MOTION PLANNING OF A MECHANICAL SNAKE USING NEURAL NETWORKS

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Barış Fidan

July 1998

TJ
211.35
.F53
1998

MOTION PLANNING OF A MECHANICAL SNAKE USING NEURAL NETWORKS

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND

ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Barış Fidan

July 1998

Tj
211.35
.F53
1998

B 042941

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. M. Erol Sezer(Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. A. Bülent Özgüler

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Dr. Billur Barshan

Approved for the Institute of Engineering and Sciences:

[illegible]

Prof. Dr. Mehmet Baray V.
Director of Institute of Engineering and Sciences

ABSTRACT

MOTION PLANNING OF A MECHANICAL SNAKE USING NEURAL NETWORKS

Barış Fidan
M.S. in Electrical and Electronics Engineering
Supervisor: Prof. Dr. M. Erol Sezer
July 1998

In this thesis, an optimal strategy is developed to get a mechanical snake (a robot composed of a sequence of articulated links), which is located arbitrarily in an enclosed region, out of the region through a specified exit without violating certain constraints. This task is done in two stages: Finding an optimal path that can be tracked, and tracking the optimal path found. Each stage is implemented by a neural network. Neural network of the second stage is constructed by direct evaluation of the weights after designing an efficient structure. Two independent neural networks are designed to implement the first stage, one trained to implement an algorithm we have derived to generate minimal paths and the other trained using multi-stage neural network approach. For the second design, the intuitive multi-stage neural network back propagation approach in the literature is formalized.

Keywords : Mechanical snake, articulated robot arm, motion planning, path planning, minimal path, back propagation, neural network, multi-stage neural network.

ÖZET

MEKANİK BİR YILANIN DEVİNİMİNİN YAPAY SİNİR AĞLARI İLE TASARLANMASI

Barış Fidan
Elektrik ve Elektronik Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Prof. Dr. M. Erol Sezer
Temmuz 1998

Bu tezde, kapalı bir alanın içine gelişigüzel yerleştirilen bir mekanik yılanın (eklemlerle birleştirilmiş bir dizi milden oluşan robot), bazı sınırlamaları bozmadan, bu alanın belirlenmiş bir çıkışından dışarı çıkarılmasını sağlayacak bir eniyi yöntem geliştirilmiştir. Bu yöntem iki evreden oluşmaktadır: İzlenebilecek eniyi yolun bulunması ve bulunan eniyi yolun izlenmesi. Bu evrelerin her biri bir yapay sinir ağı tarafından yürütülmektedir. İkinci evreyi yürüten yapay sinir ağı etkili bir yapının tasarlanmasından sonra, bu yapı içindeki ağırlıkların değerlerinin doğrudan bulunması ile oluşturulmuştur. Birinci evreyi yürütmek için birbirinden bağımsız iki yapay sinir ağı tasarlanmıştır. Bunların ilki en kısa yolları üretmek için geliştirdiğimiz bir algoritmayı yerine getirmek üzere eğitilmiştir. İkincinin eğitiminde ise çok evreli sinir ağları yaklaşımı kullanılmıştır. İkinci tasarı için, literatürde sezgisel olarak yer alan çok evreli sinir ağlarında geriyayılım yaklaşımı netleştirilmiştir.

Anahtar Kelimeler : Mekanik yılan, eklemli robot kolu, devinim tasarımı, yol tasarımı, en kısa yol, geriyayılım, yapay sinir ağı, çok evreli sinir ağı.

ACKNOWLEDGEMENT

I would like to express my deep gratitude to my supervisor Prof. Dr. Mesut Erol Sezer for his guidance, suggestions and invaluable encouragement throughout the development of this thesis.

I would like to thank to Prof. Dr. Arif Bülent Özgüler and Assoc. Prof. Dr. Billur Barshan for reading and commenting on the thesis and for the honor they gave me by presiding the jury.

I would like to thank to my family especially for their moral support.

Sincere thanks are also extended to everybody who has helped in the development of this thesis in some way.

to my parents,
and my brother

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

In robotics, the uttermost aim is to construct a physical device to simulate aspects of human behavior involving interaction with the world, such as manipulation or locomotion. To make a robot useful, an efficient control unit is required to control the robot's motions and the forces to be applied to the environment. Since the aim is to simulate human behavior, it seems feasible to construct emulators of the control units of humans as efficient robotic control units.

This idea directed the researchers towards the neural structures in living organisms and motivated them to study construction of artificial neural networks, networks with structures similar to those in humans containing processor units which mimic biological neurons. Usage of artificial neural networks in robotics and control has been popular for four decades. Because of their ability to solve highly nonlinear problems, neural networks have been used not only in robotics and control but also in many other areas such as optimization, signal processing, computer systems, statistical physics and communication [1, 2, 3, 4, 5].

Design of a useful neural network can be realized in two stages; choosing a suitable network structure and adjustment of the parameters, called weights, in this structure. In general, the weights can not be selected directly to fulfill an arbitrary task. In such a case, weights are adjusted by means of a training process. In the training process, the neural network is stimulated by the environment and the weights of the network are updated according to the result of the stimulation repetitively till the desired performance is obtained.

Most of the training techniques in the literature are gradient descent based. The basic technique used in this thesis is also a popular gradient descent based technique, Back Propagation. In basic Back Propagation algorithm, a set of input-desired output pairs is needed to train the neural network. However, for many tasks, such a set is not available. Such a case is the problem of driving a discrete-time plant from an initial state $\mathbf{z}_0$ to a desired state $\mathbf{z}_d$ in an unknown number of steps.

In this general problem, the only information in hand to train the controller is the desired outputs for the last steps. Desired outputs for the intermediate steps and number of steps to reach the desired final state are not available for any of the initial states in the training set. Using a multi-stage structure is beneficial in this case.

A multi-stage neural network is an integrated neural network composed of a number of cascaded neural networks. In [6] and [7], Nguyen and Widrow use a K-stage neural network to represent a K-step process to solve the problem above and train a controller which backs a trailer truck to a loading dock using this approach. However, the equivalent errors in the method are not well-defined and explicit formulation of the back propagation through K-stages is not given. In this work, we will introduce a similar approach to apply Back Propagation in multi-stage neural networks with identical stages and formulate the approach.

The neural networks designed in this work are used for motion planning of a mechanical snake, a robot with a structure and a motion pattern similar to those of a snake. A mechanical snake will be useful in situations where it is necesaary to crawl into places which are too dangerous or too narrow for people to enter, such as disaster areas and nuclear power plants with narrow vessels.

Efficiency of the motion for a mechanical snake is important, as it is for any mobile robot in general. Generating collision-free motion of satisfactory quality is one of the main areas of interest in robotics [8]. For the case of the mechanical snake, our aim is to generate smooth minimal paths, which can be tracked unobtrusively, to exit an enclosed area, and to track the generated paths efficiently.

The thesis is organized as follows: Basics of neural networks and applications of neural networks in robotics and control are reviewed and multistage neural networks are introduced in Chapter 2.

In Chapter 3, structure of the mechanical snake is determined, and the problem of exiting from the enclosed area is defined. Later, a structure for the complete controller is determined and the neural network for the path tracking task is designed directly.

In Chapter 4, an algorithm to generate minimal paths with maximum curvatures in the right half plane is derived, and this algorithm is converted to a neural network controller which generates minimal paths.

In Chapter 5, our approach to apply Back Propagation in multi-stage neural networks with identical stages is introduced, and this approach is used to solve the path generation problem.

In Chapter 6, concluding remarks are given.

Chapter 2

NEURAL NETWORKS IN ROBOTICS AND CONTROL

2.1 Multilayer Neural Networks

In many areas of research, an artificial neural network refers to a structure designed to implement or approximate an arbitrary function $\mathbf{o} = G(\mathbf{x})$, mimicking the neural structures in living creatures. This structure is composed of basic multi-input-single-output units called neurons. In the most general model for an n-input neuron, weighted summation of n inputs is biased and passed through an activation function $\varphi(.)$. The output for the neuron in Figure 2.1 is given by:

$$z = \varphi(\sum_{i=1}^{n} w_i y_i \ + \ w_b) \tag{2.1}$$

A multilayer neural network is a regular network in which the neurons are collected in a number of layers and the connections are settled between these layers. The layer which is closest to the output and which has outputs as

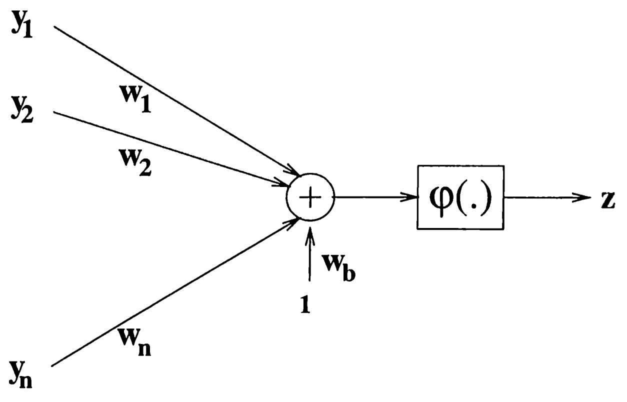


Figure 2.1: An n-input neuron.

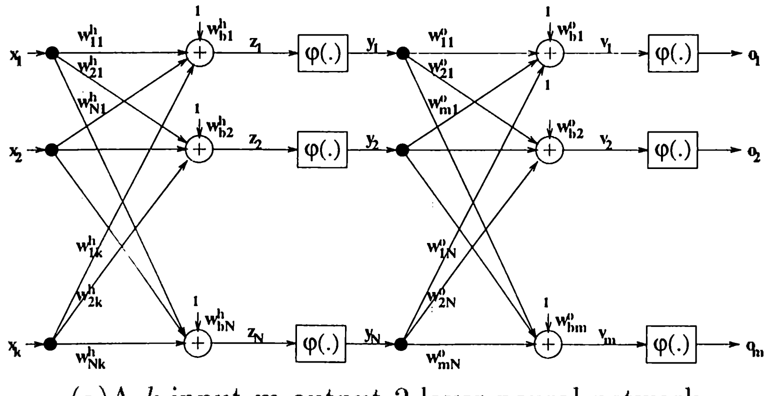
the outputs of the neural network is called the output layer. Other layers are called hidden layers. The multilayer neural networks used in this thesis are 2-layer and 3-layer neural networks. k-input-m-output 2-layer and 3-layer neural networks are pictured in Figure 2.2.

2.2 Training and Back Propagation

Design of a useful neural network can be realized in two stages. In the first stage, a suitable network structure is chosen, and in the second, the weights in the structure are adjusted so that the outputs are close enough to the desired values.

In general, direct selection of the weights to fulfil an arbitrary task is not feasible. In such cases, random initial values are assigned to the weights, and adjustment of the weights is done by means of a process of stimulation by the environment in which the neural network is embedded, which is called training.

Training processes can be divided into two groups as supervised training and unsupervised training. In supervised training, weights are adjusted according to the error measured as the distance between the actual output $\mathbf{o}$ and the desired output $\mathbf{d}$. In unsupervised training, the desired outputs are not known, and learning must be accomplished based on observations of responses to inputs about which we have marginal or no knowledge [9, page 57]. In this thesis, we


(a)A k-input-m-output 2-layer neural network

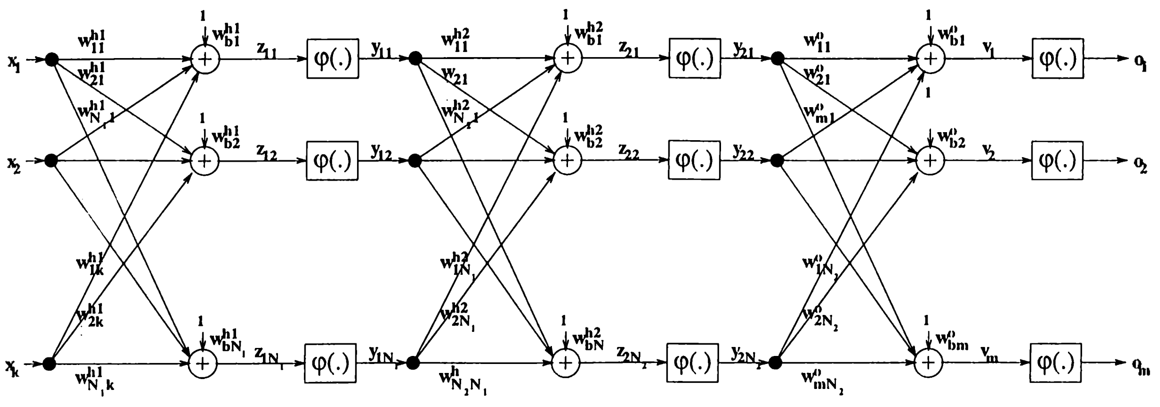

(b)A k-input-m-output 3-layer neural network

Figure 2.2: Multilayer neural networks used in the thesis.

have used supervised training processes. The distance used in these processes is the Euclidean distance, and the error function E for a desired output - actual output pair $\mathbf{d}$ - $\mathbf{o}$ is defined as:

$$E = \frac{1}{2} \|\mathbf{d} - \mathbf{o}\|^2 \tag{2.2}$$

One of the mostly used supervised training techniques is *Back Propagation.* Back Propagation is a gradient descent based algorithm, in which the weights are changed in the opposite direction of the gradient of the error function in the weight space. In the basic Back Propagation, the update for an arbitrary weight w in the neural network at step n of training process is given by

$$\Delta w(n) = -\eta \frac{\partial E(n)}{\partial w} \tag{2.3}$$

where η is a positive constant to be determined by the designer called the *learning constant.* In the thesis, we have used Back Propagation with Momentum, in which the update in the previous step is used to speed up the convergence of the weights:

$$\Delta w(n) = -\eta \frac{\partial E(n)}{\partial w} + \alpha \Delta w(n-1) \tag{2.4}$$

where $0 < \alpha < 1$ is the momentum constant to be determined. The explicit equations of Back Propagation with Momentum for the neural networks in Figure 2.2 are given in Appendix A.

In a training process using Back Propagation, firstly, a training set of inputs is determined and desired outputs for these outputs are computed. Secondly, initial weights of the neural network and parameters of the Back Propagation Algorithm are set. Initial weights are generally selected as random numbers. Then, Back Propagation is applied for each element in the training set. One run is called an *epoch.* After each epoch, the average error for the set is computed. The algorithm is run for a sufficient number of epoches, till the average error falls below a predefined value ϵ.

2.3 Applications of Neural Networks in Robotics and Control

In robotics and control problems, the final aim is generally to implement a highly nonlinear task using linear devices and devices with nonlinearities of certain types. Since dealing with such highly nonlinear problems directly is too difficult and time consuming, different techniques of approximation are developed to simplify these problems.

Until the last four decades, the usual attempt to such a nonlinear problem had been to determine some critical operating points, to linearize the equations of the problem around these points and to use the known linear system techniques. Since this attempt may not always be feasible or may be too costly, different approaches are needed.

Based on this need, researchers tried to adapt the perfect control structures and learning strategies of animals and humans to highly nonlinear systems, and usage of neural networks in control and robotics became popular in recent decades[2, 5, 10, 11, 12, 13].

In the literature of the last three decades, many examples of applications of neural networks in function approximation, optimization, system identification, adaptive control, associative memory design, pattern classification, pattern recognition, sensor-based robotics, kinematic control of robots, path planning, sensing-motion coordination can be seen[2, 4, 5, 10, 11, 13, 14, 15, 16, 17].

2.4 Multi-stage Neural Networks

A *multi-stage neural network* is a large neural network composed of a number of cascaded neural networks. Such a structure is useful if the task of the neural network to be designed can be partitioned into separate simpler subtasks. In such a case, complementary neural networks are designed for these subtasks and are combined as a single network [18, 19].

The multi-stage structure is also beneficial for control problems in which the controller has to drive the plant for a number steps to achieve the task, and only the desired output for the final step is available. In [6] and [7], Nguyen and Widrow show how the multi-stage structure can be used for controlling nonlinear dynamic systems, and apply this approach to the *Truck Backer-Upper* Problem.

A finite-dimensional discrete-time controller can be represented as in Figure 2.3. Here $\mathbf{z}_k$ and $\mathbf{z}_{k+1}$ denote the current and next states of the plant respectively, and $\mathbf{u}_k$ is the control signal given to the plant by the controller. Plant and controller equations for this scheme are as follows:

$$\mathbf{z}_{k+1} = D(\mathbf{z}_k, \mathbf{u}_k) \tag{2.5}$$

$$\mathbf{u}_k = C(\mathbf{z}_k) \tag{2.6}$$

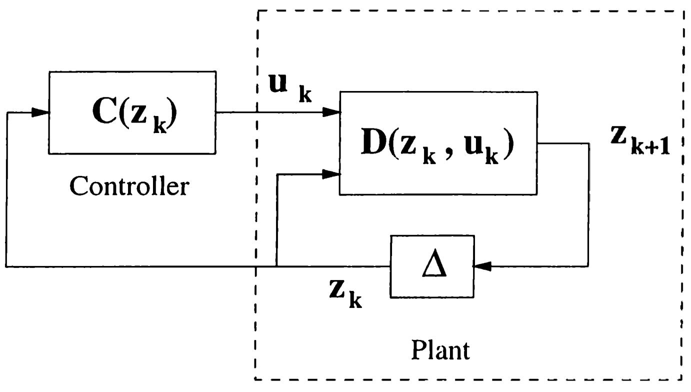


Figure 2.3: A finite-dimensional discrete-time system with a controller.

A common problem for such a scheme is to drive the plant from an initial state $\mathbf{z}_0$ to a desired state $\mathbf{z}_d$ in an unknown number of steps. In the Truck Backer-Upper example, the aim is to design a controller to back a trailer truck to a loading dock by only backing (i.e., going forward is not allowed) starting at an arbitrary initial position. The speed of the truck is assumed to be constant, and only the instantaneous steering angles of the truck determine the backing-path. If we discretize the problem, that is if the truck is moved for a number of constant-angle steps, the discrete-time controller is required to give a sequence of steering signals during the process. For this problem, $\mathbf{z}_k$ and $\mathbf{z}_{k+1}$ denote the

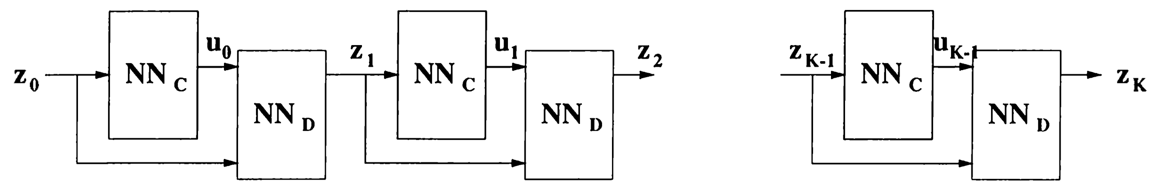


Figure 2.4: K-stage neural network representation of a K-step process.

position and the head direction of the truck-trailer in the current and the next step, $\mathbf{u}_k$ is the steering signal given by the controller, and $\mathbf{z}_d$ is the position vector of the parking dock augmented with a possible direction parameter denoting that the rear of the trailer is parallel to the dock.

