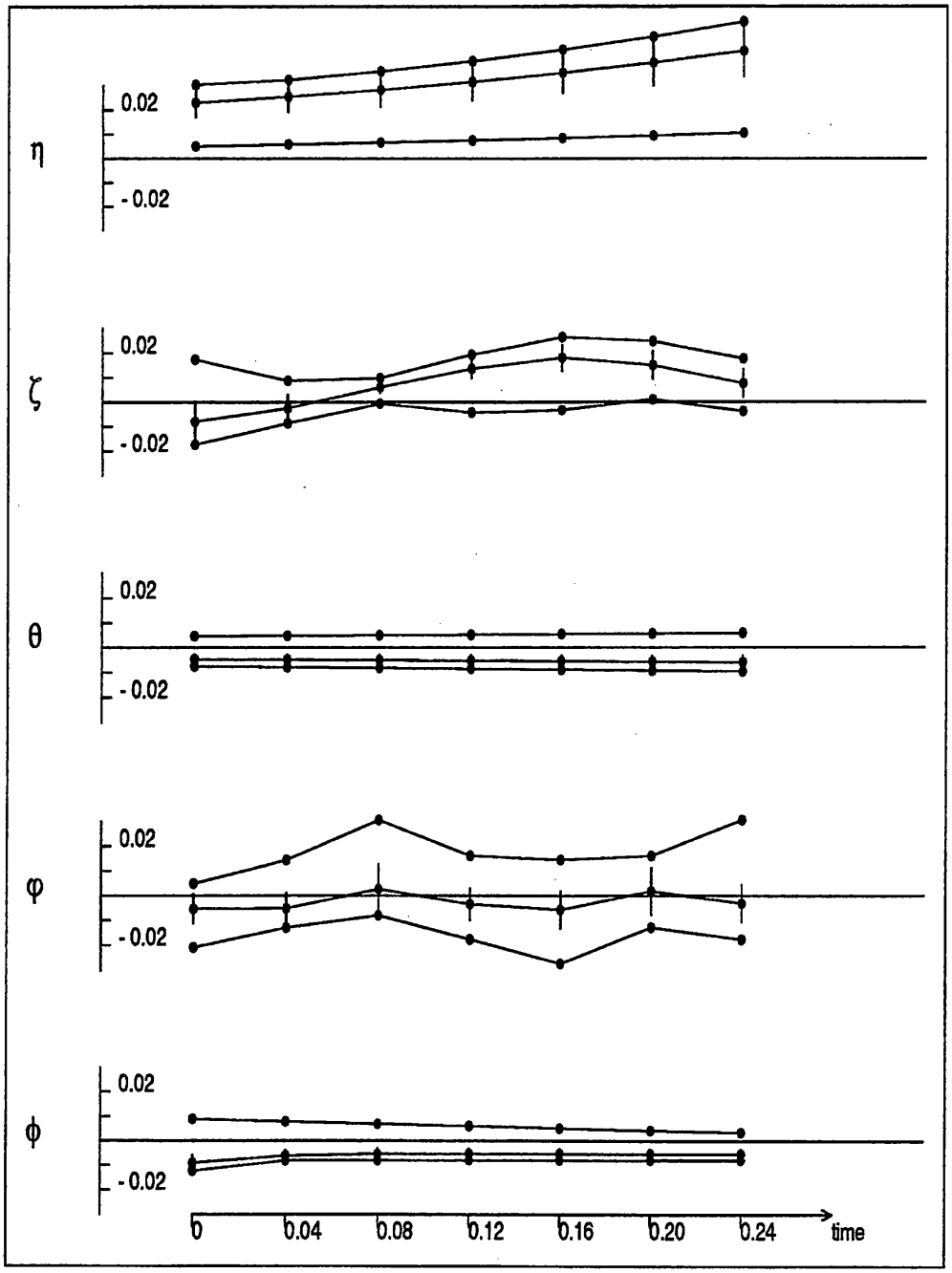



Figure 7.4: Successful use of inertial controller after two search steps. The standard deviation values decrease in comparison to Figure 7.3, indicating a more regular behaviour.

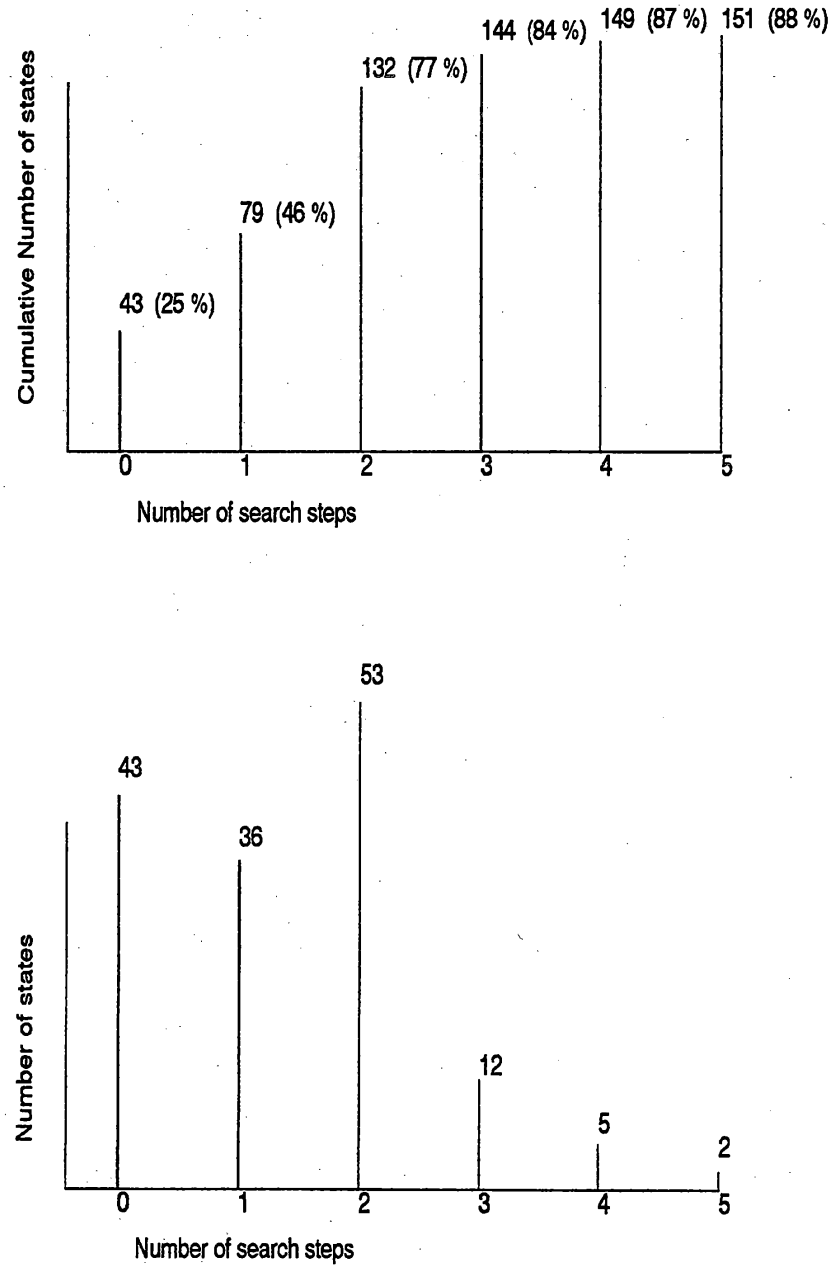



Figure 7.5: Successful use of inertial control after n search steps. The diagram on the top shows the cumulative number of states which could be controlled after n steps. The lower diagram shows how many states could be controlled after n search steps, but not after n-1 search steps.

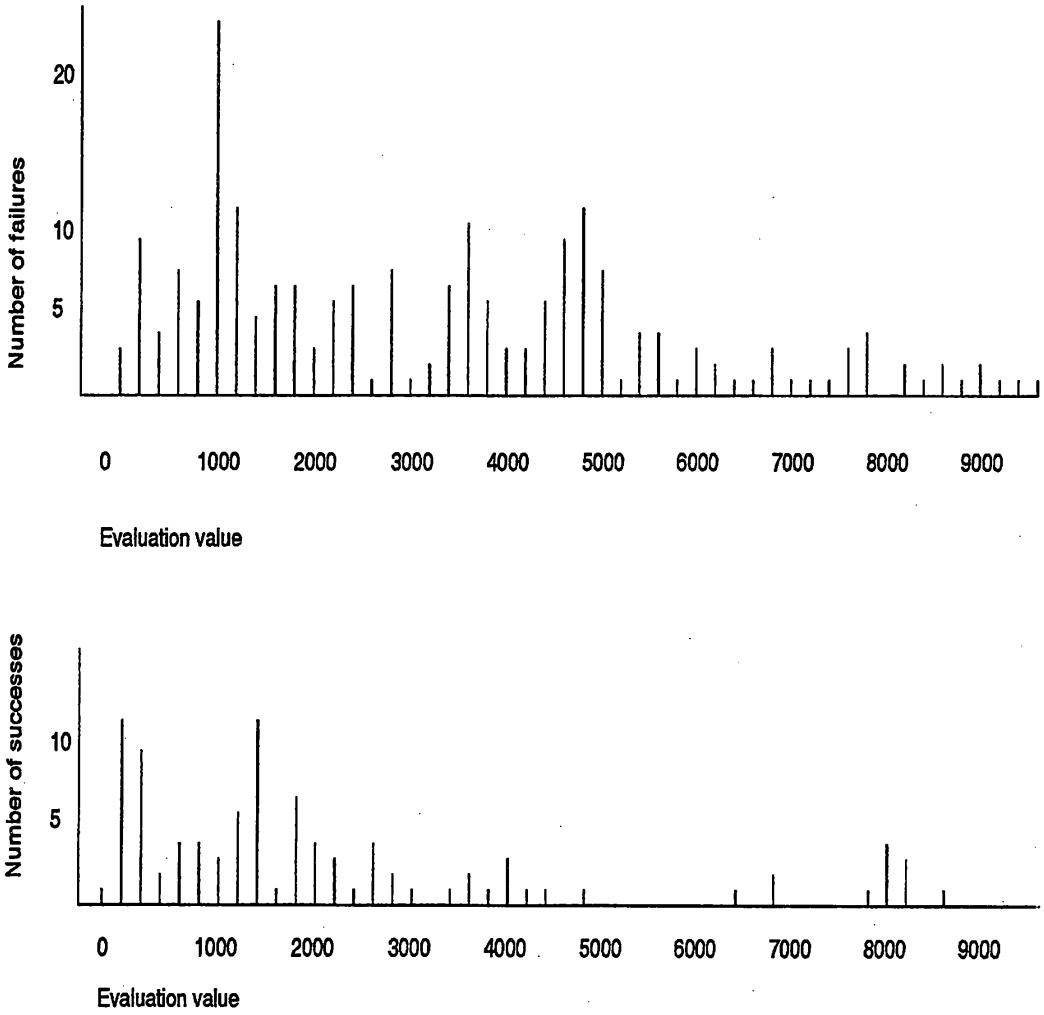



Figure 7.6: Histogram of evaluation function value for failed and successful control. This figure demonstrates that the evaluation function does not indicate whether the controller can be used successfully.

## 7.2 Positional Control

It seems that when humans walk they are typically not very much aware of whether the current position of a knee is very similar to the corresponding position of the knee in the previous step; but they can normally remember how upright they walk. They are much more aware of the height of their head or the orientation of the spine, than any position of parts of their legs. In this section we explore the efficiency of such a control algorithm. Rather than using the deviation of individual parameters we look at other "indirect" parameters: the position of the hip and the height of the foot.

During the original execution of the gait we record the position of the hip and the free foot in each state. During later runs deviation from these values will lead to corrective action. Again a look-up table is constructed which reports how changes in the motor commands affect these parameters. In the actual implementation of the positional controller tolerances $\Theta$ of 2 mm (*foot-height*), 1 mm (*hip-y-displacement*), and 1.5 mm (*foot-x-displacement*) are chosen. The corrective force corresponds to $\pm$ 0.01 $NM$.

Figures 7.7 to 7.9 show how this control technique performed. As we can see the performance is comparable to a controller using inertial control. Figure 7.7 displays the trajectory control when the robot starts with the desired posture. As one can see the robot is kept in a very stable trajectory similar to the one developed by the inertial controller. Figure 7.8 shows the inability of the positional controller to control the gait execution starting in a different start position. The controller also failed when tested after applying one search step trying to reach the start position of the gait. Figure 7.9 shows the gait execution which fails after 4 gait executions. After 2 search steps (Figure 7.10) the controller is finally able to control the gait execution.

The positional controller behaves differently from the inertial controller since its tolerances, $\Theta$, do not exactly correspond to the tolerances of the inertial control. This means that there can be deviations from the desired gait execution where only one of the two controllers becomes active. As an example there can be various joint positions which result in the same foot position.

The positional controller does perform slightly better than the inertial controller. Figure 7.11 shows how many search steps were needed in order to reach a position close enough to the target gait such that the positional controller could be used. In about 10% of the test states the positional controller needed one search step less in order to reach a posture from which the controller could be used successfully. After two search steps the positional controller could control 83% of the tested start positions. The inertial controller could control 84% of the start states after three search steps.

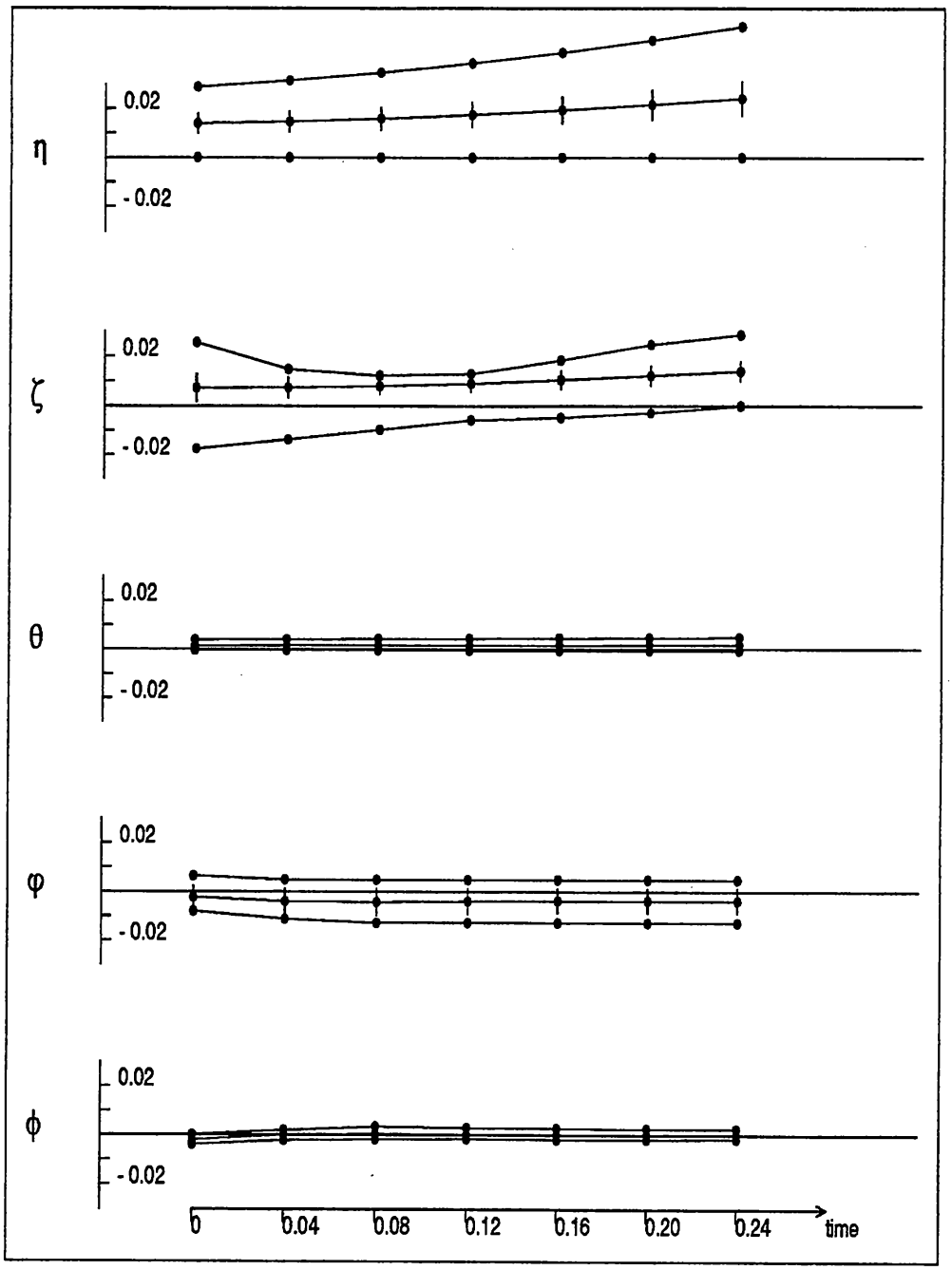



Figure 7.7: Positional control starting in the desired state.

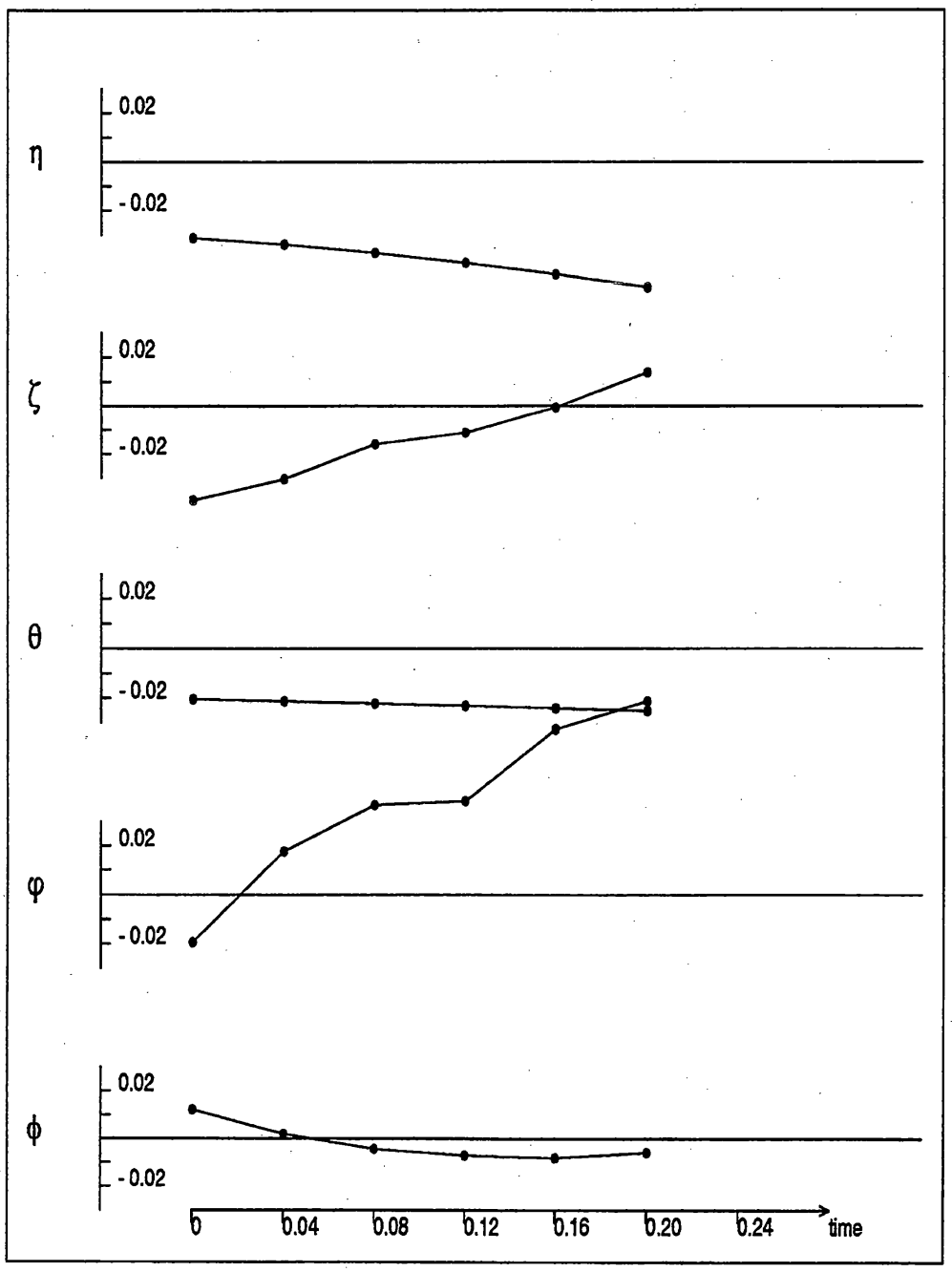



Figure 7.8: Failure of Positional Controller starting in a different start position. The controller fails after one step. Control using hip-height and displacements.

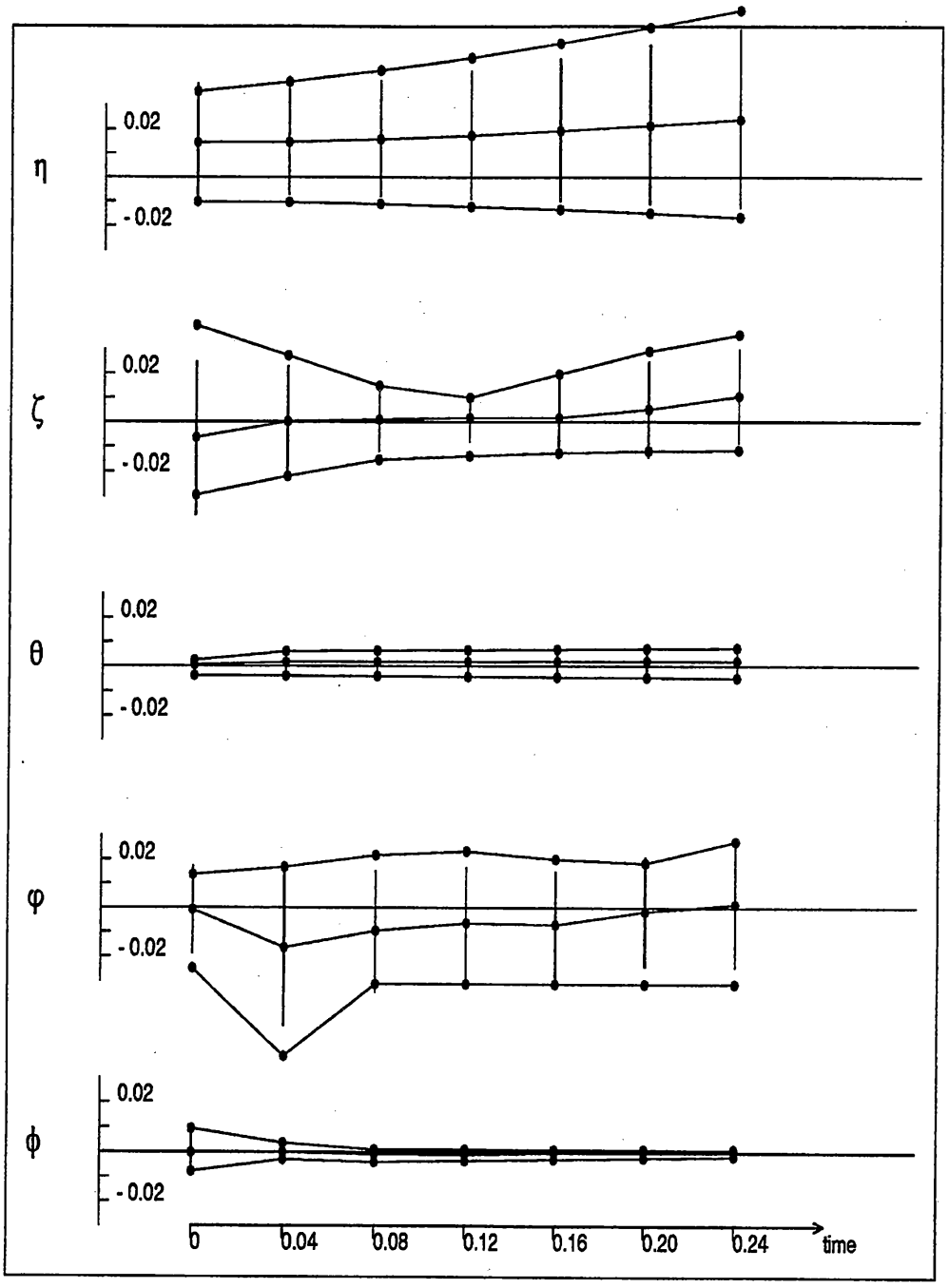



Figure 7.9: Unsuccessful use of positional controller after one search step. The robot collapsed because after 4 gait executions the free foot steps into the ground. Note that the standard deviation corresponds almost to the maximum and minimum values.

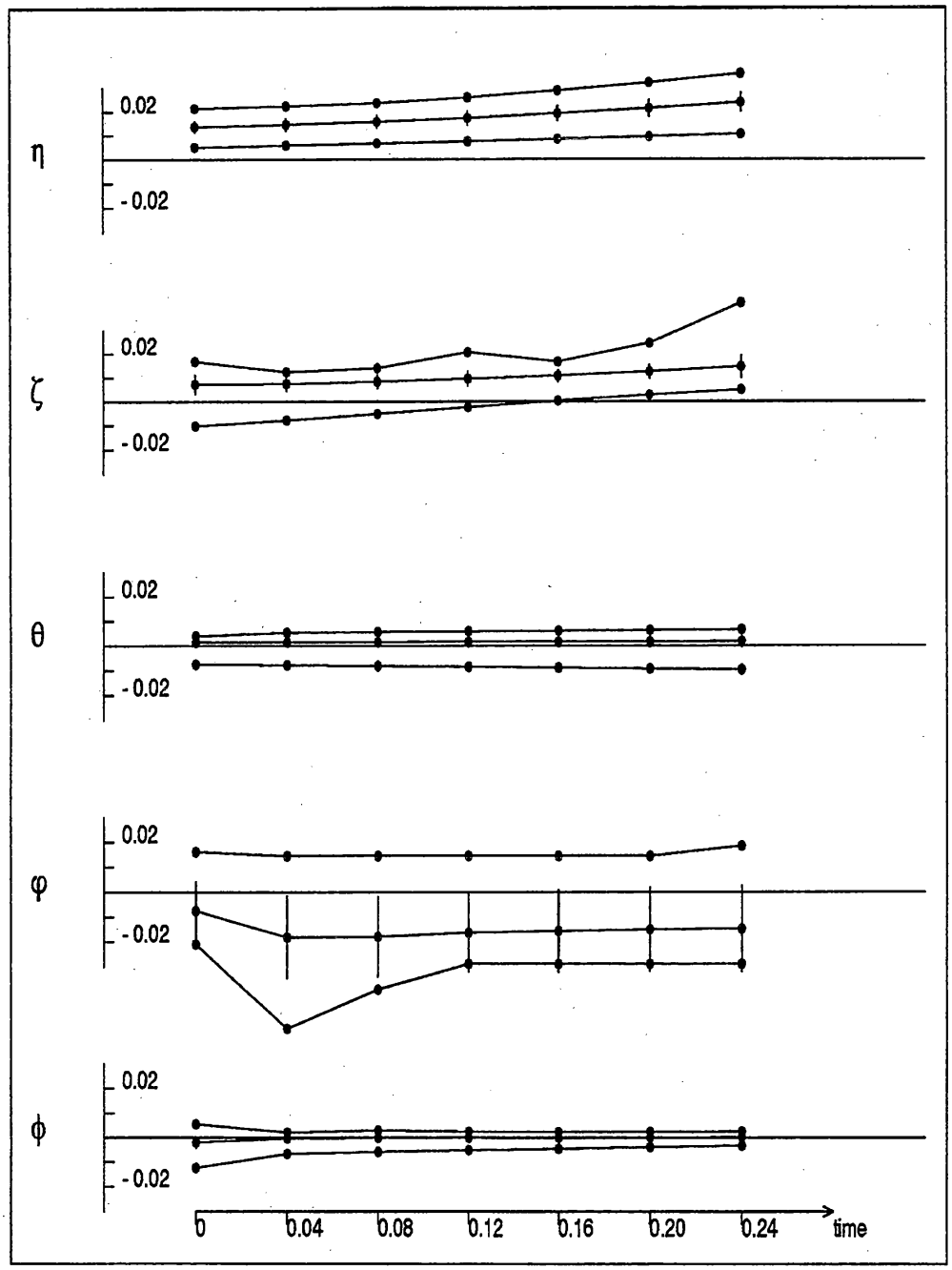



Figure 7.10: Successful use of positional controller after 2 search steps.

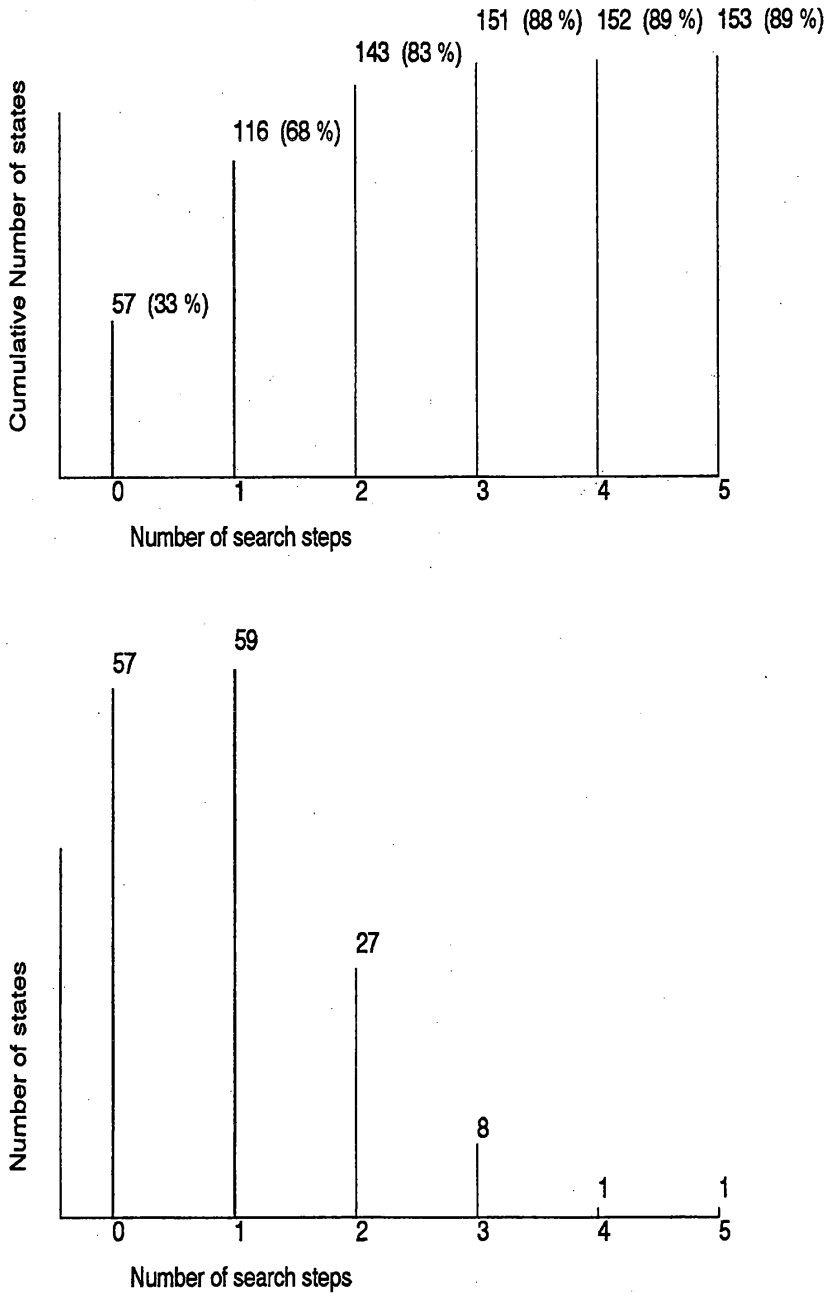



Figure 7.11: Successful use of positional control after n search steps. The diagram on the top shows the cumulative number of states which could be controlled after n steps. The lower diagram shows how many states could be controlled after n search steps, but not after n-1 search steps.

## 7.3 Testing for Robustness

The original gaits have been developed in a noise free environment. The behaviour of the robot was assumed to be absolutely deterministic. If the robot is ever to survive in a more hostile environment, then clearly this is an invalid assumption. In a realistic environment we will have to cope with such factors as:

- General wear and tear: the behaviour of the robot will be different from the originally modelled behaviour.
- Unreliable sensors: the perceived behaviour of the robot is different from the actual behaviour of the robot.

An experiment was set up in which the model of the robot itself was changed. This was done by making the robot $X$% more responsive. Thus the controller needed to be "quicker" and dangerous behaviour leading to failure could happen more easily. Every 0.04 seconds the velocity of the limbs of the robot was increased by a constant factor of $X$%. In different experiments $X$ varied between -20% and 20%.

A similar set of experiments was set up again. This time the *perception* of the robot's behaviour was different from the actual behaviour of the robot. What was tested was how far the perceived behaviour of the robot was allowed to vary from its actual behaviour before the controller made things worse.

Figure 7.12 displays the behaviour of the inertial controller, and Figure 7.13 displays the behaviour of the positional controller in these experiments. The controllers were able to cope with decreased velocities much more predictably than with increased velocities, which is not surprising. Generally the controllers were able to cope as long as the actual changes in the speed of the robot differed by less then 10% from the predicted speed changes of the limbs of the robot. Each controller's ability to handle a faulty perception of the robot's behaviour corresponds directly to the controller's tolerance, $\Theta$. Since the positional controller allowed for more tolerance, it was also less sensitive to a faulty perception.

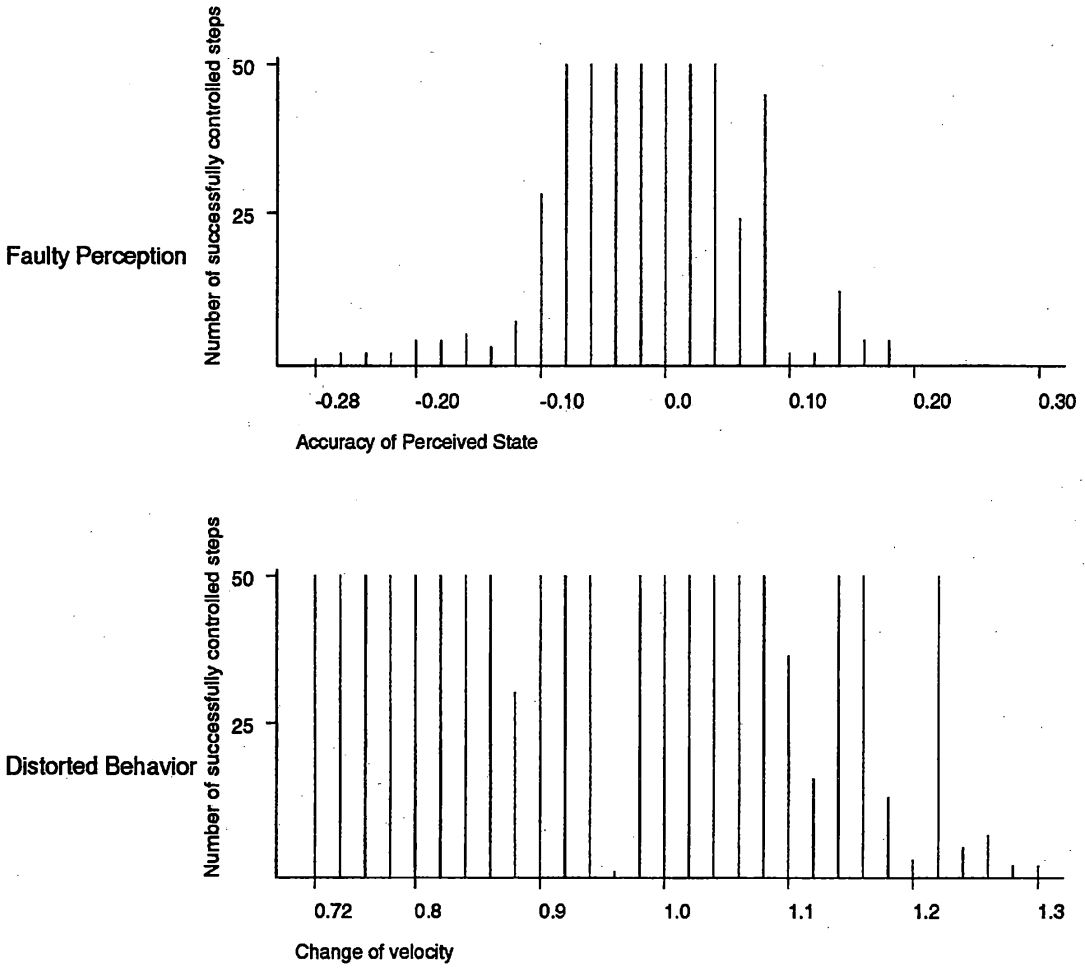



Figure 7.12: Inertial control of a biped with modified physical characteristics. In the lower graph it is assumed that due to the changed characteristics of the biped its angular velocities change by some percentage from the original model for which the controller was developed. The graphs describe the ability of the controller to control the successful execution of a number of steps (vertical axis) for various different behaviours of the robot (horizontal axis). In the upper graph the same tests are repeated for a model were due to faulty sensors the controller assumes the robot to be in a position which is different from the actual position of the robot.