In the multi-stage approach to design a neural network controller for this problem, the plant block can be approximated by an emulator neural network NN_D, and if we denote the neural network controller by NN_C, a K-step process can be represented by a K-stage neural network composed of K cascaded identical NN_C-NN_D pairs as in Figure 2.4.

Here, the actual aim is to adjust the weights of the controller neural network. The plant-emulator neural network is designed only to have a structure and an expression for the plant which can be easily used for training the controller. Instead of a neural network, any smooth function $D(.)$ emulating the plant can be used. A method to design an emulator neural network is given in [6].

For an initial point $\mathbf{z}_0$ in the training set, the system is run till a stopping event occurs. Assuming that the number of steps of the process is K, the final error $E = \frac{1}{2} \|\mathbf{z}_d - \mathbf{z}_K\|^2$ ($\mathbf{z}_d$ is the pre-determined desired output for $\mathbf{z}_0$) can be back propagated through the plant emulator to find an equivalent error for the controller in the Kth stage. This equivalent error can be used to apply Back Propagation to the controller neural network in the Kth stage. The equivalent error can be back-propagated through all stages similarly to apply the Back Propagation Algorithm in all K stages, and weight updates for each stage can be determined. Weight update for one run can be found as the sum of the individual weight updates for the K stages.

This method is proposed by Nguyen and Widrow in [6], but the equivalent error in the method is not well-defined and there is no explicit formulation of the back-propagation through K-stages. In Chapter 5, we will describe and formulate a slightly different approach to apply Back Propagation in multi-stage neural networks with identical stages.

Chapter 3

MOTION PLANNING OF A MECHANICAL SNAKE

3.1 Introduction

In robotic designs, designers are generally inspired by an animal that has the characteristics which are required to exist on the device to be designed. Snake, with its quick and calm motion ability in cluttered or tight areas, is such an animal. We will call a robot with a structure and a motion pattern similar to those of a snake a *mechanical snake*.

Such a robot will be useful when it is necessary to crawl into places which are too dangerous or too narrow for people to enter. Nuclear power plants with narrow vessels, disaster areas, military surveillance tasks are examples for these situations [20].

The basic design of a mechanical snake can be done in two steps, design of its structure and design of its motion pattern. There are various studies on the design of flexible-articulated structures. These structures are designed either to obtain a more efficient robot arm [21] or to realize the idea of producing snake-like mobile robots [22, 20].

Motion patterns of biological snakes can be divided into four groups as *lateral undulation, rectilinear locomotion, sidewinding,* and *concertina motion.* In lateral undulation, the snake undulates laterally by bending the forward part of its body to establish a wavelike muscular contraction traveling down the snake's trunk. The wavelike contractions cause the snake's body to exert lateral forces on irregularities on the path, such as small elevations and depressions, pebbles, etc. The snake in effect pushes itself off from such points to go forward. During the motion, the snake sets its body in patterns of loops according to the irregularities, with the outside of each loop forming a contact point, and these patterns constitute the S-shaped path to be followed.

Rectilinear locomotion involves the application of force somewhat downward instead of laterally. In this motion pattern, the snake fixes several series of scutes and moves the skin between them in each step. As the snake's body moves forward the skin is stretched, pulling the forward scutes of each series out of contact with the ground, while additional scutes are continuously pulled up to the rear edge of the series.

In sidewinding, the snake achieves strong contact by moving so that its body lies almost at right angles to the direction of its travel. Track of a sidewinding snake is a series of straight parallel line segments, each of which is inclined about 60° to the direction of motion, with length nearly the same as the length of the snake. The snake starts to move forward by lifting its front quarter off the ground. Later, the head arches downward as the lifted part remains off the ground; in making the first contact, the neck bends at the next track of the sequence. Successive sections of the remaining part of the snake follow along the new track which is parallel to the preceding one.

In concertina motion, the snake draws itself into an S-shaped curve and sets the curved portion of its body in static contact with the ground. Movement begins when the front part of the snake is extended by forces transmitted to the ground in the zone which remains in stationary contact. After the front part moves forward a short distance, it stops. After establishing a new zone of stationary contact in this new position, the rear end of the snake is pulled forward. In this motion pattern, the necessary force is produced by only contacting to the ground. Irregularities to exert force laterally are not needed as in the case of lateral undulation. Detailed explanations of the motion patterns of biological snakes are given in [23].

Although lateral undulation is the most common one for snakes, concertina motion seems to be the most suitable method of motion to emulate. The last property of concertina motion makes the mechanical implementation of this motion pattern easier. This fact and its suitability for progressing in narrow areas make the concertina motion appropriate to be mimicked mechanically, and we will try to mimic this motion pattern, like [20]. The mechanical snake will proceed by fixing some of its links and drawing S-shapes with free link sequences or straightening them successively.

The structure and motion pattern of the mechanical snake that we will work on will be very similar to those in [20]. The mechanical snake designed by Shan and Koren in [20] is composed of seven links which are connected by active joints allowing motion on a horizontal plane. Control of the joint angles between arms are done by DC motors, and linear solenoids with sharp tip pins are used to provide static points with the surface.

3.2 Definition of the Problem

In our work, we will deal with a similar structure except that our structure is composed of eight links of the same length and for each link, a link-solenoid is assumed to be able to fix the entire link to the surface. The last assumption can be realized by using a solenoid pair for each link. For motion planning purposes, the structure can be simplified as in Figure 3.1. Each segment is represented by a line segment of length L. Joints and segments are ordered beginning from the head so that Joint 0 is the head joint, Joint 8 is the tail joint, Segment 1 is the head segment and Segment 8 is the tail segment. Joint i is denoted by J_i for $i = 0, 1, ..., 8$. Let us denote the ray $\overrightarrow{J_{i-1}J_i}$ by L_i for $i = 1, ..., 8$. Joint angles $\theta_0, ..., \theta_7$ are defined as follows: θ_0 is the directional counterclockwise (CCW) angle from x-axis to L_1 and θ_i is the directional (CCW) angle from L_i to L_{i+1} for $i = 1, 2, ..., 7$. Similar to that in [20], $t_i(\in \{0, 1\})$ will denote the state of the solenoid of Segment i for $i = 1, ..., 8$, where 0 denotes that the segment is free and 1 denotes that it is fixed by its solenoid.

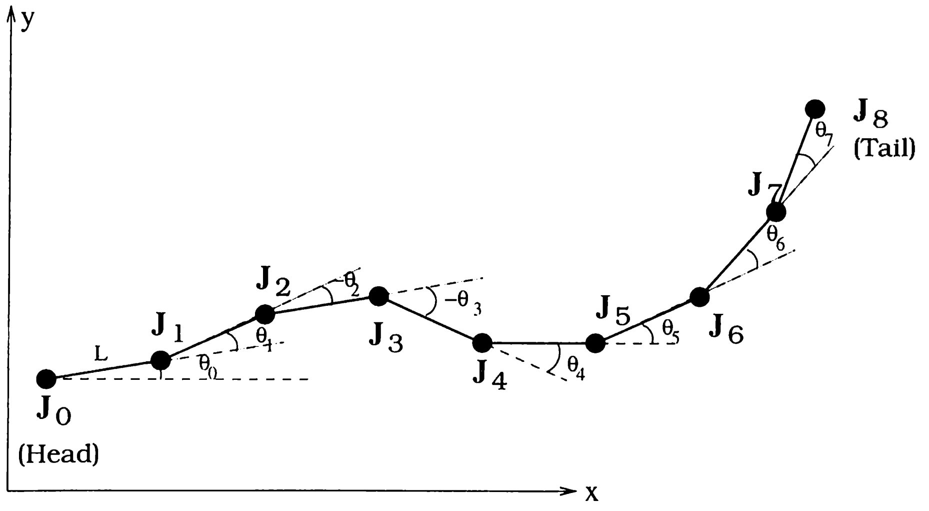


Figure 3.1: Simplified structure of the mechanical snake.

To avoid possible needs to apply large amounts of torque to swing big portions of the mechanical snake and to avoid a jack-knifed situation, in which the snake is bent over an intermediate joint and motion control is very difficult, absolute values of the interior joint angles θ_2, ..., θ_6 are limited to 60° with a tolerance of 5%.

Based on the structure and the angle-constraint given above, the basic linear motion pattern is as follows:

Step	t_1	t_2,t_3,t_4	t_5,t_6,t_7	t_8	θ_1	θ_2,θ_3	θ_4	θ_5,θ_6	θ_7
1	0	0	1	1	0°	0°	0°	0°	0°
2	1	1	0	0	0°	0°	−60°	60°	−60°
3	1	0	0	1	−60°	60°	−60°	0°	0°

Table 3.1: Segment solenoids and joint angles during a linear motion cycle.

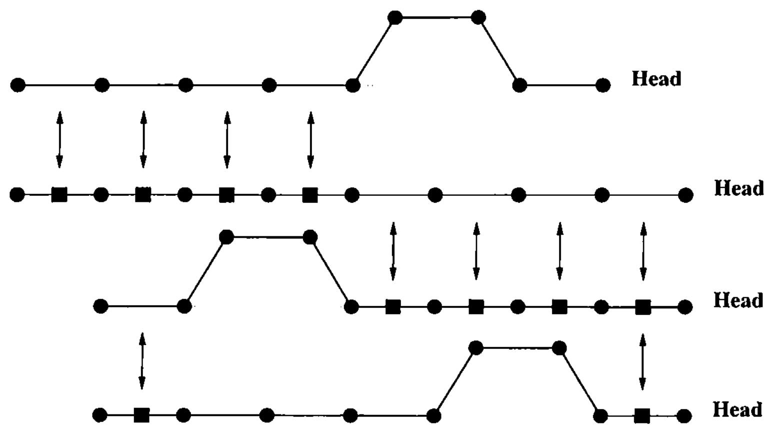


Figure 3.2: Basic linear motion pattern.

This pattern can be easily generalized for an arbitrary motion which does not violate the constraints. For a general motion pattern, the solenoids which are activated in a specific step will be the same as those activated in the same step of the basic linear motion pattern. Joint angle changes will slightly differ to move on nonlinear paths. To simplify the general motion pattern, we will adapt a convention that the sequence formed by the three links which are not on the path tracked (e.g. the line tracked for the basic linear motion pattern) is symmetric, that is, the angles of the two joints which are not found on the path are the same.

Using the structure and the motion pattern above, our problem will be to get the mechanical snake out of a rectangular region through a specified exit. The initial configuration of the snake can be arbitrary, but it is assumed that the initial configuration is suitable to follow a smooth path to get out of the region without crashing on the borders.

We specify the problem in order to have an explicit task to deal with:

Problem 3.1 *Let the length of a segment of the mechanical snake be normalized to 1. Let the rectangular region of interest, R, be the region having the corner coordinates (-1, -10), (-1, 10), (20, -10) and (20, 10), and let the exit be an interval between points (-1, -2) and (-1, 2). Let the mechanical snake initially be inside the square R_i with corners (0, -10), (0, 10), (20, -10) and (20, 10). The aim is to take the snake out of R using the specified exit. The absolute values of the angles θ_2, ..., θ_6 should never exceed* $63°$.

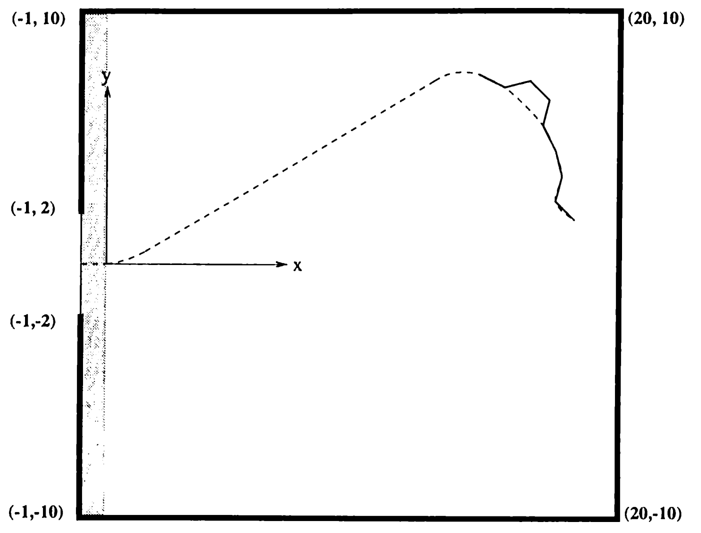


Figure 3.3: Problem 3.1.

This task can be done in two steps: Generation of the path to get out, and tracking of the generated path by the mechanical snake. A separate neural network will be designed to achieve each subtask.

3.3 Generation of Path

To have an unobtrusive motion, and to take steps significant in size, the path should be a smooth path with no sharp turns, that is which has a suitable maximum curvature κ_{max}.

To determine a specific value for the maximum curvature, we will use the angle constraint and two other constraints which make significance of steps explicit. To make the steps of the motion pattern significant, we impose that the distance between J_i and J_{i+3} is at least 2.8 for a sequence J_i, J_{i+1}, J_{i+2}, J_{i+3} of joints all of which are on the path to be tracked, and that the distance between J_i and J_{i+3} is at most 2.2 for a sequence J_i, J_{i+1}, J_{i+2}, J_{i+3} in which J_{i+1}, J_{i+2} are outside the path to be tracked.

Using the three constraints, the maximum curvature is determined as $\kappa_{max} = 0.447$ (the minimum radius of curvature is $R_{min} = \frac{1}{\kappa_{max}} = 2.236$).

Derivation of the maximum curvature from the constraints is given in Appendix C. Note that further algorithms and neural network structures are independent of this specific value of κ_{max}, and the algorithms can be run and the neural networks can be trained for different minimum radii of curvature by only setting R_{min} to the specific value of the minimum radius of curvature.

Suppose that the initial structure is adjusted so that the snake can be represented by a sequence of seven pieces of curves, each of which is either a line segment of length 1 or an arc of radius R_{min} and chord length 1 (A neural network to make this adjustment for an arbitrary initial condition will be introduced later). For such a configuration, seven of the joints are at the ends of these curve pieces. Let us order the pieces as Piece 1, ..., Piece 7 from head to tail, and denote the end points of these pieces as V_0, ..., V_7 in the same order. Let P_i denote both the ray $\overrightarrow{V_{i-1}V_i}$ and the type of Piece i for $i = 1, ..., 7$, where the piece-type definitions are given in Figure 3.4(a). The directional (CCW) angle from x-axis to the ray which is tangent to Piece 1 at V_0 and which follows the direction of Piece 1 is denoted by ϕ_0, and the directional (CCW) angle from P_i to P_{i+1} is denoted by ϕ_i for $i = 1, ..., 6$. Note that the convention used in defining ϕ_0 is different from the convention used in defining ϕ_i for $i = 1, ..., 6$, and θ_i for $i = 0, ..., 7$. Using this notation, a configuration is represented by 10 parameters, position and direction of the head, and types of the seven pieces $(x_{J_0}, y_{J_0}, \theta_0, P_1, ..., P_7)$. A sample initial configuration, which is represented by $(x, y, 180°, -1, -1, 1, -1, 0, 1, 1)$, is given in Figure 3.4(b).

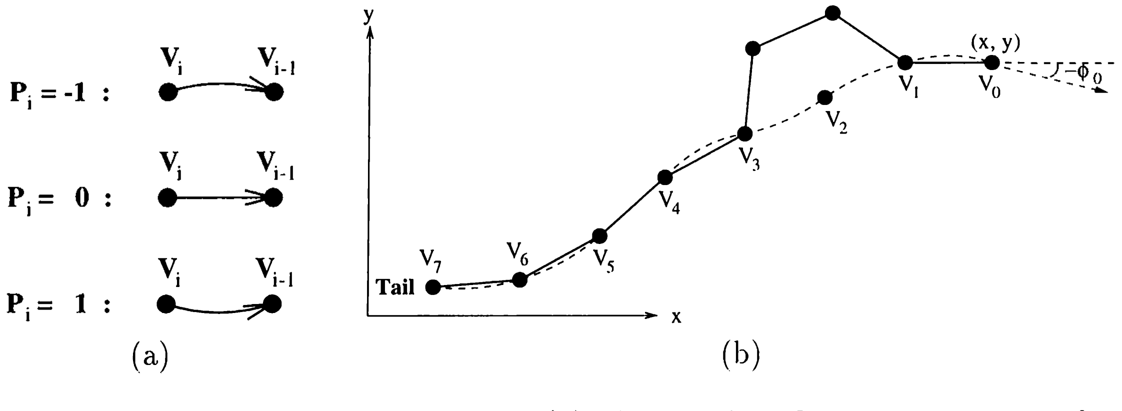


Figure 3.4: (a) Piece-type definitions. (b) The initial configuration represented by $(x, y, 180°, -1, -1, 1, -1, 0, 1, 1)$.

In the path generation task, our aim will be simplified to find a path in

R_i ending at the origin with a direction perpendicular to the exit interval. After that point, linear motion will be sufficient to take the snake out. In the realization of the task, the path generated will not exactly end at the origin with the specified direction, but these values will be close enough to the desired ones to exit from the specified interval.

So, the path generation problem can be formulated as finding a path $X(t) = (x(t), y(t))$ in the right half plane such that $X(0) = X_0$, $\phi(0) = \phi_0$, $X(t_f) = 0$ and $\phi(t_f) = 180°$ for some t_f, and $\kappa(t) \leq \kappa_{max}$, $\forall t \in [0,\ t_f]$. Here, $\phi(t) = \text{atan2}(\dot{y}(t), \dot{x}(t))$, where atan2(.,.) is the signed arctan function, defined same as the C^{++} command `atan2`, and $\kappa(t')$ is the curvature of the path $X(t)$ at $t = t'$.

This problem will be approached in two different ways, and two different neural networks will be designed for path generation in the next two chapters. Both of the paths generated using these two approaches will be compositions of the three arc types defined above, with smooth connections. Path generator neural network can be represented by the block given in Figure 3.5, for both approaches.

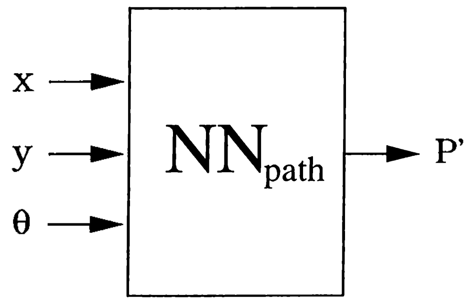


Figure 3.5: Path generator neural network.

The inputs of NN_{path} in the complete neural network will be $x = x_{J_0}$, $y = y_{J_0}$, $\theta = \phi_0$, and the output P' will be type of the next curve piece to be tracked, the P_1 for the next step.

3.4 Structure of the Complete Controller

The algorithm that will be realized by the complete controller to achieve the specified task is summarized as follows:

Algorithm 3.1 *Given an arbitrary configuration of the snake at an arbitrary location, represented by the (sensed) variables* x_{J_0}, y_{J_0}, θ_0, θ_1, ..., θ_7:

1. *Adjust the initial configuration* (x_{J_0}, y_{J_0}, θ_0, θ_1, ..., θ_7) *so that a seven pieces of curves representation* $(x_{J_0}, y_{J_0}, \theta_0, P_1, ..., P_7)$ *is available.*
2. *While* $x_{J_0} > 0$:
   - (a) *Using* θ_0 *and* P_1, *obtain* ϕ_0.
   - (b) *Apply* x_{J_0}, y_{J_0} *and* ϕ_0 *to* NN_{path} *as inputs to get the type of the next piece,* P', *as output.*
   - (c) *Using* θ_1, θ_2, ..., θ_7, P_1, P_2, ..., P_7, P', *get the joint angle increments to be given in the three steps of the motion cycle.*
   - (d) *Update the curve piece types as* $P_{1_{new}} = P'$, $P_{2_{new}} = P_1$, ..., $P_{7_{new}} = P_6$ *for the next step.*
   - (e) *Realize the 3-step motion cycle using the joint angle increments obtained in Step 2(c).*
   - (f) *Get new values of* x_{J_0}, y_{J_0}, θ_0, θ_1, ..., θ_7 *using sensors.*
3. *While* $x_{J_0} > -1$:
   - (a) *Assign the type of the next piece to be tracked as* $P' = 0$.
   - (b) *Apply steps 2(d), 2(e) and 2(f).*
4. *Stop.*

The task is accepted to be finished, when the head of the snake is departed from the region R. Consequently, a path consisting of the same three types of curve pieces which is suitable for the environment can be generated for controlling the motion outside the region.

A neural network to adjust the initial configuration will be directly designed in the last section of this chapter. The complete neural network controller scheme for the adjusted initial configuration is given in Figure 3.6. Vector variables in this scheme are defined as follows:

$$
\begin{aligned}
S &= [x_{J_0},\ y_{J_0},\ \theta_0,\ \theta_1,\ \dots,\ \theta_7]^T && (3.1)\\
S_J &= [\theta_1,\ \theta_2,\ \dots,\ \theta_7]^T && (3.2)\\
P &= [P_1,\ P_2,\ \dots,\ P_7]^T && (3.3)
\end{aligned}
$$

ΔS_{J1}, ΔS_{J2} and ΔS_{J3} are the increment vectors to be added to S_J in Steps 1, 2 and 3 of a motion cycle.

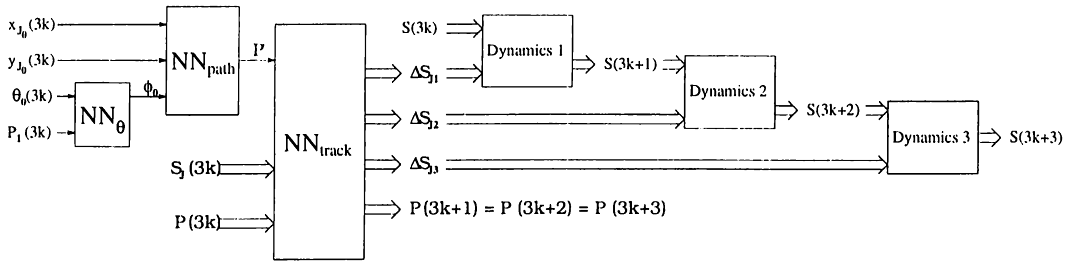


Figure 3.6: Complete neural network controller scheme for the adjusted initial configuration.

The blocks Dynamics 1, Dynamics 2 and Dynamics 3 simply realize the Steps 1, 2 and 3 of a motion cycle. That is, Dynamics i activates the solenoids for Step i according to Table 3.1 and forces the joint motors to apply the increment ΔS_{Ji}, for $i = 1, 2, 3$. The neural network blocks NN_{track} and NN_θ will be described in the next section. They are designed to implement Steps 2(a), 2(c) and 2(d) of Algorithm 3.1.

3.5 Tracking the Generated Path

The neural network block NN_{track} is designed to get the joint angle increments to be given in the three steps of the motion cycle, using θ_1, θ_2, ..., θ_7, P_1, P_2, ..., P_7, P', and to update the curve piece types for the next cycle.

We will use the two facts expressed below to design this block. Proofs of these facts will be given in Appendix B.

Fact 3.1 *In any configuration, value of ϕ_i is given by*

$$\phi_i = -\beta(P_i + P_{i+1}) \tag{3.4}$$

for $i = 1, ..., 6$, where the constant β is defined as

$$\beta = \arcsin\left(\frac{1}{2\, R_{min}}\right) \tag{3.5}$$

Proof: See Appendix B. □

Corollary 3.1 *For a configuration where Segments i and $i+1$ lie on (have the same end points with) Pieces k and $k+1$, the joint angle θ_i is given by*

$$\theta_i = -\beta(P_k + P_{k+1}) \tag{3.6}$$

where β is defined by Equation 3.5.

Proof: For such a configuration, J_{i-1}, J_i, J_{i+1}, θ_i coincide with V_{k-1}, V_k, V_{k+1}, ϕ_k respectively, and the result immediately follows. □

Fact 3.2 *For a configuration where J_{i-1}, J_i, J_{i+3} and J_{i+4} coincide with V_{k-1}, V_k, V_{k+2}, V_{k+3} respectively, the joint angles θ_i, θ_{i+1}, θ_{i+2}, θ_{i+3} are given by either*

$$\theta_{i+1} = \theta_{i+2} = \alpha_{|P_{k+1}+P_{k+2}|} \quad (3.7)$$

$$\theta_i = -\beta(P_k + \frac{3}{2}P_{k+1} + \frac{1}{2}P_{k+2}) - \alpha_{|P_{k+1}+P_{k+2}|} \quad (3.8)$$

$$\theta_{i+3} = -\beta(\frac{1}{2}P_{k+1} + \frac{3}{2}P_{k+2} + P_{k+3}) - \alpha_{|P_{k+1}+P_{k+2}|} \quad (3.9)$$

or

$$\theta_{i+1} = \theta_{i+2} = -\alpha_{|P_{k+1}+P_{k+2}|} \quad (3.10)$$

$$\theta_i = -\beta(P_k + \frac{3}{2}P_{k+1} + \frac{1}{2}P_{k+2}) + \alpha_{|P_{k+1}+P_{k+2}|} \quad (3.11)$$

$$\theta_{i+3} = -\beta(\frac{1}{2}P_{k+1} + \frac{3}{2}P_{k+2} + P_{k+3}) + \alpha_{|P_{k+1}+P_{k+2}|} \quad (3.12)$$

where β is defined by Equation 3.5 and α_j is defined as

$$\alpha_j = \arccos\left[\cos\left(j\frac{\beta}{2}\right) - \frac{1}{2}\right] \quad (3.13)$$

Proof: See Appendix B. □

The choice among the two candidates given in Fact 3.2 will be based on the joint angle constraint. To guarantee that the absolute value of the interior one of θ_i and θ_{i+1} is below 63°, the choice in which the sign in front of the α-term is opposite to the sign of the rest of the expression for that angle will be selected.

Based on this selection rule in addition to Fact 1 and Fact 2, the values of joint angles for a cycle with NN_{track}-inputs θ_1^{old}, θ_2^{old}, ..., θ_7^{old}, P_1, P_2, ...,P_7, P', will be as follows:

Joint Angle	Initial	After Step 1	After Step 2	After Step 3
θ_1	θ_1^{old}	Q_1	Q_1	$Q_1 + Q_2/2 - s_2\alpha_{k_2}$
θ_2	θ_2^{old}	Q_2	Q_2	$s_2\alpha_{k_2}$
θ_3	θ_3^{old}	Q_3	Q_3	$s_2\alpha_{k_2}$
θ_4	θ_4^{old}	Q_4	$Q_4 + Q_5/2 - s_1\alpha_{k_1}$	$Q_3 + Q_2/2 - s_2\alpha_{k_2}$
θ_5	θ_5^{old}	θ_5^{old}	$s_1\alpha_{k_1}$	Q_4
θ_6	θ_6^{old}	θ_6^{old}	$s_1\alpha_{k_1}$	Q_5
θ_7	θ_7^{old}	θ_7^{old}	$Q_6 + Q_5/2 - s_1\alpha_{k_1}$	Q_6

Table 3.2: Joint angles during a general motion cycle.

Here Q_1, ..., Q_6, s_1, s_2, k_1 and k_2 are given by

$$\begin{aligned}
Q_1 &= -\beta(P' + P_1) && (3.14)\\
Q_i &= -\beta(P_{i-1} + P_i) \quad for\ i = 2,3,4,5,6 && (3.15)\\
s_1 &= \text{sign}(Q_4 \ + \ Q_5/2) && (3.16)\\
k_1 &= |Q_5/\beta| && (3.17)\\
s_2 &= \text{sign}(Q_3 \ + \ Q_2/2) && (3.18)\\
k_2 &= |Q_2/\beta| && (3.19)
\end{aligned}$$

where sign(.) denotes the function of a bipolar hardlimiter defined as:

$$\text{sign}(x) = \begin{cases} -1\ , & x < 0 \\ 1\ , & x \geq 0 \end{cases} \tag{3.20}$$

So the joint angle increments to be applied during a cycle are obtained as given in Table 3.3.

Having the table for angle increments, we can now design the NN_{track} circuit. Collecting Q_1, ..., Q_6 in vector Q and the new piece types P_{n1}, ..., P_{n7} in vector P_{new} as

Inc.	Step 1	Step 2	Step 3
$\Delta\theta_1$	$Q_1 - \theta_1^{old}$	0	$Q_2/2 - s_2\alpha_{k_2}$
$\Delta\theta_2$	$Q_2 - \theta_2^{old}$	0	$s_2\alpha_{k_2} - Q_2$
$\Delta\theta_3$	$Q_3 - \theta_3^{old}$	0	$s_2\alpha_{k_2} - Q_3$
$\Delta\theta_4$	$Q_4 - \theta_4^{old}$	$Q_5/2 - s_1\alpha_{k_1}$	$Q_2/2 + Q_3 - Q_4 - Q_5/2 + s_1\alpha_{k_1} - s_2\alpha_{k_2}$
$\Delta\theta_5$	0	$s_1\alpha_{k_1} - \theta_5^{old}$	$Q_4 - s_1\alpha_{k_1}$
$\Delta\theta_6$	0	$s_1\alpha_{k_1} - \theta_6^{old}$	$Q_5 - s_1\alpha_{k_1}$
$\Delta\theta_7$	0	$Q_6 + Q_5/2 - s_1\alpha_{k_1} - \theta_7^{old}$	$-Q_5/2 + s_1\alpha_{k_1}$

Table 3.3: Joint angle increments during a general motion cycle.

$$Q = [Q_1,\ Q_2,\ ...,\ Q_6]^T \tag{3.21}$$

$$P_{new} = [P_{n1},\ P_{n2},\ ...,\ P_{n7}]^T \tag{3.22}$$

NN_{track} can be divided into subblocks as pictured in Figure 3.7.

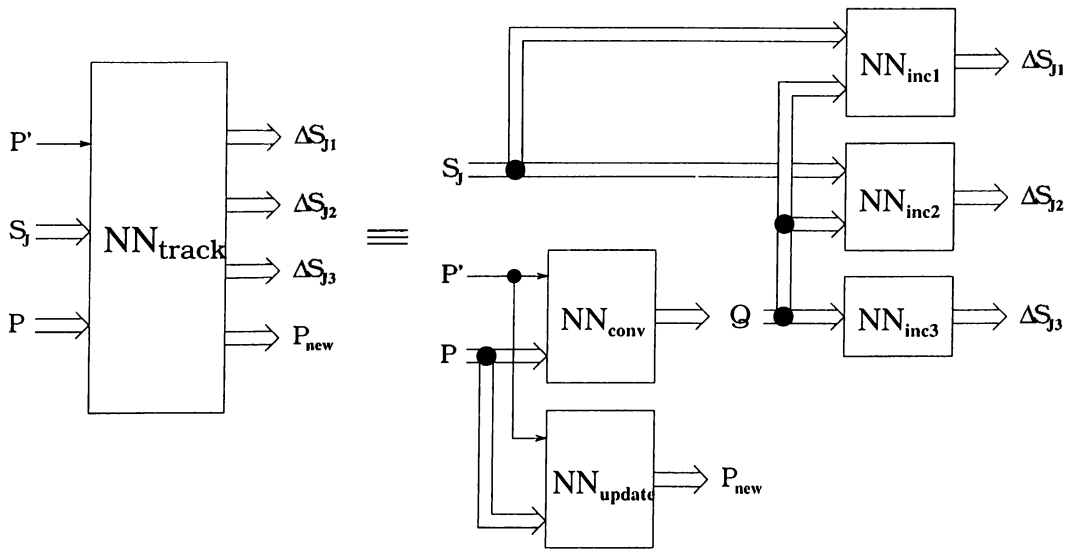


Figure 3.7: NN_{track}.

Here, NN_{conv} is used for deriving the vector Q from P and P'. NN_{update} updates the vector P for the next cycle. NN_{inc1}, NN_{inc2} and NN_{inc3} compute the increments to be given to the joint angles in Step 1, Step 2 and Step 3 of the current cycle respectively.

In the rest of this section, neural network structures of these subblocks and structure of NN_θ, which is used to obtain ϕ_0, will be given. The activation functions used in these blocks are unipolar and bipolar hardlimiters. Note that each one of these two types can be constructed as a simple neural network with a single activation function block of the other type. So one type of activation block is sufficient to construct the subblocks of NN_{track}.

NN_{update} and NN_{conv}:

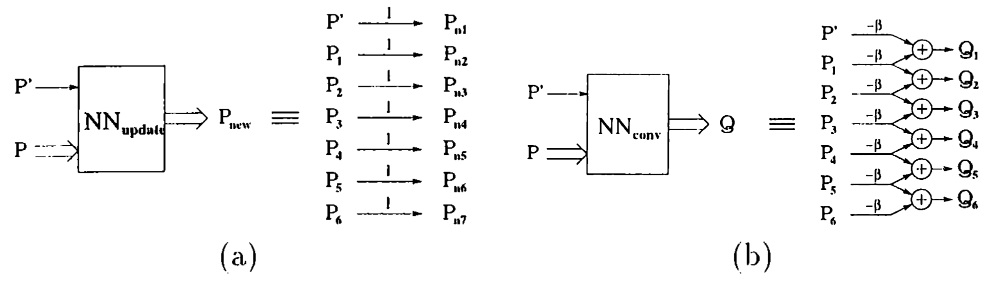


Figure 3.8: (a) NN_{update}. (b) NN_{conv}.

NN_{inc1}, NN_{inc2} and NN_{inc3}:

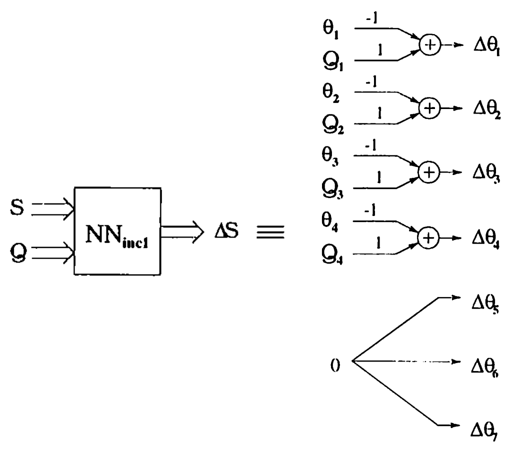
Let us define a function $\gamma(x,\ y)$ as

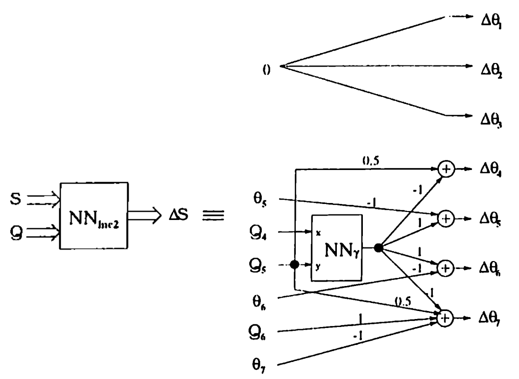
$$\gamma(x,\ y) \ = \ sign(x \ + \ y/2)\alpha_{|y/\beta|} \qquad (3.23)$$

where

$$\alpha_i \ = \ \arccos\left[\cos(\frac{i\beta}{2}) - \frac{1}{2}\right] \quad \text{for } i \ = \ 0,1,2 \qquad (3.24)$$

$\gamma(x,\ y)$ will be implemented by the neural network NN_γ, whose structure will be described later. The structures for NN_{inc1}, NN_{inc2} and NN_{inc3} including NN_γ are given in Figure 3.9.


(a)


(b)

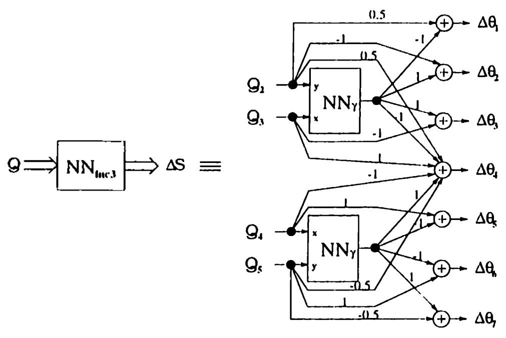

(c)

Figure 3.9: (a) NN_{inc1}. (b) NN_{inc2}. (c) NN_{inc3}.

NN_γ:

This block is designed as shown in Figure 3.10(a), where the block NN_α will be designed to implement the function

$$\alpha(a,\ b) \ = \ a\ \alpha_{|b|} \tag{3.25}$$

NN_α:

This block is designed as two subblocks, $NN_{\alpha 1}$ to implement

$$\alpha_1(a) = |a| \tag{3.26}$$

for $a \in \{-2, -1, 0, 1, 2\}$ and $NN_{\alpha 2}$ to implement

$$\alpha_2(a,\ b) = a\ \alpha_b \tag{3.27}$$

for $a \in \{-1, 1\}$ and $b \in \{0, 1, 2\}$. $\alpha_2(.)$ is tabulated in Table 3.4.

a	-1	-1	-1	1	1	1
b	0	1	2	0	1	2
o	$-\alpha_0$	$-\alpha_1$	$-\alpha_2$	α_0	α_1	α_2

Table 3.4: Possible input-output pairs for $NN_{\alpha 2}$.

$NN_{\alpha 1}$ and $NN_{\alpha 2}$ designed based on this data is given in Figure 3.10.