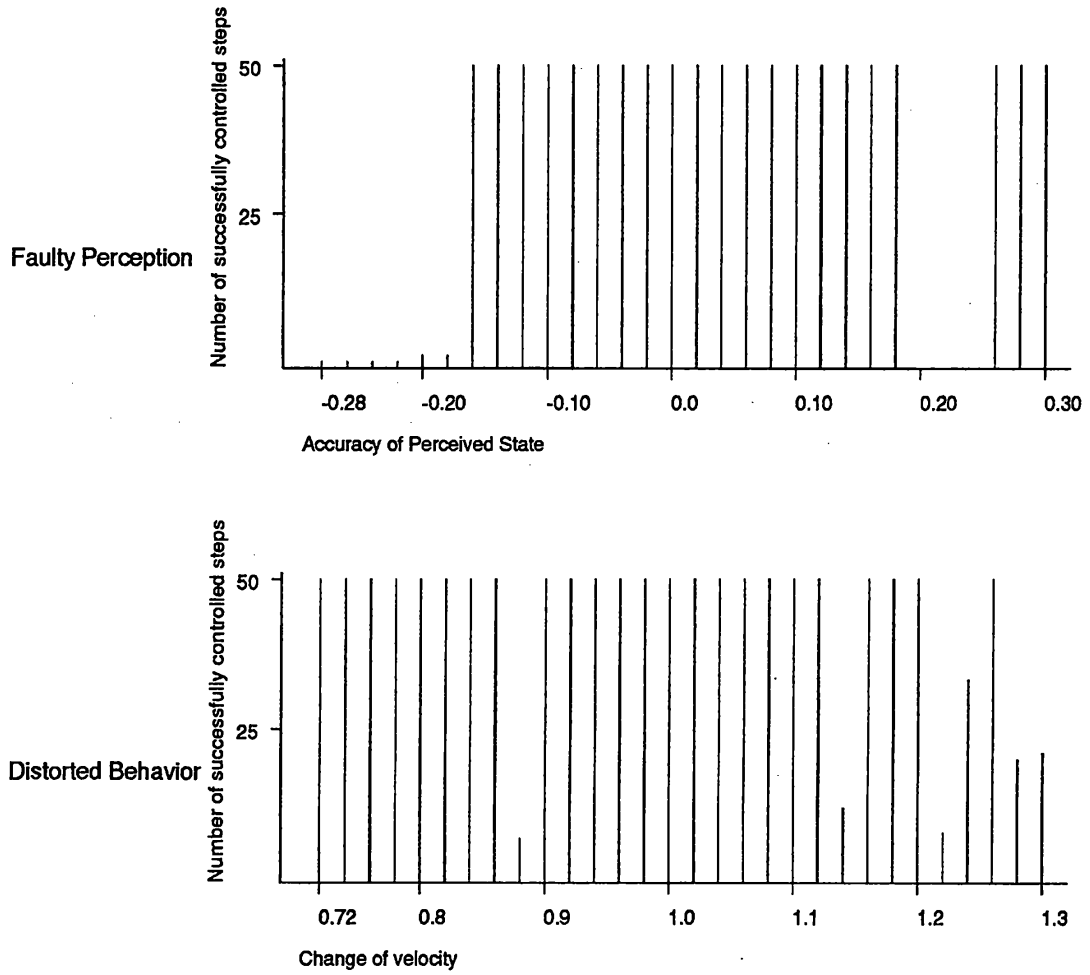



Figure 7.13: Positional control of a biped with modified physical characteristics. Again positional control produces a more robust control than inertial control.

## 7.4 Summary

The look-up table based controller has a fixed response the moment a parameter goes out of bounds. This fixed control response does not reflect the amount of *error* of the corresponding parameter. Therefore the performance of the controller depends to some extent on the error tolerance $\Theta$. $\Theta$ has to be chosen in such a way that the controller only becomes active if the error corresponds to the force applied by the controller. The present implementation of positional control appears to be slightly superior to inertial control: since positional control allows for a wider range of values for each parameter (there are various postures resulting in the same hip position), it becomes more tolerant towards minor errors and performs better.

Look-up table based inertial and positional bang bang control are an easy and robust way to control biped gait execution. Combining look-up table control and search for steps results in the ability of the biped to execute a given gait from most positions in its state space. However there is no obvious way of indicating whether a given position is close enough to the desired trajectory so that the gait can be executed. Therefore the program has to search for a number of steps leading the robot close to the trajectory before the controller takes over. Once the controller controls the execution of a stored gait the robot is able to execute gaits in constant time and without search.

Biped robot walking on a planar surface will therefore be controlled by combining the search for steps and the controller. As soon as the controller proves able to control the biped and the parameter values converge to some frequency response pattern, the search is switched off and the biped is controlled only by the controller.

# Chapter 8

# Interpreting and Modifying Gaits as Qualitative Functions

At this point the robot has learned to execute and control gaits. The latter were developed in order to enable it to cross an obstacle-free horizontal plane. However the robot is unable to cross obstacles by relying on these gaits alone or on a search of the inner state space. Even so the sequence of actions needed to step forward is not very different from the stepping over a small obstacle. It is therefore interesting to see how the original ability of the robot to walk on a horizontal plane can be *adapted* in order to cross an obstacle. This adaptation will be based on modifications of the *entire gait*. Thus the robot changes its bias from the search for motor commands to the search for gaits and gait modifications. This will provide it with a powerful mechanism to speed up and improve the search process.

Research on biological motor control [Bro86b] reveals that motion is organised in the form of activity templates. Any activity learnt by an animal or a human is stored as a pattern of muscular activities. For example we have a pattern stored away somewhere that describes how to write the letter "a". If we want to write the letter a bit larger or a bit faster, then we will still use the same pattern, but the frequency of the pattern and the amplitude of the pattern will change [VT80].

In this chapter the term *behaviour* will be used to describe a sequence of movements made by the biped robot such as a step forward. These behaviours usually correspond to the motion patterns which the robot learnt to control.

Some movements are related, but their relationship is of a more general nature than a variation of frequency and amplitude. This is true in the world of biped walking: the steps of the robot may be long or short, but in each case the basic procedure is the same: lift one leg off the ground and bring it into a

more upright position, then lower it back onto the ground and execute a support exchange command. On crossing obstacles some of the robot's steps may be considerably different from others, but all will obey the same basic principles. In order to climb upwards the stepping motion is curtailed or shortened because the foot is no longer returned to its original height. This movement differs from the original by more than frequency and amplitude.

What is needed is a way to derive these new, modified behaviours from old ones, without distorting too much of the original behaviour. The aim therefore is to change as few parameters as possible. If the robot searches for new behaviours as *modifications* of old behaviours, then the new search space will be the set of possible behavioural modifications rather than the original state space of the robot. If this set of behavioural modifications is smaller or easier to search than the original state space then this can lead to considerable improvement in the search process.

At the same time these behavioural modifications must be easy to implement: if the robot "designs" a certain trajectory which is similar to some other trajectory, then it still has to find the sequence of operators (motor commands) which will enable it to execute this trajectory. If, however, the modifications are applied directly to the original motor commands, then this problem will no longer arise. This is particularly important in the case of the biped robot model used in this thesis: it does not have a servo mechanism and thus has to *search* for suitable motor commands if it wants to follow a certain trajectory.

Therefore new gaits will be developed from old gaits by modifying the *motor commands* (or torques) applied to the robot. The number of control dimensions (the number of motors) as well as the range of control torques is considerably smaller than the state space of the robot. Provided the number of modifications remains small, this should result in a powerful mechanism by which the robot can solve new, related tasks.

Related recent work has focussed on the generation of trajectory modifications. Y.F. Zheng [Zhe90] proposes to use the van der Pol oscillator to (manually) generate gaits for biped walking. These gaits are then changed by using a neural network to modify the constants of the van der Pol oscillator. As a result he generated gait *trajectories*, which are then used to servo the robot joints. This approach is not applicable to this thesis, where the servoing itself is considered to be non-trivial.

## 8.1 Qualitative Equivalence

This section will define various gait modifications which preserve the qualitative equivalence between the original gait and the resultant modified gait. These modifications have to meet several criteria:

- The definition of qualitative equivalence has to be operational. Once a qualitatively equivalent gait has been identified its execution should be easy.
- Many locomotive patterns can be regarded as periodic functions. A gait modification should exist that modifies the amplitude of this periodic function. This would generate pattern modifications in a similar way to the modification of human or animal motion patterns, and would be especially useful for when the robot needs to reach a goal state similar to the goal state of the original gait. A definition of qualitative output equivalence ($\approx_o$) will be given which is aimed at generating such a modification.
- Many new situations will impose new goals for the robot. Therefore any gait modification which lets the robot reach the original goal state will be too conservative. The robot needs the ability to generate gaits which allow it to achieve new goal states. Nevertheless these gaits should preserve many of the properties of the old gait. A definition of qualitative input equivalence ($\approx_i$) will be given which is aimed at generating new trajectories while attempting to preserve some properties of the old gait.

We will introduce the following definition of qualitative equivalence $\approx_i$ between gaits: we are given a linear dynamic system of the form

$$\ddot{X} = a_1 X + a_2 \dot{X} + bU$$

where $a$ and $b$ are matrices and $X$ and $U$ are vectors. $X$ denotes the state variables of the system, and $U$ denotes the input to the system. Now assume that the system is fed a series of input signals $\mathcal{U} : U_1 \ldots U_n$. The *behaviour* of the system under input $\mathcal{U}$ is now described by

$$\mathbf{B} \equiv \ddot{X}_i = a_1 X_{i-1} + a_2 \dot{X}_{i-1} + bU_{i-1}, i : 1 \ldots n.$$

Let $U_i^1 \in \mathcal{U}_1$ be the input to behaviour $\mathbf{B}_{u_1}$ at step $i$ and let $U_i^2 \in \mathcal{U}_2$ be defined in the same way. Two behaviours $\mathbf{B}_{u_1}$ and $\mathbf{B}_{u_2}$ of a dynamic system under inputs $\mathcal{U}_1$ and $\mathcal{U}_2$ will be considered to be qualitatively equivalent $\mathbf{B}_{u_1} \approx_i \mathbf{B}_{u_2}$ if

$$U_i^1 = U_i^2 + k, i : 1 \ldots n$$

where $k$ is a small constant. In other words two behaviours are input equivalent if one differs from the other only because constantly a slightly higher (or lower) input was fed to the system. Since $\mathcal{U}_1$ and $\mathcal{U}_2$ can be of different length, they only have to be qualitatively equivalent for the duration of the smaller of the two input sequences.

The important aspect of this definition of qualitative equivalence is the fact that the *input* to the system is *similar* to other input sequences. This does not necessarily mean that the resulting behaviour of the system is similar in terms of its state space. As an example consider a cart-pole system. Assume the pole is leaning forward with an inclination $\theta$ and the cart is accelerated in such a way that the angular velocity of the pole becomes zero. In this instance the pole is balanced, but by applying a slightly higher acceleration this balance will be lost. Despite this the behaviour would be considered to be qualitatively equivalent!

The above definition of qualitative equivalence is defined in terms of the input to the system. A second definition of qualitative equivalence can be based on the output of the system. This definition is based on the fact that many locomotive patterns are rhythmic or cyclic. It is therefore possible to describe the state transitions of the robot by some periodic function. The robot's new behaviour $\mathbf{B}_1$ can be considered to be output equivalent to its old behaviour $\mathbf{B}_2$ ($\mathbf{B}_1 \approx_o \mathbf{B}_2$), if the functions describing the two behaviours differ only in their respective amplitude.

### 8.1.1 Gait Transformations which Preserve Qualitative Equivalence

The resulting gait modifications can be derived as follows: a gait is defined as a start position P followed by a sequence of motor commands. For example gait $G$ can be defined as

$$G = (P, u_1, u_2, ..u_n)$$

where each motor command $u_i$ is a triplet of torques

$$u_i = m_{i1}, m_{i2}, m_{i3}.$$

Various gait modification heuristics will be defined to preserve the input equivalence between two gaits: one (*add*) that uniformly increases the torque used in the motor commands, one (*sub*) which uniformly decreases it, one (*lengthen*) which adds an additional motor command to the gait, and one (*shorten*) which cuts the last command. Another heuristic (*amplify*) will be defined which aims at the preservation of output equivalence. This heuristic splits the motor commands in half and increases the first half and decreases the second half or vice versa.

Assuming the original gait is defined as above, and the torque $m_i, 1 \leq i \leq 3$ refers to the $i$-th torque in a motor command, then the gait modifications can be defined as follows:

$$\mathbf{add}(\mathrm{G}, \mathrm{i}) : G = (P, \mathrm{plus}(u_1, i), \mathrm{plus}(u_2, i), \ldots \mathrm{plus}(u_n, i))$$

where $\mathrm{plus}(u_j, i)$ is defined as increasing the $i$-th torque $m_i$ in the motor command $u_j$ by one unit.

$$\mathbf{sub}(\mathrm{G}, \mathrm{i}) : G = (P, \mathrm{minus}(u_1, i), \mathrm{minus}(u_2, i), \ldots \mathrm{minus}(u_n, i))$$

where $\mathrm{minus}(u_j, i)$ is defined as decreasing the $i$-th component of $u_j$ by one unit. The modification

$$\mathbf{shorten}(\mathrm{G}) : G = (P, u_1, u_2, ..u_{n-1}).$$

shortens the gait by one time-slice.

$$\mathbf{lengthen}(\mathrm{G}) : G = (P, u_1, u_2, ...u_n, u_{n+1})$$

adds a further motor command to the gait. The value of $u_{n+1}$ is computed from the average of the previous commands:

$$m_{n+1,i} = \sum_{k=1}^{n} m_{k,i}/n.$$

These heuristics preserve most of the properties of the original gait. Since only the torques applied to one joint are effected, most parameters of the robot will remain uneffected. These heuristics also preserve the qualitative input properties $\approx_i$ of the gait since they introduce constant change of the input torques in one direction.

The only gait transformation of $\mathbf{B}_1$ which produces an output equivalent gait $\mathbf{B}_2$ where $\mathbf{B}_1 \approx_o \mathbf{B}_2$ is the *amplify* heuristic.

$$\mathbf{amplify}(\mathrm{G}, \mathrm{i}) : G = (P, \mathrm{plus}(u_1, i), \ldots \mathrm{plus}(u_{\frac{n}{2}}, i), \mathrm{minus}(u_{\frac{n}{2}+1}, i), \ldots \mathrm{minus}(u_n, i)).$$

$$\mathbf{amplify}(\mathrm{G}, \mathrm{i}) : G = (P, \mathrm{minus}(u_1, i), \ldots \mathrm{minus}(u_{\frac{n}{2}}, i), \mathrm{plus}(u_{\frac{n}{2}+1}, i), \ldots \mathrm{plus}(u_n, i)).$$

In this case the first half of the gait is increased and the second half is decreased (or vice versa). The reason for the inclusion of this gait transformation heuristic is that by increasing and then decreasing the torque applied to the system it is hoped to produce an amplification of the underlying periodic behaviour of the system only, and therefore to preserve the qualitative output equivalence $\approx_o$.

### 8.1.2 Example of Qualitative Equivalence Heuristic

In the current model of the biped robot two motors effect the movement of the free leg around the roll and pitch axis, and a third motor moves the hip around the roll axis. All three motors receive signals, coded as integers between 0 and 9, which are then translated into specific torques exerted by the motor. If a step takes 7 time slices, the corresponding gait could be the following pair:

$$((\theta, \psi, \phi, \eta, \zeta)\ (9.7.3, 7.7.7, 5.5.7, 9.5.7, 7.5.7, 3.3.7, 3.3.7))$$

$\theta, \psi$, and $\phi$ are the angles of the support leg, the hip and the free leg around the roll axis, $\eta$ and $\zeta$ describe the angles of the support leg and the free leg around the pitch axis. The first triplet of commands, (9.7.3), represents a torque associated with signal "9" that is exerted from the motor controlling the free leg around the pitch axis, torque "7" is exerted towards the hip around the roll axis, and torque "3" swings the free leg around the roll axis.

In this chapter the behaviour of the robot will be documented by using a plot of the position of the robot's free foot. Figure 8.1 shows the execution of the actual trajectory: the original position of the biped is drawn and consequent positions are shown only as dots. Each dot indicates where the free foot of the robot would be. The start values for $\theta, \psi, \phi, \eta$ and $\zeta$ were -2.5, 5.0, -4.8, -0.2, and -6.0 respectively. The command sequence was the same as above.

Figures 8.2 to 8.5 document the effects of various heuristics. For most of the rest of this chapter the documentation of the robot's behaviour will be restricted to plotting only the position of the free foot. Using this representation Figures 8.2 to 8.5 document the effects of the various heuristics introduced in Section 8.1.1. The heuristics have been applied to the gait depicted in Figure 8.1.

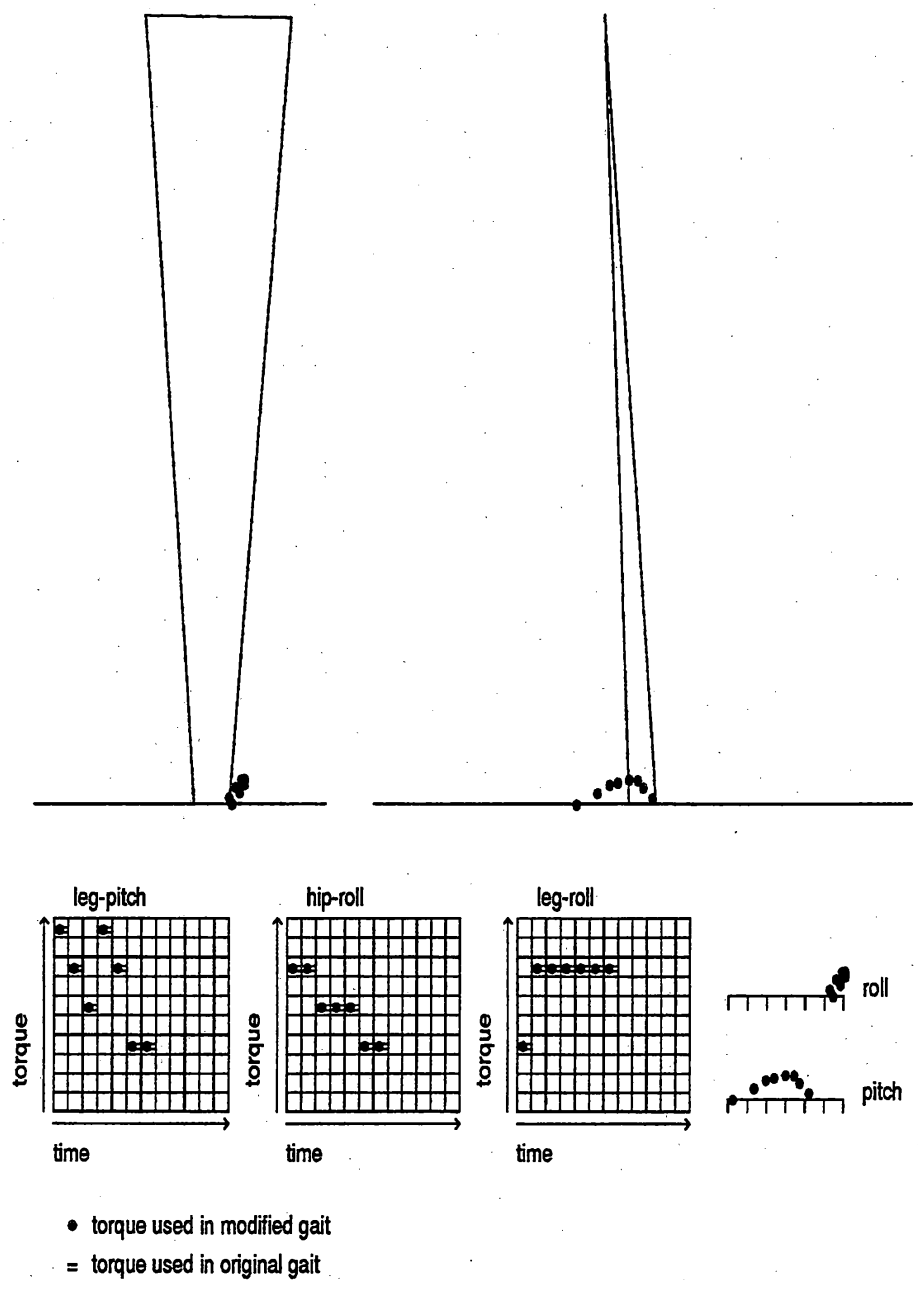



Figure 8.1: The robot executes one step. Small black dots indicate the position of the free foot during the gait execution.

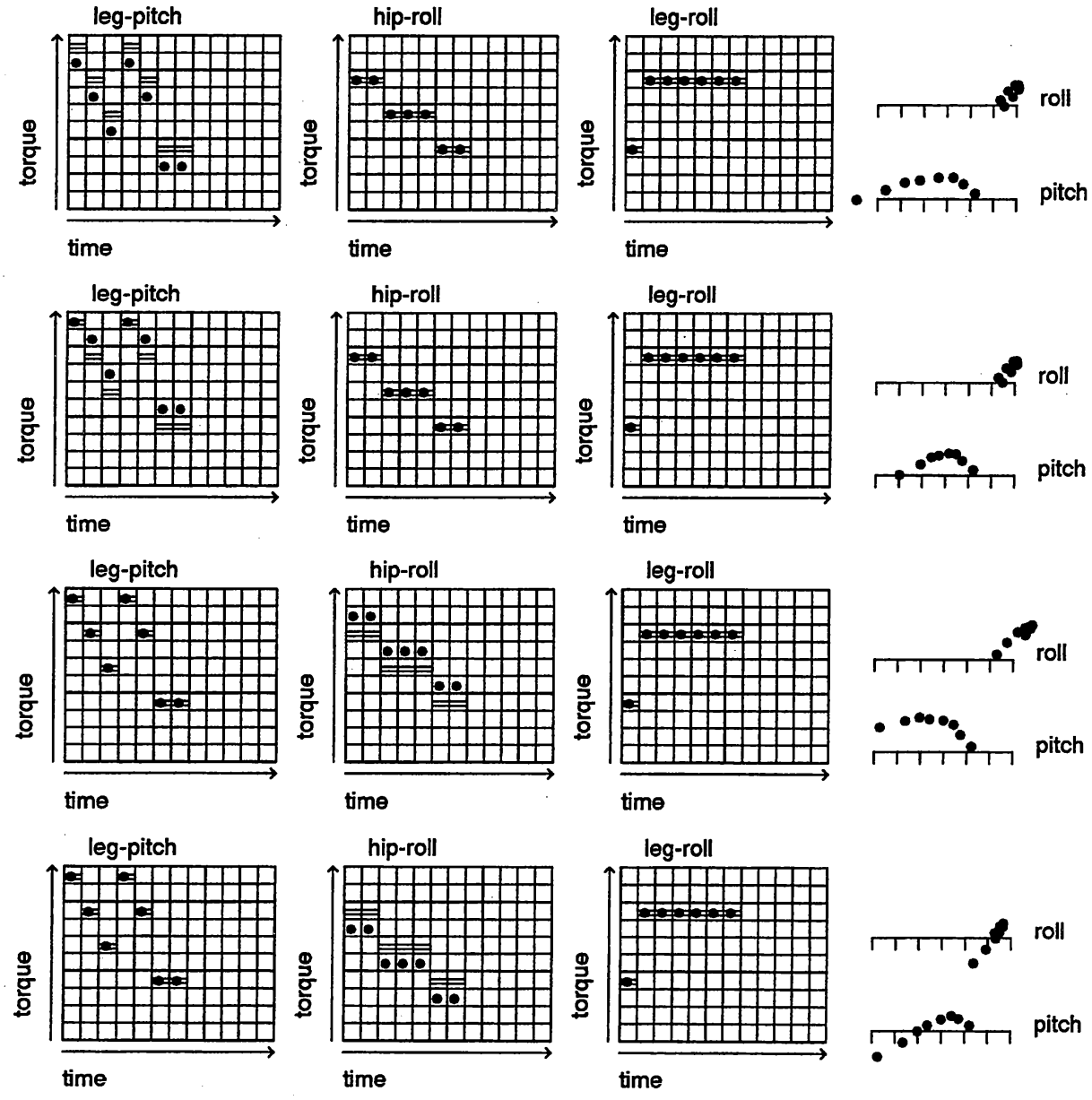



Figure 8.2: Application of various gait modifications: applying stronger and weaker torques to the hip axis and the leg around the pitch axis.

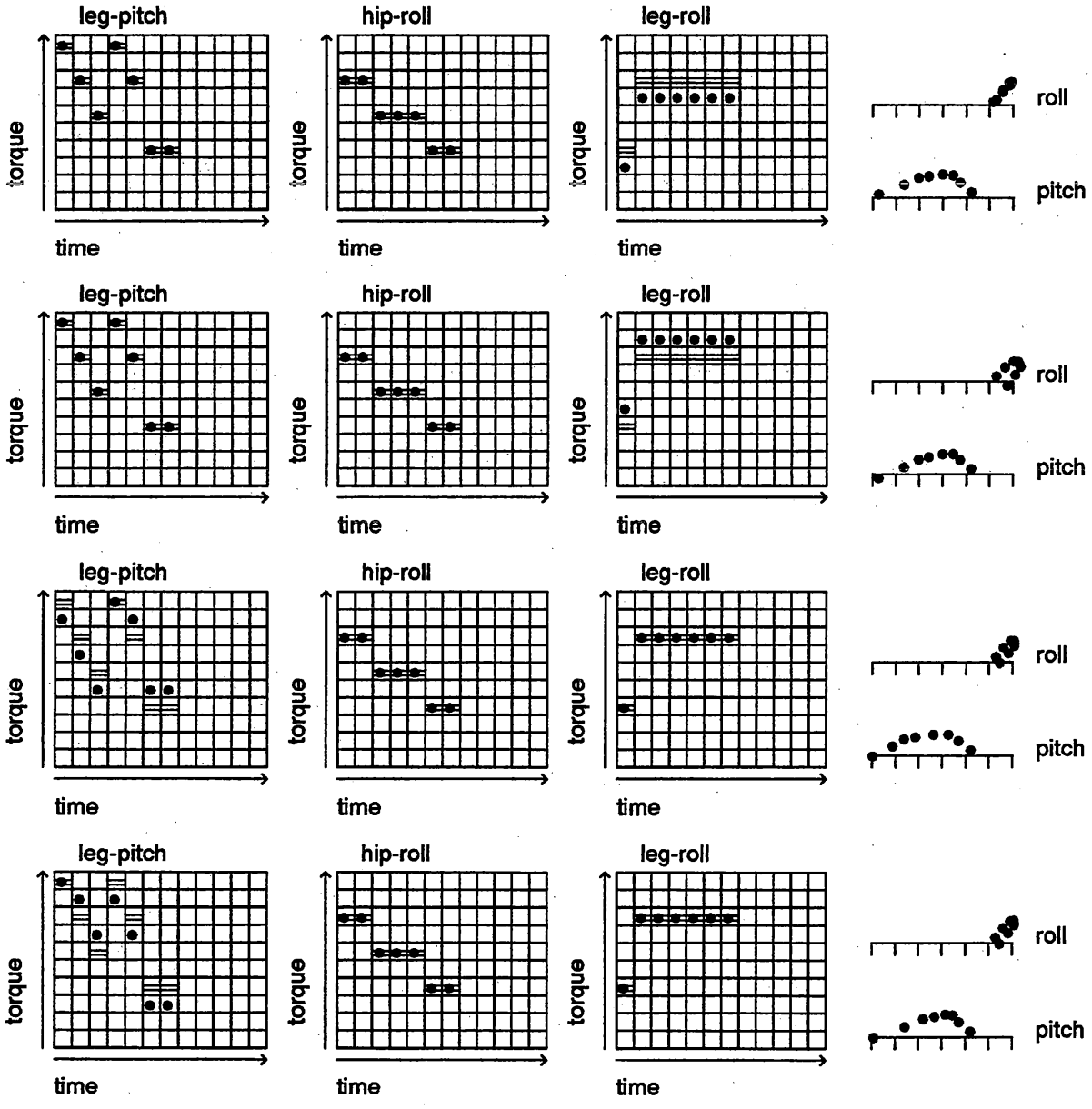



Figure 8.3: The continued application of various gait modifications: applying softer and stronger torques to the leg around the roll axis and amplifying the torques applied to the leg around the pitch axis.

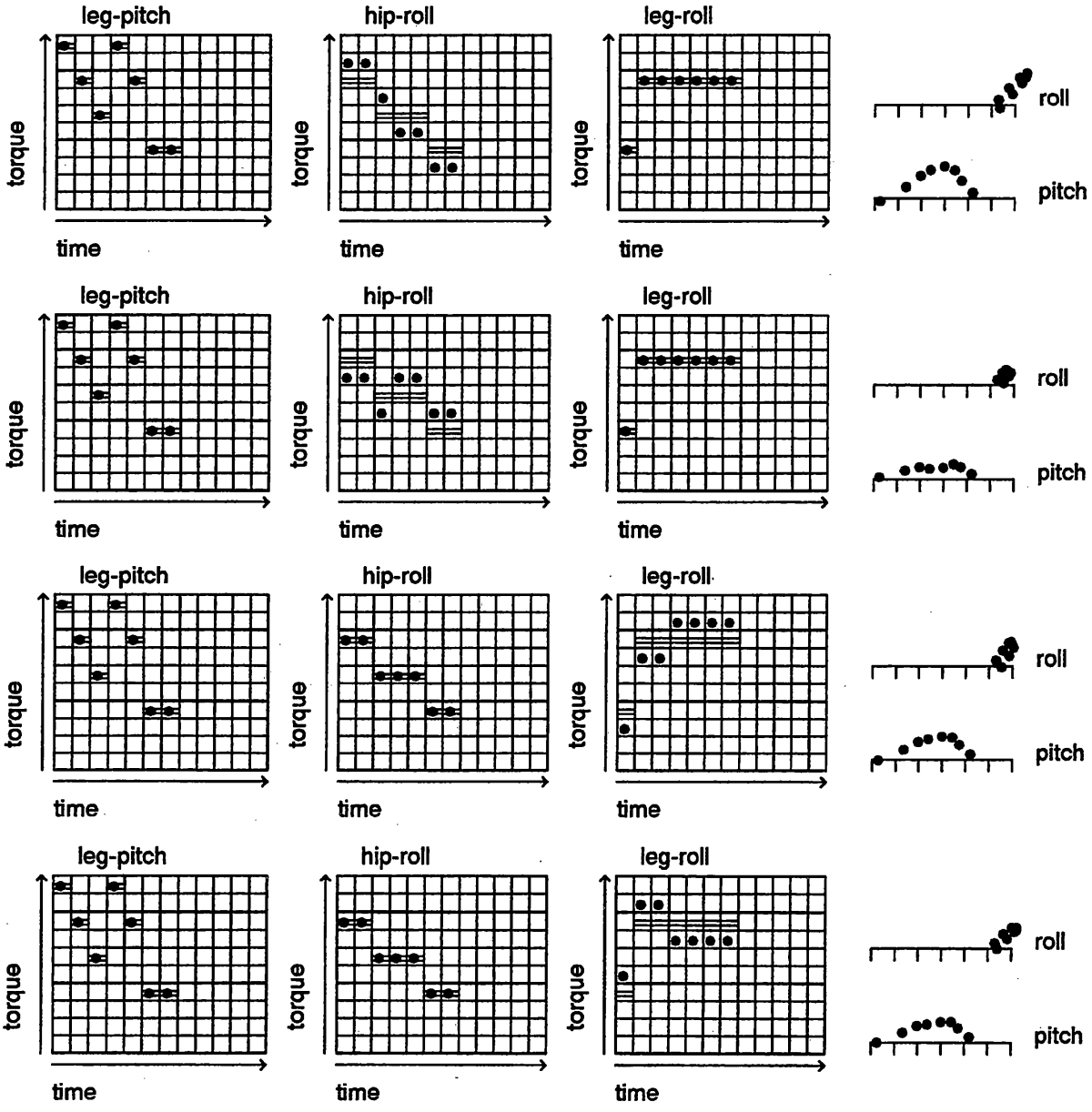



Figure 8.4: The continued application of various gait modifications: amplifying the torques applied to the leg and the hip around the roll axis.

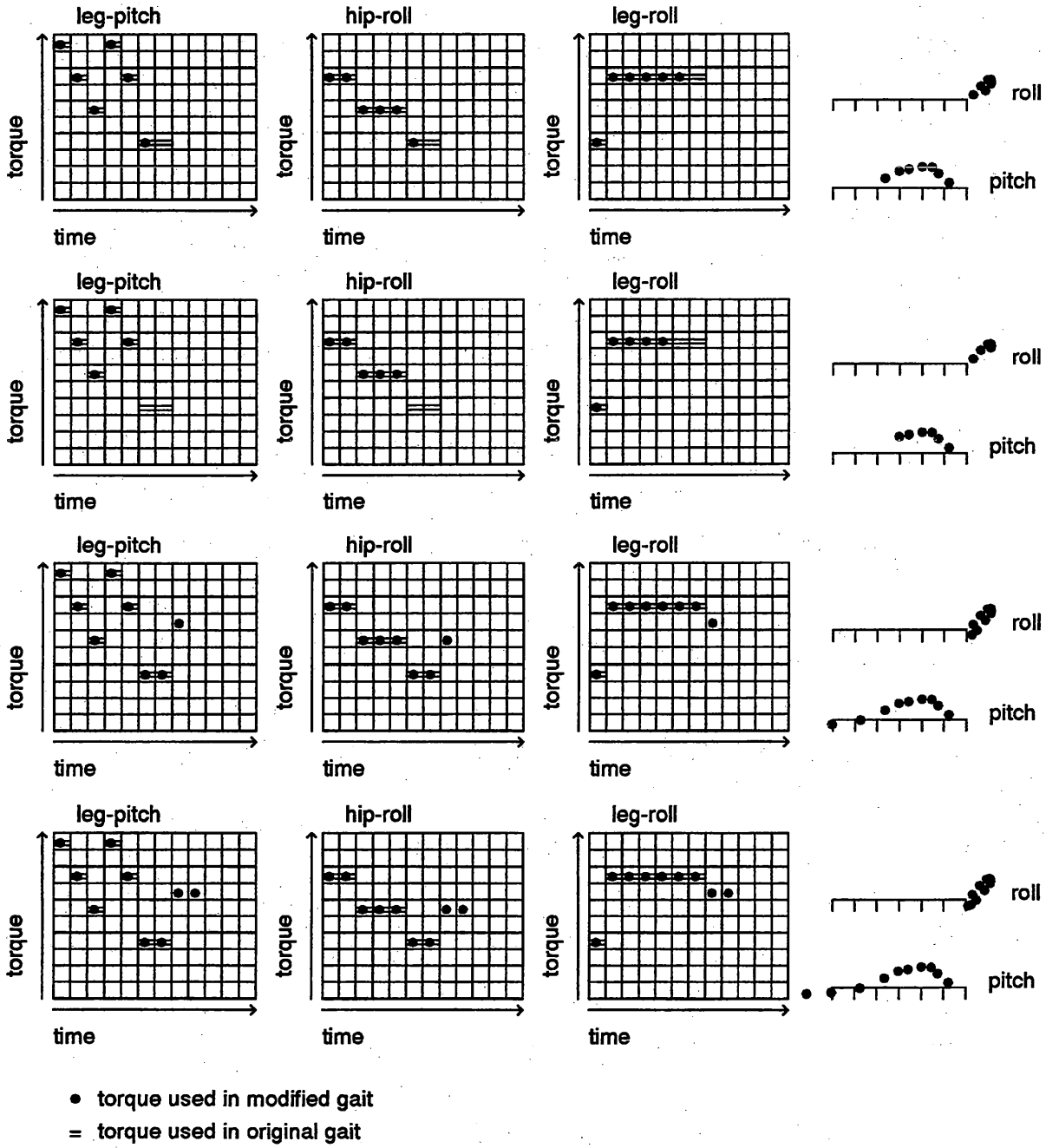



Figure 8.5: The continued application of various gait modifications: adding and subtracting motor commands.