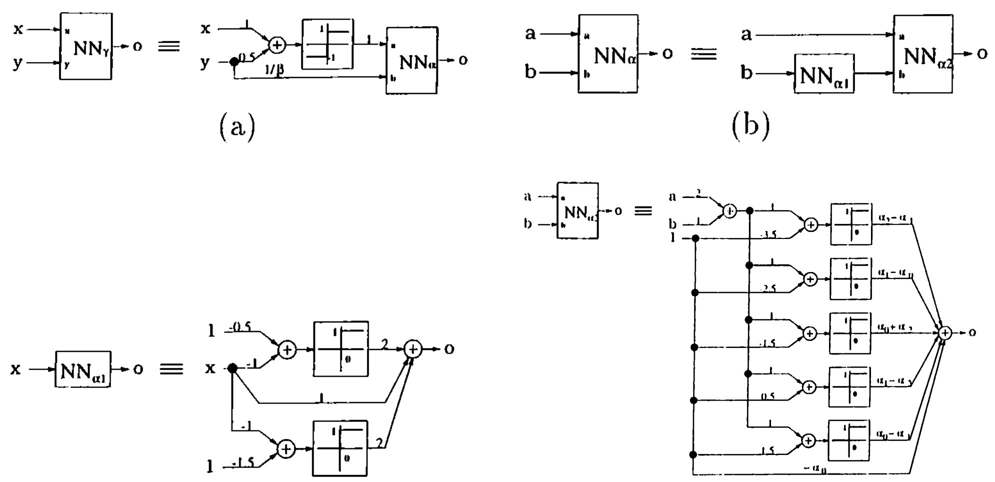


Figure 3.10: (a) NN_γ. (b) NN_α. (c) $NN_{\alpha 1}$. (d) $NN_{\alpha 2}$.

NN_θ:

This neural network is used to obtain ϕ_0 using θ_0 and P_1.

Thinking ϕ_0 as the direction of an extra curve piece of type 0, and using Fact 1, the difference between the angles ϕ_0 and θ_0 is found as:

$$\phi_0 - \theta_0 = \beta P_1 + \pi \tag{3.28}$$

Using this fact, the neural network is designed as in 3.11 where NN_{mod} is used to keep the output in the interval $[0,\ 2\pi)$.

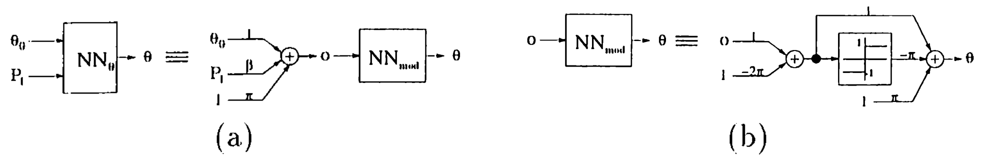


Figure 3.11: (a) NN_θ. (b) NN_{mod}.

3.6 Adjusting the Initial Configuration

Supposing that the initial configuration is sufficiently smooth with representative curve with maximum curvature close to κ_{max}, we developed an algorithm to adjust the configuration so that the representative curve is converted into a sequence of curve pieces of specified types, which is very close to the initial one. Adjustment will be done in one motion cycle:

Algorithm 3.2 *Given an arbitrary initial configuration of the mechanical snake with joint angles θ_1^{old}, ..., θ_7^{old}:*

1. $\Delta\theta_5 = \Delta\theta_6 = \Delta\theta_7 = P_2 = P_3 = 0$
2. If $\theta_4^{old} < -60° - \frac{\beta}{2}$, then $P_4 = 1$, $\Delta\theta_2 = 60° - \theta_2^{old}$, $\Delta\theta_3 = 60° - \theta_3^{old}$, $\Delta\theta_4 = -60° - \beta - \theta_4^{old}$;

else if $-60^\circ - \frac{\beta}{2} \leq \theta_4^{old} < -60^\circ + \frac{\beta}{2}$, then $P_4 = 0$, $\Delta\theta_2 = 60^\circ - \theta_2^{old}$, $\Delta\theta_3 = 60^\circ - \theta_3^{old}$, $\Delta\theta_4 = -60^\circ - \theta_4^{old}$;

else if $-60^\circ + \frac{\beta}{2} \leq \theta_4^{old} < 0^\circ$, then $P_4 = -1$, $\Delta\theta_2 = 60^\circ - \theta_2^{old}$, $\Delta\theta_3 = 60^\circ - \theta_3^{old}$, $\Delta\theta_4 = -60^\circ + \beta - \theta_4^{old}$;

else if $0^\circ \leq \theta_4^{old} < 60^\circ - \frac{\beta}{2}$, then $P_4 = 1$, $\Delta\theta_2 = -60^\circ - \theta_2^{old}$, $\Delta\theta_3 = -60^\circ - \theta_3^{old}$, $\Delta\theta_4 = 60^\circ - \beta - \theta_4^{old}$;

else if $60^\circ - \frac{\beta}{2} \leq \theta_4^{old} < 60^\circ + \frac{\beta}{2}$, then $P_4 = 0$, $\Delta\theta_2 = -60^\circ - \theta_2^{old}$, $\Delta\theta_3 = -60^\circ - \theta_3^{old}$, $\Delta\theta_4 = 60^\circ - \theta_4^{old}$;

else if $\theta_4^{old} \geq 60^\circ + \frac{\beta}{2}$, then $P_4 = -1$, $\Delta\theta_2 = -60^\circ - \theta_2^{old}$, $\Delta\theta_3 = -60^\circ - \theta_3^{old}$, $\Delta\theta_4 = 60^\circ + \beta - \theta_4^{old}$;

3. If $\theta_1^{old} < -60^\circ - \frac{\beta}{2}$, then $P_1 = 1$, $\Delta\theta_1 = -60^\circ - \beta - \theta_1^{old}$;
   else if $-60^\circ - \frac{\beta}{2} \leq \theta_1^{old} < -60^\circ + \frac{\beta}{2}$, then $P_1 = 0$, $\Delta\theta_1 = -60^\circ - \theta_1^{old}$;
   else if $-60^\circ + \frac{\beta}{2} \leq \theta_1^{old} < 0^\circ$, then $P_1 = -1$, $\Delta\theta_1 = -60^\circ + \beta - \theta_1^{old}$;
   else if $0^\circ \leq \theta_1^{old} < 60^\circ - \frac{\beta}{2}$, then $P_1 = 1$, $\Delta\theta_1 = 60^\circ - \beta - \theta_1^{old}$;
   else if $60^\circ - \frac{\beta}{2} \leq \theta_1^{old} < 60^\circ + \frac{\beta}{2}$, then $P_1 = 0$, $\Delta\theta_1 = 60^\circ - \theta_1^{old}$;
   else if $\theta_1^{old} \geq 60^\circ + \frac{\beta}{2}$, then $P_1 = -1$, $\Delta\theta_1 = 60^\circ + \beta - \theta_1^{old}$.

4. Apply Step 1 of the motion cycle.

5. $\Delta\theta_1 = \Delta\theta_2 = \Delta\theta_3 = \Delta\theta_4 = 0$

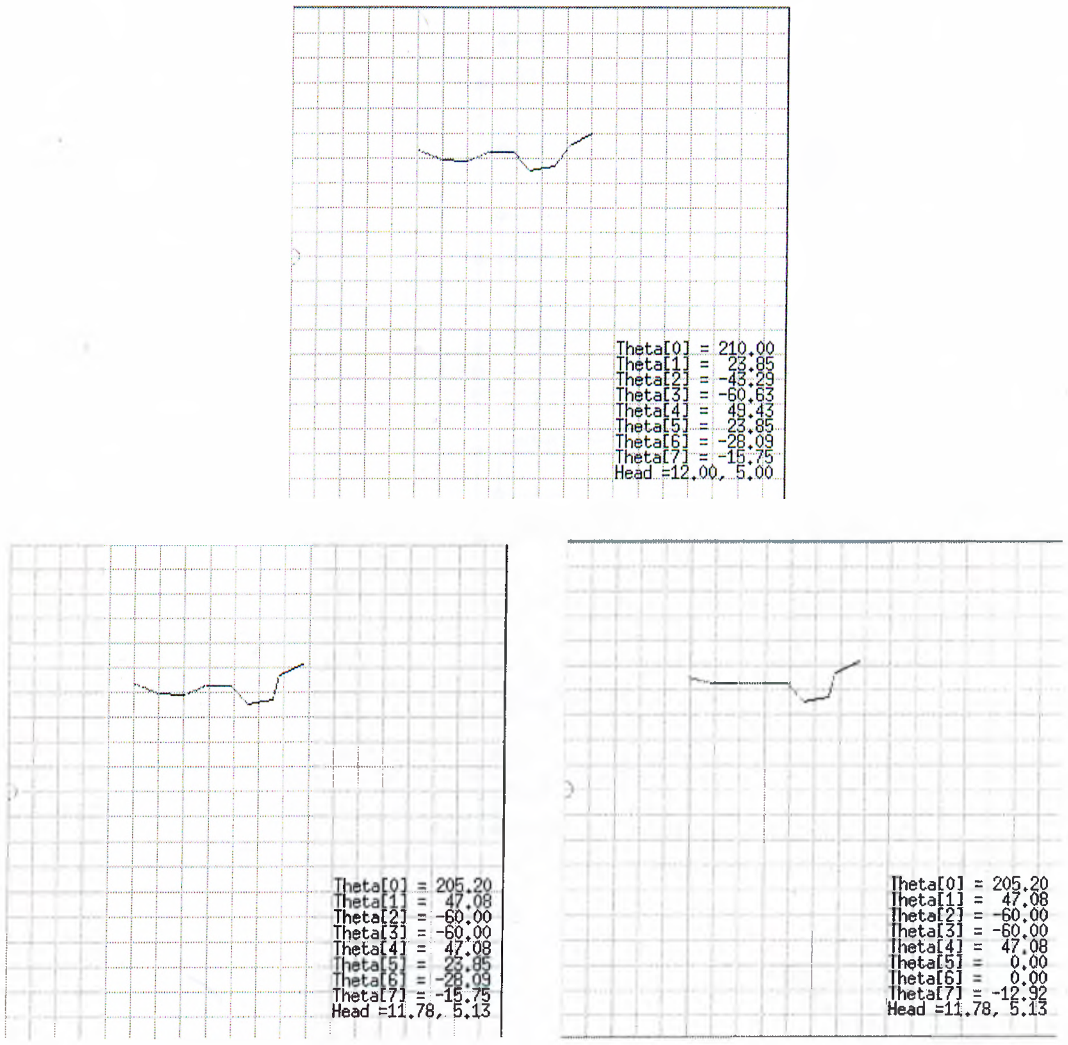
6. For $i = 2,3,4$:
   If $\theta_{i+3}^{old} < -\frac{\beta}{2} - P_{i+2}\beta$, then $P_{i+3} = 1$, $\Delta\theta_{i+3} = -(P_{i+2} + 1)\beta - \theta_{i+3}^{old}$;
   else if $-\frac{\beta}{2} - P_{i+2}\beta \leq \theta_{i+3}^{old} < \frac{\beta}{2} - P_{i+2}\beta$, then $P_{i+3} = 0$, $\Delta\theta_{i+3} = -P_{i+2}\beta - \theta_{i+3}^{old}$;
   else if $\theta_{i+3}^{old} \geq \frac{\beta}{2} - P_{i+2}\beta$, then $P_{i+3} = -1$, $\Delta\theta_{i+3} = -(P_{i+2} - 1)\beta - \theta_{i+3}^{old}$.

7. Apply Step 2 of the motion cycle.

8. $\Delta S_{J3} = 0$

9. Apply Step 3 of the motion cycle.

Based on this algorithm, the scheme for the adjustment of the initial configuration and the neural network structures used for adjustment are given in Figures 3.13 and 3.14. To illustrate the algorithm, a sample adjustment process controlled by these neural networks is pictured in Figure 3.12.

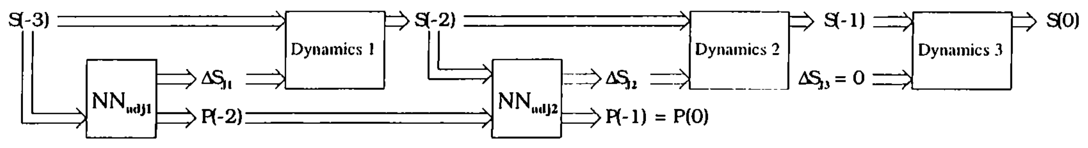


```
Theta[0] = 210.00
Theta[1] =  23.85
Theta[2] = -43.29
Theta[3] = -60.63
Theta[4] =  49.43
Theta[5] =  23.85
Theta[6] = -28.09
Theta[7] = -15.75
Head =12.00, 5.00
```

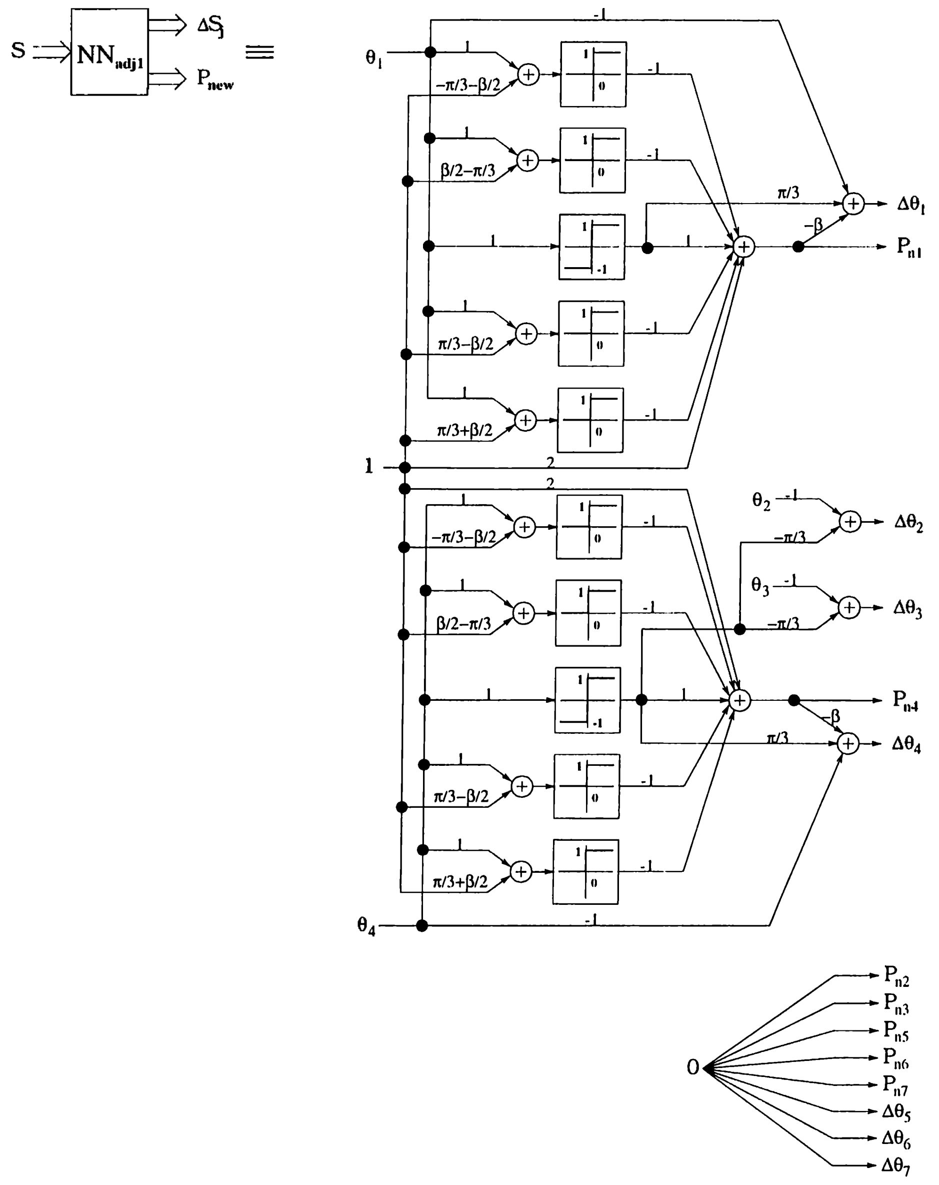
```
Theta[0] = 205.20
Theta[1] =  47.08
Theta[2] = -60.00
Theta[3] = -60.00
Theta[4] =  47.08
Theta[5] =  23.85
Theta[6] = -28.09
Theta[7] = -15.75
Head =11.78, 5.13
```

```
Theta[0] = 205.20
Theta[1] =  47.08
Theta[2] = -60.00
Theta[3] = -60.00
Theta[4] =  47.08
Theta[5] =   0.00
Theta[6] =   0.00
Theta[7] = -12.92
Head =11.78, 5.13
```

Figure 3.12: A sample adjustment process.



(a)



(b)

Figure 3.13: (a) The scheme for the adjustment of the initial configuration. (b) NN_{adj1}.

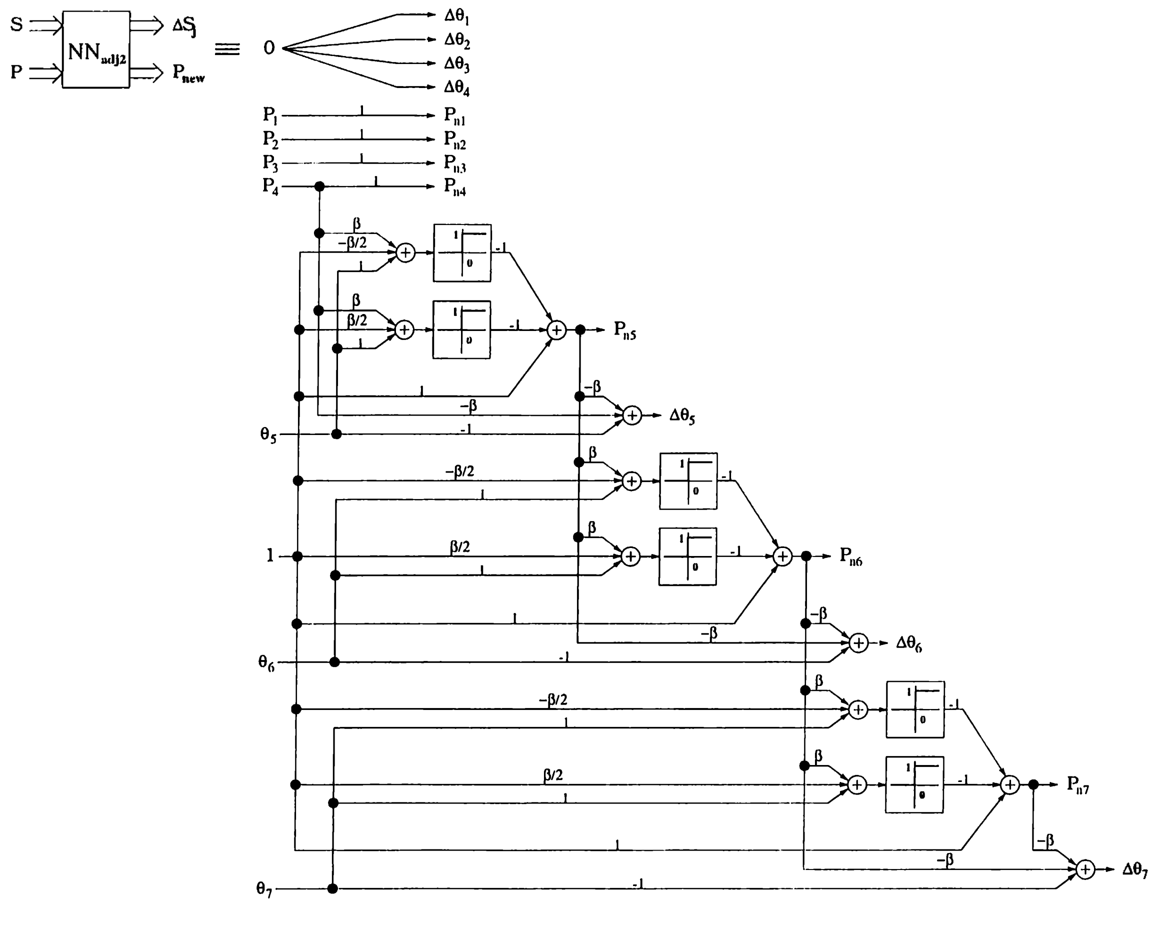


Figure 3.14: NN_{adj2}.