## 8.2 Searching for Obstacle Crossings Using the Qualitative Equivalence Heuristic

Whenever the robot encounters an obstacle its normal gait execution will fail and the robot will have to search for alternative strategies to cross the obstacle. These alternatives will be provided by generating new gaits based on the gait modifications introduced in Section 8.1.1. The robot will search for a successful obstacle crossing by searching over the space of qualitatively equivalent gaits.

Originally the robot will only "know" one gait, which is the standard gait which it developed in order to cross a planar surface. This gait will be the starting point for the search. Using the gait modifications as operators the robot will perform a hillclimbing search until it reaches an executable gait. The evaluation function for the search will be the displacement of the robot towards and across the obstacle. The gait found by the search will be executed. If this is sufficient to cross the obstacle then the robot will proceed with its usual gait. Otherwise it will repeat the search for an executable gait. The difference is that at this point the robot will now have two gaits in its repertoire: the standard gait and the just developed obstacle crossing gait. If the robot starts to search for a further step by using modifications of both the standard gait and the newly acquired obstacle gait, then this search will be called *training*.

Training introduces a trade-off: by storing each successfully used gait the breadth of the search increases while the depth of the search decreases. The more training occurs the more likely it becomes that the required gait can be created from one previously learned by applying only one or two modifications.

This trade-off is discussed by Iba [Iba89], who uses static and dynamic filters to keep the number of new macros (which correspond to gaits in this section) under control. Static filters define fixed criteria which every new macro has to satisfy. These criteria include domain dependent knowledge, a maximum length for each macro, and degree of redundancy. A dynamic filter assigns credit to each macro that has been used in obtaining a solution. After a number of tests all macros below a certain credit rating are discarded.

All the algorithms and heuristics that are used in this section will be documented using a fence as obstacle. This was done in order to limit the number of diagrams. All other types of obstacles (steps, trenches and slopes) could be crossed with less search effort. The documentation will view only the free foot of the biped along the pitch and the roll axis. Black dots indicate the position of the foot as the biped crosses the obstacle from the right to the left. This is done in order to make the documentation of the experiments more concise. All graphs use the same scale which is approximately 1:4.

### 8.2.1 Searching from Scratch

For a first test the gait modifications were used to generate obstacle crossings. The search started from the gait displayed in Figure 8.1 and finally succeeded in generating a successful fence crossing. Figure 8.6 and Figure 8.7 show the successful crossing of a small fence.

It is interesting to note the preconditions for such a successful obstacle crossing. First the extent to which the original gait influences the success of the search will be investigated. If the original gait is already very similar to an obstacle crossing then the search will obviously become much easier. At the same time it could be argued that the hillclimbing search will produce a series of successive gait modifications which pull the free leg of the biped further and further over the obstacle. For this purpose the robot will be given two different gaits. One pair of "strong" gaits that lifts the free leg of the biped about 10mm above the ground (the biped itself is only 308mm tall), and a pair of "weak" gaits that lifts the leg about 6mm above the ground. Figure 8.8 and Figure 8.9 show the respective behaviour.

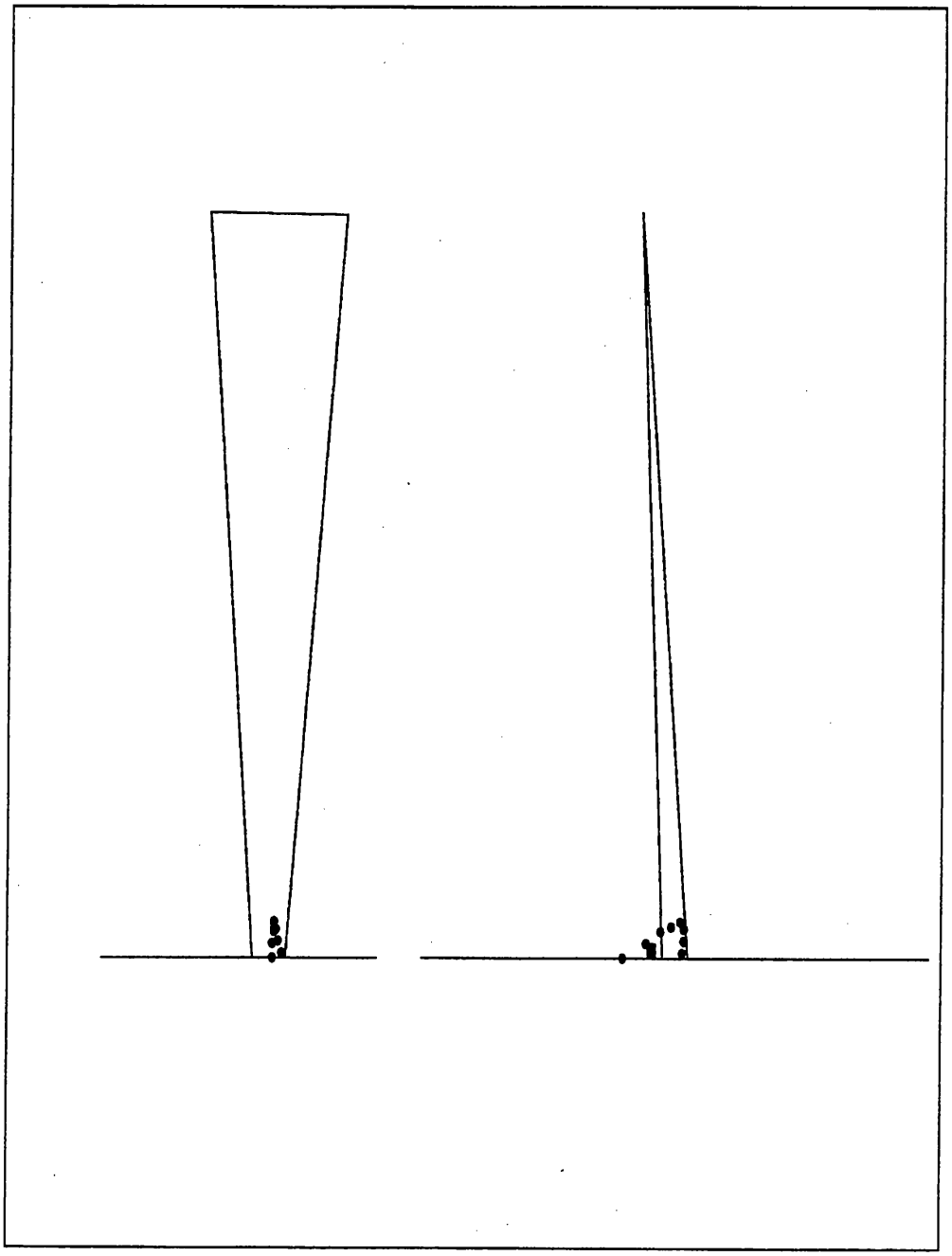


Figure 8.6: The biped crosses a small fence. The gait has been found by applying various successive gait modifications. Small black dots indicate the position of the free foot during the gait execution.

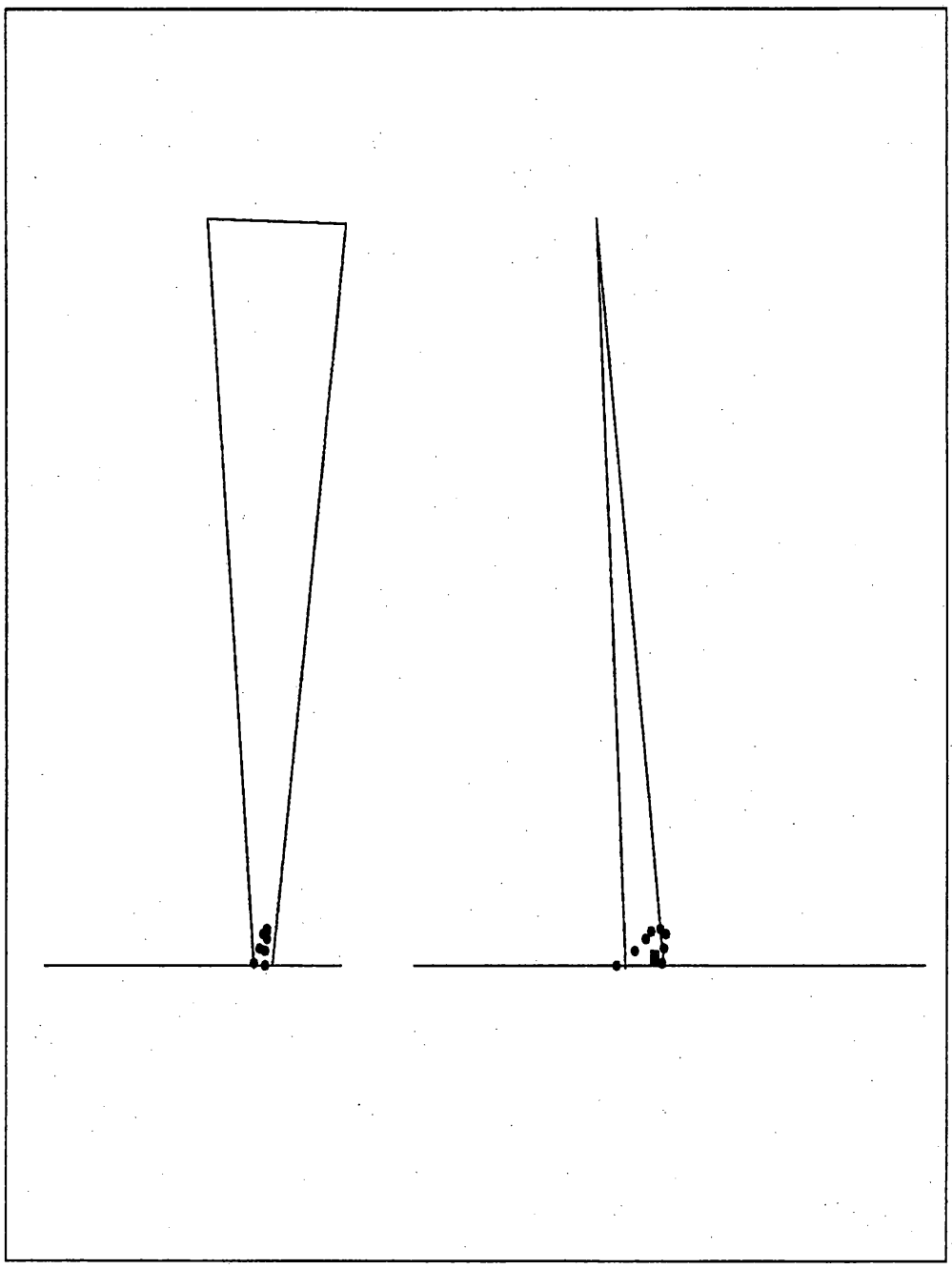


Figure 8.7: The second step of the biped over the fence. The gait modifications provide a gait which first pulls away the free leg from the obstacle and then lifts it to the other side.

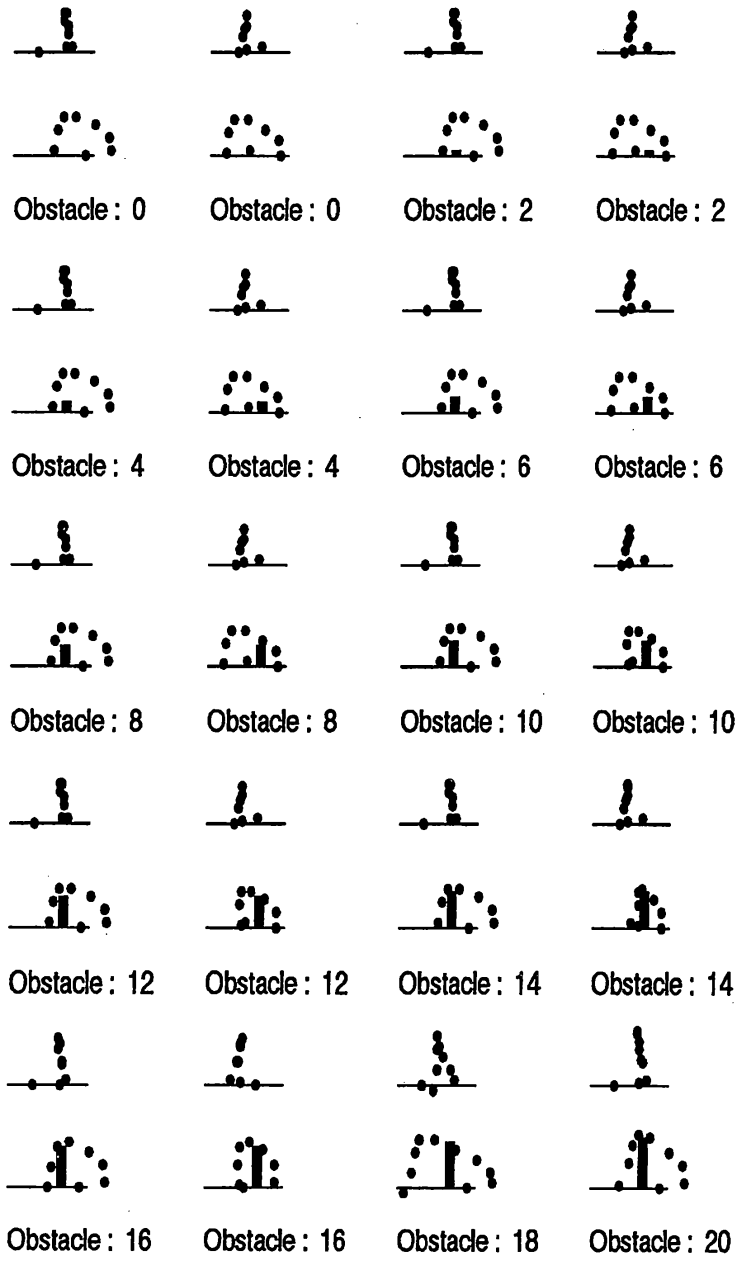



Figure 8.8: The robot tries to cross obstacles of different heights discarding the experience gained in previous attempts. It only uses the set of "strong gaits". Successful obstacle crossing gaits are *not* stored. The robot fails to lift the second leg over the obstacle at heights greater than 16.

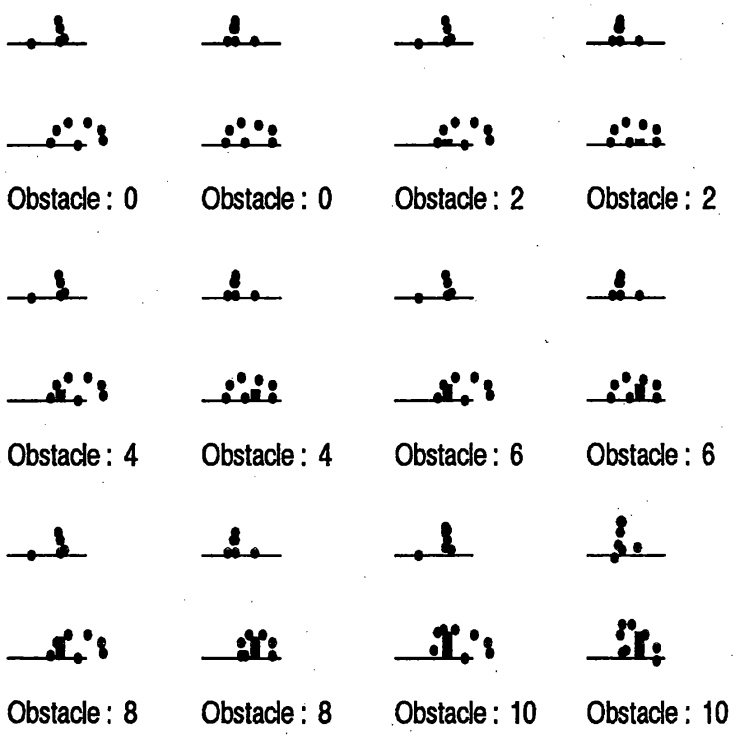



Figure 8.9: The biped tries to cross obstacles of different heights discarding the experience gained in previous attempts. It only uses the set of "weak gaits". Fences of a height of up to 10mm are crossed successfully.

### 8.2.2 The Effect of Training

The fence crossing experiments were repeated. This time the robot stored every successful gait. The search algorithm for new obstacle crossing gaits searched for modifications of all previously successful gaits. One would expect the performance of the robot to improve since the search covers a wider range of gaits. This is only partially true as Figure 8.10 and Figure 8.11 show. The wider range of gaits leads into local minima and in some cases the performance of the biped deteriorates.

This can be seen in Figure 8.11: without training, the biped crosses fences of up to 10mm height. Surprisingly with training the behaviour is (for an obstacle height of 10mm) worse than it was when no additional training gaits were available! The biped is able to cross the obstacle with its first step, but is not able to cross the obstacle with its second leg when the obstacle height exceeds 10mm. However, the robot succeeds in lifting at least one leg across the obstacle when the obstacle height is greater than 10mm. Without training the robot did not achieve this. Compare this behaviour with the one documented in Figure 8.9.

In a second set of experiments the biped was put in the same posture, but once 5mm closer and once 5mm further away from the obstacle. Again the robot tried to find gait modifications which would enable it to cross the obstacle, and successful gaits were used to widen the search.

Figure 8.12 shows how the robot starts its search 5mm closer to the obstacle. It successfully crosses fences of up to 12mm height, and the overall performance is superior to the one displayed in Figure 8.11. Similarly Figure 8.13 and Figure 8.14 show the performance of the robot when it starts 5mm further away from the obstacle. For heights greater than 4mm the robot fails to lift the second leg over the fence. These results underline the sensitivity of the qualitative equivalence heuristics with respect to the distance of the obstacle. It shows that the gaits cannot be "stretched" to cross a wider obstacle, so the robot has to use some gait simply to come closer to the obstacle and then to try again. In this case it ends up in a position which is unsuitable for any of the gaits which it has developed so far, and the search fails. In some cases the robot effectively moves the second leg *away* from the obstacle. This is because the search algorithm executes the best executable gait available. If it cannot execute a gait that leads it closer to or over the obstacle, then the robot executes a step away from the obstacle.

This time only the weak pair of gaits is documented. Using a strong pair of gaits yields similar results when the robot is 5mm closer to the obstacle. However the use of a strong pair of gaits enables the robot to cross fences of up to 14mm height when the robot starts 5mm further away from the obstacle.

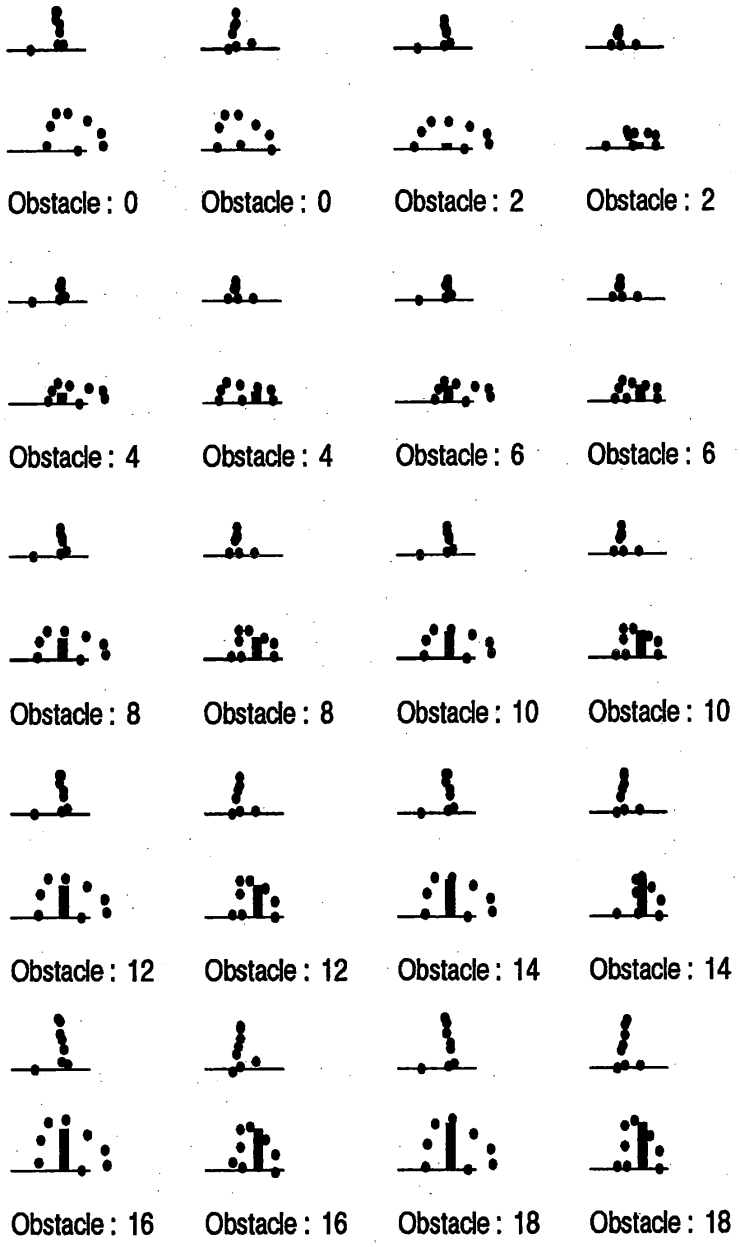



Figure 8.10: Using strong gaits and accumulating experience during the trial. The robot succeeds in crossing a 18mm high fence but fails for any higher fences. Without training the robot failed at fences higher than 16mm.

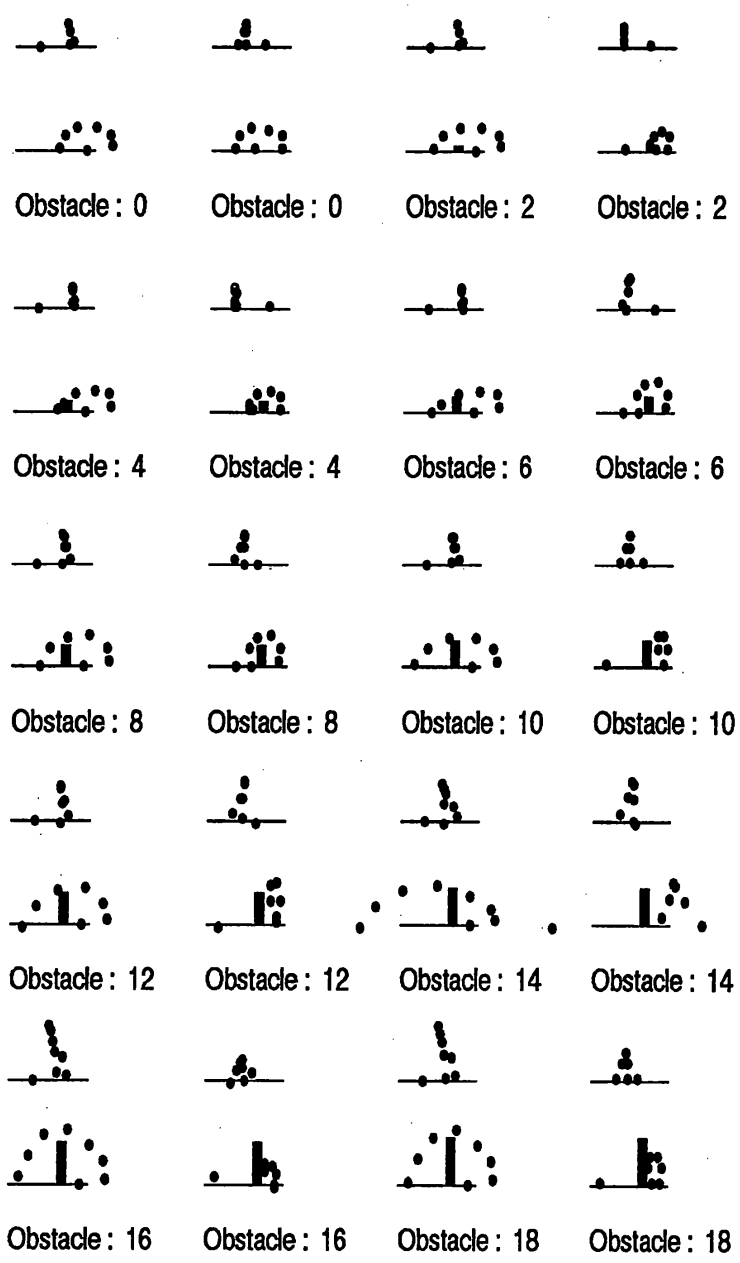



Figure 8.11: Using weak gaits and experience from previous training examples in order to cross an obstacle.

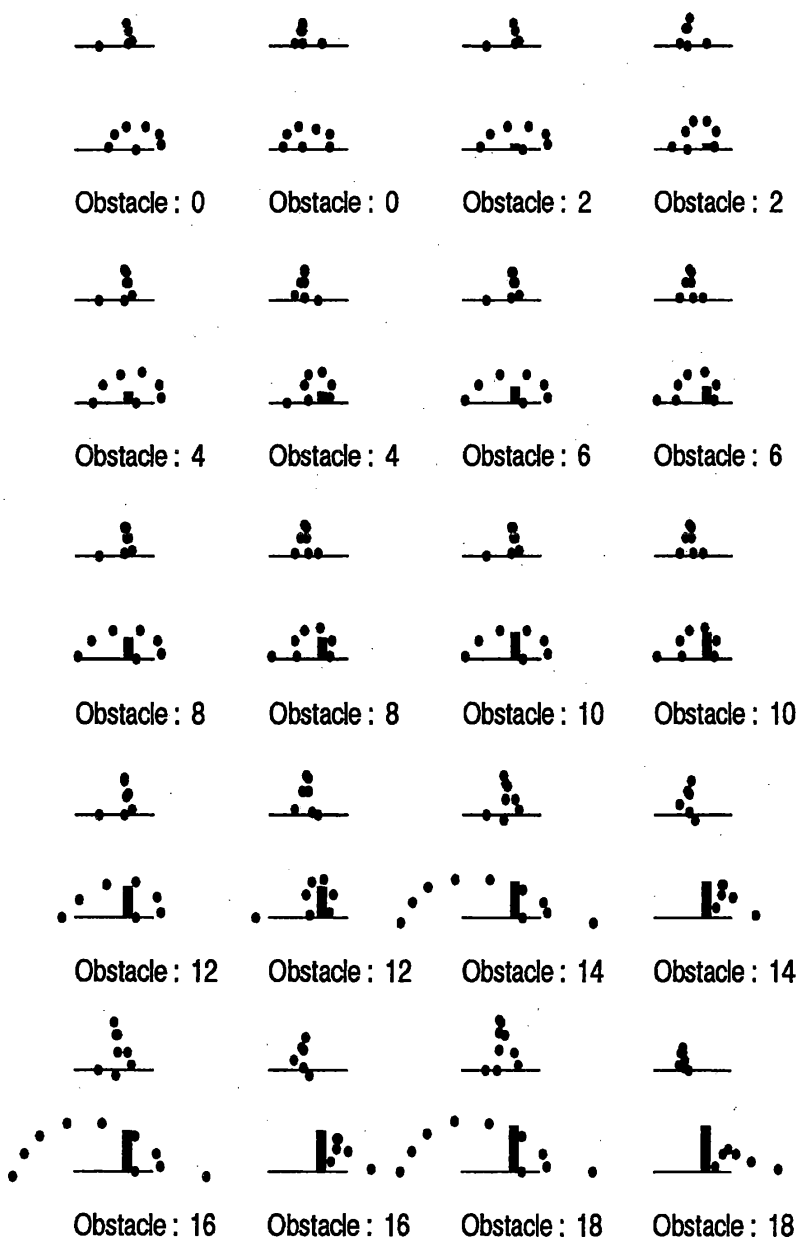



Figure 8.12: The robot crossing the obstacle, with the start position 5 units closer to the obstacle. Each successful training example is recorded, and originally the set of "weak" gaits was used. The behaviour is good for heights of up to 12 units but then deteriorates drastically: the second leg is actually moved away from the obstacle.

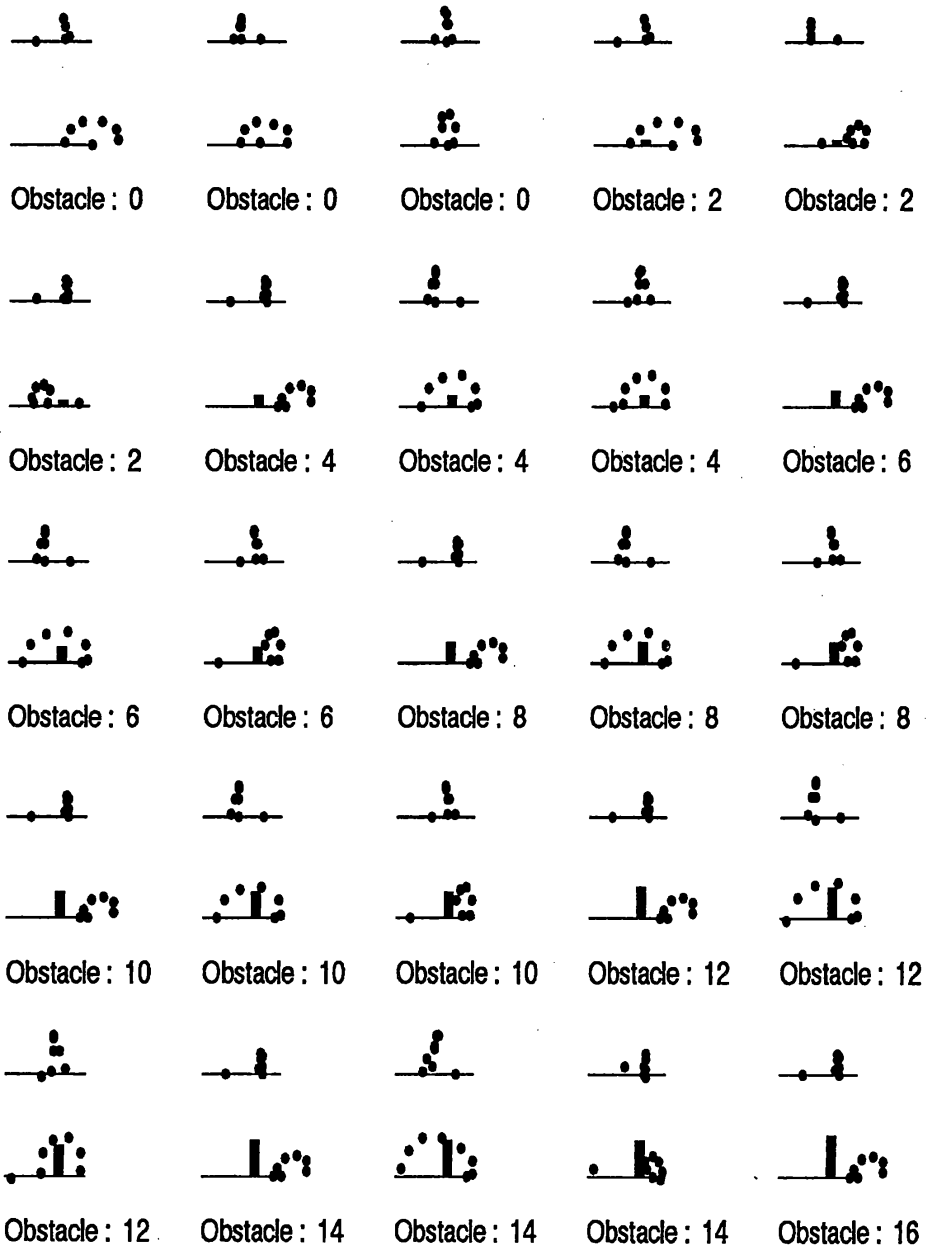



Figure 8.13: The robot crossing the obstacle, with the start position 5 units further away from the obstacle. Each successful training example is recorded, and originally the set of "weak" gaits was used. This time 3 consecutive steps are recorded. The robot fails to cross obstacles higher than 4mm.

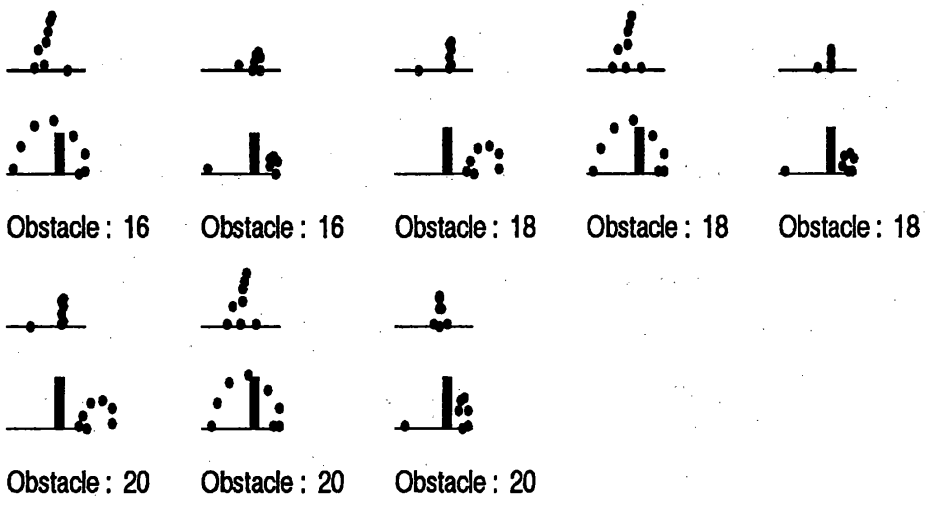



Figure 8.14: The continuation of Figure 8.13. Only one foot can be lifted over the obstacle.

### 8.2.3 Robustness: the Transfer of Training Data

So far qualitative equivalence has been tested with the biped always in the same start position. In all previous experiments the angles of the joints are the same at the beginning of all obstacle crossings. However, it cannot always be assumed that the biped reaches an obstacle in precisely this posture. It was thought to be worthwhile to explore to what extent successful gaits can be used in new postures.

Fortunately Chapter 7 demonstrated that it is possible to control the robot in such a way that it remains close to the desired trajectory. Even if the robot approaches the obstacle from a posture which is different to the start posture of the obstacle crossing gait, on average two steps will be enough to bring the robot close to the desired trajectory, and hence at the end of the second and any later step the robot will be in a posture *similar* to the start position of the obstacle crossing gait. In this section it will be tested whether such a position is similar enough. As Section 8.2.2 demonstrated the distance to the obstacle is important for the success of the obstacle crossing. Being in the right posture is of little help if the displacement relative to the obstacle does not suit the gait.

Figure 8.15 shows the robot attempting to cross the fence after it started in a posture different from the start position in the original gait. This new start position has been derived by starting the robot several steps away from the fence and then letting it step (using the gait controller) towards the fence. The robot tries the qualitative equivalence heuristics after the gait controller failed to execute further steps. Figure 8.16 shows the successful obstacle crossings of the biped using the strong pair of gaits.

Using the weak pair of gaits the robot failed to cross obstacles higher than 4mm. This is documented in Figure 8.17. This very poor performance reveals that some gaits are completely inadequate for use in new start positions.