Chapter 4

MINIMAL PATH APPROACH

In the path generation task, our aim is to design a neural network to find a path in R_i ending at the origin with a direction perpendicular to the exit interval. In the realization of the task, the path generated will not exactly end at the origin with the specified direction, but these values will be close enough to the desired ones to exit from the specified interval. In Section 3.3, the path generation problem was formulated as finding a path $X(t) = (x(t), y(t))$ in the right half plane such that $X(0) = X_0$, $\phi(0) = \phi_0$, $X(t_f) = 0$ and $\phi(t_f) = 180°$ for some t_f, and $\kappa(t) \leq \kappa_{max}$, $\forall t \in [0,\ t_f]$.

The design in this chapter will be based on finding such a path which has the minimum possible length. First, an algorithm to find the minimum path will be given. Then, this algorithm will be used to train a multilayer neural network used for path generation.

4.1 Minimal Paths on the Right Half Plane

The general problem of finding minimal-length smooth paths with bounded curvature on a region in $\mathbf{R}^2$ with or without obstacles is attacked by many researchers. This problem can be formulated as follows [24]:

Problem 4.1 *Let $\Omega \subset \mathbf{R}^2$ be a closed set of polygons representing the obstacles, and let $\partial\Omega$ and $\mathrm{int}(\Omega)$ denote the boundary and interior of Ω respectively. Let us consider the class of paths $X(t) = (x(t), y(t)) \in \mathbf{R}^2$ satisfying the constraints:*

$$
\begin{aligned}
&X(0) = IP,\ \phi(0) = \phi_0, \\
&\exists t_f > 0, \text{ such that } X(t_f) = FP,\ \phi(t_f) = \phi_f, \\
&X(t) \in \mathbf{R}^2 \backslash \mathrm{int}(\Omega),\ \forall t \in [0, t_f], \\
&\|\dot{X}(t)\| = \sqrt{(x^2(t) + y^2(t))} = 1,\ \forall t \in [0, t_f], \\
&\|\dot{X}(t_1) - \dot{X}(t_2)\| \leq R_{min}^{-1}|(t_1 - t_2)|,\ \forall t_1, t_2 \in [0, t_f],
\end{aligned}
$$

where R_{min} is the minimum acceptable radius of curvature, IP is the initial point and FP is the final point. The problem is to find a path $X^*(t)$ defined for $t \in [0, t_f^*]$ such that t_f^* is minimum among t_fs of all paths in this class.

Our region of interest is obstacle-free, that is $\Omega = \emptyset$ in our case. The forms of the minimal length paths for this case were determined by Dubins [24, 25]:

Theorem 4.1 [25] *Every minimum length planar curve satisfying the conditions of* Problem 4.1 *with $\Omega = \emptyset$ is necessarily a C^1 curve which is either*

1. *an arc of a circle of radius R_{min} followed by a line segment, followed by an arc of a circle of radius R_{min}*
2. *a sequence of three arcs of circles of radius R_{min}*
3. *a subpath of a path of either of these two types*

Our algorithm to find minimal paths on the right half plane will be based on this theorem. Paths of Form 2 rarely occur. In our case, since the final point FP is the origin and $\phi(t_f) = 180°$, and the initial point IP should be in the right half plane, the rate of occurrence of Form 2 is much smaller. Furthermore these paths can be replaced by slightly longer paths which are unions of two paths of Form 1 or Form 3. Because of these facts, dealing with only Forms 1 and 3 will not effect the result much whereas the algorithm will be simplified.

Since we are looking for a path of Form 1 or Form 3 of 4.1, this path can be either:

1. an arc of a circle of radius R_{min} followed by a line segment, followed by an arc of a circle of radius R_{min} (Form 1)
2. an arc of a circle of radius R_{min} followed by a line segment
3. a line segment followed by an arc of a circle of radius R_{min}
4. an arc of a circle of radius R_{min} followed by another arc of a circle of radius R_{min}
5. an arc of a circle of radius R_{min}
6. a line segment

If we let the arcs and the segment to have length 0, the first case will include the other cases. For example, a path of Case 4 can be thought as an arc of a circle of radius R_{min} followed by a line segment of length 0, followed by an arc of a circle of radius R_{min}. Furthermore, the path should completely lie on the right half plane. The algorithm will define these three curve pieces and the command sequence to follow these curve pieces, if such a path exists. If there is no such path and if there is a directed arc in the right half plane beginning at IP with tangential direction ϕ_0 and ending with tangential direction $0°$, this arc will be followed, then a line segment will be followed till reaching a point IP' for which a minimal path composed of three curve pieces exists. The tangential direction at IP' will be eventually $0°$.

The output of the algorithm will be a sequence of five parameters defining the path: CP_1, ..., CP_5. If a minimal path exists, $CP_1 = CP_2 = 0$, CP_3 and CP_5 indicate the angles to be proceeded on respectively the first and the second arc (the third curve piece). Signs of CP_3 and CP_5 determine the orientations of the arcs, negative meaning CW and positive meaning CCW. CP_4 is the length of the line segment (the second curve piece) of the path. If a minimal path can be found after maneuvering as described above, CP_1 and CP_2 are the parameters for the arc and the line segment proceeded for steering to a point for which a minimal path exists. CP_3, CP_4 and CP_5 are defined the same as the case of existence of a minimal path. Signs and values of CP_1 and CP_2 have the same indications as CP_3 and CP_4 respectively. If there is no such path, all of the five parameters are 0.

Given $IP = (x,\ y)$ and $\phi_0 = \theta$, there are two circles with radii R_{min} tangent to the ray originating from IP with inclination θ. Let us denote the circle on the left of the ray by $LC(x, y, \theta)$ and the one on the right by $RC(x, y, \theta)$. Similarly, there are two fixed circles tangent to the ray originating from $FP = (0,\ 0)$ with inclination 180°. We will denote the one on the upper half plane by FC_1 and the one on the lower half plane by FC_2.

So, given $IP = (x,\ y)$ and $\phi_0 = \theta$, if a minimal path exists, the first curve piece will be an arc on either $LC(x, y, \theta)$ or $RC(x, y, \theta)$, the third piece will be another arc on either FC_1 or FC_2, and the second piece will be a line segment tangent to these arcs. Moreover the orientation of motion can only be CCW on $LC(x, y, \theta)$, CW on $RC(x, y, \theta)$, CW on FC_1 and CCW on FC_2.

Given two circles C_1 and C_2, there are at most four line segments which are tangent to both of the circles (having end points on the circles). Orientation pair of arcs of a smooth path containing one of these line segments on C_1 and C_2 is different from that for another element of the set of tangent line segments. So given one of the oriented initial circles $LC(x, y, \theta)$ and $RC(x, y, \theta)$, and one of the oriented final circles FC_1 and FC_2, there is at most one candidate for a path having portions on these oriented circles.

Let us consider $LC(x, y, \theta)$. If it is in the lower half plane, a minimal path having its first arc on this oriented circle can not have a portion on FC_1. If $LC(x, y, \theta)$ intersects with the right half plane, a path having its first piece on $LC(x, y, \theta)$ and its last piece on FC_2 does not exist. So there is at most one candidate for a minimal path having its first piece on $LC(x, y, \theta)$.

By symmetry, the same fact is valid for also $RC(x, y, \theta)$. That is there is at most one candidate for a minimal path having its first piece on $RC(x, y, \theta)$.

So for given $IP = (x, y)$ and $\phi_0 = \theta$, there are at most two candidates for a minimal path, one having its first arc on $LC(x, y, \theta)$ and the other having its first arc on $RC(x, y, \theta)$. A sample situation is given in Figure 4.1. In this situation, there are two candidates for the minimal path: Path 1 and Path 2. The shorter one, Path 1, is the desired minimal path.

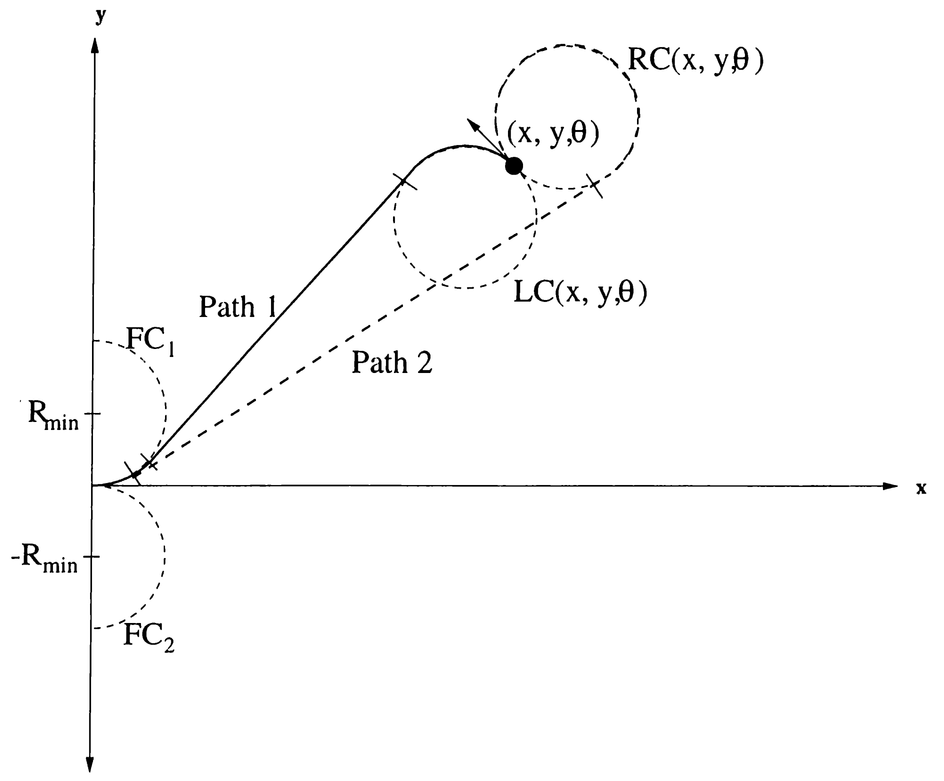


Figure 4.1: Minimal path candidates for $IP = (x, y)$ and $\phi_0 = \theta$.

Based on these facts and some further analysis, we have developed the algorithm given below. Location of $LC(x, y, \theta)$ and $RC(x, y, \theta)$, and the cases in the algorithm are illustrated in Figure 4.2 and Figure 4.3.

Algorithm 4.1 *Given an initial point* $IP = (x, y)$ *and initial direction* $\phi_0 = \theta$*:*

1. Set the default values of the five parameters defining the path and flags indicating the existence of candidate paths as:

$$CP_1 = CP_2 = CP_3 = CP_4 = CP_5 = 0$$
$$Candidate\_1\_exists = Candidate\_2\_exists = FALSE$$

2. Calculate the coordinates of the centers of $LC(x, y, \theta)$ and $RC(x, y, \theta)$ as:

$$\begin{aligned} x_{LC} &= x - R_{min}\sin(\theta) \\ y_{LC} &= y + R_{min}\cos(\theta) \\ x_{RC} &= x + R_{min}\sin(\theta) \\ y_{RC} &= y - R_{min}\cos(\theta) \end{aligned}$$

3. To determine the first candidate having its first arc on $LC(x, y, \theta)$, defined by parameters $CP1_1$, ..., $CP1_5$ and having total length $length_1$:

   (a) If $(y_{LC} \geq -R_{min})$ and $(x_{LC}^2 + (y_{LC} - R_{min})^2 \geq 4R_{min}^2)$:

   i.

$$\Theta_1 = \text{atan2}(R_{min} - y_{LC}, -x_{LC}) - \arccos(2R_{min}/\sqrt{x_{LC}^2 + (y_{LC} - R_{min})^2})$$

   ii. If $((y_{LC} \geq (1 - \sqrt{3})R_{min})$ and $(x_{LC} \geq R_{min}))$ or $((y_{LC} < (1 - \sqrt{3})R_{min})$ and $(x_{LC} \geq -R_{min}\ \text{mincos}(\theta - \frac{\pi}{2}, \Theta_1)))$:

$$\begin{aligned} CP1_3 &= (\Theta_1 - \theta + \frac{\pi}{2})_{2\pi} \\ CP1_4 &= \sqrt{x_{LC}^2 + (y_{LC} - R_{min})^2 - 4R_{min}^2} \\ CP1_5 &= -(\Theta_1 - \frac{\pi}{2})_{2\pi} \\ length_1 &= CP1_4 + R_{min}(|CP1_3| + |CP1_5|) \\ Candidate\_1\_exists &= TRUE \end{aligned}$$

   (b) Else if $(y_{LC} < -R_{min})$:

i.

$$\Theta_1 = \arcsin(x_{LC}/\sqrt{x_{LC}^2 + (y_{LC} + R_{min})^2})$$

ii. If $(x_{LC} \geq -R_{min} \operatorname{mincos}(\theta - \frac{\pi}{2}, \Theta_1))$:

$$\begin{aligned} CP1_3 &= (\Theta_1 - \theta + \frac{\pi}{2})_{2\pi} \\ CP1_4 &= \sqrt{x_{LC}^2 + (y_{LC} + R_{min})^2} \\ CP1_5 &= (\frac{\pi}{2} - \Theta_1)_{2\pi} \\ length_1 &= CP1_4 + R_{min}(|CP1_3| + |CP1_5|) \\ Candidate\_1\_exists &= TRUE \end{aligned}$$

4. To determine the second candidate having its first arc on $RC(x, y, \theta)$, defined by parameters $CP2_1$, ..., $CP2_5$ and having total length $length_2$:

(a) If $(y_{RC} \leq R_{min})$ and $(x_{RC}^2 + (y_{RC} + R_{min})^2 \geq 4R_{min}^2)$:

i.

$$\Theta_1 = \operatorname{atan2}(-R_{min} - y_{RC}, -x_{RC}) + \arccos(2R_{min}/\sqrt{x_{RC}^2 + (y_{RC} + R_{min})^2})$$

ii. If $((y_{RC} \leq (\sqrt{3} - 1)R_{min})$ and $(x_{RC} \geq R_{min}))$ or $((y_{RC} > (\sqrt{3} - 1)R_{min})$ and $(x_{RC} \geq -R_{min} \operatorname{mincos}(\Theta_1, \theta + \frac{\pi}{2}))$:

$$\begin{aligned} CP2_3 &= -(\theta + \frac{\pi}{2} - \Theta_1)_{2\pi} \\ CP2_4 &= \sqrt{x_{RC}^2 + (y_{RC} + R_{min})^2 - 4R_{min}^2} \\ CP2_5 &= (\frac{3\pi}{2} - \Theta_1)_{2\pi} \\ length_2 &= CP2_4 + R_{min}(|CP2_3| + |CP2_5|) \\ Candidate\_2\_exists &= TRUE \end{aligned}$$

(b) Else if $(y_{RC} > R_{min})$:

i.

$$\Theta_1 = \arcsin(x_{RC}/\sqrt{x_{RC}^2 + (y_{RC} - R_{min})^2})$$

ii. If ($x_{RC} \geq -R_{min}$ mincos$(\Theta_1, \theta + \frac{\pi}{2})$):

$$\begin{aligned} CP2_3 &= -(\theta + \frac{\pi}{2} - \Theta_1)_{2\pi} \\ CP2_4 &= \sqrt{x_{RC}^2 + (y_{RC} - R_{min})^2} \\ CP2_5 &= -(\Theta_1 + \frac{\pi}{2})_{2\pi} \\ length_2 &= CP2_4 + R_{min}(|CP2_3| + |CP2_5|) \\ Candidate\_2\_exists &= TRUE \end{aligned}$$

5. If not($Candidate\_1\_exists$ or $Candidate\_2\_exists$):
If ($-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$) or ($x \geq R_{min}(1 - |\sin\theta|)$):

(a) If ($\theta \leq \pi$):

$$\begin{aligned} x' &= x + R_{min}\sin\theta \\ y' &= y - R_{min}\cos\theta + R_{min} \\ CP_1 &= -\theta \end{aligned}$$

(b) Else:

$$\begin{aligned} x' &= x - R_{min}\sin\theta \\ y' &= y + R_{min}\cos\theta - R_{min} \\ CP_1 &= 2\pi - \theta \end{aligned}$$

(c) If ($x'^2 + y'^2 \leq 4R_{min}^2$):

$$CP_2 = \sqrt{4R_{min}^2 - y'^2} - x'$$

(d) Set $x = x' + CP_2$, $y = y'$, $\theta = 0°$ and return to Step 2.

6. If ($Candidate\_1\_exists$):

(a) If ($Candidate\_2\_exists$) and ($length_1 > length2$):

$$CP_3 = CP2_3$$

$$\begin{aligned} CP_4 &= CP2_4 \\ CP_5 &= CP2_5 \end{aligned}$$

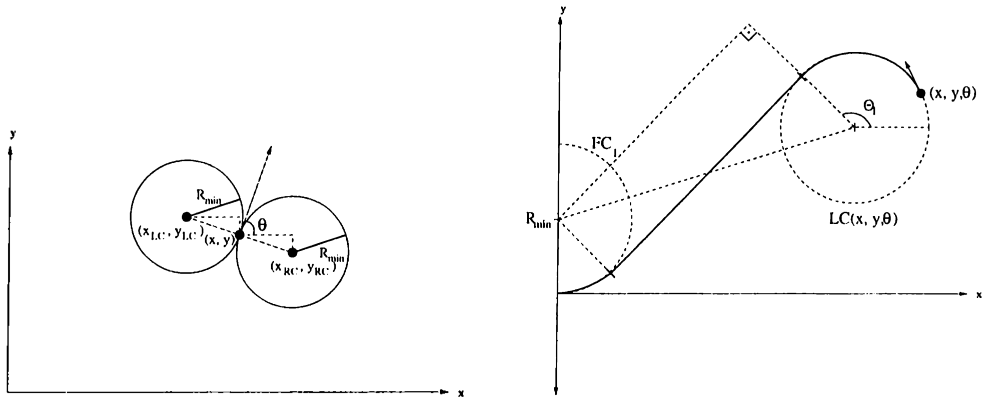
(b) Else:

$$\begin{aligned} CP_3 &= CP1_3 \\ CP_4 &= CP1_4 \\ CP_5 &= CP1_5 \end{aligned}$$

7. Else if $(Candidate\_2\_exists)$:

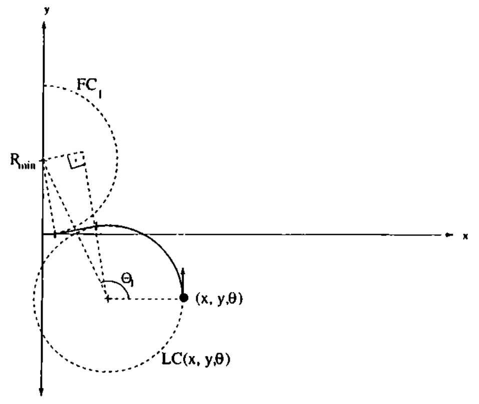
$$\begin{aligned} CP_3 &= CP2_3 \\ CP_4 &= CP2_4 \\ CP_5 &= CP2_5 \end{aligned}$$

Here, $(x)_{2\pi}$ gives the equivalent angle of x in $[0, 2\pi)$, and $\text{mincos}(\alpha_1, \alpha_2)$ gives the minimum value of $\cos(\alpha)$ while α is tracing the CCW unit circular arc starting at angle α_1 and ending at angle α_2.

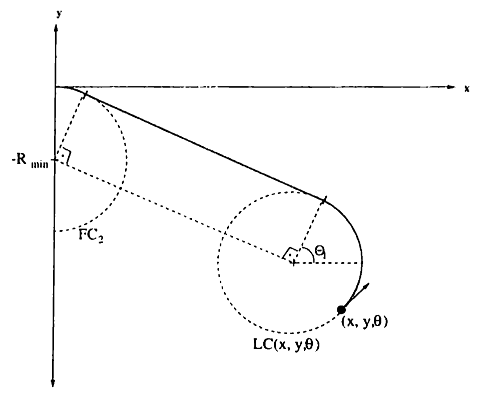


(a) (b) $x = 11,\ y = 6,\ \theta = 117°$

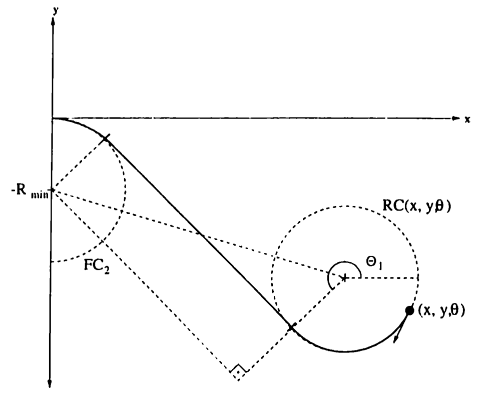
Figure 4.2: Algorithm 4.1: (a) location of $LC(x, y, \theta)$ and $RC(x, y, \theta)$ and (b) sample minimal path for the first case in Step 3(a).ii.



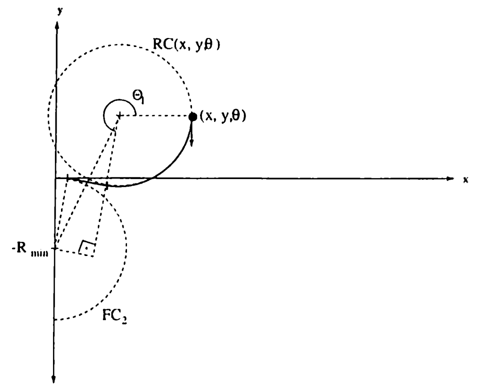
(a) $x = 4.25,\ y = -2,\ \theta = 90°$



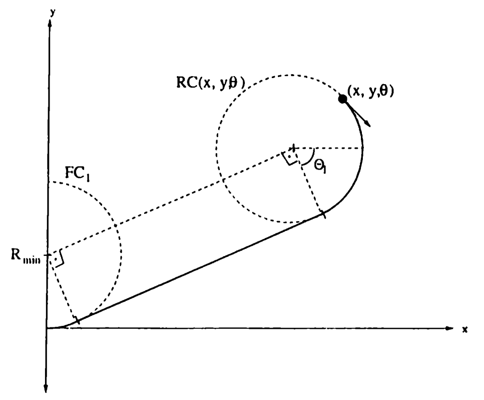
(b) $x = 9,\ y = -7,\ \theta = 45°$



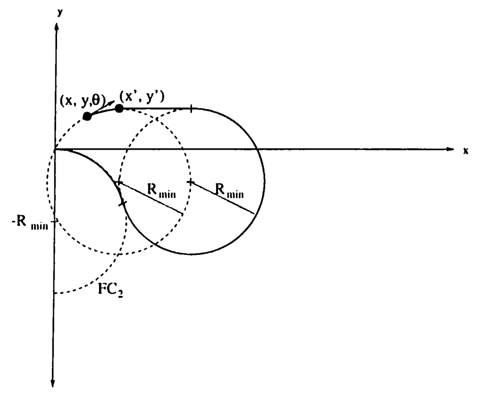
(c) $x = 11,\ y = -6,\ \theta = 243°$



(d) $x = 4.25,\ y = 2,\ \theta = 270°$



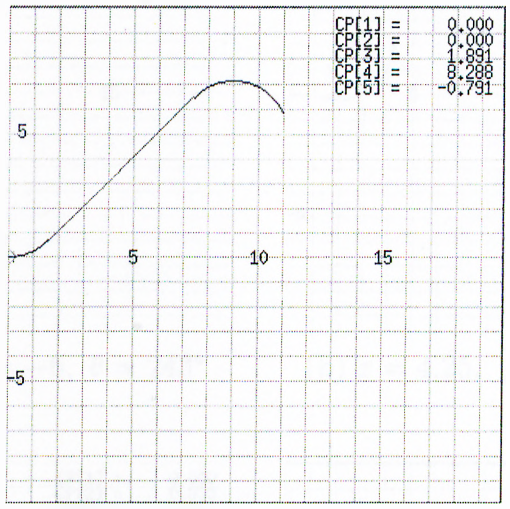
(e) $x = 9,\ y = 7,\ \theta = 315°$



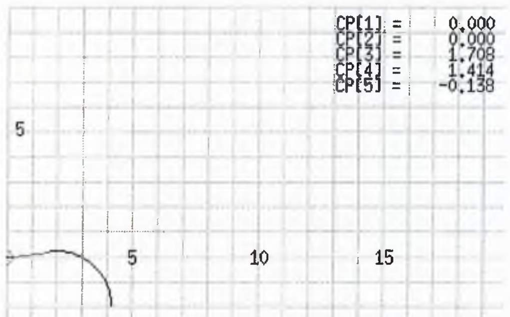
(f) $x = 1,\ y = 1,\ \theta = 30°$

Figure 4.3: Algorithm 4.1: Sample minimal paths for (a) the second case in Step 3(a).ii, (b) Step 3(b), (c) the first case in Step 4(a).ii, (d) the second case in Step 4(a).ii, (e) Step 4(b) and (f) Step 5.

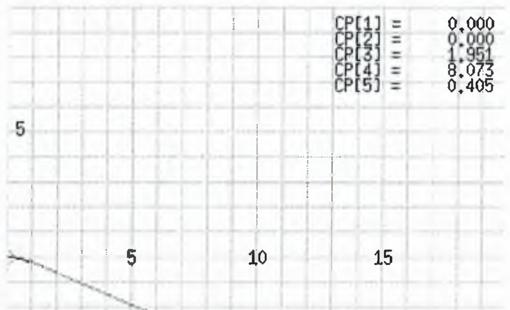
We have written the code *path.h* to implement this algorithm directly and the code *path.cpp* to see the resultant path. For the case of Step 5 of the algorithm, to prevent possible discretization problems, CP_2 is lengthened by 0.1 in these codes. Such an extension guarantees the end of the subpath with the parameter CP_2 to be the initial point of a minimal path. Outputs of *path.cpp* for the seven examples in Figure 4.2 and Figure 4.3 are given in Figures 4.4 and 4.5.


```
CP[1] =    0.000
CP[2] =    0.000
CP[3] =    1.891
CP[4] =    8.288
CP[5] =   -0.791
```

(a) $x = 11,\ y = 6,\ \theta = 117°$

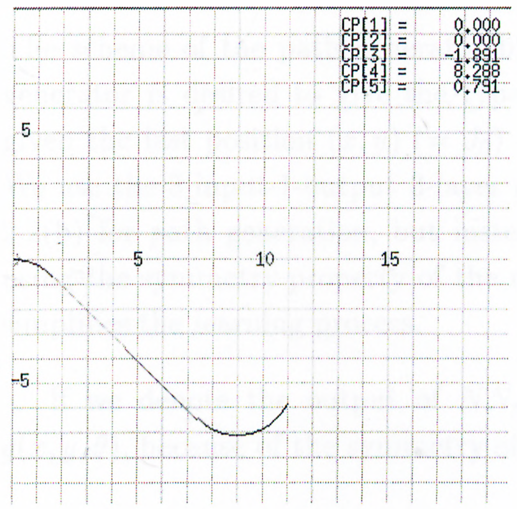

```
CP[1] =    0.000
CP[2] =    0.000
CP[3] =    1.708
CP[4] =    1.414
CP[5] =   -0.138
```

(b) $x = 4.25,\ y = -2,\ \theta = 90°$


```
CP[1] =    0.000
CP[2] =    0.000
CP[3] =    1.951
CP[4] =    8.073
CP[5] =    0.405
```

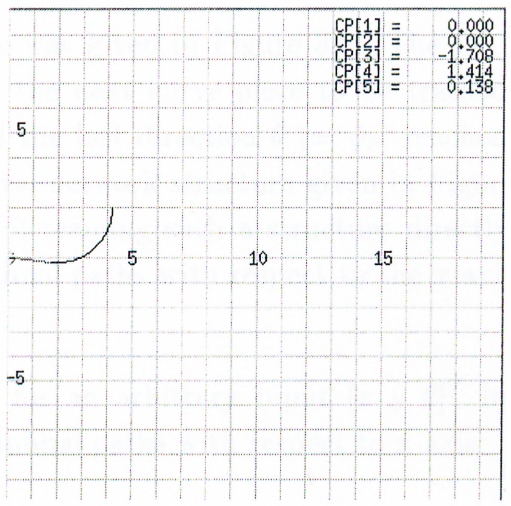
(c) $x = 9,\ y = -7,\ \theta = 45°$

Figure 4.4: Outputs of the C^{++}-code of Algorithm 4.1 for the examples illustrated in Figures 4.2(b), 4.3(a) and 4.3(b).



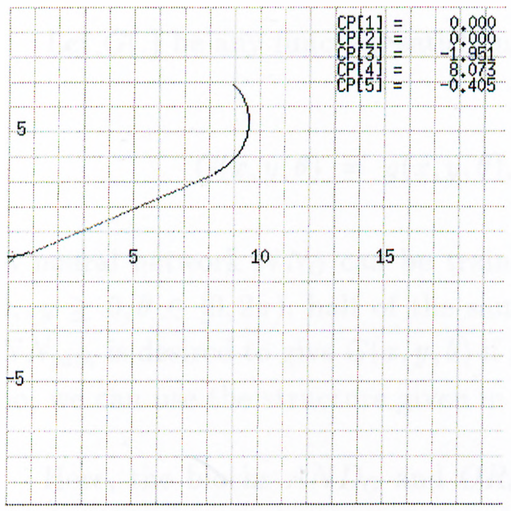
```
CP[1] =     0.000
CP[2] =     0.000
CP[3] =    -1.891
CP[4] =     8.288
CP[5] =     0.791
```

(a) $x = 11,\ y = -6,\ \theta = 243°$



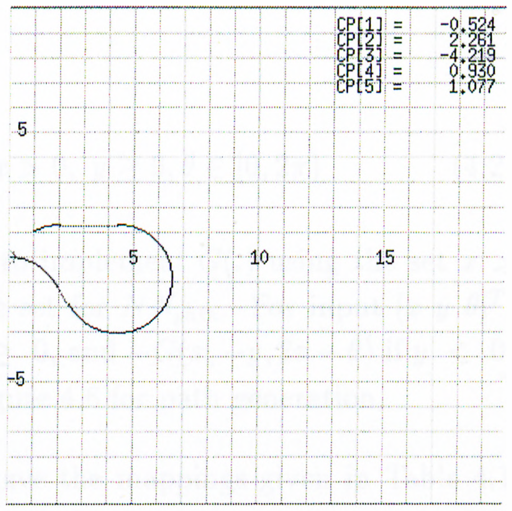
```
CP[1] =     0.000
CP[2] =     0.000
CP[3] =    -1.708
CP[4] =     1.414
CP[5] =     0.138
```

(b) $x = 4.25,\ y = 2,\ \theta = 270°$



```
CP[1] =     0.000
CP[2] =     0.000
CP[3] =    -1.951
CP[4] =     8.073
CP[5] =    -0.405
```

(c) $x = 9,\ y = 7,\ \theta = 315°$



```
CP[1] =    -0.524
CP[2] =     2.261
CP[3] =    -4.219
CP[4] =     0.930
CP[5] =     1.077
```

(d) $x = 1,\ y = 1,\ \theta = 30°$

Figure 4.5: Outputs of the C^{++}-code of Algorithm 4.1 for the examples illustrated in Figures 4.3(c), 4.3(d), 4.3(e) and 4.3(f).

4.2 Design of the Path Generator Neural Network

The scheme of the path generator block was given in Figure 3.5. The path generator is desired to give the type of the next curve piece to be tracked, given the last position (x, y) $(= J_0)$ and the last direction θ $(= \phi_0)$ reached. We use the 3-input-1-output 3-layer neural network structure given in Figure 2.2(b) for path generation, but instead of training such a structure directly, using the symmetry property of the minimal paths with respect to the x-axis simplifies the learning process.

If we denote the output of NN_{path} for an arbitrary input $[x \ y \ \theta]^T$ by $g(x, y, \theta)$, because of the symmetry of minimal paths with respect to x-axis:

$$g(x, y, \theta) \;=\; -g(x, -y, 2\pi - \theta) \tag{4.1}$$

Let $G(.)$ be any function satisfying

$$G(x, y, \theta) \;=\; g(x, y, \theta), \; \forall x > 0, \; y \geq 0, \theta \in [0, 2\pi) \tag{4.2}$$

A neural network NN_G can be trained to implement $G(.)$ using the set $x > 0$, $y \geq 0, \theta \in [0, 2\pi)$, half of the sample set for NN_{path}. For input $[x \ y \ \theta]^T$, NN_{path} is desired to give $G(x, y, \theta)$ if $y \geq 0$, and $-G(x, -y, 2\pi - \theta)$ if $y < 0$. So, we can use the structure given in Figure 4.6 for path generation.

Here, the blocks AND_G and OR_G have the mappings given in Table 4.1 ($\times$ denotes that the input pair is not applicable) and the structures given in Figure 4.7.

Based on this structure, we have trained a 3-input-1-output 3-layer neural network with the structure given in Figure 2.2(b) to implement $G(.)$. The numbers of neurons in the first and the second hidden layers of NN_G are $N_1 = 100$ and $N_2 = 5$. We have used

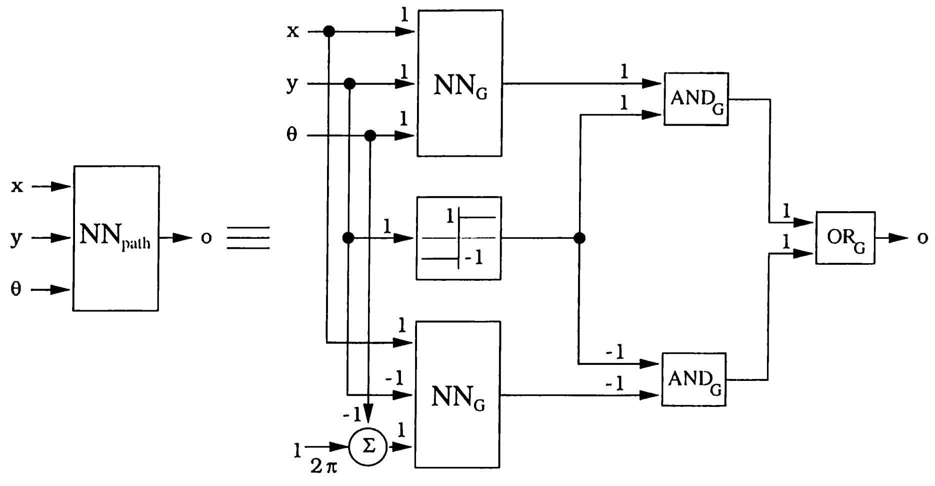


Figure 4.6: Construction of NN_{path} using NN_G blocks.

x	-1	-1	-1	0	0	0	1	1	1
y	-1	0	1	-1	0	1	-1	0	1
AND_G	-1	-1	-1	-1	$\times$	0	-1	0	1
OR_G	-1	0	1	0	$\times$	$\times$	1	$\times$	$\times$

Table 4.1: Mappings for the blocks AND_G and OR_G.

$$\varphi(x) = \tanh\left(\frac{\lambda x}{2}\right) = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}} \tag{4.3}$$

with $\lambda = 0.1$ as the activation function.

This neural network is trained using Algorithm 4.1. For each (x, y, θ)-triple in the training set, the desired output is determined according to the

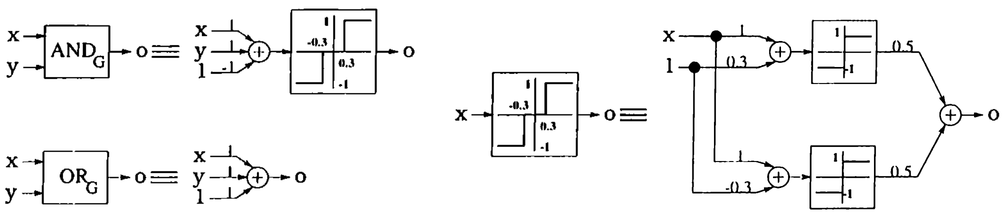


Figure 4.7: Blocks AND_G and OR_G, and the 3-state hardlimiter used in AND_G.

first nonzero-length curve piece of the output of Algorithm 4.1 for (x, y, θ). The desired output is 0 if this curve piece is a line segment (CP_2 or CP_4), -1 if it is a CW arc (CP_1, CP_3 or CP_5 with negative value), 1 if it is a CCW arc (CP_1, CP_3 or CP_5 with positive value). In this approach, the desired resultant path is a smooth approximation of the minimal path composed of curve pieces of the three types defined in Section 3.3.

Let us partition our sample set $\{(x, y, \theta) \mid 0 < x < 20, \ -10 < y < 10, 0 \leq \theta < 2\pi\}$ into three regions according to the desired outputs matched to the points in these regions. Let the regions R_{-1}, R_0, R_1 be the collections of all of the points for which the desired output is -1, 0 and 1 respectively. R_0 is just a proper subset of the 2-dimensional boundary separating R_{-1} and R_1. Furthermore R_{-1} and R_1 and are highly nonlinearly separated. The elements of the boundary separating R_{-1} and R_1 which are not in R_0 are those for which the two candidates for minimal path have equal lengths. Because of these facts, selecting input-desired output pairs keeping the three regions as they are is not suitable to train a neural network with feasible structure. We have expanded the 2-dimensional boundary separating R_{-1} and R_1 to a 3-dimensional slice with thickness of 1 unit. All of the triples in this slice are assigned the desired output value 0. The resultant sample set is shown in Figure 4.8 on a collection of constant-θ-planes.