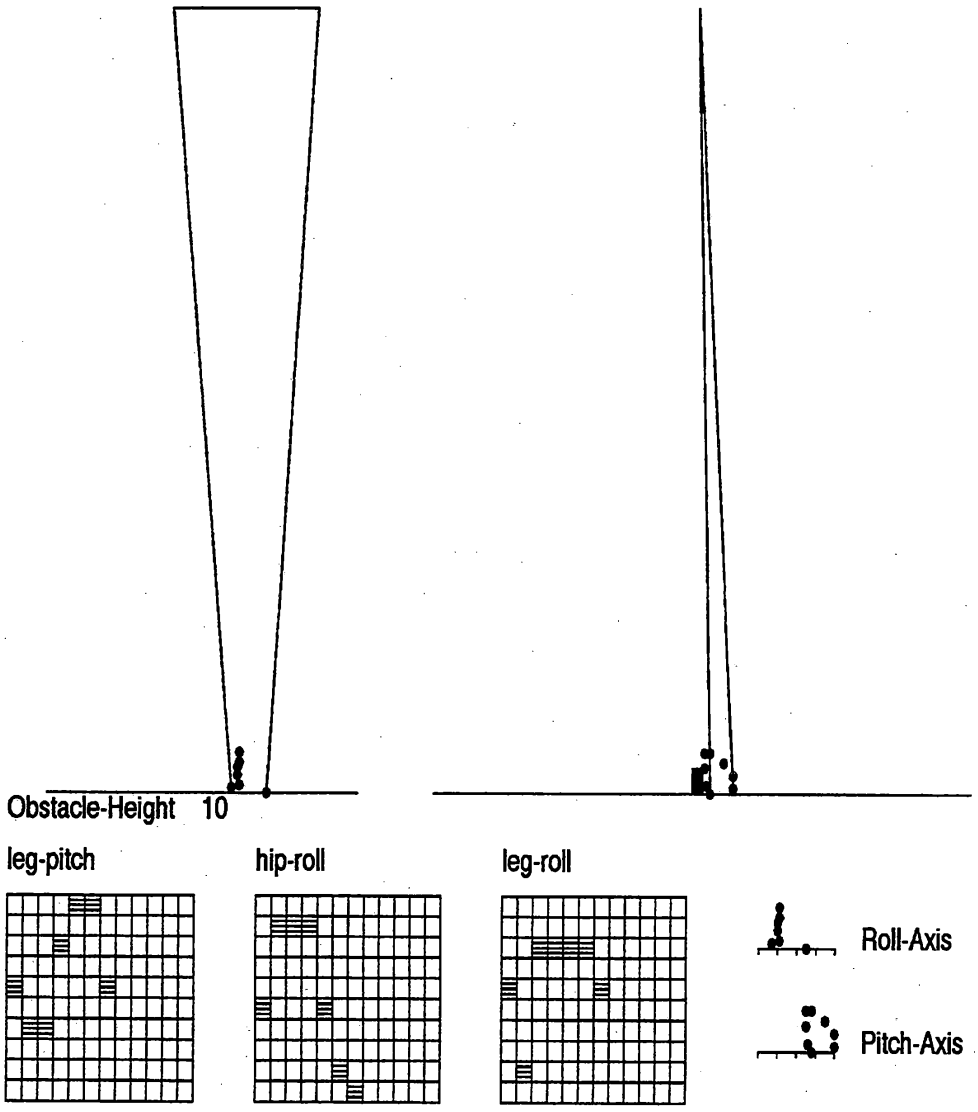



Figure 8.15: This is the position from which the robot will try to cross the obstacle in the next two experiments. This position was derived from when the robot was *searching* to get into the position in which it started in the previous examples. Thus it represents a position similar to the ones we are likely to encounter if we are using the previously described gaits. In this actual example the robot tries unsuccessfully to cross an obstacle and brings the foot safely down in front of the obstacle.

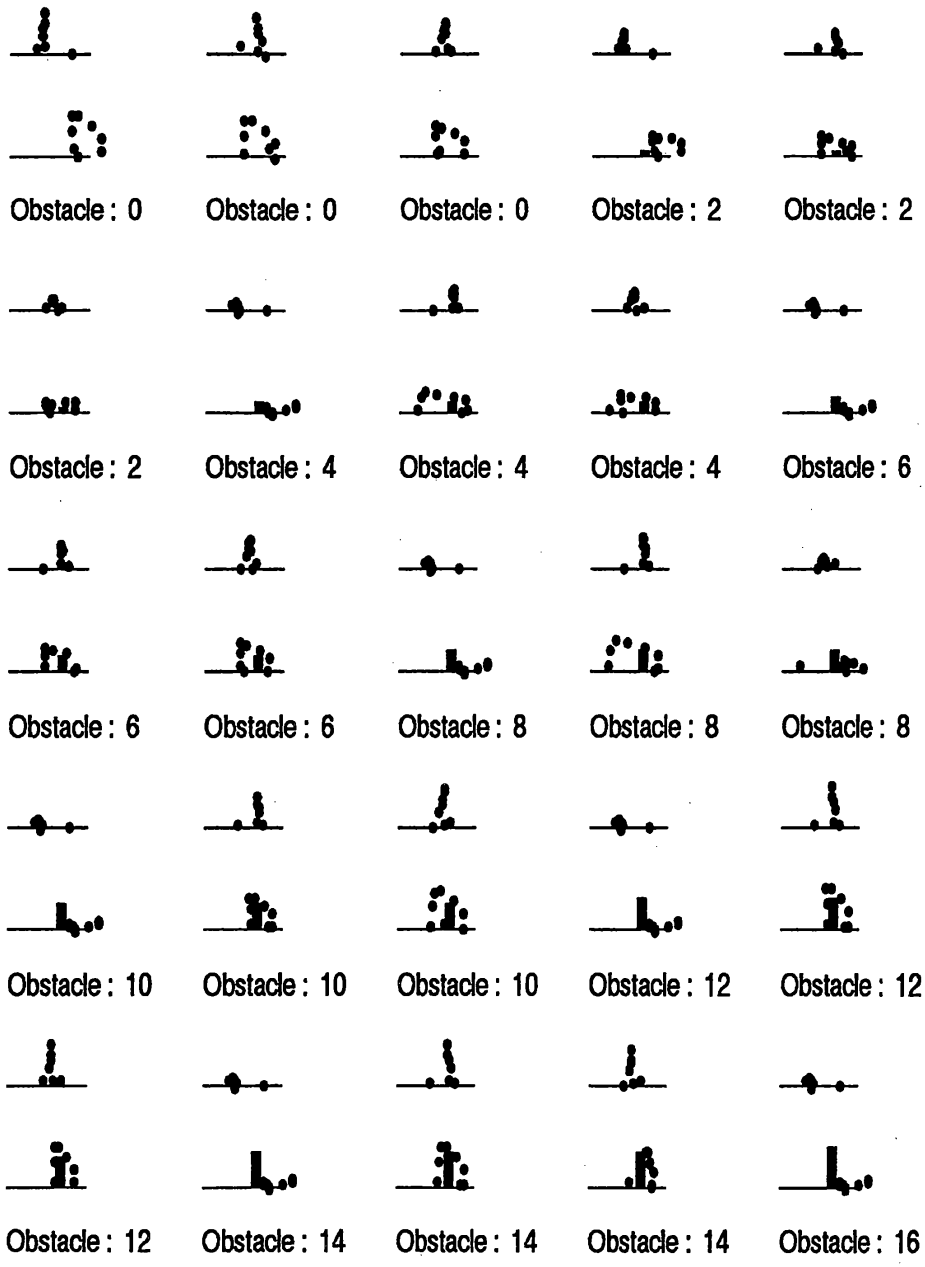



Figure 8.16: The set of strong gaits is used in order to cross a series of increasingly higher obstacles. Successful gaits are stored and used. The position described in Figure 8.15 is used as a start state. Observe that the biped fails to cross the fence at a fence height of 8mm. The biped does not succeed to lift any leg over fences which are higher than 14mm.

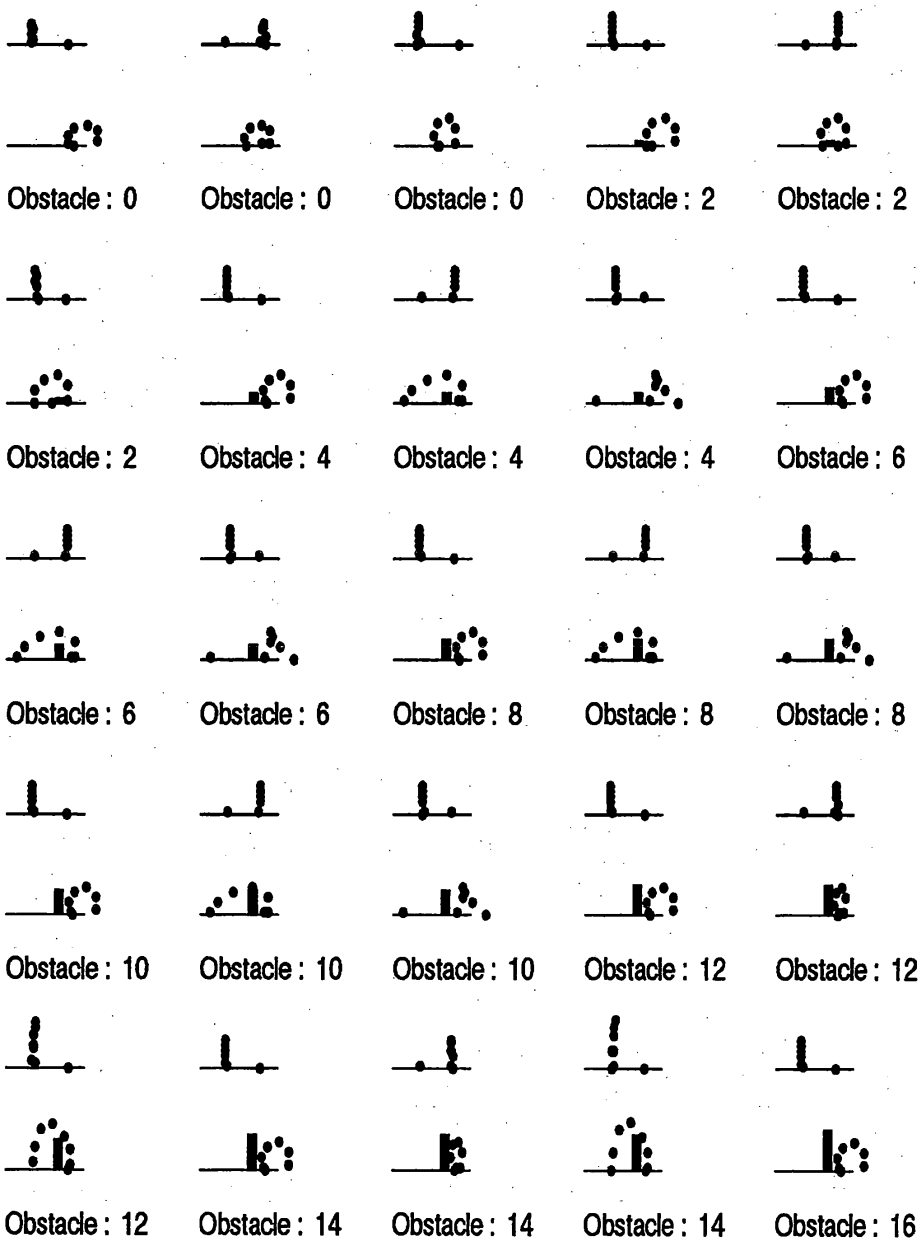



Figure 8.17: The set of weak gaits is used in order to cross a series of increasingly higher obstacles. Again the position described in Figure 8.15 is used as a start state. The obstacle crossing performance is very poor.

## 8.3 Discussion

This chapter described a sequence of experiments which pointed out the power and limitations of the qualitative equivalence heuristics. Although the heuristics were able to achieve obstacle crossings, it proved difficult to develop a generic obstacle crossing ability. We will briefly summarise the main results:

- Improvement. The use of gaits and the search via gait modifications enables the robot to develop a behaviour that is unobtainable using the original search method.
- Brittleness. A single gait does not guarantee a certain behaviour unless the preconditions for the execution of the gaits are exactly matched. In many cases one leg is successfully lifted over the fence but in consecutive steps all that happens is that the legs are actually moved further apart from each other. The robot ends up standing with one leg on each side of the obstacle, both of them moving further away from each other and therefore leading into disaster.
- Locality and lack of power. The search for new gaits based on gait modifications depends heavily on the property of the gait from which the search started. If a gait is modified in order to achieve a desired behaviour which is distinctively different from the original gait then the search is almost guaranteed to fail. Gait modifications have to be more powerful if they are to be used extensively.
- Training improves performance. It is possible to train for a certain behaviour by providing the program with tasks of increasing complexity. The provision of training examples is similar to a guided search through the search space, where only neighbouring areas leading towards the correct solution are searched.
- Training is not very powerful. Training for a certain behaviour improves the performance but does not provide a general solution to the problem of obstacle crossing. In fact it only seems useful where the desired behaviour is similar to an already learned behaviour.
- Overtraining can occur. If the robot learns too many obstacle crossing gaits it will often find and use gaits that do not lead to failure but rather into a situation that is even further away from the goal. Running the same experiment, recording each time more actual steps, results in a different, in some cases even worse behaviour. This is due to the fact that the heuristic which selects the gait only looks at the height of the foot after the execution of the gait. The more gaits the robot stores, the more likely it is to find one that places the foot precisely on the surface without crossing the obstacle.

Thus the effects of overtraining can be limited by introducing an evaluation function which only selects steps which contribute to the obstacle crossing. One way of doing this is to introduce a selection procedure which only accepts gaits which lead the robot towards and over the obstacle.

- Complex evaluation functions are needed. One of the reasons for the occasional failure of the robot to find a gait which lifts its legs *over* the obstacle is the fact that its current search procedure uses a very simple evaluation function. Of all *executable* gaits the robot chooses the one which brings it closer or across the obstacle. This however introduces a severe limitation: if the robot can immediately execute a gait which corresponds to a step backwards, then the robot will execute this step despite the fact that searching slightly deeper into the space of gait modifications would yield a step forwards.

  It is clearly insufficient to search for a behaviour that is "only" specified in so far as it has to bring both feet back to the ground. Whether or not the obstacle is crossed could be included along with other features like the similarity to the original start position. This would almost erase the overtraining symptoms. It should be stressed again that the goal of this chapter is to develop a simple obstacle crossing heuristic without feeding in too much information. One can always try to find and code an evaluation function which is nothing less than a precise instruction on how to cross the obstacle.

- Filters. Gait modifications have to be evaluated and filtered in order to reduce the growth of the search space. An algorithm like Holland's "bucket brigade" credit assignment algorithm [BGH87] could lead to considerable speed-up. One could then also search over the space of gait modifications in order to extract useful *and* powerful operators. Experiments have shown that the biped normally only uses 4 out of 16 gait modifications. These gaits could then be used as the start point for further gait modifications, i.e. one could create new heuristics that do the same thing but are more powerful.

- Limited use. Gaits should be memorised for standard situations like walking in the plane. In this case they lead to considerable speed-up and help to avoid search almost completely. In order to tackle obstacles gait modification is useful as a search technique but the resulting gaits are relatively difficult to use in other situations.

As a result, we are in a position where the robot is able to cross *some* obstacles. The robot couldn't develop a generic obstacle crossing strategy since the success of the qualitative equivalence heuristics depends heavily on the robot being in the right position with the right posture. The failure of the robot

can be attributed to two main shortcomings: first, the algorithm produces too many different gaits and the search suffers from a large branching factor. Second, the training data are not classified or analysed and it is therefore easy to use them out of context. Thus, we have two choices for the development of a *generic* obstacle crossing capability:

(1) In order to investigate further into the development of qualitatively equivalent behaviours, the robot would need to develop the ability to administer a reasonably large set of gaits. These gaits would either ensure that the robot reaches the obstacle in the very position from which it has a working obstacle crossing gait, or they would ensure that the search for obstacle crossing gaits was powerful enough to adapt an existing gait to the current obstacle. In either case the reasoning of the robot would be on the level of behaviours and behavioural modifications.

(2) Alternatively one can look at the existing obstacle crossings and consider them as a successful *example* of how to cross an obstacle. The task of the robot is then to transfer this experience to new situations. Here the robot has the choice of deriving new *behaviours* from old successful ones, or it can derive a new strategy for the search for *actions* (individual motor commands) which constitute the obstacle crossing behaviour.

In the next chapter we will choose the second alternative and use existing obstacle crossings as *examples*. These examples will be analysed in terms of the individual actions which constitute the obstacle crossing behaviour. By doing so we will loose one level of abstraction, but as a result we will gain a generic obstacle crossing capability.

# Chapter 9

# Learning from Examples

At this stage the robot has been able to successfully cross various obstacles. However this obstacle crossing performance was relatively brittle. In order to be able to use the qualitative equivalence heuristic the robot had to stand more or less in a certain posture and with a certain distance in front of the obstacle. In practice it proved to be difficult to approach the obstacle in exactly this way. In order to be able to deal with arbitrary obstacle combinations it is therefore necessary to improve the robot's obstacle crossing capabilities.

In this chapter we will use the obstacle crossings based on the qualitative equivalence heuristic as *examples* of successful obstacle crossings. These examples will then be analysed in terms of how the robot's behaviour changed from its normal behaviour during the obstacle crossing. In order to analyse the changes of the robot's behaviour in more detail, we will analyse the activities of the robot on the level of the individual motor commands. The basic idea will be to see how the choice of a motor command differs depending on whether the robot is crossing an obstacle or executing a normal step on a horizontal surface. We will analyse how these two commands differ from each other and memorise these differences along with the point in the state space where they were observed. In other words we will label the state space in terms of how the robot's choice of actions differs from its normal activities in an obstacle free environment. If the labelling of the state space is sufficiently dense, then it will be possible to generalise a strategy of how to select a motor command if the robot is in a certain area of the state space. Section 9.1 will describe how the different choices of motor commands will be analysed and how the result of this analysis will be operationalised such that the robot can derive a generic obstacle crossing capability.

Originally the robot developed a gait which enabled it to walk on a obstacle free horizontal plane. Whenever the robot is unable to use this gait or search for a step based on the weighting of the inner state space we assume that the

robot is in an *obstacle space*. The obstacle space is defined as a hyperrectangle inside the state space where the robot is unable to execute its standard gaits. An important question for the robot is how to identify the exact location of the individual obstacle spaces, so that it can change its behaviour accordingly. Section 9.2 describes how the obstacle space of each obstacle can be identified.

## 9.1 Analysing Obstacle Crossings

Assume the robot has somehow developed a way to cross a particular obstacle, for example a step. This obstacle crossing could have been achieved by using the help of a teacher or human operator, or the gaits developed by using the qualitative equivalence heuristic might have been used. In any case the robot would not have been able to cross this obstacle by using its standard search based on the inner state space (otherwise the situation wouldn't count as an obstacle).

This specific obstacle crossing will constitute an *example* of how to cross such an obstacle. Using several of these examples the robot will analyse their properties, compare them to its normal behaviour (in the inner state space) and try to generalise an obstacle crossing strategy. In other words: whenever the robot encounters a situation similar to an obstacle crossing example, it will compare its present state with the corresponding state in the obstacle crossing example. It will then see what it did in the obstacle crossing example, and try to do something similar with the current obstacle. This section will describe how the robot achieves these *similar* behaviours. (NB in this section the term *behaviour* is used to denote the choice of action or motor command in a given situation.)

The main emphasis of this section is to give an *operationalisation* of how to achieve *similar* behaviours. Since the robot will rarely be in a state that is exactly equivalent to some state in an obstacle crossing example, it has to match its present state with the states in the obstacle crossing example, and it has to develop a behaviour which is similar to the behaviour in the matching state. This section will define similarities between behaviours by trying to construct evaluation functions which would result in a similar choice of operators during the search for a step forwards.

The comparison of an obstacle crossing with the robot's behaviour in the inner state space has to be defined more precisely. Imagine the robot in some posture $P$ in the obstacle space of some obstacle $O$. Because the robot is *inside* the obstacle space of $O$ it will at least once take an action that is different from the action that it would have taken if it were in the same posture $P$ *outside* the obstacle space $O$. The aim is to compare the actual action of the robot in

some posture $P$ inside the obstacle space with the action in the same posture $P$ in some state away from any obstacle (ie. in the inner state space).

Assume the obstacle crossing gait is a sequence $\mathcal{C}$ of motor commands $u_1, u_2, \ldots, u_n$ and a starting state $\mathcal{S}_0$. This leads the robot through a series of states $\mathcal{S}_0..\mathcal{S}_n$:

$$u_1 \circ \mathcal{S}_0 \rightarrow \mathcal{S}_1$$

$$u_2 \circ \mathcal{S}_1 \rightarrow \mathcal{S}_2$$

$$\ldots$$

$$u_n \circ \mathcal{S}_{n-1} \rightarrow \mathcal{S}_n$$

If the robot searched for steps as if the obstacle were not present, then it would choose for any state $\mathcal{S}_i$ a motor-command $u'_{i+1}$, resulting in a state $\mathcal{S}'_{i+1}$. It would select a motor command which brings the robot as close as possible to some goal state, and the distance to the goal would be measured by an evaluation function based on a weighting of the inner state space.

Provided the robot is given some goal state, then for each state $\mathcal{S}_i$ there will always be exactly one best successor state $\mathcal{S}'_{i+1}$ with respect to a given weighting of the inner state space and the goal state. This state transition $\mathcal{S}_i \rightarrow \mathcal{S}'_{i+1}$ will be called the *standard state transition* of $\mathcal{S}_i$.

This will be used in order to describe an obstacle crossing gait: each state transition $\mathcal{S}_i \rightarrow \mathcal{S}_{i+1}$ will be compared with the result of the standard state transition of $\mathcal{S}_i$, namely $\mathcal{S}'_{i+1}$. Assuming each state is defined as a tuple of parameters

$$\mathcal{S}'_{i+1} : x'^{i+1}_1, x'^{i+1}_2, \ldots x'^{i+1}_n$$

$$\mathcal{S}_{i+1} : x^{i+1}_1, x^{i+1}_2, \ldots x^{i+1}_n,$$

then the qualitative difference $\mathcal{S}_{i+1} \ominus \mathcal{S}'_{i+1}$ can be expressed as the difference per parameter filtered by some threshold $\theta$:

$$\mathcal{S}_{i+1} \ominus \mathcal{S}'_{i+1} = \Delta_\theta(x^{i+1}_1, x'^{i+1}_1), \Delta_\theta(x^{i+1}_2, x'^{i+1}_2), .., \Delta_\theta(x^{i+1}_n, x'^{i+1}_n)$$

where $\Delta_\theta$ is defined as the qualitative difference between the two parameters filtered through the threshold $\theta$:

$$\Delta_\theta(a,b) = \begin{cases} 0 & \text{if } |a-b| < \theta \\ + & \text{if } a-b > \theta \\ - & \text{if } a-b < -\theta \end{cases}$$

This qualitative difference $\mathcal{S}_{i+1} \ominus \mathcal{S}'_{i+1}$ describes how the state transition chosen by the robot in state $\mathcal{S}_i$ differs depending on whether or not there is an obstacle. If state $\mathcal{S}_i$ is inside the obstacle space then the next state will be state $\mathcal{S}_{i+1}$. Otherwise, if state $\mathcal{S}_i$ is outside the obstacle space, then the state transition will be from $\mathcal{S}_i$ to $\mathcal{S}'_{i+1}$. The qualitative difference $\mathcal{S}_{i+1} \ominus \mathcal{S}'_{i+1}$ describes whether the individual parameter values in $\mathcal{S}_{i+1}$ are greater or less than their respective values in $\mathcal{S}'_{i+1}$.

The qualitative difference $\mathcal{S}_{i+1} \ominus \mathcal{S}'_{i+1}$ is however dependent on the state $\mathcal{S}_i$, with the state $\mathcal{S}_{i+1}$ being taken from the obstacle crossing example, and the state $\mathcal{S}'_{i+1}$ being computed on the assumption that state $\mathcal{S}_i$ isn't close to any obstacle. It is important to note that once a state $\mathcal{S}_i$ has been annotated with such a qualitative difference list, it is possible to *approximate* the successor state $\mathcal{S}_{i+1}$. Therefore the qualitative difference between $\mathcal{S}_i$'s actual successor $\mathcal{S}_{i+1}$ and its successor using the standard state transition $\mathcal{S}'_{i+1}$ will be called the *virtual evaluation function* used at state $\mathcal{S}_i$. The robot originally doesn't know why the successor state $\mathcal{S}_{i+1}$ was chosen, but it assumes that some evaluation function was used in order to select the best of all possible successor states of $\mathcal{S}_i$. Since it doesn't know much about this evaluation function the term *virtual* will be used.

In other words: *the virtual evaluation function of state $S_i$ is the qualitative difference between state $S_i$'s standard successor $S'_{i+1}$ and its actual successor $S_{i+1}$*. The virtual evaluation function is basically a labelling of the state space which will be used in order to compute the best possible successor state. In order to generalise from the obstacle crossing examples, the robot analyses the successful obstacle crossings and annotates each state $\mathcal{S}_i$ in the successful crossings with the virtual evaluation function which was used to get to its successor state $\mathcal{S}_{i+1}$. *Whenever in the future the robot finds itself in a state similar to $\mathcal{S}_i$ it will use this virtual evaluation function for its next state transition.*

As an example imagine the robot approaches a wall and starts to lift its free leg. At some point the free leg will be lifted higher than it would be during the execution of a normal gait away from the wall. Thus there will be some state where the leg will be lifted instead of being lowered. This state will be labelled with the qualitative difference between its actual successor state (*lift-leg*) and its "normal" standard successor state (*lower-leg*). Let us call this virtual evaluation function *"lift-leg-higher-than-normal"*. Now whenever the robot approaches a wall again and finds itself in a similar state, it will attempt to find a successor state in which it also lifts the leg higher than normal. How-

ever there could be various possible actions which would lead to a lifting of the leg, and we need to give a more precise definition of how to compute the successor state base on a virtual evaluation function.

A state transition from a state $\mathcal{S}_i$ based on a virtual evaluation function $v_i$ is computed as follows: we assume the robot has $m$ operators $u_1, .., u_m$ available to generate the successor states of $\mathcal{S}_i$. Applying these operators to state $\mathcal{S}_i$

$$u_1 \circ \mathcal{S}_i \xrightarrow{u_1} \mathcal{S}_{i+1}^1$$

$$u_2 \circ \mathcal{S}_i \xrightarrow{u_2} \mathcal{S}_{i+1}^2$$

$$\ldots$$

$$u_m \circ \mathcal{S}_i \xrightarrow{u_m} \mathcal{S}_{i+1}^m$$

results in a set of possible successor states $\mathcal{S}_{i+1}^k, k : 1..m$. Furthermore the standard transition from $\mathcal{S}_i$ can be computed, let this state be $\mathcal{S}'_{i+1}$. For each state $\mathcal{S}_{i+1}^k, k : 1..m$ the qualitative difference $v_k$

$$v_k = \mathcal{S}_{i+1}^k \ominus \mathcal{S}'_{i+1}$$

is computed. Let $\mathcal{S}_{v_i}$ be the set of all those states $\mathcal{S}_{i+1}^k$ with a qualitative difference equal to $v_i$.

$$\mathcal{S}_{v_i} = \{\mathcal{S}_{i+1}^k | \mathcal{S}_{i+1}^k \ominus \mathcal{S}'_{i+1} = v_i\}.$$

For each state in $\mathcal{S}_{v_i}$ the Euclidian distance to the state $\mathcal{S}'_{i+1}$ is computed, and the state furthest away from state $\mathcal{S}'_{i+1}$ is selected as successor state of state $\mathcal{S}_i$. This state has been chosen since the idea is to make the robot behave *differently* from its normal behaviour. By choosing the state furthest away from state $\mathcal{S}'_{i+1}$ the robot chooses an action which maximises this difference.

### 9.1.1 Constructing the Virtual Evaluation Function: an Example

As an example assume that there is a three dimensional state space with parameters $\phi$, $\psi$, and $\chi$. The robot goes through a series of states $\mathcal{S}_1$ to $\mathcal{S}_6$. Suppose the standard state transition of the robot is defined as

$$(\phi, \psi, \chi) \rightarrow (\phi + 5, \psi - 5, \chi).$$

Thus the robot would "normally" increase the first parameter by 5, decrease the second by 5 and leave the third parameter unchanged. That is if the robot is in a state $\mathcal{S}_i : (20, 30, 40)$, then the result of its standard state transition $\mathcal{S}_i \rightarrow \mathcal{S}'_{i+1}$ is state $\mathcal{S}'_{i+1} : (25, 25, 40)$.

Now assume that the robot approaches an obstacle $O$ and displays the following behaviour:

| State | $\phi$ | $\psi$ | $\chi$ |
|---|---|---|---|
| $S_1$ | 20 | 30 | 40 |
| $S_2$ | 26 | 24 | 40 |
| $S_3$ | 31 | 24 | 45 |
| $S_4$ | 35 | 24 | 50 |
| $S_5$ | 40 | 20 | 40 |
| $S_6$ | 45 | 15 | 30 |

Next the standard transitions for each state $S_1..S_6$ are computed. For example the standard transition for state $S_3 : (31, 24, 45)$ is $S'_4 : (36, 19, 45)$ The standard transitions $S'_{i+1}$ for each of these states are:

| State | $\phi$ | $\psi$ | $\chi$ |
|---|---|---|---|
| $S'_2$ | 25 | 25 | 40 |
| $S'_3$ | 31 | 19 | 40 |
| $S'_4$ | 36 | 19 | 45 |
| $S'_5$ | 40 | 19 | 50 |
| $S'_6$ | 45 | 15 | 40 |
| $S'_7$ | 50 | 10 | 30 |

The difference between $\mathcal{S}_{i+1}$ and $\mathcal{S}'_{i+1}$ is computed:

| State | $\phi$ | $\psi$ | $\chi$ |
|---|---|---|---|
| $S_2 - S'_2$ | 1 | -1 | 0 |
| $S_3 - S'_3$ | 0 | 5 | 5 |
| $S_4 - S'_4$ | -1 | 5 | 5 |
| $S_5 - S'_5$ | 0 | 1 | -10 |
| $S_6 - S'_6$ | 0 | 0 | -10 |
| $S_7 - S'_7$ | nil | nil | nil |

In order to arrive at the qualitative difference the values are filtered by some threshold $\theta$. In this example the value chosen for $\theta$ is 2. The qualitative difference of $S_7 - S'_7$ can be discarded because state $S_7$ is unknown.

| State | $\phi$ | $\psi$ | $\chi$ |
|---|---|---|---|
| $S_2 - S'_2$ | 0 | 0 | 0 |
| $S_3 - S'_3$ | 0 | + | + |
| $S_4 - S'_4$ | 0 | + | + |
| $S_5 - S'_5$ | 0 | 0 | − |
| $S_6 - S'_6$ | 0 | 0 | − |

As a result the virtual evaluation functions for each state $S_1, \ldots, S_5$ can be given. Again the virtual evaluation function for state $S_6$ can not be given because the successor state $S_7$ is unknown.

| State | virtual evaluation function |
|---|---|
| $S_1$ | $(0, 0, 0)$ |
| $S_2$ | $(0, +, +)$ |
| $S_3$ | $(0, +, +)$ |
| $S_4$ | $(0, 0, -)$ |
| $S_5$ | $(0, 0, -)$ |

Therefore the robot assumes that whenever it is close to an obstacle of type $O$ it has to use three different strategies: whenever it is in a state similar to state $S_2$ or $S_3$ it uses the virtual evaluation function of the type $(0, +, +)$ and whenever it is close to the states $S_4$ or $S_5$ it uses the virtual evaluation function $(0, 0, -)$. Otherwise the robot uses the normal evaluation function $(0, 0, 0)$.

### 9.1.2 Finding the Successor State Based on the Virtual Evaluation Function

Suppose the robot is in a state similar to $\mathcal{S}_i$ and tries to imitate the behaviour displayed in state $\mathcal{S}_i$. Since the robot is not in exactly the same state, it is not obvious which command it should use in order to mimic the behaviour displayed in state $\mathcal{S}_i$.

It is possible to *approximate* the state transition $\mathcal{S}_i \rightarrow \mathcal{S}_{i+1}$ based on the standard state transition $\mathcal{S}_i \rightarrow \mathcal{S}'_{i+1}$ and the virtual evaluation function used in state $\mathcal{S}_i$. If the robot has a set of possible motor commands (or operators) $u_1, u_2, .., u_m$, then applying these operators to state $\mathcal{S}_i$ will result in the following states:

$$u_1 \circ \mathcal{S}_i \rightarrow \mathcal{S}^1_{i+1}$$

$$u_2 \circ \mathcal{S}_i \rightarrow \mathcal{S}^2_{i+1}$$

$$\ldots$$

$$u_m \circ \mathcal{S}_i \rightarrow \mathcal{S}^m_{i+1}$$

Assuming the standard state transition leads to state $\mathcal{S}'_{i+1}$, then for each state $\mathcal{S}^k_{i+1}, k : 1..m$ the qualitative difference $\mathcal{S}^k_{i+1} \ominus \mathcal{S}'_{i+1}$ is computed. Each state $\mathcal{S}^k_{i+1}$ where this qualitative difference is equivalent to the virtual evaluation function of state $\mathcal{S}_i$ is a potential successor state of $\mathcal{S}_i$, and from these states a state will be selected where the Euclidian distance to state $\mathcal{S}'_{i+1}$ is greatest.

Following the last example the robot might have five different operators available:

$$\begin{aligned}
(\phi, \psi, \chi) &\xrightarrow{op_1} (\phi + 5, \psi - 5, \chi) \\
(\phi, \psi, \chi) &\xrightarrow{op_2} (\phi + 5, \psi + 5, \chi + 3) \\
(\phi, \psi, \chi) &\xrightarrow{op_3} (\phi + 5, \psi + 5, \chi + 7) \\
(\phi, \psi, \chi) &\xrightarrow{op_4} (\phi, \psi - 5, \chi) \\
(\phi, \psi, \chi) &\xrightarrow{op_5} (\phi + 5, \psi - 5, \chi - 8)
\end{aligned}$$

Assume the robot is in state $S_2 : (26, 24, 40)$ and therefore has the following 5 possible successor states:

| State | $\phi$ | $\psi$ | $\chi$ |
|---|---|---|---|
| $S_3^1$ | 31 | 19 | 40 |
| $S_3^2$ | 31 | 29 | 43 |
| $S_3^3$ | 31 | 29 | 47 |
| $S_3^4$ | 26 | 19 | 40 |
| $S_3^5$ | 31 | 19 | 32 |

Assuming that the standard transition from state $S_2$ would lead to state $S_3^1$ the following qualitative differences $S_3^k \ominus S_3^1$ can be computed:

| State | $\phi$ | $\psi$ | $\chi$ |
|---|---|---|---|
| $S_3^1 \ominus S_3^1$ | 0 | 0 | 0 |
| $S_3^2 \ominus S_3^1$ | 0 | + | + |
| $S_3^3 \ominus S_3^1$ | 0 | + | + |
| $S_3^4 \ominus S_3^1$ | − | 0 | 0 |
| $S_3^5 \ominus S_3^1$ | 0 | 0 | − |

The virtual evaluation function used for the transition from state $S_2$ to state $S_3$ was (0,+,+), and therefore states $S_3^2$ and $S_3^3$ are possible successors of state $S_2$. However the Euclidian distance between $S_3^3$ and $S_3^1$ is greater then the Euclidian distance between $S_3^2$ and $S_3^1$, and therefore state $S_3^3$ is chosen as the successor state of state $S_2$. In other words when the robot reaches state $S_2$ and it is supposed to use the virtual evaluation function (0,+,+), it will select operator $u_3$ in order to generate the next state.

Using the current example the robot would reach state $S_3^3 : (31, 29, 47)$, while the actual state transition in the example was $S_3 : (31, 24, 45)$. This demonstrates that the virtual evaluation function concept does not guarantee a state transition that recreates a certain example. It is, however, good enough to approximately recreate a certain type of state transition. Applying the concept of the virtual evaluation function to biped obstacle crossing will reveal the usefulness of these concepts.

## 9.2 Identifying the Obstacle Space

At this point the robot has identified the set of virtual evaluation functions $\mathcal{V}$ which was used during various obstacle crossings. The next task will be to identify where in the state space which virtual evaluation function was used. The underlying assumption of this section is that each virtual evaluation function is used in one single hyperrectangle of the state space. It is assumed that it is possible to identify these hyperrectangles by using linear decision functions. This is a relatively strong assumption concerning the description of the state space of the robot. However it is in line with one of the premises of this thesis[1] which stated that the state space description is so rich that all necessary data are directly readable, which means that for each information on the state of the robot that might be needed at some point, there will be a parameter where this information is directly accessible. No information needed to describe the state of the robot or its behaviour has to be computed by combining various parameters.

Therefore the task of identifying the obstacle space is reduced to the task of identifying the relevant dimensions and then identifying the boundaries of the obstacle space along these dimensions.

The obstacle space was originally defined as the part of the state space (which is the set of all available parameters and their values) in which the robot is unable to carry out its normal search for steps on the basis of the weighting of the inner state space (the inner state space corresponds to the 5 joint positions of the robot). This will happen whenever the robot is so close to an obstacle that it influences the executability of it standard gait. In the following sections we will not only identify the hyperrectangles in which the robot needs to change its search strategy, but also we will split the obstacle space into a disjoint set of hyperrectangles, where within each hyperrectangle the robot will use exactly one virtual evaluation function for its search.

### 9.2.1 Identifying the Dimensions of the Obstacle Space

In this thesis a survival oriented approach to credit assignment is chosen: the survival time heuristic orders parameters based on their impact on the survival of the robot. All parameters which have a certain impact on the average survival time of the robot constitute the *inner state space*. A simplified version of this approach will suffice in order to establish the dimensions of the obstacle space for each type of obstacle.

At this point the robot has a set of examples of successful obstacle crossings.

[1] see Section 2.1, page 6

These examples are defined by a start state $\mathcal{S}_0$ and a sequence of motor commands $C : u_1, u_2, .., u_n$. Applying the set of motor commands to state $\mathcal{S}_0$ results in a successful obstacle crossing. Some of the parameters in the description of state $\mathcal{S}_0$ describe the original *posture* of the robot at the beginning of the obstacle crossing.

In order to discover the dimensions of the obstacle space the robot will test the sensitivity of the obstacle crossing gait to changes of parameters outside the inner state space. This means that the robot will execute the same gait (ie. execute the same commands starting in the same posture), but it will individually alter all other parameters and observe whether these parameters will have an impact on the executability of the gait. Provided a certain parameter is changed and the robot fails to execute the obstacle crossing gait, then this parameter will be included into the obstacle space. Executability of the gait refers to the fact that all commands can be executed and the robot reaches a state in which it can execute a support-exchange command. This definition leaves enough space to tolerate all those parameters which have a small impact on the performance of the robot but do not on average affect the robot's survival during an obstacle crossing. The parameters of the inner state space are also included within the obstacle space, since they have an impact on the robot's survival even if there is no obstacle. Thus they don't need to be tested.

### 9.2.2 Identification of the Obstacle Space Boundaries

Once the dimensions of the obstacle space have been identified the robot tries to identify which hyperrectangles of the state space correspond to which virtual evaluation function. The underlying assumption is that the state space of the robot is sufficiently regular and it is possible to identify relatively few regions where a certain virtual evaluation function has to be used.

Let there be $n$ dimensions in the obstacle space of the robot. Let $\mathcal{S}$ be the set of all states of all successful obstacle crossing exemplars. Let $\mathcal{V} : v_1, .., v_m$ be the set of all virtual evaluation functions annotated to states in $\mathcal{S}$ except for the virtual evaluation function consisting only of zeros (ie. where the robot follows its standard transition). For each virtual evaluation function $v_i$ in $\mathcal{V}$, inspect each state in $\mathcal{S}$ that has been annotated with $v_i$. Let the set of these states be $\mathcal{S}_{v_i}$. For each dimension $j$ of the $n$ dimensions of the obstacle space find the maximum and minimum value of the corresponding parameter in $\mathcal{S}_{v_i}$, and call them $\max_{v_i}(j)$ and $\min_{v_i}(j)$.

Assuming a state $S$ is described as

$$S = (x_1, x_2, .., x_n),$$

then the hyperrectangle $\mathcal{P}(v_i)$ identified with the virtual evaluation function

$v_i$ will be described by the set of all states for which the following holds:

$$\mathcal{P}(v_i) = \{S|S = (x_1, x_2, .., x_n), x_j < \max_{v_i}(j) \wedge x_j > \min_{v_i}(j), j : 1..n\}$$

As an example assume that there is a small obstacle space described by a parameter called *distance-to-obstacle* and another one called *height-of-obstacle*. An examination of the successful obstacle crossing examples identifies a virtual evaluation function which will be called *lift-leg*. Assume the maximum value of the parameter *distance-to-obstacle* when the virtual evaluation function *lift-leg* is used is 10, and its smallest value when *lift-leg* is used is -100. If the corresponding values for *height-of-obstacle* are 50 and 10 respectively, than the hyperrectangle identified with the virtual evaluation function *lift-leg* is described as follows:

$$\begin{aligned}\mathcal{P}(lift-leg) = \{S \mid\ & S = (x_1, x_2, .., x_n) \wedge \\ & 10 < \text{height-of-obstacle} < 50 \wedge \\ & -100 < \text{distance-to-obstacle} < 10\}\end{aligned}$$

### 9.2.3 Refining Obstacle Space Boundaries

At this point the obstacle spaces have been split into various hyperrectangles, each of which represents the use of one virtual evaluation function. In a last step the borders of these hyperrectangles will be refined: it will be examined whether dropping some of the dimensions of each obstacle space is permissible, and the hyperplanes which make up the boundaries of each hyperrectangle will be adjusted in order to improve the performance of the robot.

These adjustments will be carried out individually for each dimension of the obstacle space for each hyperrectangle indicating the use of some virtual evaluation function $v$. Let this dimension be expressed by the parameter $x_d$, and let the two hyperplanes which limit the use of $v$ be

$$A \leq x_d \leq B$$

with $A$ and $B$ being some constants. The robot will be presented with a set of obstacle crossing tasks and the parameters $A$ and $B$ will be adjusted using a hillclimbing algorithm with the number of successful crossings as an evaluation function.

### 9.2.4 The Treatment of Overlapping Obstacle Spaces

Once the hyperrectangles for each virtual evaluation function have been identified it is possible that two such hyperrectangles overlap: this would in effect

mean that the robot could be required to carry out two contradictory activities like lower and lift the leg. The conflict resolution heuristic is to test whether the obstacle crossing capabilities of the robot remain uneffected if the overlapping region is removed from one of the two hyperrectangles. This is in effect an extension of the boundary adjustment of Section 9.2.3 above.

The justification for this heuristic is based on the origin of the obstacle crossing exemplars from which the virtual evaluation functions have been derived. If for example the robot executes a long step over a fence then it might not immediately lower the leg behind the obstacle. Thus part of its behaviour will differ from its normal behaviour without the need to do so. It would be perfectly acceptable (and in fact more stable) if the robot lowered the leg as soon as it has crossed the fence. However the qualitative equivalence heuristics which generated the obstacle crossing exemplars were just looking for *any* successful obstacle crossing, and thus some solutions were non-optimal. At the same time it is possible to observe an obstacle crossing where the robot lowers its leg at exactly the same position. Thus one could end up with overlapping hyperrectangles for different virtual evaluation functions. However one of them could clearly be removed because of the non-optimal nature of the original example.

The underlying assumption is that it is possible to identify disjoint hyperrectangles for each virtual evaluation function. The results in Section 9.4 will confirm this assumption. Otherwise the robot would necessarily have to take different decisions at different times despite being in the same state. This would imply that the state description of the robot lacks some necessary information about the robot, which contradicts one of the assumptions of this thesis, namely that all relevant information is directly accessible.

### 9.2.5 Comments

Let the inductive learning algorithm described in this section be called *boundary adjuster*. It is based on a number of rather strong assumptions: first is the assumption that the use of a virtual evaluation function can be described by a linear decision function in one parameter. Therefore the boundary adjuster can not learn disjunctive concepts. The second assumption is that the hyperrectangles for each virtual evaluation function do not overlap. The boundary adjuster is even weaker than the perceptron learning algorithm, because it requires the decision functions to coincide with a dimension of the state space.

However, the reasons for using the boundary adjuster are its ease of use and the fact that it is sufficient in order to enable the biped robot to learn to cross obstacles. It also gives its results in a very simple and understandable form (upper and lower boundaries per parameter rather than multi-parameter de-

cision functions). This makes debugging and explanation easier.

Various algorithms exist for such applications where the boundary adjuster would fail. A. Moore discusses nearest neighbour techniques [Moo90] and shows its application to robotics. Other popular techniques are decision tree techniques like ID3 [Qui83, Qui86], as well as neural nets [Was89]. For a more detailed discussion see Section 2.4.

The main difference between the boundary adjuster and some classification techniques is the fact that the boundary adjuster selects its own set of training data. Techniques like ID3 or neural nets work on a given set of training data. (For an excellent treatment of the choice of training data for nearest neighbour learning see again A. Moore's thesis [Moo90].) The boundary adjuster also operates in a state space which is full of redundant parameters. In contrast to this most classification systems assume "...that the features are relevant to the prediction or category decision" ([Sal88], page 56).

Section 9.4 will show that a simple algorithm like the boundary adjuster is sufficient to generate a reasonably robust obstacle crossing behaviour.

## 9.3 Adjusting Step Length for Obstacle Crossings

Obstacle crossing requires that the robot stands with the correct distance to the obstacle before initiating the obstacle crossing gait. If the robot is too close to the obstacle then it will not be able to lift its foot quick enough over the obstacle, and as a result it will crash into the obstacle. Similar problems occur if the robot is initiating the obstacle crossing behaviour too far away from the obstacle.

Experiments have shown that it is quite important for the biped to initiate the obstacle crossing with the right distance to the obstacle. The virtual evaluation function tells the robot what to do once the robot is in the right position. However it does not take care of bringing the robot into the right position in front of the obstacle. About 20% of all attempted obstacle crossings failed because the robot was standing the wrong distance away from the obstacle. In such cases the robot needed to be about half a step length further away (or closer) to the obstacle, otherwise starting one step earlier (or later) would have solved the problem. Thus step length adjustment becomes a necessity.

Robots which navigate in rough terrain must make use of a limited number of available footholds, and consequently the robot must plan ahead to ensure that it places its feet in the right position. Very recently Hodgkins and Raibert

[HR91] discuss control algorithms to manage step length adjustments for a dynamic biped robot. Pal and Jayarajan [PJ91] use the $A^*$ algorithm to search for foot placements for a planar quadruped robot.

The approach taken in this thesis is a combination of backtracking and step length adjustment. We recall from Chapter 6 that the biped searches for new motor commands until it reaches a local minimum with respect to its goal position. It then finishes the current step, executes a support exchange command and starts again. If the biped has to find a step with a different step length then it simply ignores the first local minimum which it encounters. This is equivalent to not executing the support exchange command between two steps. Thus the new step becomes slightly longer and should provide the robot with the necessary displacement. Whenever the robot fails to cross an obstacle it backtracks to the last position in which it executed a normal gait for planar surfaces. This time it will execute a step with different step length as just described.

In the experiments with the virtual evaluation function described in Section 9.4 the biped makes use of this step length adjustments. However the biped was not allowed to backtrack more than 5 times during an entire obstacle crossing experiment. On average the biped needed to backtrack twice per obstacle.

Figure 9.1 shows the amount of backtracking used in a more complex test series. Here the biped was given the task of crossing 175 randomly created test surfaces. The distance between each obstacle was 90mm. 79 test surfaces could be crossed without backtracking, and another 29 could be crossed with only one backtracking step. Backtracking occurred twice on 9 test surfaces, and three times on 5 test-surfaces. 29 test surfaces could not be crossed with less than 20 backtracking steps.

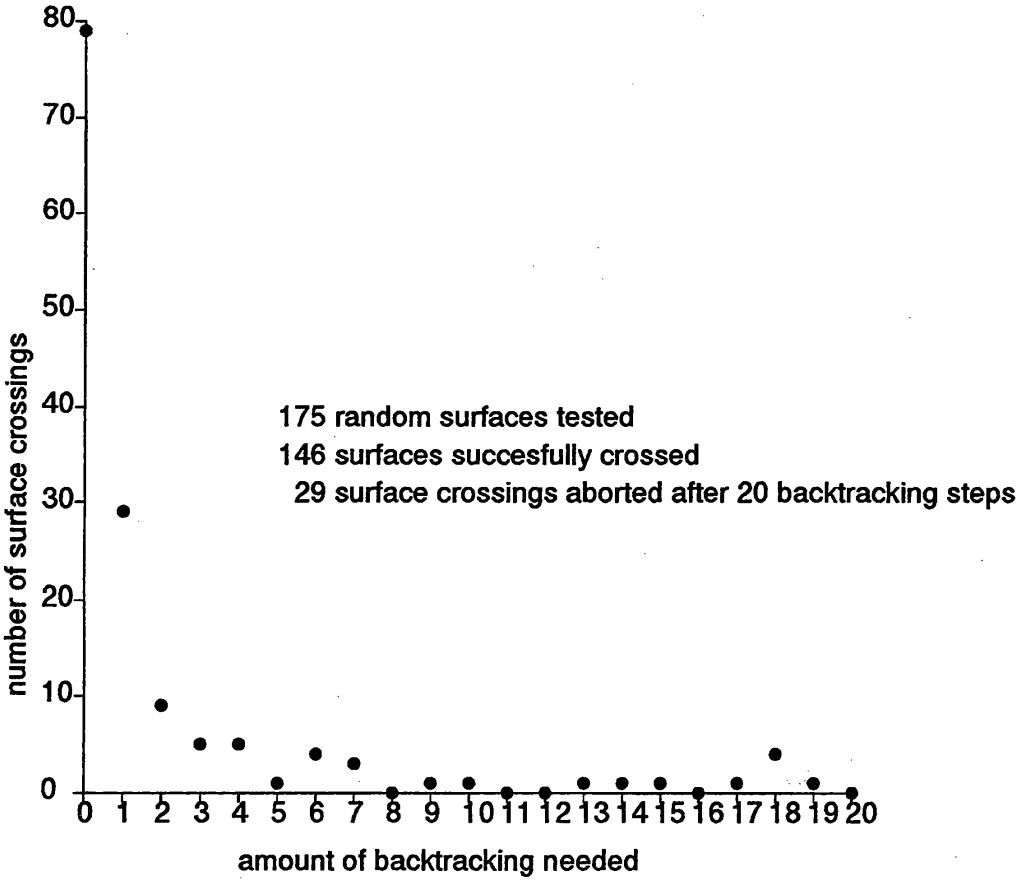



Figure 9.1: Number of backtracking needed to cross a surface with 4 randomly chosen obstacles.

## 9.4 Learning Biped Obstacle Crossing

It is finally time to discuss the application of the virtual evaluation function and the boundary adjuster to dynamic biped walking. This section discusses how the biped crosses various types of obstacles. Each type of obstacle is inspected individually, and the performance of the robot is discussed. The test of the obstacle crossing capability follows the same pattern each time:

1. An obstacle crossing using search and the qualitative equivalence heuristic are inspected, and the virtual evaluation functions are analysed.
2. For each virtual evaluation function the corresponding hyperrectangles of the state space are constructed.
3. The ability of the robot to cross this type of obstacle using the virtual evaluation functions is tested. The robot encounters the same type of obstacle at various positions and the performance of the robot is measured.

These data provide insight into various aspects of this chapter. They show how many different virtual evaluation functions can be detected. The experiments also show that a small set of obstacle crossing examples is sufficient to generate the original hyperrectangles identified with each virtual evaluation function. It is also demonstrated that a small number of experiments suffices in order to adjust the hyperrectangles. Finally the results demonstrate the robot's ability to cope with different obstacles.

A short note on the documentation of the movements of the biped: in the figures throughout most this chapter the surface and the position of the free foot of the biped are plotted. Occasionally the foot is plotted below the surface. This is due to the fact that the coarseness of the motor commands makes it impossible to place the foot exactly on the surface. Therefore moves are allowed which result in the robot placing its foot slightly (1mm or 2mm) under the surface. Therefore occasionally the foot position is plotted marginally below the surface line. Implementing the algorithms on a real robot would require to initiate a support exchange command before the robot's free leg hits the surface. The support exchange command would then ensure that the free leg of the robot drops the remaining one or two millimeters, and thus the robot ends up with its foot *on* the surface.

Every figure shows only one attempt of the biped to cross the obstacle. The high number of dots is due to the fact that the position of the free foot is plotted every 0.04 seconds.

### 9.4.1 Crossing Steps

#### Virtual Evaluation Functions Derived from the Qualitative Equivalence Heuristic

The task of crossing a step differs depending on whether it is a step up or a step down. In the first case the biped has to ensure that it lifts both legs high enough, in the second case it has to lower the legs quickly enough. Figures 9.2 to 9.5 show the original biped obstacle crossing.

Figure 9.2 shows how the qualitative equivalence heuristic was used in order to train the robot to cross a step upstairs. Figure 9.3 shows the use of the same heuristic in order to discover steps downstairs. The qualitative equivalence heuristic picked the first successful obstacle crossing, it did not search further for obstacle crossings which resulted in a posture close to the original posture.

#### Identifying the Used Virtual Evaluation Functions

In a next step the obstacle crossings were analysed in terms of the virtual evaluation function which could be identified with each state encountered during each successful obstacle crossing. Figure 9.4 and Figure 9.5 show the crossing of a step with a height of 10mm and -10mm respectively.

Taking a closer look at the virtual evaluation function annotated to each state reveals that the biped used a different behaviour only for one of the two steps which constitute the obstacle crossing. In Figure 9.4 and Figure 9.5 the virtual evaluation function notation refers to the parameters $\eta, \zeta, \theta, \psi$, and $\phi$. We notice that only two virtual evaluation functions have been used, namely (000+0), which corresponds to an increased value of $\psi$, which in turn corresponds to a lifting of the hip around the roll axis. Similarly (000-0) corresponds to a lowering of the leg around the roll axis.

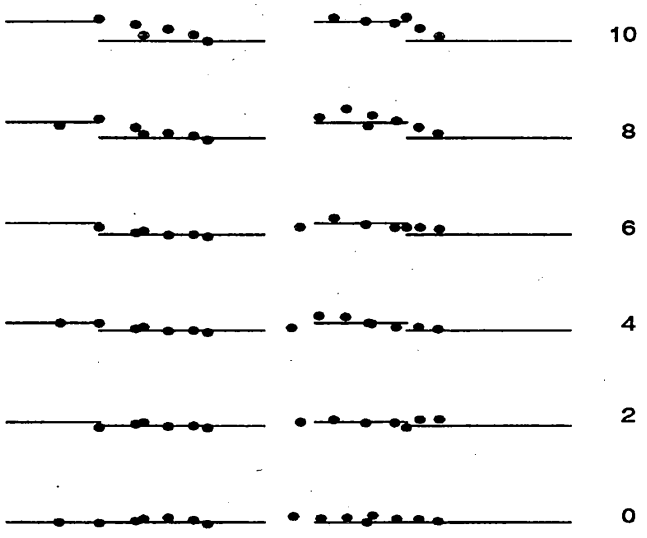



Figure 9.2: Using the qualitative equivalence heuristic to cross a step upstairs. The biped walks from the right to the left.

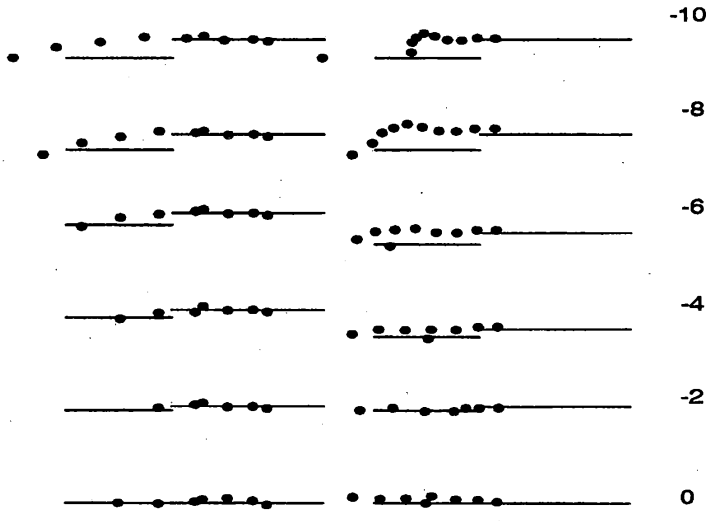
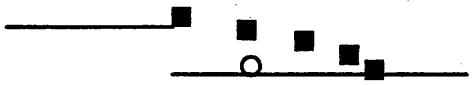



Figure 9.3: Using the qualitative equivalence heuristic to cross a step downstairs.



The biped lifts the first leg up the step.
In every position the biped uses 000+0 as virtual evaluation function,
which is equivalent to lifting the hip around the roll axis.

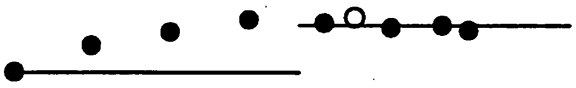
The biped lifts the second leg up the step by using normal search.
Thus in every position the biped uses 00000 as virtual evaluation function.

○ support foot position

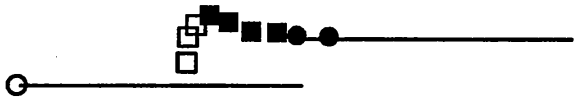
● free foot position using normal evaluation function 00000

■ free foot position using virtual evaluation function 000+0

Figure 9.4: Identifying the virtual evaluation functions of a step upstairs.



The biped lowers the first leg down the step by using normal search.



The biped lowers the second leg. This is done in three pahses.
First it uses the normal evaluation function to get the foot to the edge.
Next it lifts the leg slightly, using 000+0 as virtual evaluation function,
before finally lowering the leg, using 000-0 as virtual evaluation function.

○ support foot position

● free foot position

□ free foot position using virtual evaluation function 000-0

■ free foot position using virtual evaluation function 000+0

Figure 9.5: Identifying the virtual evaluation functions of a step downstairs.

### Hyperrectangles for Virtual Evaluation Functions

For each of the two virtual evaluation functions the corresponding hyperrectangle of the state space had to be identified. Once the robot enters this area of the state space it has to use the corresponding virtual evaluation function to generate its next set of motor commands. The original hyperrectangles defining the obstacle space were discovered using the annotation of all states encountered during the application of the qualitative equivalence heuristic.

For the virtual evaluation function (000+0) the following boundaries were originally discovered:

$$42 \geq \text{distance-to-step} \geq -34$$

$$7.3 \geq \text{foot-height} \geq -1.0$$

$$0.04 \geq \eta \geq -0.11$$

$$0.12 \geq \zeta \geq -0.04$$

$$0.01 \geq \theta \geq -0.11$$

$$0.16 \geq \psi \geq 0.04$$

$$-0.05 \geq \phi \geq -0.12$$

These boundaries were then adjusted. Three new test obstacles of the same type (step up or step down (height 10mm)) had to be crossed using the virtual evaluation functions, and the smallest[2] hyperrectangle was identified, in which the virtual evaluation function could still lead to a successful crossing of all three test obstacles. This resulted in the following hyperplanes:

$$25 \geq \text{distance-to-step} \geq -6$$

$$-0.04 \geq \eta \geq -0.11$$

[2]smallest: smallest number of boundary hyperplanes, and smallest distance between two hyperplanes of the same dimension.

Running the same program for the virtual evaluation function (000-0) resulted originally in

$$9 \geq \text{distance-to-step} \geq -34$$

$$1.6 \geq \text{foot-height} \geq -1.6$$

$$0.02 \geq \eta \geq -0.08$$

$$0.07 \geq \zeta \geq 0.01$$

$$0.01 \geq \theta \geq -0.11$$

$$0.1 \geq \psi \geq -0.03$$

$$-0.11 \geq \phi \geq -0.12$$

which was reduced to

$$-6 \geq \text{distance-to-step} \geq -34,$$

thus it was ensured that the two hyperrectangles for (000-0) and (000+0) did not overlap.

### Crossing new Steps

Finally the new obstacle spaces and their virtual evaluation functions were tested in a set of new random obstacles. Figure 9.6 shows how the biped successfully crosses 5 different steps upstairs, while Figure 9.7 displays how the biped succeeds in crossing 5 different steps downstairs.

The data show how the robot progresses from right to left. The dots indicate the position of the free foot. The obstacle positions in the different trials differ between 5mm and 50mm. The small differences between the obstacle positions were chosen in order to avoid having the biped always encounter the obstacle at a "convenient" point during its stride. Each figure shows the foot positions during a single attempt to cross the obstacle. The large number of dots is due to the 0.04 second frequency with which the foot positions were plotted. The same type of visualisation (one walk and plotting of the foot position per diagram) will be used in all similar diagrams in this and the next chapter. Thus all diagrams where the dots indicating the foot position reach the left end of the surface demonstrate that the obstacles were crossed on the first attempt. If the robot fails then the foot position will be plotted up to the last "safe" position.

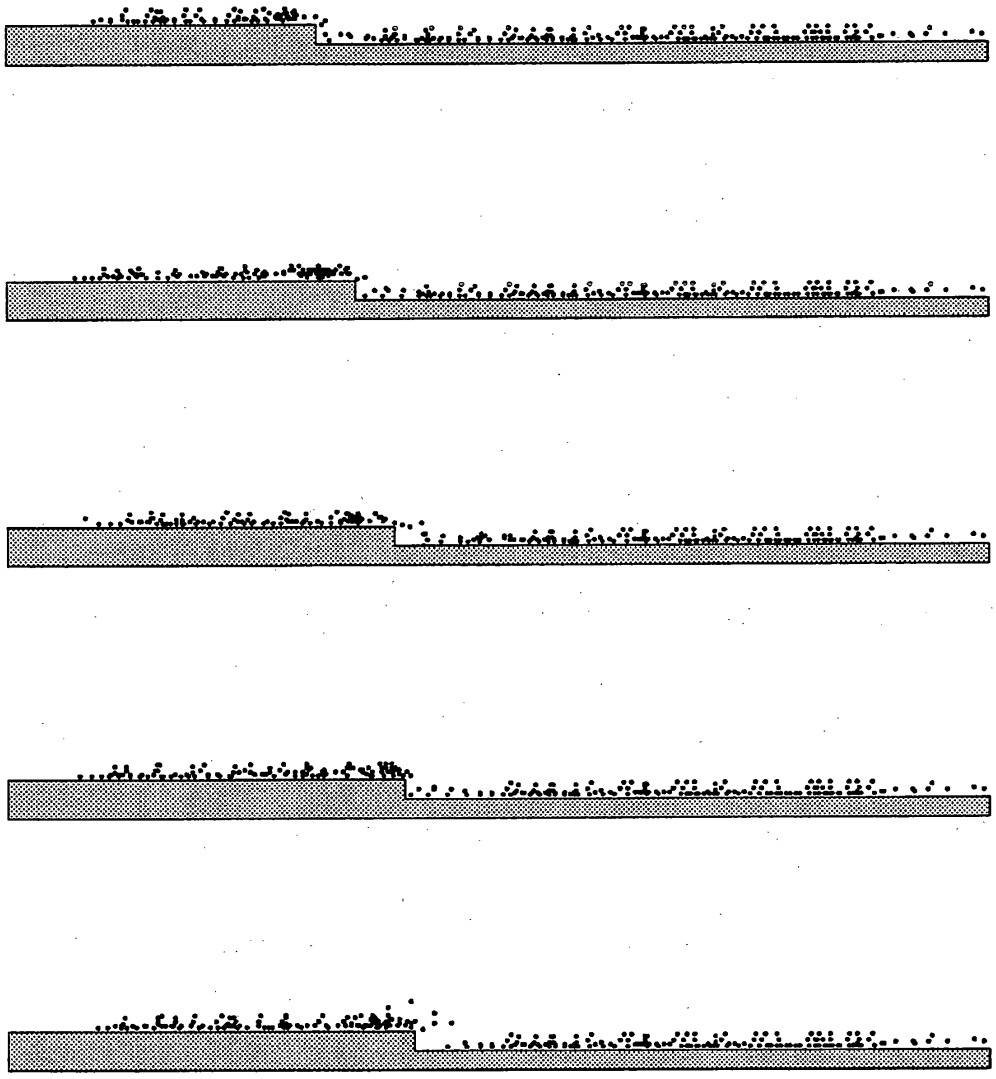


Figure 9.6: Using the virtual evaluation functions to cross a step upstairs. Each diagram represents one walk, with the position of the free foot plotted every 0.04 seconds. The biped walks from the right to the left. The location of the steps was chosen in such a way that the biped would encounter the obstacle during different moments of its stride.

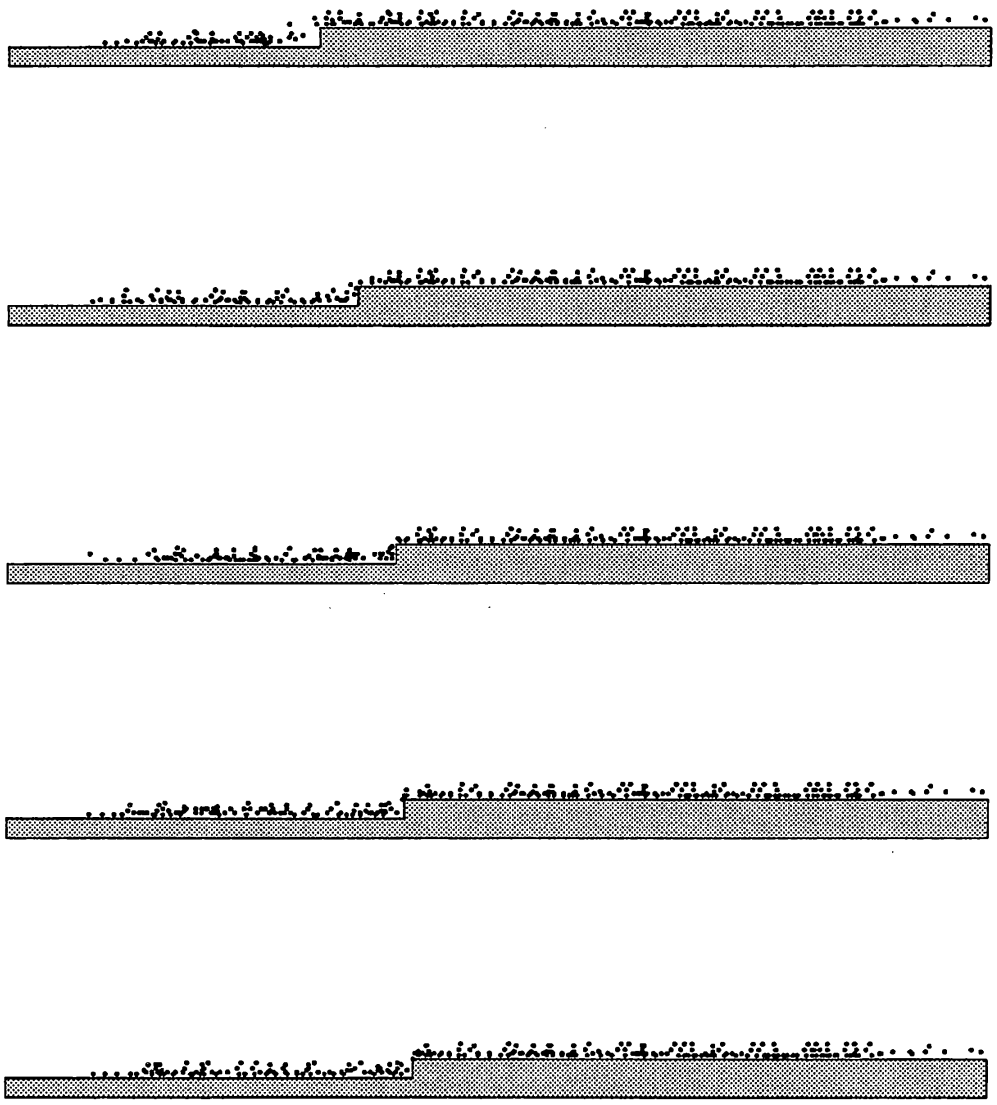


Figure 9.7: Using the virtual evaluation functions to cross a step downstairs.

### 9.4.2 Crossing Fences

#### Virtual Evaluation Functions Derived from the Qualitative Equivalence Heuristic

The same experiments were repeated in order to analyse the fence crossing capabilities of the robot. First the biped was trained to cross a fence by presenting it with a number of fences of increasing height. Using the qualitative equivalence heuristics and memorising all successful gaits the robot succeeds in discovering gaits which enable it to cross a 10mm high fence (Figure 9.8). A closer look at the successful crossing of the 10mm fence reveals that the same virtual evaluation functions were used as in the step crossings (Figure 9.9).

#### Hyperrectangles for Virtual Evaluation Functions and Crossing new Fences

For the crossing of walls the robot could already rely on some of the previous results. The virtual evaluation functions used were the same as the ones needed to cross a step. Since the hyperrectangles for these virtual evaluation functions were already identified, it had to be tested whether they were applicable to the crossing of a fence. Generalising the variables *distance-to-step* and *distance-to-wall* to *distance-to-obstacle* allowed us to apply the hyperrectangles developed for the crossing of steps. All that had to be tested was whether the respective boundaries of the hyperrectangles needed adjustment. Thus the robot was present with 5 randomly positioned fences of 10mm height. The robot used the hyperrectangles for virtual evaluation functions developed in Section 9.4.1 for the crossing of steps. As the results in Figure 9.10 show it proved to be sufficient to use these hyperrectangles and virtual evaluation functions.

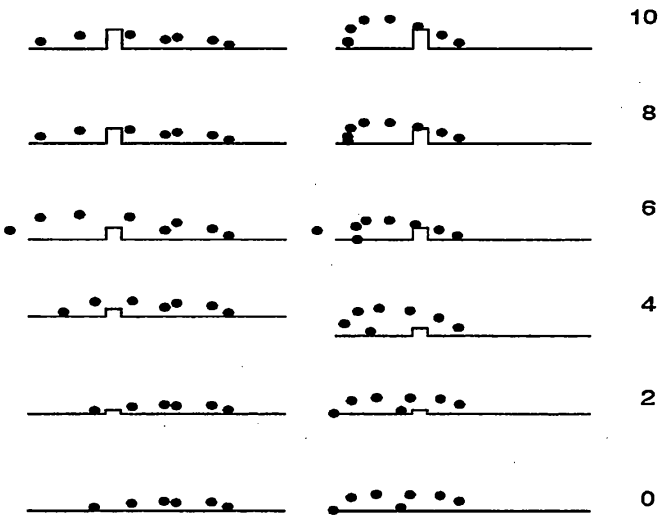
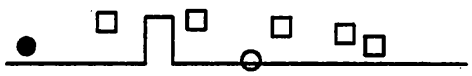
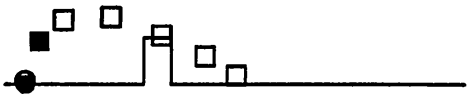



Figure 9.8: Using the qualitative equivalence heuristic to cross a fence. The height of the fence is written to the right of each plot.



The biped lifts the first leg over the wall.
In every position the biped uses 000+0 as virtual evaluation function, which is equivalent to lifting the hip around the roll axis.



The biped lifts the second leg over wall.
Again, in every position execpt for the last one the biped uses 000+0 as virtual evaluation function.

- ○ support foot position
- ● free foot position using the normal evaluation function
- □ free foot position using virtual evaluation function 000+0
- ■ free foot position using virtual evaluation function 000-0

Figure 9.9: Identifying the virtual evaluation functions of a fence crossing.

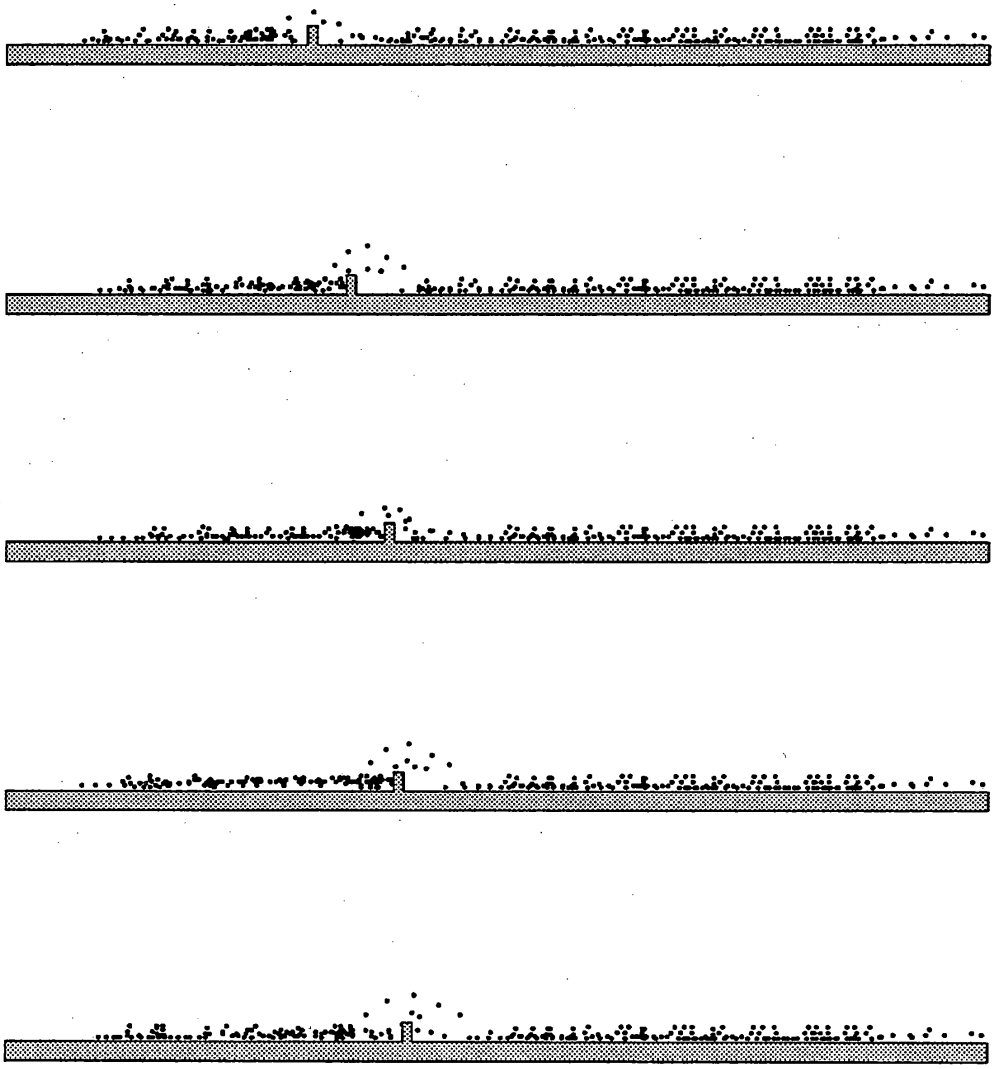


Figure 9.10: Crossing a set of random fences. As in the previous examples the the fences have been placed relatively close to each other so that the biped encounters them during different phases of its stride.