25000 triple-desired output pairs are chosen randomly from the upper half part of the sample set, in which $x \geq 0$, to construct the training set. After composing the training set, the Back Propagation Algorithm, whose equations are given in Appendix A, is applied to train the 3-layer path generator neural network. The learning and momentum constants are chosen as $\eta = 0.5$ and $\alpha = 0.3$. The neural network is trained for totally 45000 epoches.

4.3 Results

During the training, we have saved the average error for each epoch and the neural network weights after epoches 100, 500, 1000, 2000, ..., 45000. Plot of the average error for the first 10000 epoches is given in Figure 4.9. After epoch

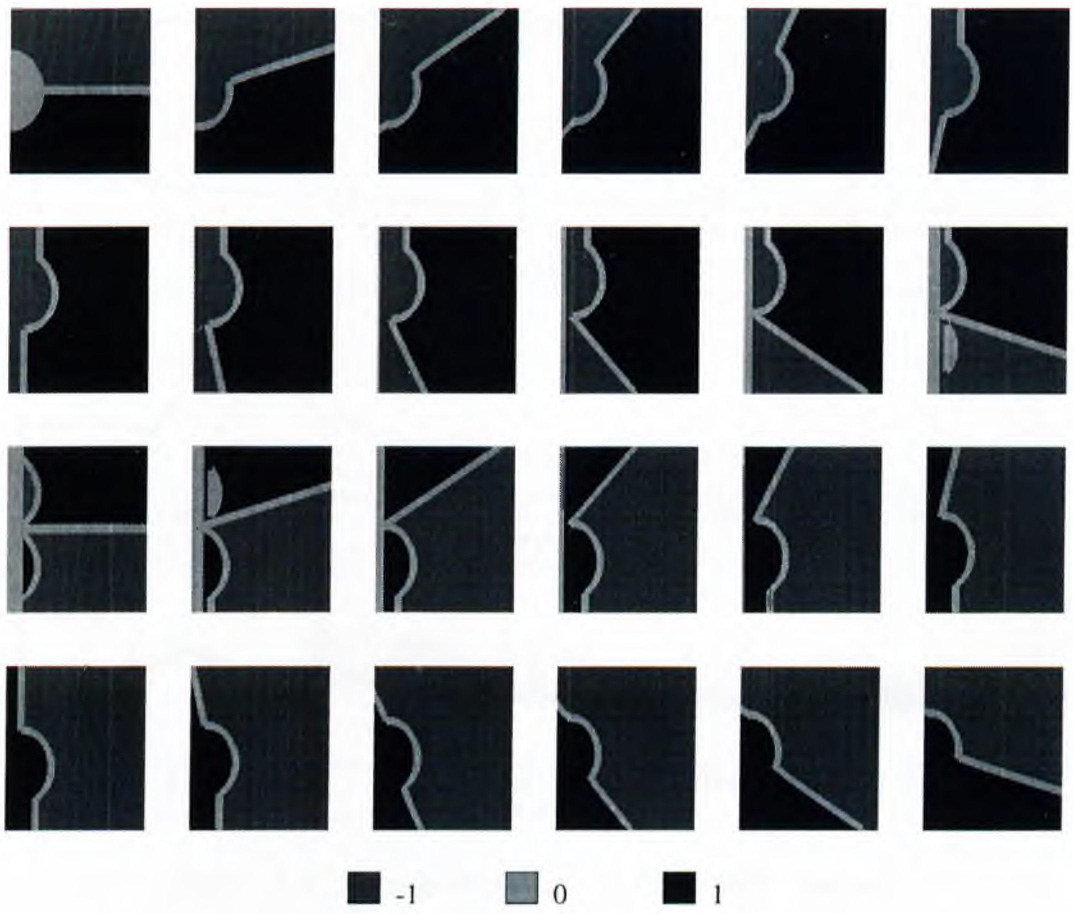


Figure 4.8: Desired outputs for $(x, y) \in R_i$ (defined in Problem 3.1) for $\theta = 0, \frac{\pi}{12}, \ldots, \frac{23\pi}{12}$.

10000, the average error continues to decrease but the decrease rate is very small and the fluctuation among successive epoches increases significantly.

Using the codes *path.cpp*, *pathnn_d.cpp* and *pathnn.cpp*, we have also checked the path generation performance of the neural network for the saved weights. The best result in this sense is obtained using the weights obtained after epoch 9000. Although the average error continues to decrease slightly after that, the weight sets of further epoches have generated paths significantly different from the desired minimal paths for some sample initial points. The minimal paths (outputs of Algorithm 4.1), the desired outputs obtained using the original forms of R_{-1}, R_0 and R_1, and the outputs using the neural network weights of epoches 100, 1000 and 9000 for some sample initial triples

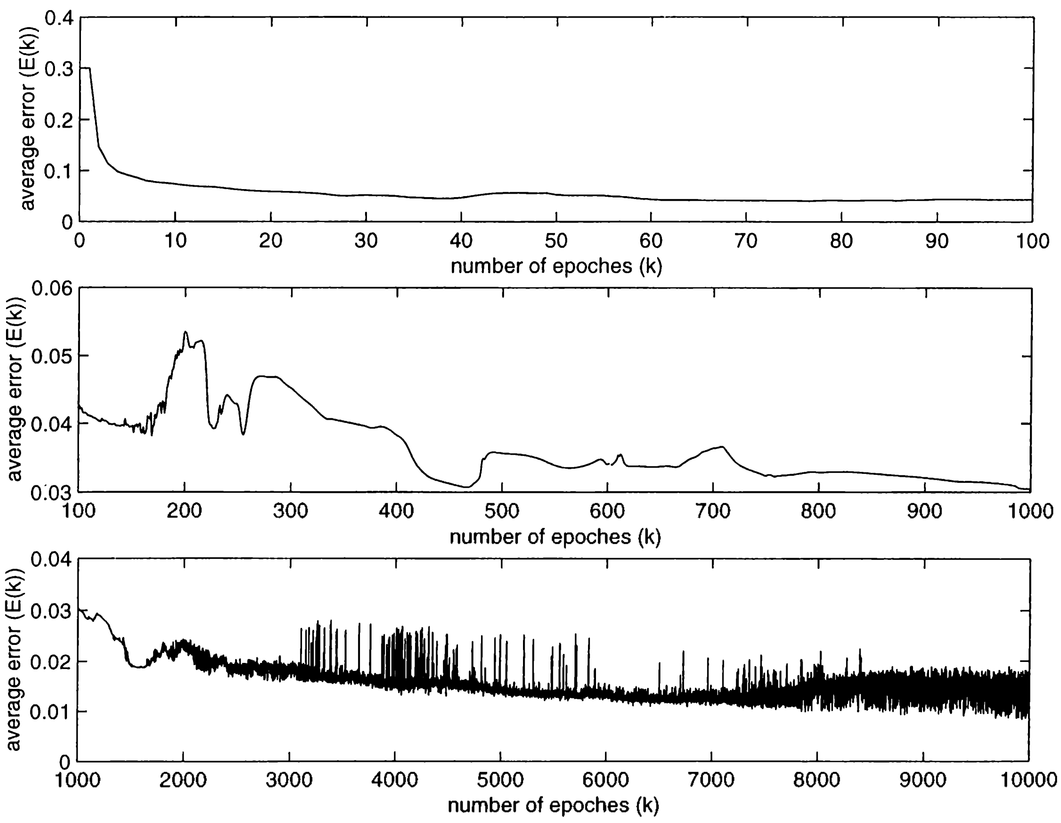


Figure 4.9: Average error for the first 10000 epoches.

are pictured in Figure 4.10 and Figure 4.11. The generated smooth paths are shown as sequences of line segments which are either curve pieces of type 0 themselves or chords of the pieces of type -1 or type 1.

The path generator neural network having the weights obtained after epoch 9000 is combined with the neural network designed in Chapter 3 to track the generated path to construct the complete neural network controller. A sample process using this complete neural network controller is pictured in Figure 4.12.

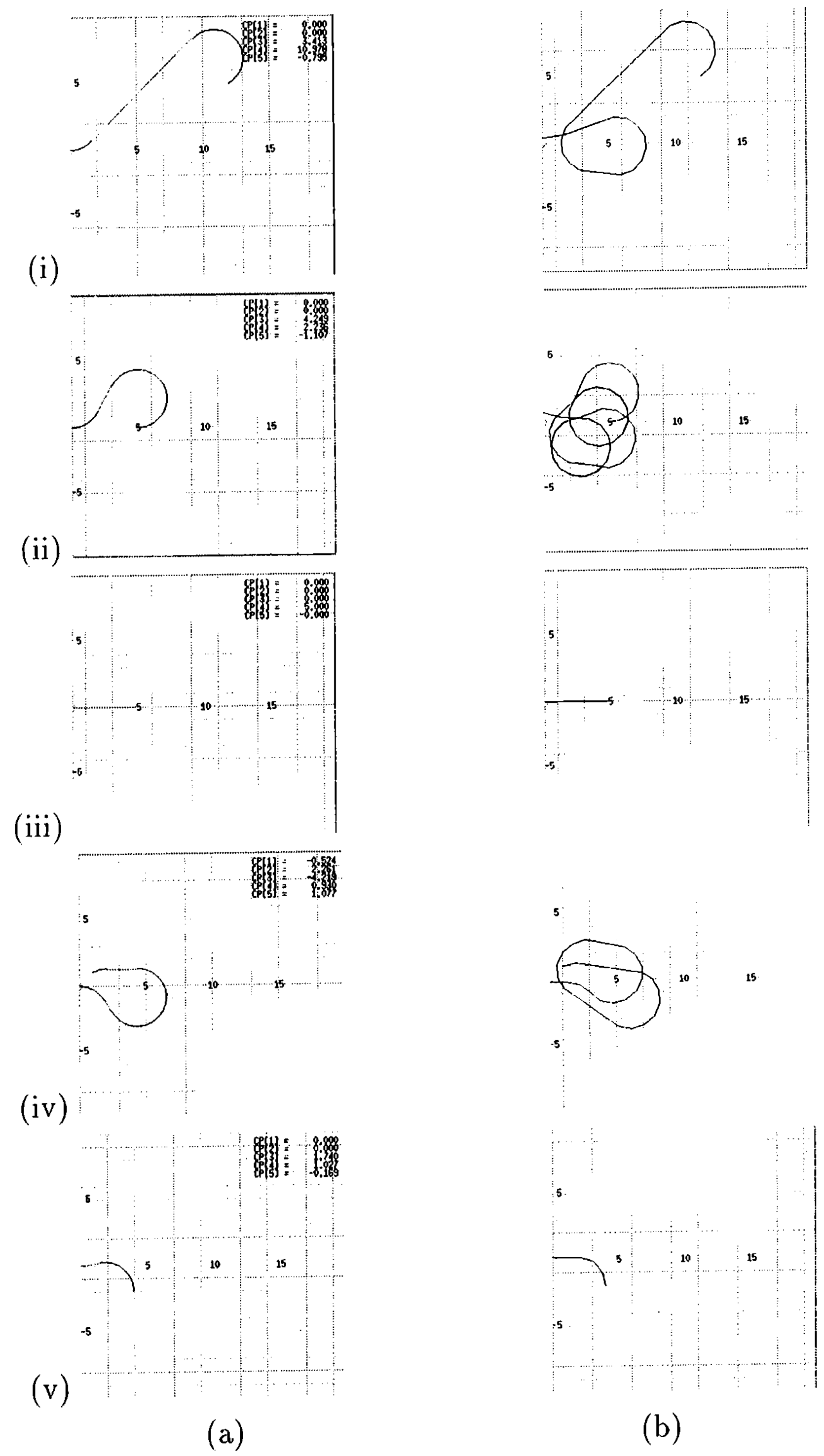


Figure 4.10: (a) Minimal paths and (b) the desired outputs obtained using the original forms of R_{-1}, R_0 and R_1 for the initial triples (i) $x = 12$, $y = 5$, $\theta = 30°$, (ii) $x = 5$, $y = 0$, $\theta = 0°$, (iii) $x = 5$, $y = 0$, $\theta = 180°$, (iv) $x = 1$, $y = 1$, $\theta = 30°$, (v) $x = 4$, $y = -2$, $\theta = 90°$.

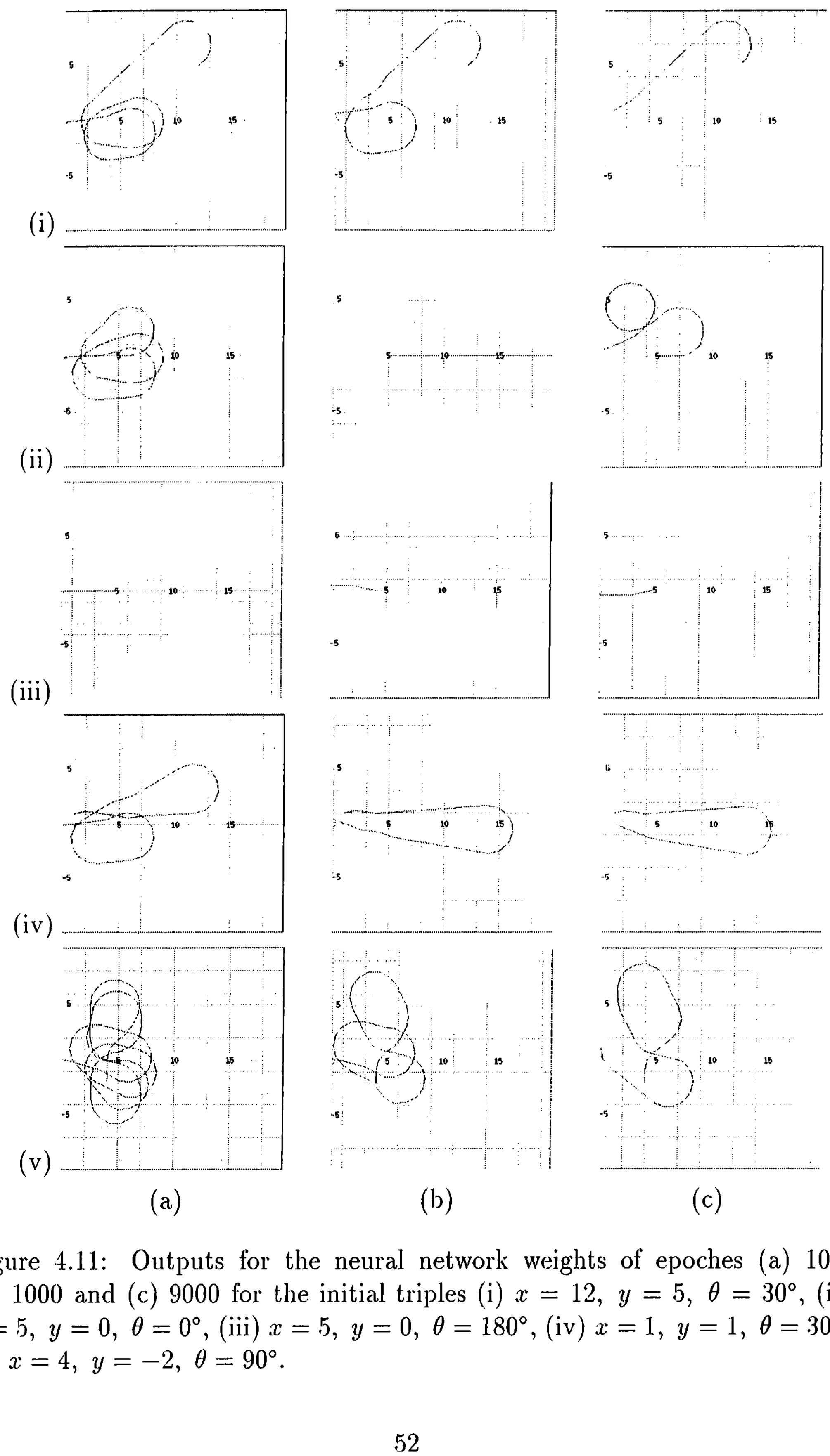
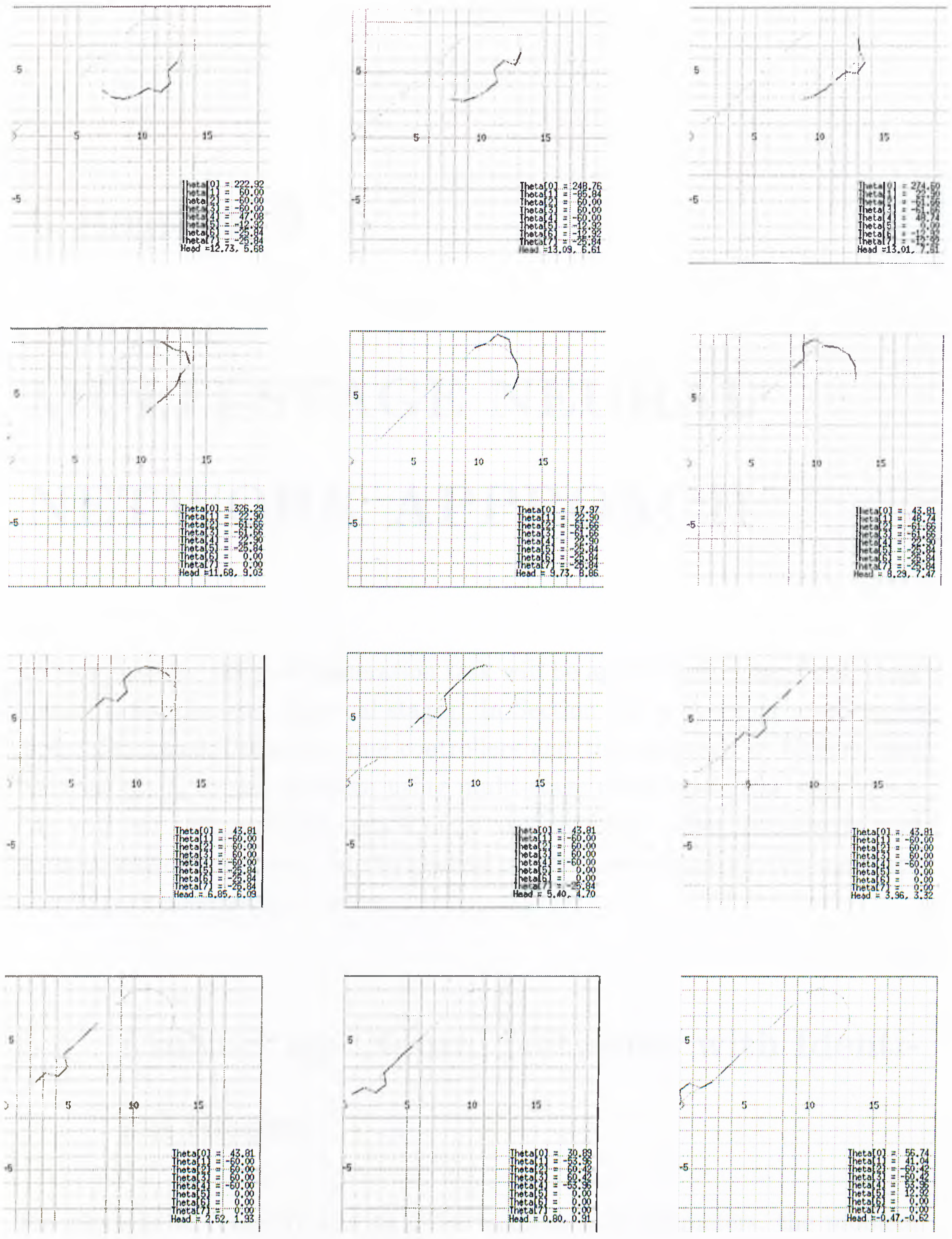