### 9.4.3 Crossing Slopes

The crossing of slopes proved to be no problem for the biped. The robot was able to cross slopes of up to about 120% by using the normal search procedure for the inner state space. Figure 9.11 shows the biped crossing slopes of different inclination. Once the slopes became steeper than 120% they could be treated like steps and crossed accordingly, provided they were not too high.

### 9.4.4 Crossing Trenches

Crossing trenches proved to be the fairly easy as well. Here the main problem was the fact that the normal gait execution could end in a state where the free foot is just over the trench and hence no support exchange was feasible. In this case the robot had to backtrack to a previous support-exchange state and create a "short" step such that later steps would lead over the trench (see Section 9.3). Figure 9.12 shows the biped crossing small trenches of 5mm width.

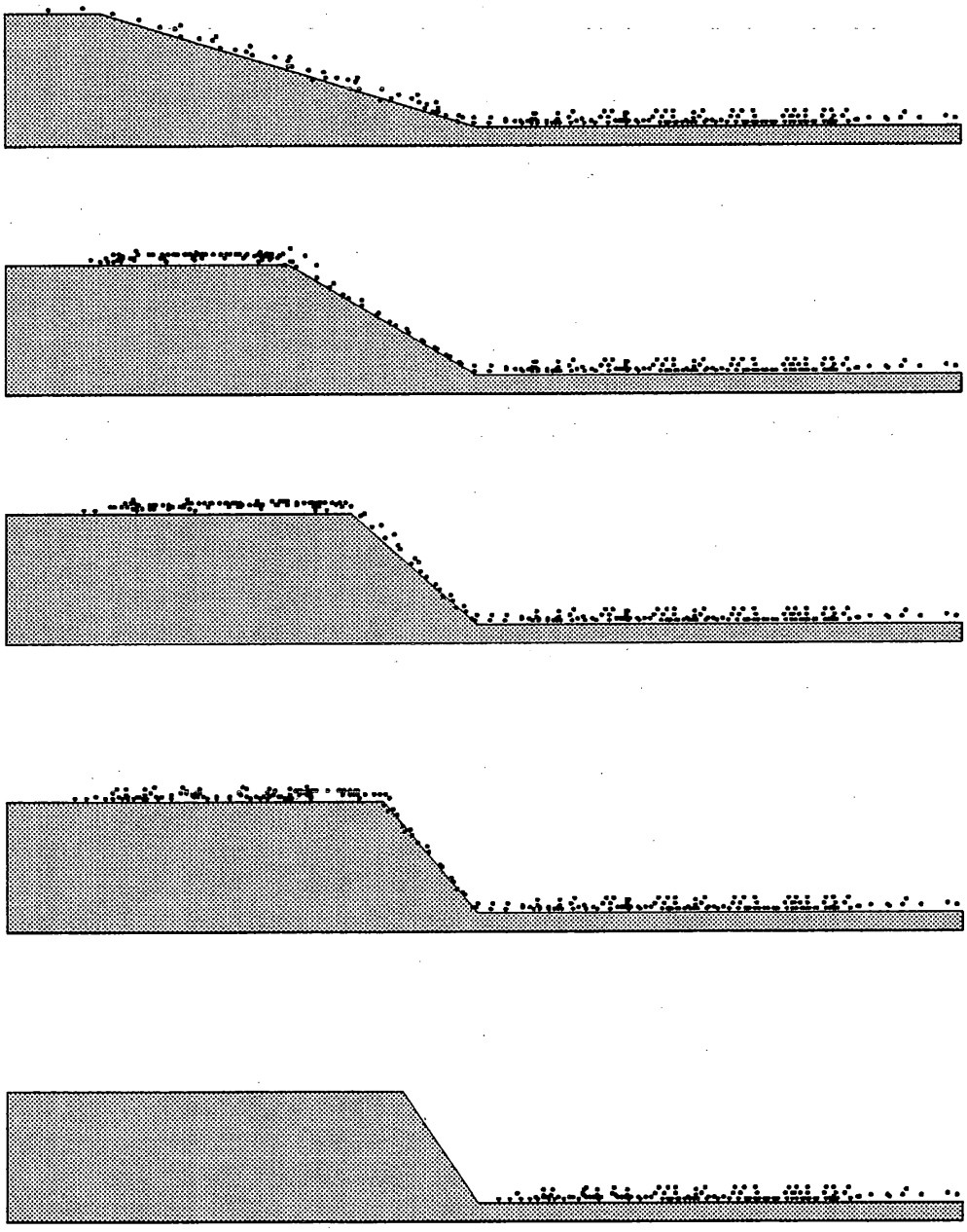


Figure 9.11: Crossing a set of slopes with increasing inclination.

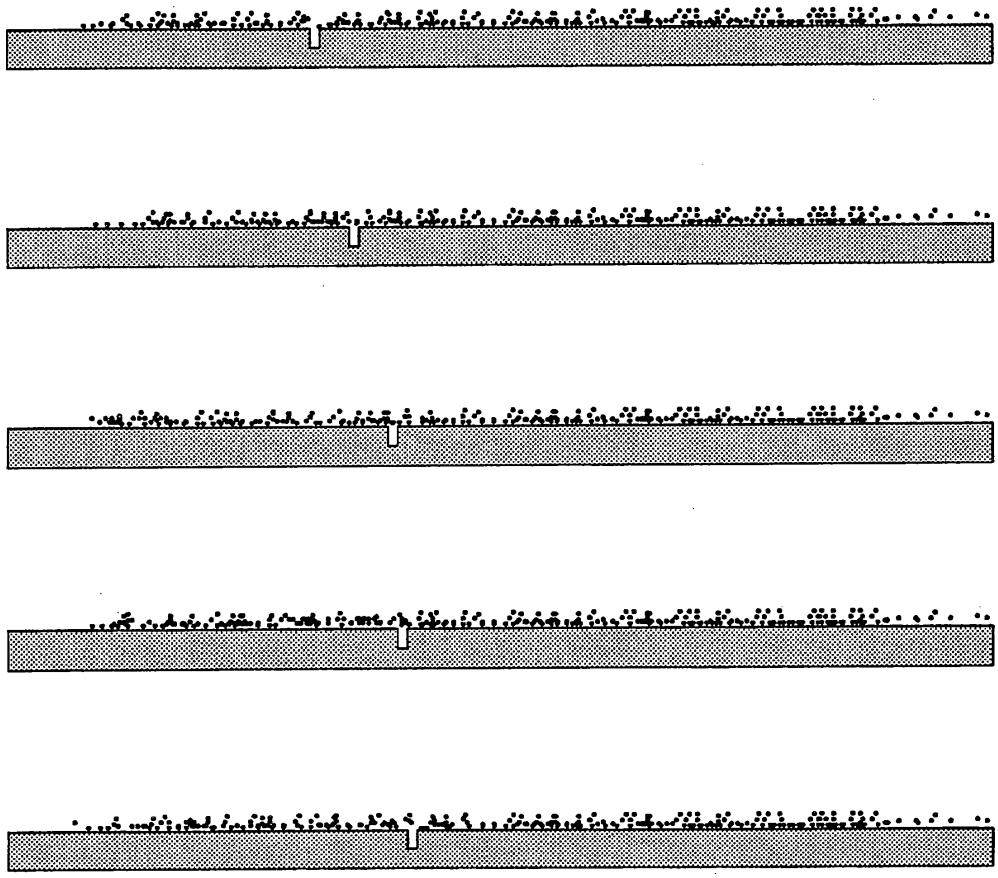


Figure 9.12: Crossing a set of trenches.

## 9.5 Discussion

The qualitative equivalence heuristics proved to be relatively brittle with respect to the posture of the robot at the beginning of the gait. They were also brittle with respect to the relative distance between the obstacle and the robot. Thus a small variation in the search technique was introduced which allowed the robot to create steps with a step-length different from the one of the standard gait. However the execution of such a step leads the robot into a position which is even more different from the start position of the standard gait. Thus the robot is able to vary its position relative to the obstacle, but this isn't always enough to successfully apply the qualitative equivalence heuristics for the obstacle crossing.

The concept of a virtual evaluation function overcomes these limitations. Based on a small number of successful obstacle crossings the robot developed a generic obstacle crossing strategy. This chapter demonstrated that based on a small series of successful obstacle crossings generic obstacle crossing behaviour can be derived.

The robot used no other data than the obstacle crossings displayed in Figure 9.2 and Figure 9.3. Based on these 10 obstacle crossing exemplars the robot identified all necessary virtual evaluation functions. From these obstacle crossings the robot also derived a first description of the hyperrectangles of the obstacle space which are identified with each virtual evaluation function. Using 3 new test crossings the robot was able to adjust the boundaries of these hyperrectangles to such a degree that a generic obstacle crossing capability was achieved.

Up to now the robot has been presented with one obstacle, and thus after the obstacle crossing it had plenty of time to adjust itself with respect to its normal gait. Introducing a series of obstacles will result in a test of the biped's ability to recover quickly enough from the disturbances to its gait execution created by individual obstacles. It will also give a more impressive documentation of the biped's ability to cope with difficult surfaces. The obstacle crossing capabilities of the robot with respect to random surfaces will be tested in more detail in Chapter 10.

# Chapter 10

# Crossing Random Obstacles

## 10.1 Random Obstacles on Even Terrain

It is finally time to demonstrate the generic obstacle crossing capabilities of the biped. Using the virtual evaluation functions developed in Chapter 9, the biped will be presented with a number of randomly generated test surfaces. Each of these surfaces contains up to 4 random obstacles; any one of these obstacles is either a 10mm fence, a step which goes 10mm up or down, or a slope up or down. The space between the obstacles is a horizontal plane. The distance between obstacles is 90mm; this distance was chosen because it proved to be sufficiently large for the robot to resume its standard gait execution. Shorter distances between obstacles can prevent the robot from regaining its normal posture. In this case the deviation from the normal start-position increases with each obstacle until the robot reaches a position where the torques of the robot are not powerful enough to bring the robot back into an upright position.

Typically the robot spreads its legs around the roll axis while it crosses an obstacle. If given enough space the robot needs up to about ten steps in order to adjust the legs around the roll axis; however if the robot encounters the next obstacle before the legs are adjusted, it spreads its legs even further and is unable to adjust them any longer.

The obstacle crossings demonstrated in this section are chosen from 175 randomly generated surfaces. Out of these 175 surfaces only 29 could not be crossed. This is equivalent to a failure rate of 16%. By moving the obstacles further away from each other all obstacle combinations could eventually be crossed.

The robot's movements across the obstacle course is represented as in previous

chapters: the robot walks from the right to the left, and every 0.04 seconds the position of the free foot is plotted as a dot. All regions with an above average concentration of black dots indicate that the robot is making less progress forwards. This is due to the fact that the robot spends time to get itself back into a posture that is close to the start position of its standard gait. Usually such concentrations of dots can be observed (1) at the end of an upwards slope, (2) behind a fence, or (3) behind a step upstairs. Especially after the crossing of a step or a wall the robot finds itself in a posture where it actually leans in the wrong direction. Thus it has to use the limited space between its current position and the obstacle in order to make a turn and regain a posture which leans into the other direction.

In order to give a better understanding of what exactly is happening in these "above average concentrations of black dots" we will look at such an obstacle crossing in more detail: Figures 10.1 to 10.3 give a detailed example of the robot's behaviour during the crossing of a wall. In Figure 10.1 the robot is seen as it approaches the obstacle (position 1). It lifts the first leg over the wall (position 2) and then corrects its own position around the pitch axis and the wall (positions 3 and 4). Figure 10.2 then shows how the robot lifts the second leg over the obstacle (position 5). In positions 6 and 7 the robot actually moves backwards towards the wall in order to regain a posture which enables it to continue to walk to the left; in position 8 it has reached this posture. Finally Figure 10.3 documents how the robot now attempts to adjust its posture further by correcting errors around the roll axis. In positions 9 to 11 the robot pulls both legs further inwards around the roll axis until in position 12 it reaches a posture similar to that before the obstacle crossing (which may be compared with position 1 in Figure 10.1). These last 3 steps show how the controller enables the robot to resume its normal gait trajectory.

Figure 10.4 to 10.7 present some random examples of the successfully crossed obstacle combinations. We can see that the robot could cope with combinations of steps, fences and slopes. Trenches were not included because to do so would make the obstacle combinations less difficult. The scale used in the figures is approximately 1 : 4, which is the same scale as used in most figures shown in previous chapters.

Figure 10.8 shows obstacle combinations which could not be crossed if the distance between obstacles was 90mm or less. Figure 10.9 documents how these obstacle combinations could be successfully crossed with a wider distance between the individual obstacles. In Figure 10.9 the distance between obstacles is widened to 120mm.

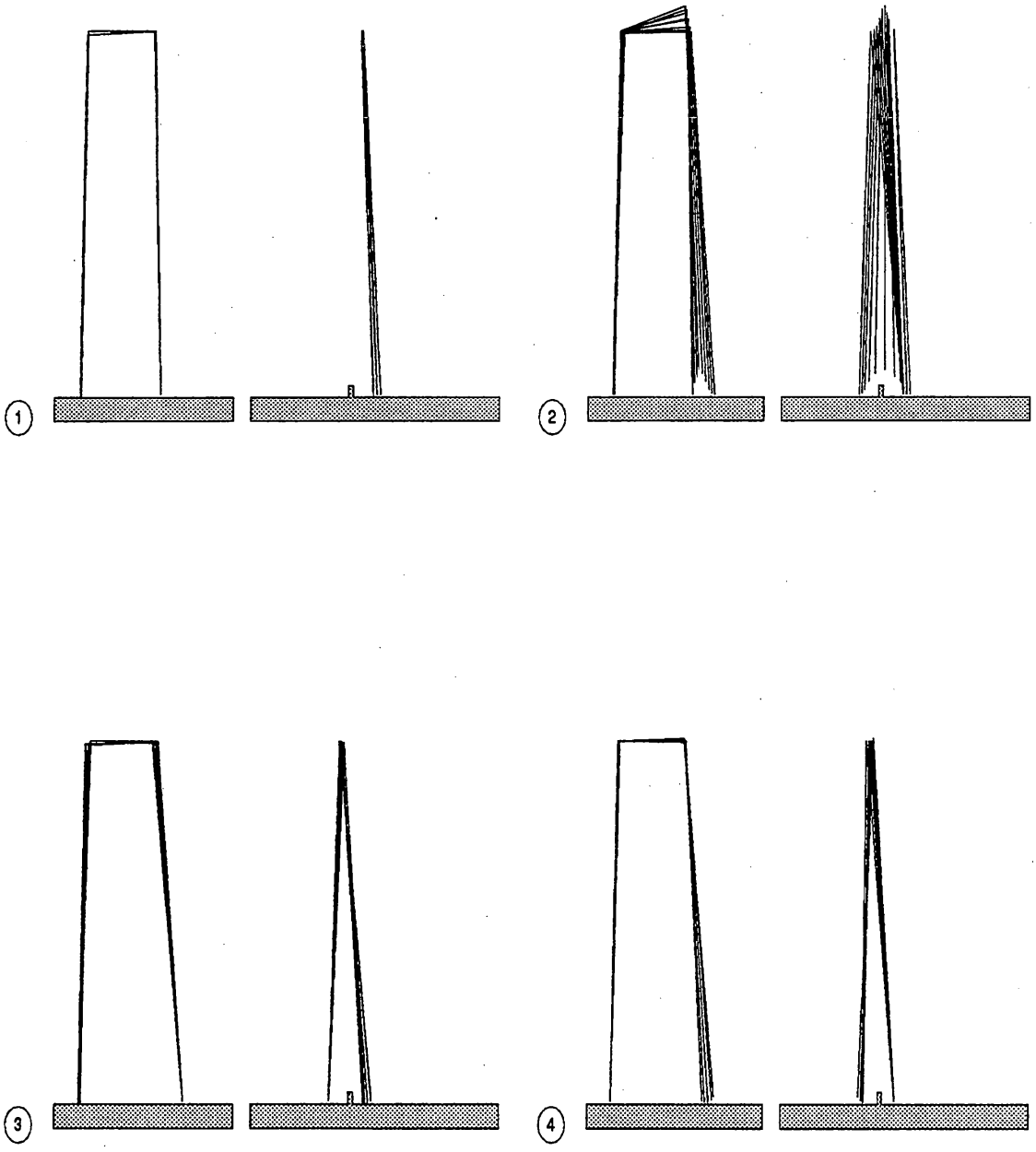



Figure 10.1: The robot lifts the first leg over a wall and then corrects its posture.

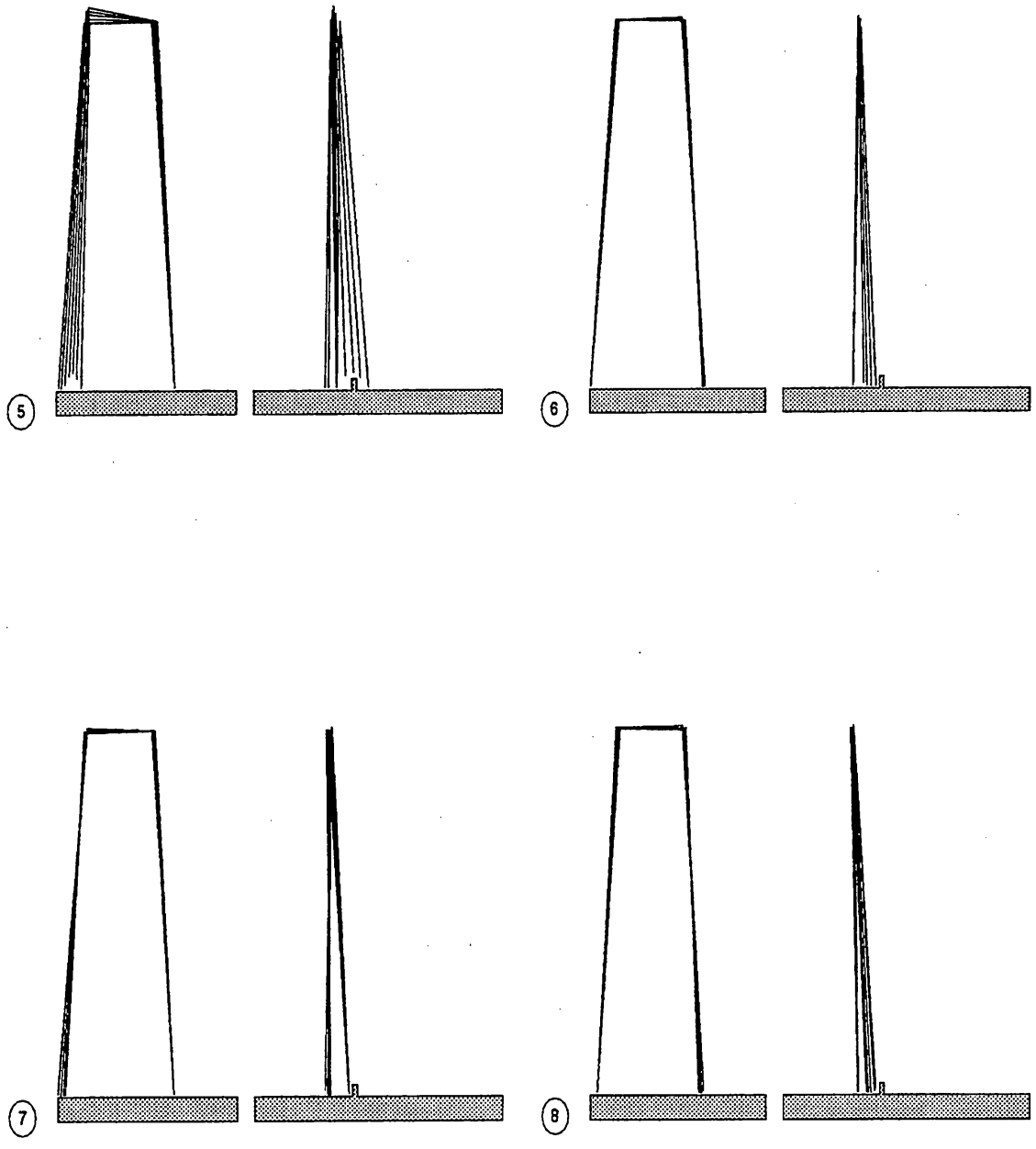



Figure 10.2: The robot lifts the second leg over the wall and resumes a posture leaning towards the left.

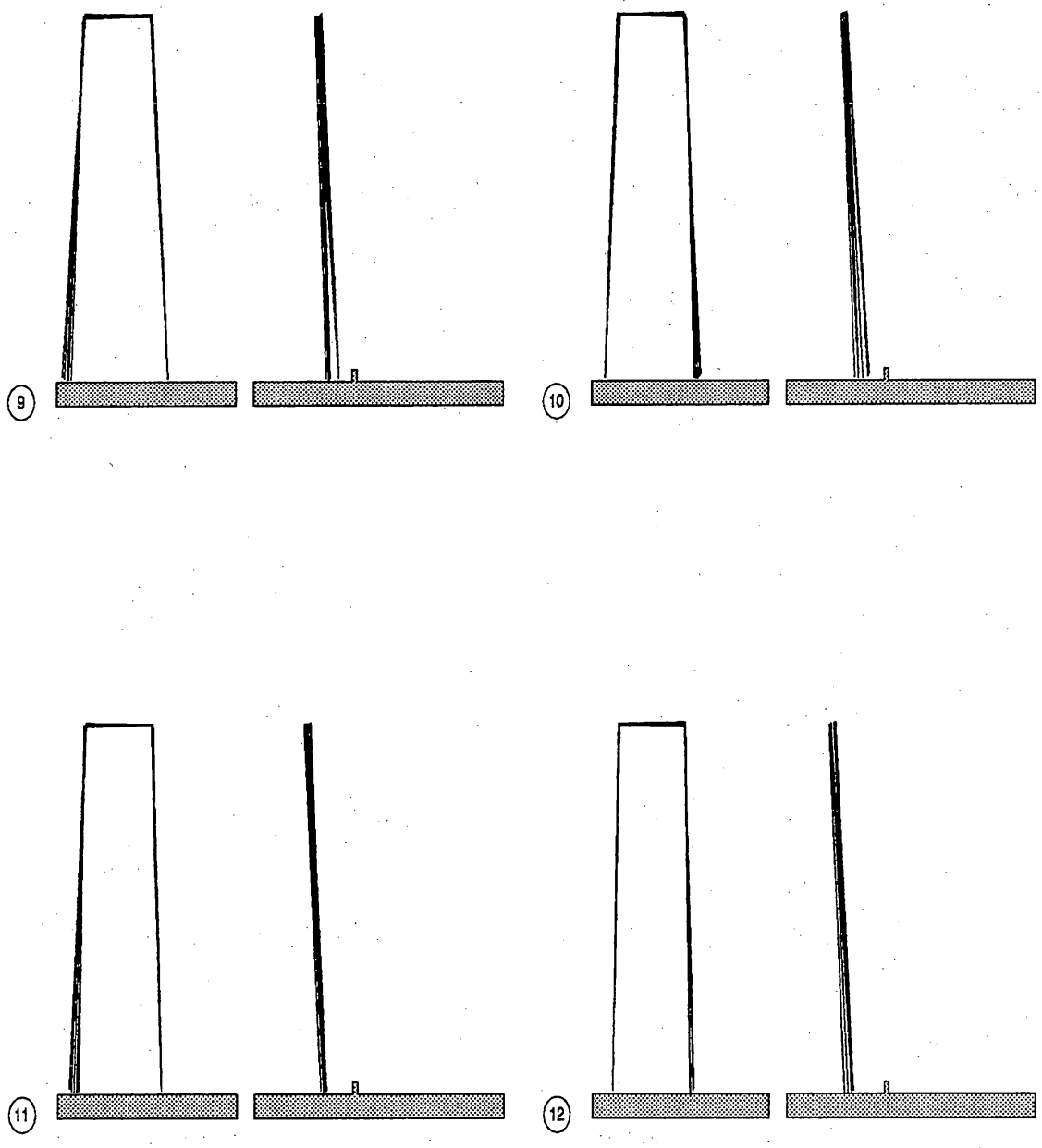



Figure 10.3: The robot finished the obstacle crossing and needs another 3 steps to resume its standard gait.

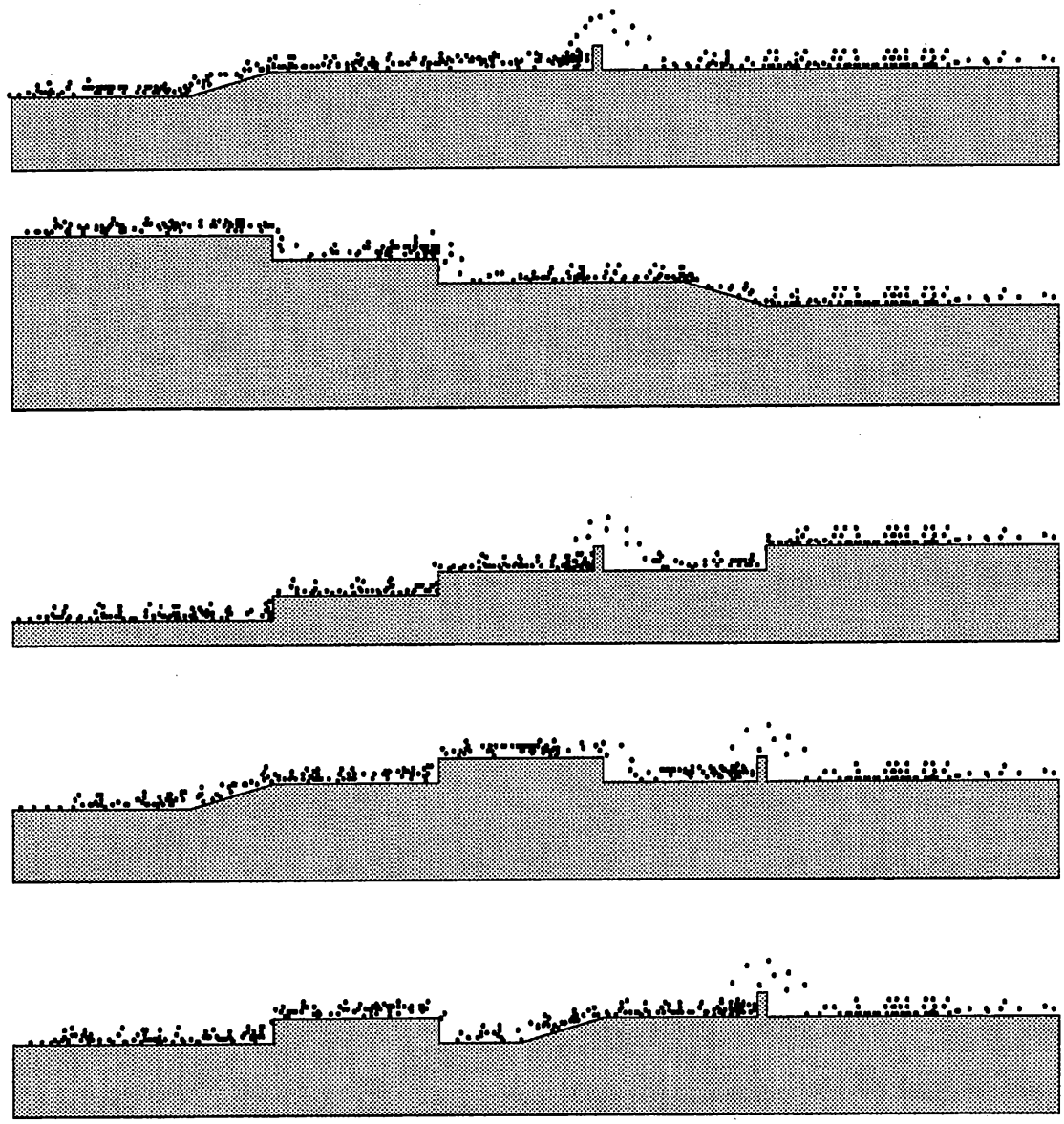


Figure 10.4: The robot crosses a sequence of random obstacles. The robot walks from the right to the left. Small black dots indicate the position of the free foot.

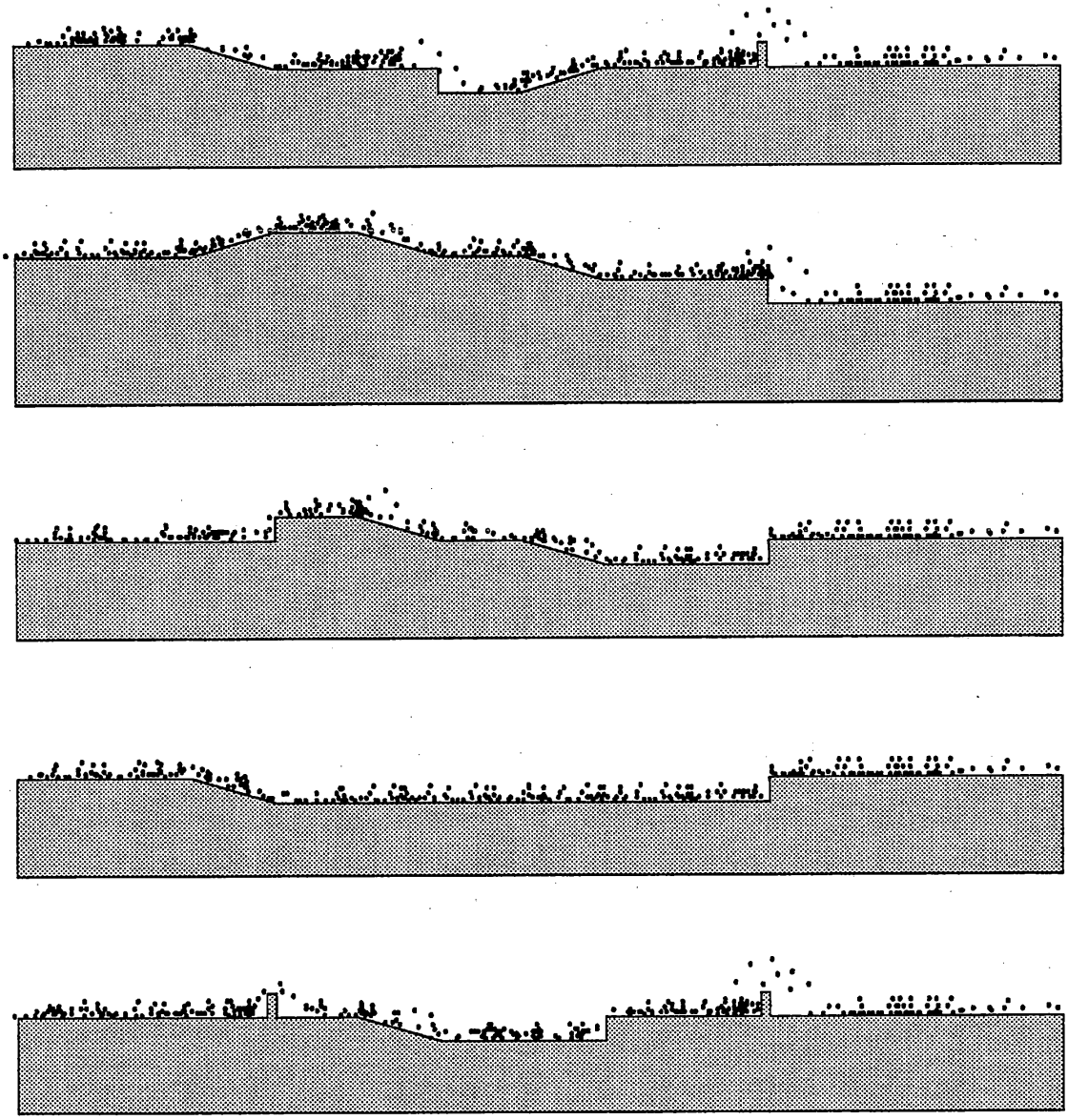


Figure 10.5: The robot crosses another sequence of random obstacles.

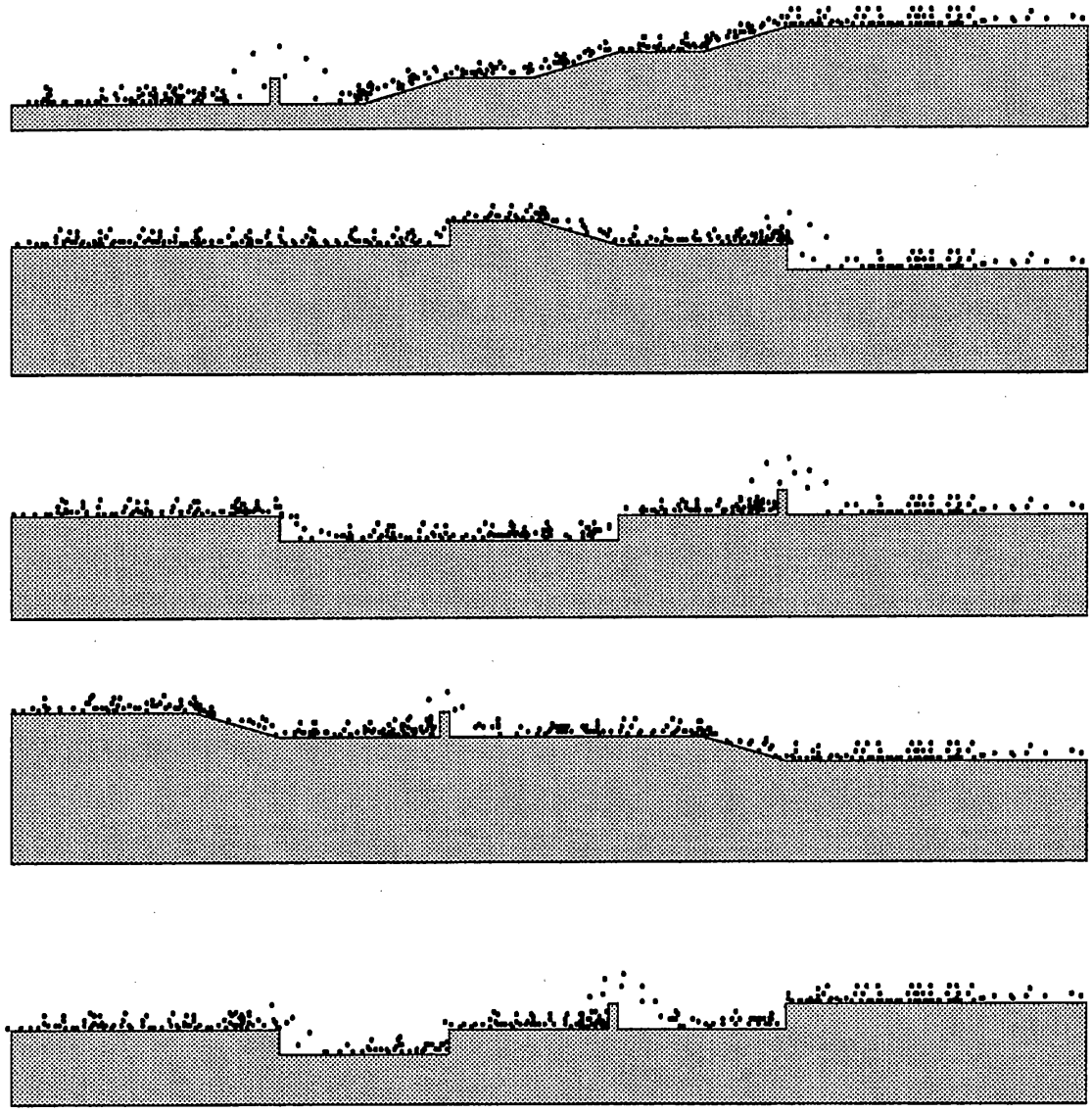


Figure 10.6: The robot crosses a third sequence of random obstacles.

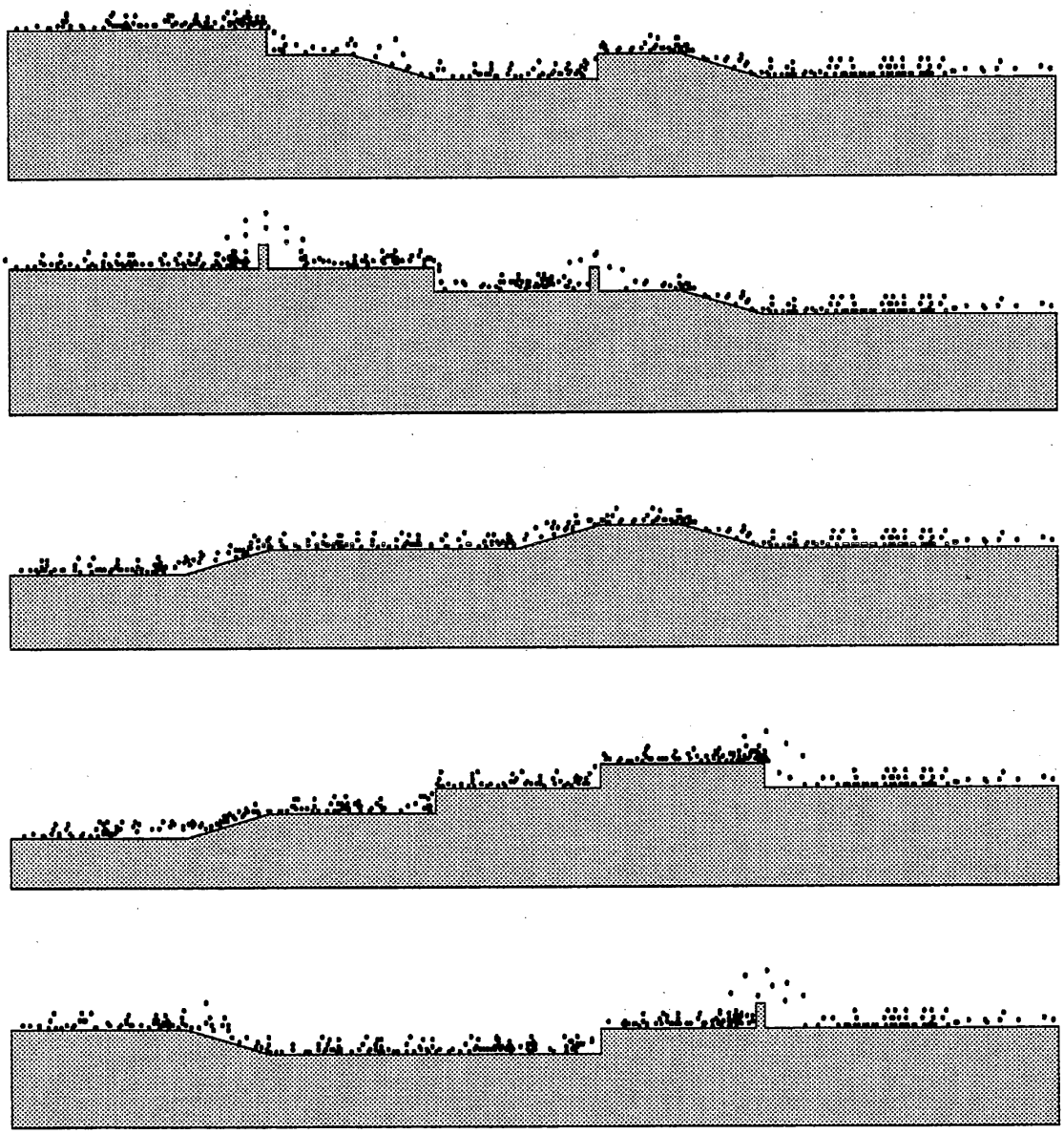


Figure 10.7: The robot crosses a fourth sequence of random obstacles.

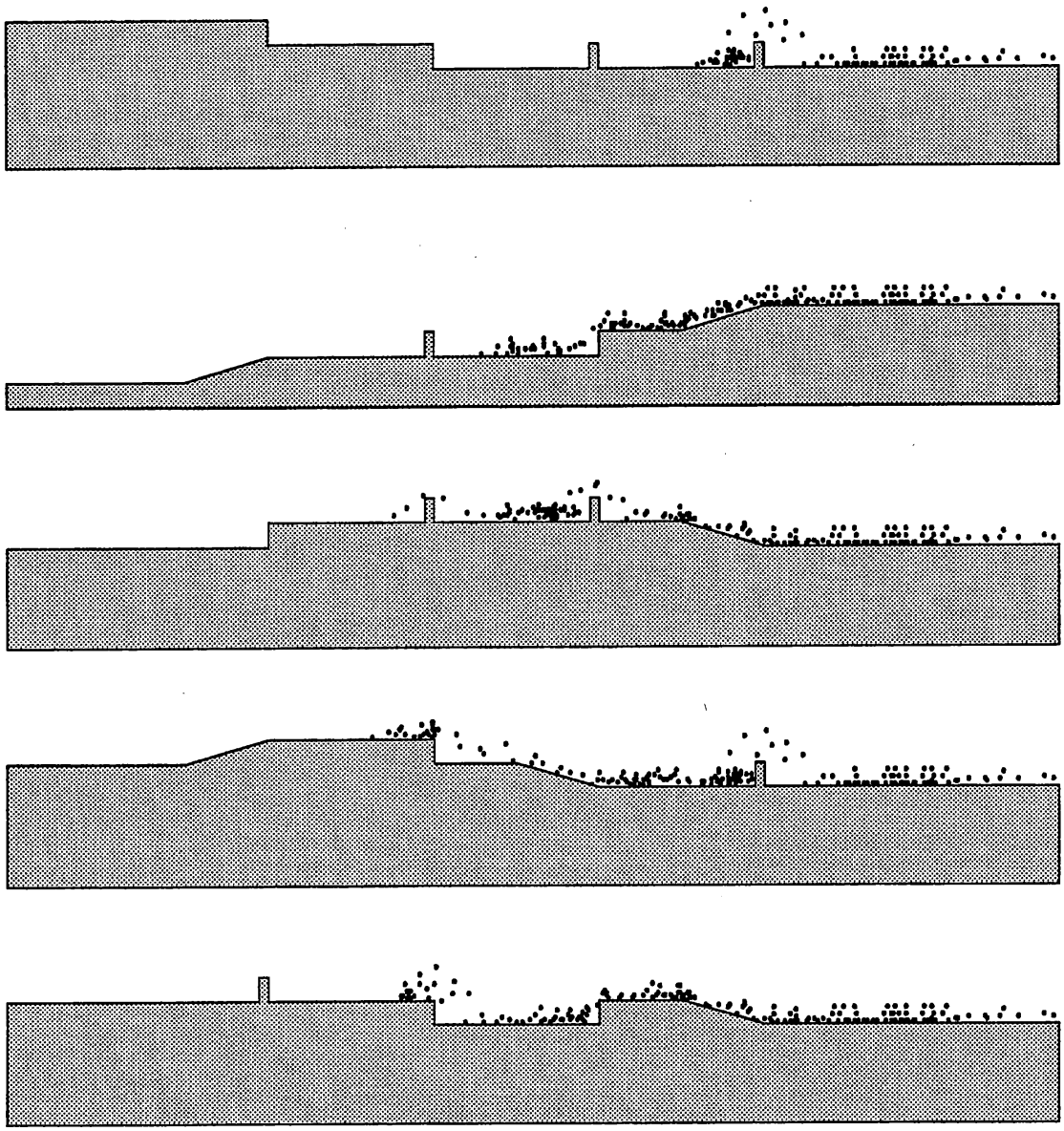


Figure 10.8: Sample of obstacle combinations which the robot was originally unable to cross

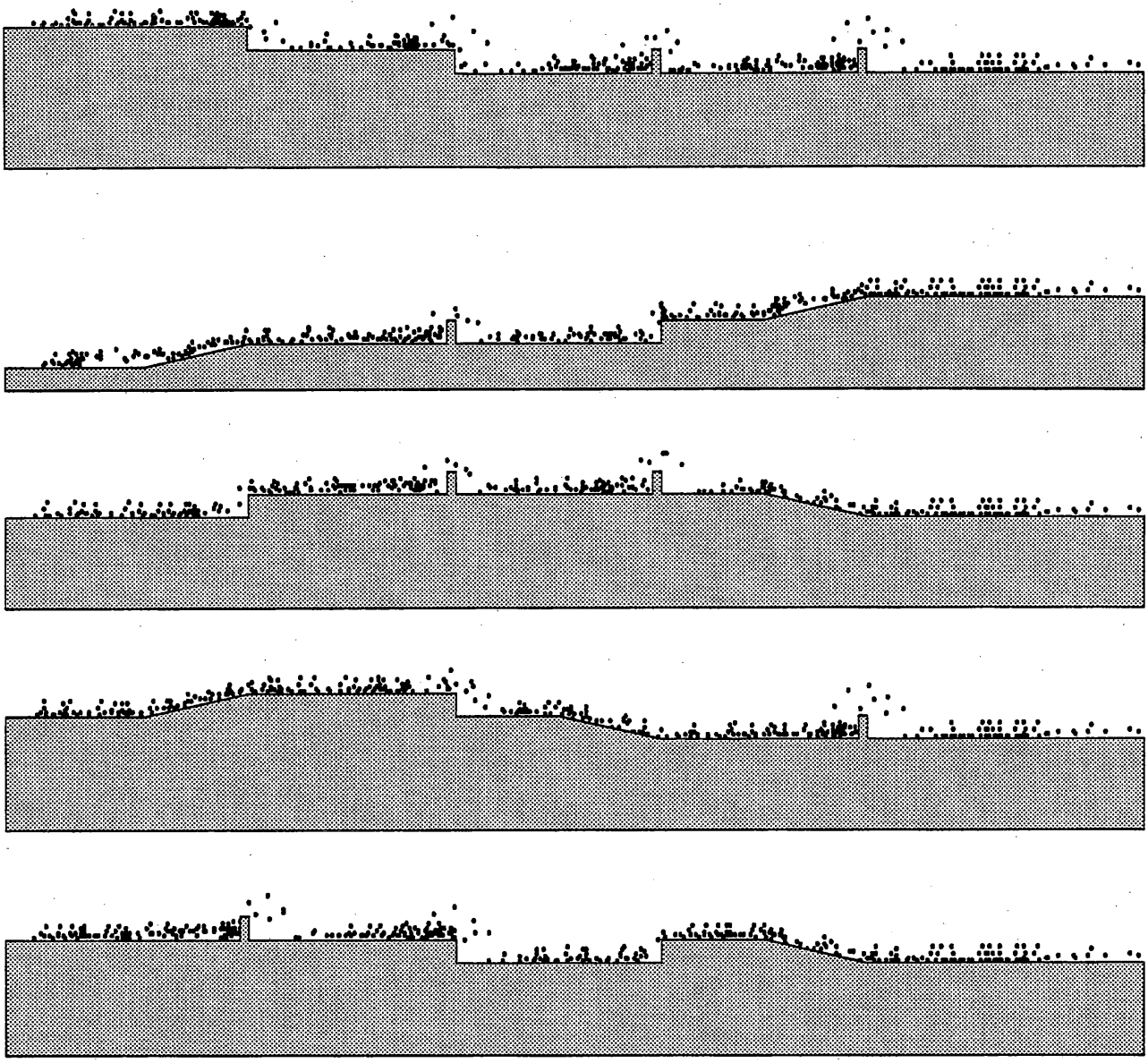


Figure 10.9: The same set of obstacles as in Figure 10.8. This time the distance between obstacles has been increased from 90mm to 120mm and the robot is now able to cross all obstacle combinations.

## 10.2 Rough Terrain

Section 10.1 considered smooth terrain; we now turn our attention to rough terrain which more closely approximates a surface that might be found outdoors. A set of such surfaces was generated by concatenating slopes whose width and steepness varied randomly. Figure 10.10 and Figure 10.11 represent a sample of such randomly generated, rugged surfaces. The robot was able to cross all of the surfaces that were tested. This is due to the fact that during most of the terrain crossing simple search for steps sufficed to move the robot forwards.

It is important to note that the robot was modelled in such a way that the foot of the support leg had optimal grip on any surface. Thus the robot was able to stand on rather steep slopes. This in part accounts for the ease with which the robot is able to navigate through rough terrain.

In general the robot proceeded by trying to apply its standard gait. Whenever this failed the robot searched for steps using the weighting of the inner state space. When the terrain elevation in front of the robot differed by more than 2mm from the elevation of the current position of the robot then the robot assumed it had approached an obstacle; in this case it searched to see whether it was in some obstacle space, if so the robot used the corresponding virtual evaluation function.

## 10.3 Real-Time Considerations

The code generating the obstacle crossings, the gait execution and search programs was written in C; the step-length adjustment and overall control functions were written in Lucid Common Lisp. The code ran on a Motorola 68020 based Hewlett Packard HP9000/350 workstation with 8 Mb memory. The average time to cross obstacle combinations like such as those described earlier in this chapter was 120 seconds. Without step-length adjustments, using the gaits found in a previous trial, the robot was 30 seconds faster. This demonstrates the small overhead for backtracking and step-length adjustment. This is about 10 times slower than would be needed for the real-time *execution* (excluding search) of the obstacle crossing. If the obstacle crossing had been performed by the real robot then the robot would be crossing 50cm (the length of the obstacle path) in about 9 seconds, which corresponds to an average speed of 200 meters per hour.

It would be interesting to see what computing resources will be required to obtain this obstacle crossing performance (including search for step-length ad-

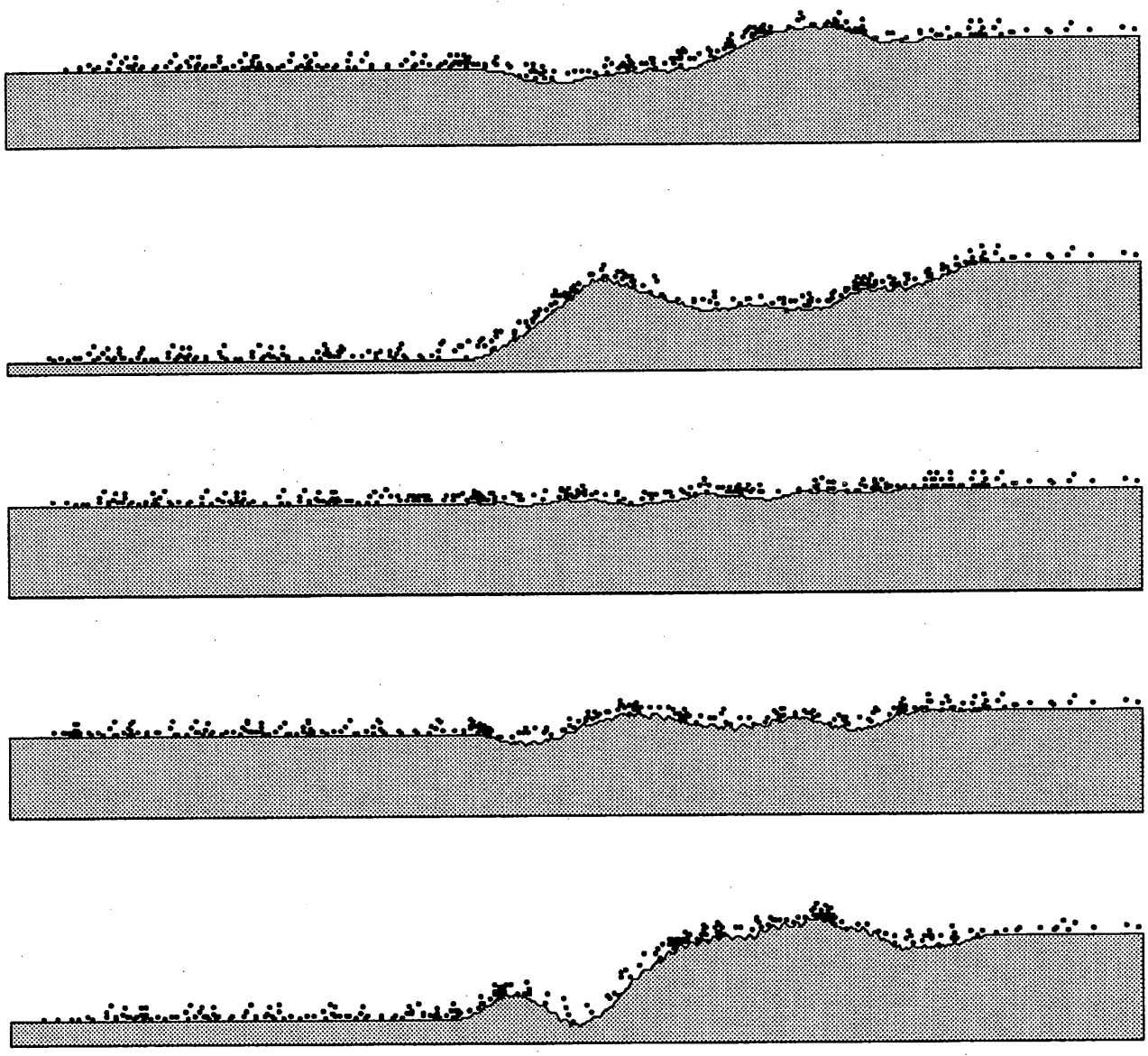


Figure 10.10: The robot crosses a set of randomly generated "rough" surface.

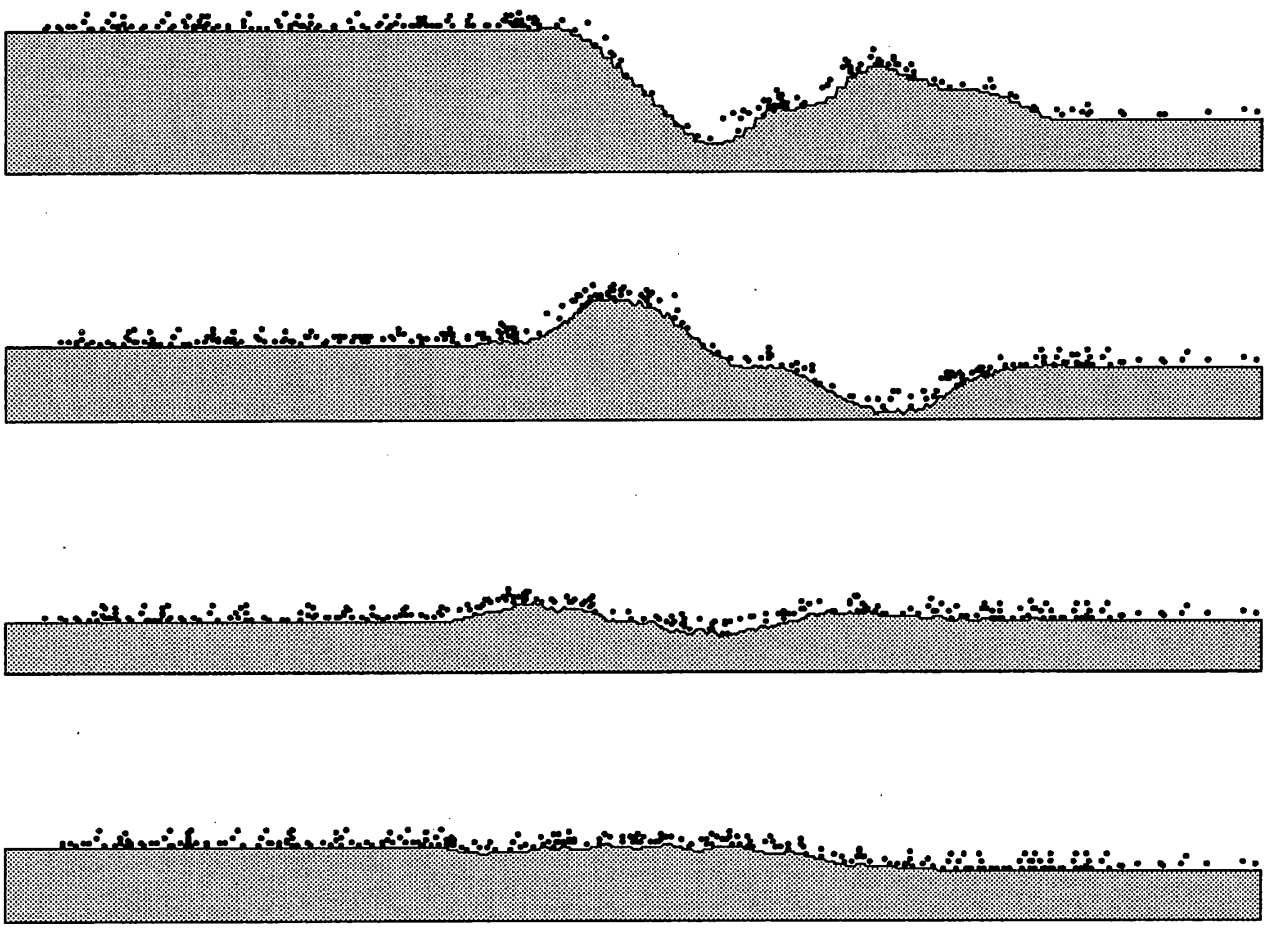


Figure 10.11: The robot crosses another set of random surfaces.

justment) in real time. This will be based on the assumption that all sensor data are immediately available and all communication to the physical robot can be carried out in real time. It is further assumed that the state space has been labelled with the appropriate virtual evaluation functions.

For obstacle crossings using the virtual evaluation function the robot has to be able to plan several steps ahead before committing to the execution of some gait. The duration of a gait execution varies between about 0.1 and 0.3 seconds, and thus the robot has to compute the effects of several steps ahead within this time frame. This is needed in order to decide whether a step-length adjustment has to be carried out, such that the robot reaches the next obstacle in the right posture and with the correct distance to the obstacle.

The amount of backtracking required to select the correct gait to approach the next obstacle is very small - usually at most one or two backtrack steps were needed to cross an individual obstacle. This is equivalent to an upper limit on the number of step-length adjustments. The small amount of backtracking justifies the assumption that the search for obstacle crossing gait combinations can start reasonably close to the obstacle; therefore there is an upper boundary on the necessary number of steps for which the robot needs to look-ahead. Thus it is possible to enumerate all possible step combinations which could lead the robot over the obstacle and search all these step combinations in parallel.

Since each step can be searched for in constant time (the hillclimbing does not allow for backtracking) the pre-planning of a gait execution can be done in the order $\theta(n^b)$ time steps where $n$ is the length of the look-ahead and $b$ the number of adjustments used by the step-length adjuster.

Assuming that choosing the best out of 30 motor commands requires 3000 computations (100 computations per motor command and no exploitation of the possible parallelism and pipelining - a very generous estimate), and further assuming that each step requires 10 motor commands, looking ahead 10 steps will require 300000 computations. Assuming these computations correspond to machine instructions, a 30 MIPS machine would therefore be able to manage a 10 step look-ahead in 0.01 seconds. This is clearly within the range of current (1991) micro-processor technology. Experimental results suggest that not more than 2 step-length adjustments per obstacle are necessary. Thus 100 such look-aheads are needed, but all these look-aheads can be run independently and in parallel.

## 10.4 Summary

This chapter provided the experimental proof that the biped acquired a generic obstacle crossing capability. This enabled the robot to cross previously unencountered obstacles. The limitations of the robot are the limitations of its physical characteristics and do not depend on the combination of obstacles. Provided the robot was given enough time to recover from one obstacle crossing then it encountered no difficulties in crossing the next obstacles. Currently the obstacle crossing performance of the robot is therefore limited by the distance between obstacles and the height of the obstacle. Both of these features do directly correspond to the physical characteristics of the robot itself: if the robot failed this was due to the fact that the robot lacked the power to bring itself quickly enough into an upright position. The same is true if one increases the height of a step or a fence; provided the robot is given more powerful torques to control its joints it can successfully cross higher obstacles.

# Chapter 11

# Conclusion

## 11.1 Contributions

Previous work in robotic learning is limited in the sense that programs learn actions rather than functions: based on a state space with relatively few dimensions the robot acquires a model of the effects of its individual actions. Based on this knowledge the robot then has to search for a sequence of actions which allows it to reach a certain goal. Whenever the robot has to execute a task *similar* to one which it previously executed, it has to search again for all the actions needed to perform this task [Moo91, Sut91]. This emphasises the need for techniques which enable a robot to deal with increasingly complex state spaces and to transfer previous solutions to new tasks. It leads to the main approach of this thesis which is to couple these two aspects and transfer behavioural knowledge from lower dimensional state spaces into more complex higher dimensional state spaces.

This thesis followed a hierarchical approach in order to decompose automatically the state space. The basic idea was that there is a small "core" of dimensions of the state space (the inner state space) for which the robot needs to develop an original strategy of how to achieve its goals. Exploring regions of the state space outside this original core of dimensions can then be done by incrementally modifying the behaviours which succeeded inside the inner state space. We kept a close connection between the behaviours of the robot and the dimension of the state space: each behaviour (seen as a sequence of activities) is identified with a set of parameters to which this behaviour is sensitive. Changes to any of these parameters lead with high probability to the failure of the robot during the execution of the corresponding behaviour.

This approach was then applied to the domain of biped robot walking. Here the notion of the inner state space and its associated behaviour corresponds to the

kinematic parameters of the robot and the execution of a gait on a horizontal obstacle free surface. Learning this gait was accomplished by searching the inner state space using an evaluation function based on the importance of the parameters for the survival of the robot. The robot then improved its gait and developed a controller for the gait execution. We were able to show that simple look-up table based bang-bang control was sufficient to generate a stable and robust ability to walk in an obstacle free environment.

We then extended the activities of the robot: in order to cross obstacles the robot needed to incorporate the parameters describing these obstacles into its reasoning. From this we developed the notion of an obstacle space, defined as the part of the state space where the normal behaviour of the robot leads to failure. The robot learned to cross various obstacles by modifying existing behaviours in a systematic way. The robot's behaviour was seen as similar to a periodic function, and it is possible to identify the frequency (stride), amplitude (lifting of leg) and displacement (step-length adjustment) of a gait. By slightly varying such aspects of a periodic function it became possible to modify the original function while keeping most of its properties. This was the motivation for the design of the qualitative equivalence heuristics. Modifying parts of a gait, while keeping most of its original properties, enabled the robot to use previously acquired knowledge from simpler domains (walking on a horizontal obstacle free surface) and apply it to new, more complicated situations (obstacle crossings).

In a last step we refined the definition of the obstacle space even further: the obstacle space is now divided into hyperrectangles. Each of these hyperrectangles corresponds to a virtual evaluation function, describing how the actions of the robot differ inside the hyperrectangle from the robot's normal actions outside the obstacle space. The virtual evaluation function concept provided a tool to analyse and compare behaviours. It further enabled the robot to operationalise its analysis of obstacle crossing exemplars. Thus, based on a small set of obstacle crossing exemplars the robot was able to generate a generic obstacle crossing capability.

## 11.2 Discussion

### 11.2.1 Scaling Up: Higher Dimensional State Spaces

Very large dimensional state spaces do not pose an immediate problem to the algorithms of this thesis. The size of the overall search space does not really matter as long as the robot is able to identify an inner state space which is easy to search. The same holds for the search over the space of qualitatively equivalent behaviours: the size of the original state space of the robot does not matter as long as the space of behavioural modifications remains relatively small or easy to search.

However these assumptions may not always hold. It is possible that the required behaviour of the robot is so complex that the search has to rely on an exhaustive enumeration of the search space. In this case the complexity of the search grows together with the search space, and the exponential character of this growth will make any complex search prohibitively expensive. One way to alleviate this is by introducing additional domain knowledge into the search program.

Thus the applicability of the algorithms discussed in this thesis share the problems of all search techniques: if the search space is benevolent (a smooth surface and an effective evaluation function), then simple hillclimbing can be very efficient. If the search space is riddled with local minima, then nothing short of an exhaustive search will do. However the wide spread use of hillclimbing search techniques suggests that such rough or chaotic search spaces are relatively rare. Additionally the technique to weight the individual parameters of the search space, based on their impact on the survival of the robot, proved to be a powerful tool in order to improve the search process.

### 11.2.2 Abstraction

We were able to generate a mechanism to develop a high level description of the robot's behaviour. By aggregating successful operator sequences into macros and then searching the space of macro modifications it became possible to change to a more abstract description of the robot. However we only partially used this abstract view of the robot to generate a generic obstacle crossing capability: the search for a generic obstacle crossing capability has been implemented in such a way that behavioural modifications using the qualitative equivalence heuristic were employed to generate a set of successful obstacle crossing exemplars. Once these exemplars were obtained the focus of the program changed and a labelling of the state space (virtual evaluation

function) was derived from these obstacle crossing exemplars. The labelled state space could then be used for an efficient search for obstacle crossing *actions*. This interaction between high level behavioural patterns and improved search for actions and action sequences deserves further attention and should be used as a starting point for further work.

### 11.2.3 Perspective

This thesis presented a novel set of algorithms for learning complex robotic tasks. We were able to show how large state spaces could be stripped of irrelevant parameters, we generated new behaviours from old ones, adapted them to new situations and analysed them so that learning from examples could take place. Taking all this together we enabled a robot to learn dynamic biped walking in obstacle cluttered environments.

The experimental results were very encouraging, and they demonstrated that relatively simple symbolic algorithms were able to generate solutions to tasks as complex as biped obstacle crossing. The incremental search over a search space of increasing size was made possible by the transfer of abstract descriptions of the robot's behaviour into new, more complex environments. This technique offers the possibility to search efficiently in search spaces where local minima would force less abstract search methods to give up. As a result we were able to demonstrate a previously unreported ability to learn a generic biped obstacle crossing capability.

Previous work in robotic learning has been concerned with the acquisition of a world model based on which the robot then searched for actions leading to its goal [Moo90], alternatively a control law was learned by the robot [Lee91, BAS83], or in a subsumption architecture a stimulus response pattern was learned [MB90]. Progress was limited to relatively small dimensional search spaces. This dissertation provided a way to integrate these previous results into the search of large, regular, and incremental state spaces. It now becomes possible to use these previous approaches in order to search the inner state space or to search over the space of behavioural modifications, because the algorithms of this thesis structure the state space into small, manageable search spaces, and therefore provide a possibility to remove the curse of high dimensionality.

## 11.3 Future Work

### 11.3.1 More and Better Behaviours

At the moment the obstacle crossing performance of the robot is improved by analysing the individual obstacle crossing examples and deriving a virtual evaluation function from these examples. The virtual evaluation function concept was powerful enough to develop a generic obstacle crossing capability. However no analysis of the resulting obstacle crossing *gaits* has yet been carried out. Such an analysis would have to concentrate on the relationship between obstacles, gaits and the start positions of these gaits. Nearest neighbour pattern matching techniques as discussed by Moore [Moo90] could then be used to develop a yet more powerful representation of the entire obstacle crossing *gait*. The result would be a repertoire of obstacle crossing gaits which could be modified using the qualitative equivalence heuristic to negotiate previously unencountered obstacles.

This would allow us a final shift from the space of robot states and motor commands to a space of gaits, obstacles and gait modifications. Techniques like the survival time heuristic could then be used again to give a weighting to the space of all gaits and their modifications. This would enable the robot to use an improved evaluation function for the search for gait modifications.

### 11.3.2 Switching Between Behaviours

Chapter 6 demonstrated the robot's ability to switch from one gait to another by *searching* for actions which would lead it into the start position of the new gait. However the complexity of the search space can make such a search very difficult, and a *spanning tree* of behaviours and the transitions between them might be necessary. Such a spanning tree for biped gaits has been constructed and it allowed the robot to switch between gaits where normal search would have failed. For example the robot was able to change from a fast gait $G_{\text{fast-forward}}$ into gait $G_{\text{fast-backward}}$ which moves quickly into the opposite direction by choosing gaits with slower respective speeds as intermediary goals. Thus the robot went through a sequence of gaits $G_{\text{fast-forward}} \rightarrow G_{\text{slow-forward}} \rightarrow G_{\text{slow-backward}} \rightarrow G_{\text{fast-backward}}$. However a direct change from gait $G_{\text{fast-forward}}$ into gait $G_{\text{fast-backward}}$ failed. It will be interesting to analyse this switching between behaviours further and investigate a higher level description of behavioural transitions.

### 11.3.3 The Application to Different Domains

The algorithms which we discussed in this thesis have been developed to learn incremental regular behavioural patterns in large state spaces. Domain specific knowledge has only been introduced by defining the goals of the search processes. Apart from this the algorithms depend only on the benevolence of the underlying search space. There are many domains which appear to share these properties. Regular activities like cycling, swimming or flying (insects or aero-planes) would therefore be interesting candidates for the application of the algorithms.

It will also be interesting to move into applications with a larger set of behaviours, for example a biped which can also jump and run[Rai86b]. The availability of a large set of behaviours will make it possible to investigate a number of interesting questions: how many different behaviours need to be stored, and how orthogonal will these behaviours be? Up to what point can behaviours again be combined into higher level behaviours? What type of synergy can be observed? What type of problems can be solved using incremental behavioural learning? Answering such questions will let us develop a better understanding of behaviour based robots and give us a clearer view of the challenges on the way towards autonomous robots.

## 11.4 Epilogue

The solution of complex robotic tasks in complex environments requires a set of various tools, and at the moment it is hard to see that a single research paradigm or unifying theory can solve all the problems of an autonomous robot. For the time being hybrid approaches using different tools and paradigms appear to be the most promising way to combine the various aspects of robotic tasks in order to build autonomous robots.

Maybe one day an all encompassing theory of autonomous agents will be available and autonomous robots will roam on distant planets and perform tasks to the benefit of mankind. Hopefully the application of this technology will not include autonomous hordes of robotic fighters for 21st century trench warfare. This will remain the responsibility of the individual researcher.

# Bibliography


[ACP88] S. Addanky, R. Cremonimi, and J.S. Penberthy. Contexts: Dynamic identification of common parameters in distributed analysis of complex devices. Technical Report RC 14273 (#63862), IBM T.J. Watson Research Center, Yorktown Heights, N.Y., 1988.

[AD74] P.R. Adby and M.A.H. Dempster. *Introduction to Optimization Methods*. Chapman and Hall, London, 1974.

[Ada71] J.A. Adams. A closed-loop theory of motor learning. *Journal of Motor Behavior*, 3(2):111–149, 1971.

[AF88] S. Anderson and A.M. Farley. Plan abstraction based on operator generalization. In *AAAI 88*, pages 100–104, Austin, Tx, 1988.

[AGL87] W.W. Armstrong, M. Greene, and R. Lake. Near-real-time control of human figure models. *IEEE Computer Graphics and Applications*, pages 52–61, June 1987.

[AK89] J. Altman and J. Klein. New models for motor control. *Neural Computation*, 1:173–183, 1989.

[AKA91] D.W Aha, D. Kibler, and M.K. Albert. Instance based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[AKN88] H. Adachi, N. Koyachi, and E. Nakano. Mechanism and control of a quadruped walking robot. *IEEE Control Systems Magazine*, 8(5):14–19, 1988. Special issue on machine intelligence and robotics.

[Alb75a] J.S. Albus. Data storage in the cerebellar model articulator controller. *Transactions of the ASME : Journal of Dynamic Systems, Measurement and Control*, pages 228–233, September 1975.

[Alb75b] J.S. Albus. A new approach to manipulator control. *Transactions of the ASME : Journal of Dynamic Systems, Measurement and Control*, pages 220–227, September 1975.

[Alb79] J.S. Albus. Mechanisms of planning and problem solving in the brain. *Mathematical Biosciences*, 45:247–293, 1979.

[Ama68] S. Amarel. On representation of problems of reasoning about actions. *Machine Intelligence*, 3:131–171, 1968.

[AMP87] Y.S. Abu-Mostafa and D. Psaltis. Optical neuro computers. *Scientific American*, 256:66–73, 1987.

[AR88] J.A. Anderson and E. Rosenfeld. *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, Ma., 1988.

[Ark87] R.C. Arkin. *Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-Made Environments*. PhD thesis, University of Massachusetts at Amherst, 1987.

[Ark89] R.C. Arkin. Motor-schema based mobile robot navigation. *The International Journal of Robotics Research*, 8(4):92–112, 1989.

[AS83] D. Angluin and C.H. Smith. Inductive inference : Theory and methods. *Computing Surveys*, 15(3):221–269, September 1983.

[AS86] H. Asada and J-J.E. Slotine. *Robot Analysis and Control*. J. Wiley, New York, 1986.

[Bal86] D.H. Ballard. Parallel logical inference and energy minimization. In *AAAI 86*, pages 203–208, Philadelphia, 1986.

[Bar89a] H.G. Barrow. AI, neural networks and early vision. *AISB Quarterly Newsletter*, 69:7–25, 1989.

[Bar89b] A.G. Barto. Connectionist learning for control: An overview. Technical Report COINS–TR–89–89, University of Massachusetts at Amherst, 1989.

[Bar89c] J. Barwise. Mathematical proofs of computer system correctness. *Notices of the American Mathematical Society*, 36(7):844–851, 1989.

[BAS82] A.G. Barto, C.W. Anderson, and R.S. Sutton. Synthesis of nonlinear control surfaces by a layered associative search network. *Biological Cybernetics*, 43:175–185, 1982.

[BAS83] A.G. Barto, C.W. Anderson, and R.S. Sutton. Neuron like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics 13*, 13:835–846, 1983.

[Bav88] B. Bavarian. Introduction to neural networks for intelligent control. *IEEE Control Systems Magazine,* 8(2):3–7, 1988. Special section on neural networks for systems and control.

[BCL+90] H. Berenji, Y-Y. Chen, Ch-Ch. Lee, J-S. Jang, and S. Murugesan. A hierarchical design to designing approximate reasoning-based controllers for dynamic physical systems. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence,* pages 362–369, Cambridge, Ma., 1990.

[BCS90] R.D. Beer, H.J. Chiel, and L.S. Sterling. A biological perspective on autonomous agent design. *Robotics and Autonomous Systems,* 6:169–186, 1990.

[Ben75] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM,* 18(9):509–517, 1975.

[Ber76] D.P. Bertsekas. *Dynamic Programming.* Prentice-Hall, Englewood Cliffs, 1976.

[Ber90a] H. Berenji. Neural networks and fuzzy logic in intelligent control. In *Proceedings of 5th IEEE International Symposium on Intelligent Control,* pages 916–920, Philadelphia, Penn., 1990.

[Ber90b] H.R. Berenji. Machine learning in fuzzy control. In *Proceedings of the International Conference on Fuzzy Logic & Neural Networks,* pages 231–234, IIZUKA, Japan, 1990.

[BF88] R.K. Belew and S. Forrest. Learning and programming in classifier systems. *Machine Learning,* 3:193–223, 1988.

[BG87] J-P. Banquet and S. Grossberg. Probing cognitive processes through the structure of event related potentials during learning: an experimental and theoretical analysis. *Applied Optics,* 26(23):4931–4946, 1987.

[BGH87] L.B. Booker, D.E. Goldberg, and J.H. Holland. Classifier systems and genetic algorithms. Technical Report No. 8, Cognitive Science and Machine Intelligence Laboratory, University of Michigan, April 1987.

[BH87a] J.S. Bay and H. Hemami. Modelling of a neural pattern generator with coupled nonlinear oscillations. *IEEE Transactions on Biomedical Engineering,* BME-34(4):297–306, 1987.

[BH87b] J.S. Bay and H. Hemami. Modelling of a neural pattern generator with coupled nonlinear oscillators. *IEEE Transactions on Biomedical Engineering,* BME-34(4):297–306, 1987.

[BHK+89] J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. Whittaker. Ambler, an autonomous rover for planetary exploration. *Computer*, 22(6):18–25, 1989.

[BK89] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1186, 1989.

[Bli87] J.F. Blinn. Nested transformations and blobby man. *IEEE Computer Graphics and Applications*, pages 59–65, October 1987.

[Bol87] L. Bolc, editor. *Computational Models of Learning*. Springer-Verlag, Berlin, 1987.

[BP84] M. Brady and R. Paul, editors. *Robotics Research. The First International Symposium*. MIT Press, Cambridge, Ma., 1984.

[Bra82] M. Brady et. al., editor. *Robot Motion: planning and control*. MIT Press, Cambridge, Ma., 1982.

[Bro84] R.A. Brooks. Planning collision free motions for pick and place operations. In M. Brady and R. Paul, editors, *Robotics Research, the First International Symposium*, pages 5–37. MIT Press, Cambridge,Ma., 1984.

[Bro86a] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[Bro86b] V.B. Brooks. *The Neural Basis of Motor Control*. Oxford University Press, Oxford, 1986.

[Bro89] R.A. Brooks. A robot that walks; emergent behavior from a carefully evolved network. *Neural Computation*, 1:253–262, 1989.

[BS81] A.G. Barto and R.S. Sutton. Landmark learning, an example of associative search. *Biological Cybernetics*, 42:1–8, 1981.

[BSC89] A.J. van den Bogert, H.C. Schamhardt, and A Crowe. Simulation of quadrupedal locomotion using a rigid body model. *Journal of Biomechanics*, 22(1):33–41, 1989.

[Buc88a] S.J. Buckley. Compliance viewed as programming a damped spring. In *AAAI 88*, pages 762–767, Austin, Tx, 1988.

[Buc88b] S.J. Buckley. Fast motion planning for multiple moving robots. Technical Report RC 14099 (#63205), IBM T.J. Watson Research Center, Yorktown Heights, N.Y., 1988.

[BWP89] T.E. Bihari, T.M. Walliser, and M.R. Patterson. Controlling the adaptive suspension vehicle. *Computer*, 22(6):59–64, 1989.

[Can67] R.H. Cannon. *Dynamics of Physical Systems*. J. Wiley, New York, 1967.

[CC87] B.A. Cartwright and T.S. Colliet. Landmark maps for honeybees. *Biological Cybernetics*, 57:85–93, 1987.

[CG87] G.A. Carpenter and S. Grossberg. ART2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26(23):4919–4930, 1987.

[CHH86] B-R. Chen, M.J. Hines, and H. Hemami. Dynamic modelling for implementation of a right turn bipedal walking. *Journal of Biomechanics*, 13(3):195–206, 1986.

[Chr90] J. Christensen. A hierarchical planner that generates its own hierarchies. In *AAAI 90*, pages 1004–1009, Boston, Ma, 1990.

[CK86] J. Christensen and R.E. Korf. A unified theory of heuristic evaluation functions and its application to learning. In *AAAI 86*, pages 148–152, Philadelphia, 1986.

[CM86] W.F. Clocksin and A.J. Morgan. Qualitative control. In *ECAI 86*, pages 350–356, Brighton, 1986.

[CM89] W.F. Clocksin and A.M. Moore. Some experiments in adaptive state space robotics. In *Proceedings of AISB89 Conference*, Brighton, 1989.

[CN89] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[Coo89] D.J. Cook. ANAGRAM: An analogical planning system. Technical Report UIUCDCS–R–89–1561, University of Illinois at Urbana Champaign, 1989.

[CS88] B.S. Choi and S.M. Song. Fully automated obstacle crossing gaits for a walking machine. *IEEE Transactions on Systems, Man and Cybernetics*, 18(6):952–964, 1988.

[CSD87] P.P. Chakrabarti, S.Ghose, and S.C. DeSarkar. Heuristic search through islands. *Artificial Intelligence*, 29:339–347, 1987.

[CU87] M.E. Connell and P.E. Utgoff. Learning to control a dynamic physical system. In *AAAI 87*, pages 456–459, Seattle, 1987.

[CWB+90] H. Cruse, E. Wichmeyer, M. Bruewer, P. Brockfeld, and A. Dress. On the cost functions for the control of the human arm movement. *Biological Cybernetics*, 62:519–528, 1990.

[Dav85] L. Davis. Applying adaptive algorithms to epistatic domains. In *IJCAI 85*, pages 162–164, Los Angelos, 1985.

[Dav87] L.D. Davis, editor. *Genetic Algorithms and Simulated Annealing*. Pitman, London, 1987.

[DeJ87] K. DeJong. Learning with genetic algorithms. *Machine Learning*, 3:121–138, 1987.

[DeJ90] G. DeJong. Explanation-based learning with plausible inferencing. Technical Report UIUCDCS–R–90-1577, University of Illinois at Urbana Champaign, 1990.

[Del89] F. Delcomyn. Walking in the american cockroach: the timing of motor activity in the legs during straight walking. *Biological Cybernetics*, 60(5):373–384, 1989.

[dG90] Hugo de Garis. Genetic programming. In *ECAI 90*, pages 204–206, Stockholm, 1990.

[Dit90] Th. G. Ditterich. Machine learning. *Annual Review of Computer Science*, 4:255–306, 1990.

[DK88] M.D. Donner and J. Kalvan. Recent progress in juggling robot research. Technical Report RC 14080 (#63170), IBM T.J. Watson Research Center, Yorktown Heights, N.Y., 1988.

[DM84] R. Dechter and D. Michie. Induction of plans. Technical Report TIRM–84–006, Turing Institute, 1984.

[Don87] M.D. Donner. *Real-Time Control of Walking*. Birkhäuser, Boston, 1987.

[Dor80] R.C. Dorf. *Modern Control Theory*. Addison-Wesley, Reading, Ma., 1980.

[DS77] C. Dawson and L. Siklossy. The role of preprocessing in problem solving systems. In *IJCAI 77*, pages 465–471, Cambridge, 1977.

[DS90] M. Dorigo and U. Schnepf. A bootstrapping approach to robot intelligence. Technical Report Report n. 90–068, Politechnico Di Milano, 1990.

[Fal87] B. Faltings. Qualitative kinematics in mechanics. In *IJCAI 87*, pages 436–442, Milano, 1987.

[FB82] J.A. Feldmann and D.H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6:205–254, 1982.

[FBB89] H. Flashner, A. Beuter, and C. Boettger. Parameter optimization model of learning in stepping motion. *Biological Cybernetics*, 60(4):277–284, 1989.

[Fer89] I.A. Ferguson. 'touring machines': Rational planners in open worlds. First year progress report and thesis proposal, Computer Laboratory, University of Cambridge, 1989.

[FFGL88] J.A. Feldmann, M.A. Fanty, N.H. Goddard, and K.J. Lynne. Computing with structured connectionist networks. *Communications of the ACM*, 31(2):170–187, 1988.

[FFN87] K.D. Forbus, B. Faltings, and P. Nielsen. Qualitative kinematics, a framework. In *IJCAI 87*, pages 430–435, Milano, 1987.

[FL90] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. Technical Report CMU–CS–90–100, Carnegie Mellon University, 1990.

[Fra89] G. Frazier. Ariel: A 100 gigaflops simulator for connectionist research. personal communication, 1989.

[FS90] J. Fushuro and A. Sano. Sensor-based control of a nine-link biped. *The International Journal of Robotics Research*, 9(2):83–98, 1990.

[Fu70] K.S. Fu. Learning control systems, review and outlook. *IEEE Transactions on Automatic Control*, pages 210–221, April 1970.

[GAK88] A. Guez, J. Albert, and M. Kam. Neural network architecture for control. *IEEE Control Systems Magazine*, 8(5):22–25, 1988. Special issue on machine intelligence and robotics.

[Gal88] S.I. Gallant. Connectionist expert systems. *Communications of the ACM*, 31(2):152–169, 1988.

[GE90] H.A. Güvenir and G.W. Ernst. Learning problem solving strategies using refinement and macro generation. *Artificial Intelligence*, 44:209–243, 1990.

[GKM81] A.P. Georgopoulos, J.F Kalaska, and J.T Massey. Spatial trajectories and reaction times of aimed movements: Effects of practice, uncertainty, and change in target location. *Journal of Neurophysiology*, 46(4):725–743, 1981.

[GM89] F. Gardin and B. Meltzer. Analogical representation of naive physics. *Artificial Intelligence*, 38(2):139–159, 1989.

[Gol85] D.E. Goldberg. Dynamic system control using learning and genetic algorithms. In *IJCAI 85*, pages 588–592, Los Angelos, 1985.

[Gre86] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems Man and Cybernetics*, 16(1):122–128, 1986.

[GS90] D.M. Gorinevsky and A.Yu. Shneider. Force control in locomotion of legged vehicles over rigid and soft surfaces. *The International Journal of Robotics Research*, 9(2):4–23, 1990.

[HA81] G.E. Hinton and J.A. Anderson. *Parallel Models of Associative Memory*. Erlbaum, Hillsdayle, N.J., 1981.

[Hal89] R.G. Hall. Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence*, 39(1):39–120, 1989.

[Han82] D.J. Hand. *Kernel Discriminant Analysis*. J. Wiley, New-York, 1982.

[Has86] Z. Hasan. Optimized movement trajectories and joint stiffness in unperturbed, inertially loaded movements. *Biological Cybernetics*, 53:373–382, 1986.

[Hem85] H. Hemami. Modelling, control and simulation of human movements. *CRC CRIT. REV. BIOMED. ENG*, 13:1–34, 1985.

[HF82] J.M. Hollerbach and T. Flash. Dynamic interaction between limb segments during planar arm movements. *Biological Cybernetics*, 44:67–77, 1982.

[HHNT86] J.H. Holland, K.J. Holoyak, R.E. Nisbett, and P.R Thagard. *Induction : Processes on Inference, Learning and Discovery*. MIT Press, Cambridge, Ma., 1986.

[Hin89] G.E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

[HK90] S.A. Hutchinson and A. C. Kak. Spar: A planner that satisfies operational and geometric goals in uncertain environments. *AI Magazin*, 11(1):30–61, 1990.

[HLK87] R.J. Hall, R.H. Lathorp, and R.S. Kirk. Multiple representation approach to understanding the time behavior of digital circuits. In *AAAI 87*, pages 799–803, Seattle, 1987.

[HM90] Shigeo Hirose and Akio Morishima. Design and control of a mobile robot with an articulated body. *The International Journal of Robotics Research*, 9(2):99–114, 1990.

[HN87] R. Hecht-Nielsen. Counterpropagation networks. *Applied Optics*, 26(23):4979–4984, 1987.

[Hog85] N. Hogan. The mechanics of multi-joint posture and movement control. *Biological Cybernetics*, 52:315–331, 1985.

[Hol82] J.M. Hollerbach. Computers, brains and the control of motion. *Trends in Neuro Science*, 5:189–192, 1982.

[Hoo88] A.T. Hooke. Multiple splitting of numerical variables in inductive learning. Technical Report TIRM–88–037, The Turing Institute, 1988.

[HR90] J.K. Hodgins and M.H. Raibert. Biped gymnastics. *The International Journal of Robotics Research*, 9(2):115–132, 1990.

[HR91] J.K. Hodgins and M.H. Raibert. Adjusting step length for rough terrain locomotion. *IEEE Transactions on Robotics and Automation*, 7(3):289–298, 1991.

[HS86] G.E. Hinton and T.J. Sejnowski. Learning and relearning in boltzmann machines. In D.E. Rumelhart and J.L. McClelland et.al., editors, *Parallel Distributed Processing*, volume 1 : Foundations, pages 282–317. MIT Press, Cambridge, Ma., 1986.

[HS91] O. Holland and M. Snaith. The neural control of locomotion in a quadrupedal robot. personal communication, 1991.

[HSW90] D. Helmbold, R. Sloan, and M.K. Warmuth. Learning nested differences of intersection close concept classes. *Machine Learning*, 5:165–196, 1990.

[HT85] J.J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[HTD90] J. Hendler, A. Tate, and M. Drummond. Ai planning: Systems and techniques. *AI Magazine*, 11(2):61–77, 1990.

[HZH82] H. Hemami, Y-F. Zheng, and M.J. Hines. Initiation of walk and tiptoe of a planar nine linked biped. *Mathematical Biosciences*, 61:163–189, 1982.

[Iba89] G.A. Iba. A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3:285–317, 1989.

[Jos87] L. Joskowitz. Shape and function in mechanical devices. In *AAAI 87*, pages 611–615, Seattle, 1987.

[KFR88] M. Kawato, K. Furukawa, and R.Suzuki. Coordinates transformation and learning control for visually guided voluntary movement with iteration : A newton like method in function space. *Biological Cybernetics*, 59:161–177, 1988.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–679, May 1983.

[KIMS87] M. Kawato, M. Isobe, Y. Maeda, and R. Suzuki. A hierarchical neural network model for control and learning of voluntary movements. *Biological Cybernetics*, 57:169–185, 1987.

[Kni89] K. Knight. A gentle introduction to subsymbolic computation: Connectionism for the AI researcher. Technical Report CMU–CS–89–150, Carnegie Mellon University, 1989.

[Kno90] C.A. Knoblock. Learning abstraction hierarchies for problem solving. In *AAAI 90*, pages 923–929, Boston, Ma, 1990.

[Koh82] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[Koh87] T. Kohonen. Adaptive, associative, and self-organizing functions in neural computing. *Applied Optics*, 26(23):4910–4918, 1987.

[Koh88] T. Kohonen. *Self Organization and Associative Memory*. Springer–Verlag, Heidelberg, 1988.

[Kor85a] R.E. Korf. Depth first iterative deepening : an optimal admissible search. *Artificial Intelligence*, 27:97–109, 1985.

[Kor85b] R.E. Korf. Macro operators : A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[Kor87] R.E. Korf. Planning as search, a qualitative approach. *Artificial Intelligence*, 33:65–88, 1987.

[Kos87] B. Kosko. Adaptive bidirectional associative memories. *Applied Optics*, 26(23):4947–4960, 1987.

[Koz90] J.R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report STAN-CS-90-1314, Stanford University, 1990.

[KRPW85] J.J. Kessis, J.P. Rambaut, J. Penne, and R. Wood. Six-legged walking robots. In G. Beni and S. Hackwood, editors, *Recent Advances in Robotics*, pages 243–262. John Wiley, New York, N.Y., 1985.

[KTT88] M. Kaneko, K. Tanie, and M. Than. A control algorithm for hexapod walking machine over soft ground. *IEEE Journal of Robotics and Automation*, 4(3):294–302, 1988.

[Kui88] B.J. Kuipers. Navigation and mapping in large scale space. *AI Magazine*, 9(2):25–43, Summer 1988.

[KUIS88] M. Kawato, Y. Uno, M. Isobe, and R. Suzuki. Hierarchical neural network model for voluntary movement with applications to robotics. *IEEE Control Systems Magazine*, 8(2):8–16, 1988. Special section on neural networks for systems and control.

[KY88] J.B. Kuipers and Y.T.Byun. A robust qualitative method for robot spatial learning. In *AAAI 88*, pages 774–779, Austin, 1988.

[LA85] D. Lyons and M. Arbib. A task-level model of distributed computation for sensory-based control of complex robot systems. Technical Report COINS–TR–85–30, University of Massachusetts at Amherst, 1985.

[Lan85] P. Langley. Learning to search : from weak methods to domain specific heuristics. *Cognitive Science*, 9:217–260, 1985.

[Lau88] C. Laugier. Traitement des incertitudes en programmation automatique des robots. Technical Report RR-0933, INRIA, Groupe de Recherche Grenoble, 1988.

[LB89] Ch-Ch. Lee and H. Berenji. An approximate controller based on approximate reasoning and reinforcement learning. In *Proceedings of the IEEE International symposium on Intelligent Control*, pages 200–205, Albany, N.Y., 1989.

[Lee91] D. M. A. Lee. A neural network approach to biped locomotion. Master's thesis, The University of Western Ontario, London, Ontario, Canada, May 1991.

[Lip87] R.P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 3(4):4–22, April 1987.

[Lip89] R.P Lippmann. Pattern classification using neural networks. *IEEE Communications*, 27(11):47–64, 1989.

[LL90] T.S. Levitt and D.T. Lawton. Qualitative navigation for mobile robots. *Artificial Intelligence*, 44:305–360, 1990.

[LM84] T. Lozano Peres and M.T. Mason. Automatic synthesis of fine motion strategies. In M. Brady and R. Paul, editors, *Robotics Research, the First International Symposium*, pages 65–97. MIT Press, Cambridge, Ma., 1984.

[LRN86] J.E. Laird, P.S. Rosenbloom, and A. Newell. Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[LS86] T-T. Lee and C-L. Shih. A study of the gait control of a quadruped walking vehicle. *IEEE Journal of Robotics and Automation*, RA-2(2):61–69, 1986.

[Lue79] D.G. Luenberger. *Introduction to Dynamic Systems.* J. Wiley, New York, 1979.

[Lyo86] D.M. Lyons. *RS: A Formal Model of Distributed Computation For Sensory-Based Robot Control.* PhD thesis, University of Massachusetts at Amherst, 1986.

[Mae89] P. Maes. How to do the right thing. *Connection Science Journal,* 1(3), 1989.

[Mae91] P. Maes. A bottom-up mechanism for behavior selection in an artificial creature. In *Proceedings of the Animat Conference,* Paris, 1991.

[Mau84] M.L. Mauldin. Maintaining diversity in genetic search. In *AAAI 84,* pages 247–250, Austin, 1984.

[MB90] P. Maes and R.A. Brooks. Learning to coordinate behaviors. In *AAAI 90,* pages 796–802, Boston, Ma, 1990.

[MC68] D. Michie and R.A. Chambers. Boxes: an experiment in adaptive state space control. *Machine Intelligence,* 2:137–152, 1968.

[MC91] S. Mahadevan and J. Conell. Automatic programming of behavior-based robots using reinforcement learning. Technical Report TR-RC16359, IBM T.J. Watson Research Center, 1991.

[McG90] T. McGeer. Passive dynamic walking. *The International Journal of Robotics Research,* 9(2):62–82, 1990.

[MCM83] R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, editors. *Machine Learning, an artificial intelligence approach.* Tioga, Palo Alto, Ca., 1983.

[MCM86] R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, editors. *Machine Learning, an artificial intelligence approach,* volume 2. Morgan Kaufmann, Los Angelos, Ca., 1986.

[Mel89] B.W. Mel. Murphy : A neurally-inspired connectionist approach to learning and performance in vision-based robot motion planning. Technical Report CSSR-89-17A, Department of Computer Science and Center for Complex Systems Research, Beckman Institute, University of Illinois, 1989.

[MGK87] W.T. Miller, F.H. Glanz, and L.G. Kraft. Applications of a general learning algorithm to the control of robot manipulators. *The International Journal of Robotics Research,* 6(2):84–98, 1987.

[Mic89] D. Michie. Personal models of rationality. Technical Report TIRM–88–035, Turing Institute, 1989.

[Min90] S. Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42:363–391, 1990.

[MKSS88] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki. Feedback error learning neural network for trajectory control of a robot manipulator. *Neural Networks*, 1:251–265, 1988.

[ML86] B.C. McInnis and C-K.F. Liu. Kinematics and dynamics in robotics: A tutorial based upon classical concepts of vectorial mechanics. *IEEE Journal of Robotics and Automation*, RA-2(4):181–186, 1986.

[MMZ88] F.A. Mussa Ivaldi, P. Morasso, and R. Zaccaria. Kinematic networks. *Biological Cybernetics*, 60(1):1–16, 1988.

[Moo89] A.W. Moore. Learning robotic control. First year progress report and thesis proposal, Computer Laboratory, University of Cambridge, 1988.

[Moo90] A.W. Moore. Efficient memory-based learning for robot control. Technical Report No.209, Computer Laboratory, University of Cambridge, 1990.

[Moo91] A.W. Moore. Variable resolution dynamic programming: efficiently learning action maps in multivariate real-valued state-spaces. In *ML 91*, pages 333–337, Evanston, Ill., 1991.

[Mor88] A.J. Morgan. *The Qualitative Behaviour of Dynamic Physical Systems*. PhD thesis, University of Cambridge, 1988.

[Mor90] E. Morales. Some experiments with macro operators in the 8-puzzle. Technical Report TIRM–90–042, Turing Institute, 1990.

[MOT84] R.B. McGhee, F. Ozguner, and S.J. Tsai. Rough terrain locomotion by a hexapod using a binocular ranging system. In M. Brady and R. Paul, editors, *Robotics Research, the First International Symposium*, pages 227–251. MIT Press, Cambridge,Ma., 1984.

[MP88] M.L. Minsky and S.A. Papert. *Perceptrons*. MIT Press, Cambridge, Ma., 1988.

[MR86] J.L. McClelland and D.E. Rumelhart et.al., editors. *Parallel Distributed Processing*, volume 2 : Psychological and Biological Models. MIT Press, Cambridge, Ma., 1986.

[MS79] M.Davis and J.T Schwartz. Mathematical extensibility for theorem verifiers and proof-checkers. *Comp. and Maths. with Appls.*, 5:217–230, 1979.

[MS84] H. Miura and I. Shimoyama. Dynamical walk of biped locomotion. In M. Brady and R. Paul, editors, *Robotics Research, the First International Symposium*, pages 303–325. MIT Press, Cambridge,Ma., 1984.

[MS87] M.L. Mavrovouniotis and G. Stephanopoulos. Reasoning with orders of magnitude and approximation relations. In *AAAI 87*, pages 626–630, Seattle, 1987.

[MSW90] W.T. Miller, R.S. Sutton, and P.J. Werbos, editors. *Neural Networks for Control.* MIT Press, Cambridge, Ma., 1990.

[Muy99] E. Muybridge. *Animals in Motion.* Chapman and Hall Ltd., London, 1899.

[Muy01] E. Muybridge. *The Human Figure in Motion.* Chapman and Hall Ltd., London, 1901.

[Nel83] W.L. Nelson. Physical principles for economics of skilled movements. *Biological Cybernetics*, 46:135–147, 1983.

[Nie88] P.E. Nielsen. *A Qualitative Approach to Rigid Body Mechanics.* PhD thesis, University of Illinois at Urbana Champaign, 1988.

[NL89] H.R. Nicholls and M.H. Lee. A survey of robot tactile sensing technology. *The International Journal of Robotics Research*, 8(3):3–30, 1989.

[ODMS87] Y. Owechko, G.J. Dunning, E. Marom, and B.H. Soffer. Holographic associative memory with nonlinearities in the correlation domain. *Applied Optics*, 26(10):1900–1910, 1987.

[Omo87] S. Omohundro. Efficient algorithms with neural network behavior. Technical Report UIUCDCS–R–87–1331, Department of Computer Science, University of Illinois at Urbana Champaign, 1987.

[Pao88] Y.H. Pao. Autonomous machine learning of effective control strategies with connectionist net. *Journal of Intelligent and Robotic Systems*, 1:35–53, 1988.

[Pau81] R.P. Paul. *Robot Manipulators : mathematics, programming and control : the computer control of robot manipulators.* MIT Press, Cambridge, Ma., 1981.

[PB88a] M.G. Pandy and N. Berme. A numerical method for simulating the dynamics of human walking. *Journal of Biomechanics*, 21(12):1043–1051, 1988.

[PB88b] M.G. Pandy and N. Berme. Synthesis of human walking: a planar model for single support. *Journal of Biomechanics*, 21(12):1053–1060, 1988.

[Pea76] K. Pearson. The control of walking. *Scientific American*, 235(6):72–86, 1976.

[Pea89] B.A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269, 1989.

[Pel89] B. Pell. Machine learning in the game of go. First year progress report and thesis proposal, Computer Laboratory, University of Cambridge, 1990.

[PG88] J. Pabon and D. Gossard. Connectionist networks for learning coordinated motion in autonomous systems. In *AAAI 88*, pages 791–795, Austin, Tx, 1988.

[PJ91] P.K. Pal and K. Jayarajan. Generation of free gait – a graph search approach. *IEEE Transactions on Robotics and Automation*, 7(3):299–305, 1991.

[PS78] H.M. Power and R.J. Simpson. *Introduction to Dynamics and Control.* McGraw Hill, London, 1978.

[PSY88] D. Psaltis, A. Sideris, and A.A. Yamamura. A multilayered neural network controller. *IEEE Control Systems Magazine*, 8(2):17–21, 1988. Special section on neural networks for systems and control.

[Pur88] K.S. Purswani. *A Probabilistic Reasoning-Based Approach to Machine Learning.* PhD thesis, University of Illinois at Urbana Champaign, 1988.

[Qui83] J.R. Quinlan. Learning efficient classification procedures and their application to chess end games. In R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning, an artificial intelligence approach*, volume 1, pages 463–482. Tioga, Palo Alto, Ca., 1983.

[Qui86] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Rab89] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.

[Rai78a] M.H. Raibert. A model for sensorimotor control and learning. *Biological Cybernetics*, 29:29–36, 1978.

[Rai78b] M.H. Raibert. *Motor Control and Learning by the State Space Model.* PhD thesis, MIT, 1978.

[Rai86a] M.H. Raibert. Legged robots. *Communications of the ACM,* 29(6):499–514, 1986.

[Rai86b] M.H. Raibert. *Legged Robots That Balance.* MIT Press, Cambridge, Ma., 1986.

[Rai86c] O. Raiman. Order of magnitude reasoning. In *AAAI 86,* pages 100–104, Philadelphia, 1986.

[Rai90] M.H. Raibert. Foreword to the special issue on legged locomotion. *The International Journal of Robotics Research,* 9(2):2–3, 1990.

[RBC84] M.H. Raibert, M.B. Brown, and M. Chepponis. Experiments in balance with a 3D one legged hopping machine. *The International Journal of Robotics Research,* 3(2):75–92, 1984.

[RBM84] M.H. Raibert, H.B. Brown, and S.S. Murthy. 3-D balance using 2-D algorithms? In M. Brady and R. Paul, editors, *Robotics Research, the First International Symposium,* pages 279–301. MIT Press, Cambridge,Ma., 1984.

[RC90] L. Rendell and H. Cho. Empirical learning as a function of concept character. *Machine Learning,* 5:267–298, 1990.

[RCB86] M.H. Raibert, M. Chepponis, and H.B. Brown. Running on four legs as though they were one. *IEEE Journal of Robotics and Automation,* RA-2(2):70–82, 1986.

[RCE82] D.L. Reilly, L.N. Cooper, and C. Elbbaum. A neural model for category learning. *Biological Cybernetics,* 45:35–41, 1982.

[Ren83] L. Rendell. A new basis for state space learning systems and a successful implementation. *Artificial Intelligence,* 20:369–392, 1983.

[Ren88] L. Rendell. Learning hard concepts through constructive induction : Framework and rationale. Technical Report UIUCDCS–R-88-1426, Department of Computer Science, University of Illinois at Urbana Champaign, 1988.

[Ric83] E. Rich. *Artificial Intelligence.* McGraw-Hill, New York, 1983.

[RK82] A. Rosenfeld and A.C. Kak. *Digital Picture Processing.* Academic Press, New York, 1982.

[RM86] D.E. Rumelhart and J.L. McClelland et.al., editors. *Parallel Distributed Processing,* volume 1 : Foundations. MIT Press, Cambridge, Ma., 1986.

[Ros83] S.M. Ross. *Introduction to stochastic dynamic programming.* Academic Press, London, 1983.

[RW88] H.E. Rauch and T. Winarske. Neural networks for routing telecommunication traffic. *IEEE Control Systems Magazine,* 8(2):22–31, 1988. Special section on neural networks for systems and control.

[Sal88] S. Salzberg. Exemplar based learning: Theory and implementation. Technical Report TR–10–88, Harvard University, 1988.

[Sau89] E. Saund. Dimensionality reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 11(3):304–314, 1989.

[SB68] J.R. Slagle and P. Bursky. Experiments with a multipurpose theorem proving heuristic program. *Journal of the ACM,* 15(1):85–99, 1968.

[SB89] R.S. Sutton and A.G. Barto. Time-derivative models of pavlovian reinforcement. To appear in : *Learning and Computational Neuroscience,* J.W. Moore and M. Gabriel, Eds., MIT Press, 1989.

[Sch82] R.A. Schmidt. *Motor Control and Learning.* Human Kinetics Publishers, Inc., Champaign, Il., 1982.

[Sch90a] R.E. Schapire. The strength of weak learnability. *Machine Learning,* 5:197–227, 1990.

[Sch90b] J.H. Schmidhuber. Towards compositional learning with dynamic neural networks. Technical Report FKI-129-90, Technische Universität München, 1990.

[SdMZ90] A.C. Sanderson, L.S. Homem de Mello, and Hui Zhang. Assembly sequence planning. *AI Magazine,* 11(1):62–81, 1990.

[SF71] J.R. Slagle and C.D. Farell. Experiments in automatic learning for a multipurpose heuristic program. *Communications of the ACM,* 14(2):91–99, 1971.

[SH90] J.H. Schmidhuber and R. Huber. Learning to generate focus trajectories for attentive vision. Technical Report FKI-128-90, Technische Universität München, 1990.

[SH91] M. Snaith and O. Holland. Quadrupedal walking using trained and untrained neural networks. In *ICANN 91,* Helsinki., June 1991.

[Shr87] J. Shrager. Theory change via view application in instructionless learning. *Machine Learning,* 2:247–276, 1987.

[SI89] Y. Suganuma and M. Ito. Learning movement and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(2):258–269, 1989.

[Slo89] R.H. Sloan. *Computational Learning Theory: New Models And Algorithms*. PhD thesis, Massachusetts Institute of Technology, 1989.

[SM89] C. Sammut and D. Michie. Controlling a 'black box' simulation of a spacecraft. Technical Report TIRM–89–039, Turing Institute, 1989.

[Smi83] S.F. Smith. Flexible learning of problem solving heuristics through adaptive search. In *IJCAI 83*, pages 422–425, Karlsruhe, 1983.

[SRA84] O.G. Selfrige, E.C. Rissland, and M.A. Arbib, editors. *Adaptive Control of Illdefined Systems*. Plenum Press, New York, 1984.

[SS85] O.G. Selfridge and R.S. Sutton. Training and tracking in robotics. In *IJCAI 85*, pages 670–672, Los Angelos, 1985.

[Sut88] R.S. Sutton. Learning to predict by the methods of temporal difference. *Machine Learning*, 3:9–44, 1988.

[Sut90] R.S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. Draft, to appear in : Proceedings of the Seventh International Conference on Machine Learning, June 1990.

[Sut91] R.S. Sutton. Planning by incremental dynamic programming. In *ML 91*, pages 353–357, Evanston, Ill., 1991.

[SW86] C. Stanfill and D. Waltz. Toward memory based reasoning. *Communications of the ACM*, 29(12):1213–1228, 1986.

[Ten86] J. Tennenberg. Planning with abstraction. In *AAAI 86*, pages 76–80, Philadelphia, 1986.

[Ten88] J.D. Tenenberg. *Abstraction in Planning*. PhD thesis, University of Rochester, 1988.

[TG74] J.T. Tou and R.C. Gonzalez. *Pattern Recognition Principles*. Addison-Wesley, Reading, Ma., 1974.

[TH85] D.S. Touretzky and G.E. Hinton. Symbols among the neurons : Details of a connectionist inference structure. In *IJCAI 85*, pages 238–243, Los Angelos, 1985.

[TNR90] M. Tambe, A. Newell, and P.S. Rosenschein. The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5:299–348, 1990.

[TS89] G. Tesauro and T.J. Sejnowski. A parallel network that learns to play backgammon. *Artificial Intelligence*, 39:357–390, 1989.

[TYS91] G Toga, Y. Yamaguchi, and H. Shimizu. Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment. *Biological Cybernetics*, 65(3):147–160, 1991.

[Val84] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[VT80] P. Viviani and C. Terzuolo. Space-time invariance in learned motor skills. *Adv. Psychol.*, 1:525–533, 1980.

[Was89] P.D. Wassermann. *Neural Computing*. Van Nostrand Reinhold, New York, 1989.

[Wel88] D.S. Weld. Comparative analysis. *Artificial Intelligence*, 36(3):333–374, 1988.

[Wey86] T.E. Weymouth. *Using Object Descriptions in a Schema Network for Machine Vision*. PhD thesis, University of Massachusetts at Amherst, 1986.

[Whi87] D. Whitney. Historical perspective and state of the art in robot force control. *The International Journal of Robotics Research*, 6(1):3–13, 1987.

[Wil87a] J. Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications*, pages 12–27, June 1987.

[Wil87b] S.W. Wilson. Classifier problems and the animat problem. *Machine Learning*, 2:199–220, 1987.

[Wil87c] S.W. Wilson. Hierarchical credit allocation in a classifier system. In *IJCAI 87*, pages 217–220, Milano, 1987.

[YWW90a] J.F. Yang, D.A. Winter, and R.P. Wells. Postural dynamics in the standing human. *Biological Cybernetics*, 62:309–320, 1990.

[YWW90b] J.F. Yang, D.A. Winter, and R.P. Wells. Postural dynamics of walking humans. *Biological Cybernetics*, 62:321–330, 1990.

[Zen89] Y.F. Zeng. Acceleration compensation for biped robots to reject external disturbances. *IEEE Transactions on Systems, Man and Cybernetics*, 19(1):74–84, 1989.

[Zhe90] Y.F. Zheng. A neural gait synthesiser for autonomous biped robots. In *IEEE International Workshop on Intelligent Robots and Systems IROS'90*, pages 601–608, 1990.

[Zis86] D. Zisper. Biologically plausible models of place recognition and goal location. In J.L. McClelland and D.E. Rumelhart et.al., editors, *Parallel Distributed Processing*, volume 2 : Psychological and Biological Models, pages 432–470. MIT Press, Cambridge, Ma., 1986.

[ZL91] D. Zhu and J.-C. Latombe. New heuristic algorithms for heuristic path planning. *IEEE Transactions on Robotics and Automation*, 7(1):9–20, 1991.

[ZS90] Y.F. Zheng and J. Shen. Gait synthesis for the sd-2 biped robot to climb sloping surfaces. *IEEE Transactions on Robotics and Automation*, 6(1):86–96, 1990.