Figure 4.11: Outputs for the neural network weights of epoches (a) 100, (b) 1000 and (c) 9000 for the initial triples (i) $x = 12$, $y = 5$, $\theta = 30°$, (ii) $x = 5$, $y = 0$, $\theta = 0°$, (iii) $x = 5$, $y = 0$, $\theta = 180°$, (iv) $x = 1$, $y = 1$, $\theta = 30°$, (v) $x = 4$, $y = -2$, $\theta = 90°$.



```
Theta[0] = 222.92
Theta[1] =  60.00
Theta[2] = -60.00
Theta[3] = -60.00
Theta[4] =  47.08
Theta[5] = -12.92
Theta[6] = -25.84
Theta[7] = -25.84
Head =12.73, 5.68
```

```
Theta[0] = 248.76
Theta[1] = -85.84
Theta[2] =  60.00
Theta[3] =  60.00
Theta[4] = -60.00
Theta[5] = -12.92
Theta[6] = -12.92
Theta[7] = -25.84
Head =13.09, 6.61
```

```
Theta[0] = 274.60
Theta[1] =  22.90
Theta[2] = -61.66
Theta[3] = -61.66
Theta[4] =  48.74
Theta[5] =   0.00
Theta[6] = -12.92
Theta[7] = -12.92
Head =13.01, 7.51
```

```
Theta[0] = 326.29
Theta[1] =  22.90
Theta[2] = -61.66
Theta[3] = -61.66
Theta[4] =  22.90
Theta[5] = -25.84
Theta[6] =   0.00
Theta[7] =   0.00
Head =11.68, 9.03
```

```
Theta[0] =  17.97
Theta[1] =  22.90
Theta[2] = -61.66
Theta[3] = -61.66
Theta[4] =  22.90
Theta[5] = -25.84
Theta[6] = -25.84
Theta[7] = -25.84
Head = 9.73, 8.86
```

```
Theta[0] =  43.81
Theta[1] =  48.74
Theta[2] = -61.66
Theta[3] = -61.66
Theta[4] =  22.90
Theta[5] = -25.84
Theta[6] = -25.84
Theta[7] = -25.84
Head = 8.29, 7.47
```

```
Theta[0] =  43.81
Theta[1] = -60.00
Theta[2] =  60.00
Theta[3] =  60.00
Theta[4] = -60.00
Theta[5] = -25.84
Theta[6] = -25.84
Theta[7] = -25.84
Head = 6.85, 6.09
```

```
Theta[0] =  43.81
Theta[1] = -60.00
Theta[2] =  60.00
Theta[3] =  60.00
Theta[4] = -60.00
Theta[5] =   0.00
Theta[6] =   0.00
Theta[7] = -25.84
Head = 5.40, 4.70
```

```
Theta[0] =  43.81
Theta[1] = -60.00
Theta[2] =  60.00
Theta[3] =  60.00
Theta[4] = -60.00
Theta[5] =   0.00
Theta[6] =   0.00
Theta[7] =   0.00
Head = 3.96, 3.32
```

```
Theta[0] =  43.81
Theta[1] = -60.00
Theta[2] =  60.00
Theta[3] =  60.00
Theta[4] = -60.00
Theta[5] =   0.00
Theta[6] =   0.00
Theta[7] =   0.00
Head = 2.52, 1.93
```

```
Theta[0] =  30.89
Theta[1] = -53.96
Theta[2] =  60.42
Theta[3] =  60.42
Theta[4] = -53.96
Theta[5] =   0.00
Theta[6] =   0.00
Theta[7] =   0.00
Head = 0.80, 0.91
```

```
Theta[0] =  56.74
Theta[1] =  41.04
Theta[2] = -60.42
Theta[3] = -60.42
Theta[4] =  53.96
Theta[5] =  12.92
Theta[6] =   0.00
Theta[7] =   0.00
Head =-0.47,-0.62
```

Figure 4.12: Sample process using the complete neural network controller.

Chapter 5

MULTI-STAGE NEURAL NETWORK APPROACH

In this chapter, the path generation task will be approached using multi-stage neural networks with identical stages. In Section 2.4, a common problem for finite-dimensional discrete-time controllers and the approach of Ngüyen and Widrow [6, 7] to this problem using multi-stage neural networks were given. We will put this approach into a more explicit form. After introducing and formulating our approach for the general problem, we will apply this approach to our path generation problem.

5.1 Multi-Stage Neural Networks with Identical Stages

For the sake of integrity and to avoid confusion of parameters, the problem introduced in Section 2.4 will be redefined here using new parameters: A finite-dimensional discrete-time controller can be represented as in Figure 5.1. Here

$\mathbf{x}(k)$ and $\mathbf{x}(k+1)$ denote the current and next states of the plant respectively, and $\mathbf{o}(k)$ is the control signal given to the plant by the controller. Plant and controller equations for this scheme is as follows:

$$\begin{aligned} \mathbf{x}(k+1) &= D(\mathbf{x}(k), \mathbf{o}(k)) && (5.1) \\ \mathbf{o}(k) &= C(\mathbf{x}(k)) && (5.2) \end{aligned}$$

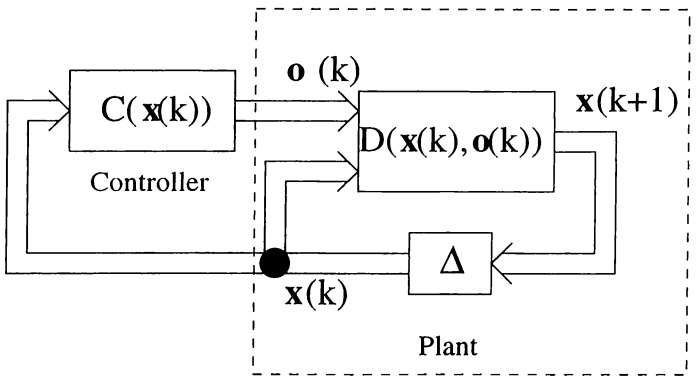


Figure 5.1: A finite-dimensional discrete-time system with a controller.

A common problem for such a scheme is to drive the plant from an initial state $\mathbf{x}(0)$ to a desired state $\mathbf{d}$ in an unknown number of steps. We want to solve this problem using a 2-layer neural network NN_C, which is the most general form of multilayer neural networks. Let $\mathbf{x}$ be of length n and $\mathbf{o}$ be of length m. Then the neural network we use will have the structure given in Figure 2.2(a), with N neurons in the hidden layer. To easily train the neural network, the plant function $D(.)$ has to be smooth (C^1). If it is not smooth, it can be approximated by a smooth function, which can be obtained either by direct derivation or using a neural network NN_D to emulate the plant. Some methods to emulate plants using multilayer neural networks are given in [6, 7, 11].

To train NN_C, we will use a training set composed of sample initial states and the states desired to be reached from these initial states. The number of steps to reach the desired state for each initial state is unknown and this number will most probably not be the same for all initial states. We can directly obtain only the final errors using the final states and their desired values.

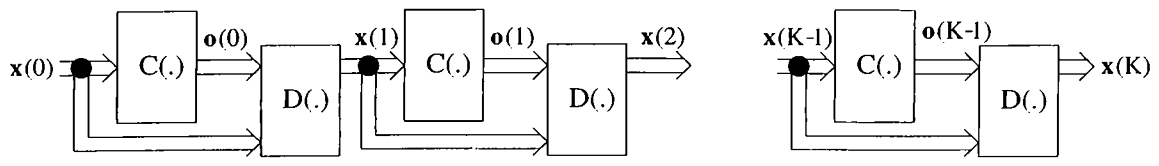


Figure 5.2: K-stage neural network representation of a K-step process.

For a specific initial state-desired state pair ($\mathbf{x}(0)$, $\mathbf{d}$), starting from $\mathbf{x}(0)$, the system will be run until a stopping criterion holds. Let this process last for K steps. A K-stage neural network structure, pictured in Figure 5.2, will be used to represent the process and to update the weights of NN_C.

The error to be backpropagated is

$$E = \frac{1}{2}(\|\mathbf{x}(K) - \mathbf{d}\|^2) = \frac{1}{2}\sum_{i=1}^{n}(x_i(K) - d_i)^2 \tag{5.3}$$

where $\mathbf{x} = [x_1\ x_2\ ...\ x_n]^T$ and $\mathbf{d} = [d_1\ d_2\ ...\ d_n]^T$. To apply Back Propagation with Momentum equation

$$\Delta w(n) = -\eta\frac{\partial E(n)}{\partial w} + \alpha\Delta w(n-1) \tag{5.4}$$

the only difficulty is to determine $\frac{\partial E}{\partial w}$ for each weight w of NN_C. Effect of each w for all of the K stages should be considered to evaluate this partial derivative:

Let us define $\mathbf{u}$ as

$$\mathbf{u} = \begin{bmatrix} \mathbf{x} \\ \mathbf{o} \end{bmatrix} = \begin{bmatrix} x_1 \\ \\ x_n \\ o_1 \\ \\ o_m \end{bmatrix} \tag{5.5}$$

and let

$$\mathbf{o}(k) = C(\mathbf{x}(k)) = \begin{bmatrix} C_1(\mathbf{x}(k)) \\ C_2(\mathbf{x}(k)) \\ \\ C_m(\mathbf{x}(k)) \end{bmatrix} \text{ and } \mathbf{x}(k+1) = D(\mathbf{u}(k)) = \begin{bmatrix} D_1(\mathbf{u}(k)) \\ D_2(\mathbf{u}(k)) \\ \\ D_n(\mathbf{u}(k)) \end{bmatrix} \tag{5.6}$$

For simplicity, let us denote partial derivatives of $C(.)$ and $D(.)$ as

$$C_{ij}(k) = \frac{\partial C_i(\mathbf{x}(k))}{\partial x_j(k))} \text{ for } i = 1, ..., m;\ j = 1, ..., n \tag{5.7}$$

$$D_{ij}(k) = \frac{\partial D_i(\mathbf{u}(k))}{\partial u_j(k))} \text{ for } i = 1, ..., n;\ j = 1, ..., n+m \tag{5.8}$$

and let us collect these partial derivatives in two matrices as

$$C' = \begin{bmatrix} \multicolumn{3}{c}{\mathbf{I}_n} \\ \hline C_{11} & C_{12} & C_{1n} \\ \vdots & \vdots & \vdots \\ C_{m1} & C_{m2} & C_{mn} \end{bmatrix} \text{ and } D' = \begin{bmatrix} D_{11} & D_{12} & D_{1(n+m)} \\ \vdots & & \vdots \\ D_{n1} & D_{n2} & D_{n(n+m)} \end{bmatrix} \tag{5.9}$$

Explicit form of C' for the neural network structure given in Figure 2.2, with N neurons in the hidden layer, is as follows:

$$C' = \begin{bmatrix} \multicolumn{3}{c}{\mathbf{I}_n} \\ \hline \dot{\varphi}(v_1)\sum_{i=1}^{N} w_{1i}^{o}\dot{\varphi}(z_i)w_{i1}^{h} & & \dot{\varphi}(v_1)\sum_{i=1}^{N} w_{1i}^{o}\dot{\varphi}(z_i)w_{in}^{h} \\ \vdots & & \vdots \\ \dot{\varphi}(v_m)\sum_{i=1}^{N} w_{mi}^{o}\dot{\varphi}(z_i)w_{i1}^{h} & \cdots & \dot{\varphi}(v_m)\sum_{i=1}^{N} w_{mi}^{o}\dot{\varphi}(z_i)w_{in}^{h} \end{bmatrix} \tag{5.10}$$

For the weights in the output layer:

$$\frac{\partial E}{\partial w_{ij}^o} = (\mathbf{x}(K) - \mathbf{d})^T \frac{\partial \mathbf{x}(K)}{\partial w_{ij}^o} \tag{5.11}$$

To evaluate $\frac{\partial \mathbf{x}(K)}{\partial w_{ij}^o}$, we first derive a recursive equation for $k = 1, 2, ..., K$:

$$\frac{\partial \mathbf{x}(k)}{\partial w_{ij}^o} = D'(k-1)\frac{\partial \mathbf{u}(k-1)}{\partial w_{ij}^o} \tag{5.12}$$

$$\frac{\partial \mathbf{u}(k)}{\partial w_{ij}^o} = C'(k)\frac{\partial \mathbf{x}(k)}{\partial w_{ij}^o} + \dot{\Phi}_i(k) y_j(k) \tag{5.13}$$

where $\dot{\Phi}_i(k)$ is defined as

$$\dot{\Phi}_i(k) = [0 \,...\, 0 \; \varphi(v_i(k)) \; 0 \,...\, 0]^T \tag{5.14}$$

a vector of length $n+m$ with $(n+i)$th entry $\varphi(v_i(k))$ and other entries 0. Using Equations 5.12 and 5.13, we get:

$$\frac{\partial \mathbf{x}(k)}{\partial w_{ij}^o} = D'(k-1)\left(C'(k-1)\frac{\partial \mathbf{x}(k-1)}{\partial w_{ij}^o} + \dot{\Phi}_i(k-1) y_j(k-1)\right) \tag{5.15}$$

To have a closed-form expression:

$$\begin{aligned}
\frac{\partial \mathbf{x}(0)}{\partial w_{ij}^o} &= 0, \quad \frac{\partial \mathbf{x}(1)}{\partial w_{ij}^o} = D'(0)\dot{\Phi}_i(0) y_j(0) \\
\frac{\partial \mathbf{x}(2)}{\partial w_{ij}^o} &= D'(1)\left(C'(1)D'(0)\dot{\Phi}_i(0)y_j(0) + \dot{\Phi}_i(1)y_j(1)\right) \\
&= D'(1)C'(1)D'(0)\dot{\Phi}_i(0)y_j(0) + D'(1)\dot{\Phi}_i(1)y_j(1) \\
\frac{\partial \mathbf{x}(3)}{\partial w_{ij}^o} &= D'(2)C'(2)D'(1)C'(1)D'(0)\dot{\Phi}_i(0)y_j(0) \\
&\quad + D'(2)C'(2)D'(1)\dot{\Phi}_i(1)y_j(1) + D'(2)\dot{\Phi}_i(2)y_j(2)
\end{aligned}$$

By induction:

$$\frac{\partial \mathbf{x}(k)}{\partial w_{ij}^o} = \sum_{p=0}^{k-1} D'(k-1)C'(k-1)...D'(p+1)C'(p+1)D'(p)\dot{\Phi}_i(p)y_j(p) \quad (5.16)$$

Using this, we get the final result as:

$$\frac{\partial E}{\partial w_{ij}^o} = (\mathbf{x}(K)-\mathbf{d})^T \sum_{p=0}^{K-1} D'(K-1)C'(K-1)...D'(p+1)C'(p+1)D'(p)\dot{\Phi}_i(p)y_j(p) \quad (5.17)$$

for $i = 1, ..., m$ and $j = 1, ..., N$. Similarly, for $i = 1, ..., m$:

$$\frac{\partial E}{\partial w_{bi}^o} = (\mathbf{x}(K)-\mathbf{d})^T \sum_{p=0}^{K-1} D'(K-1)C'(K-1)...D'(p+1)C'(p+1)D'(p)\dot{\Phi}_i(p) \quad (5.18)$$

Proceeding in the same fashion for the weights in the hidden layer, to derive a recursive equation for $k = 1, 2, ..., K$:

$$\frac{\partial \mathbf{x}(k)}{\partial w_{ij}^h} = D'(k-1)\frac{\partial \mathbf{u}(k-1)}{\partial w_{ij}^h} \quad (5.19)$$

$$\frac{\partial \mathbf{u}(k)}{\partial w_{ij}^h} = C'(k)\frac{\partial \mathbf{x}(k)}{\partial w_{ij}^h} + \Psi_i(k)\dot{\varphi}(z_i(k))x_j(k) \quad (5.20)$$

where $\Psi_i(k)$ is defined as

$$\Psi_i(k) = [0 \; ... \; 0 \; \dot{\varphi}(v_1(k))w_{1i}^o \; ... \; \dot{\varphi}(v_m(k))w_{mi}^o]^T \quad (5.21)$$

Using Equations 5.19 and 5.20:

$$\frac{\partial \mathbf{x}(k)}{\partial w_{ij}^h} = D'(k-1)\left(C'(k-1)\frac{\partial \mathbf{x}(k-1)}{\partial w_{ij}^h} + \Psi_i(k-1)\dot{\varphi}(z_i(k-1))x_j(k-1)\right) \quad (5.22)$$

By induction, we obtain a closed-form expression similar to Equation 5.16:

$$\frac{\partial \mathbf{x}(k)}{\partial w^o_{ij}} = \sum_{p=0}^{k-1} D'(k-1)C'(k-1)...D'(p+1)C'(p+1)D'(p)\Psi_i(p)\dot{\varphi}(z_i(p))x_j(p) \tag{5.23}$$

Using this equation:

$$\frac{\partial E}{\partial w^h_{ij}} = (\mathbf{x}(K)-\mathbf{d})^T \sum_{p=0}^{K-1} D'(K-1)C'(K-1)...D'(p+1)C'(p+1)D'(p)\Psi_i(p)\dot{\varphi}(z_i(p))x_j(p) \tag{5.24}$$

for $i = 1, ..., N$ and $j = 1, ..., n$. Similarly, for $i = 1, ..., N$:

$$\frac{\partial E}{\partial w^h_{bi}} = (\mathbf{x}(K)-\mathbf{d})^T \sum_{p=0}^{K-1} D'(K-1)C'(K-1)...D'(p+1)C'(p+1)D'(p)\Psi_i(p)\dot{\varphi}(z_i(p)) \tag{5.25}$$

Having Equations 5.17, 5.18, 5.24 and 5.25 to determine partial derivatives of the final error with respect to the weights of the neural network NN_C, we can apply Equation 5.4 to train NN_C.

5.2 Application to the Path Generation Problem

We will use a path generator neural network similar to the one in Section 4.2 with the block diagram given in Figure 3.5. The path generator is again desired to give the type of the next curve piece to be tracked, given the last position (x, y) and the last direction θ reached. This time, a 3-input-1-output 2-layer neural network with structure given in Figure 2.2(a) will be used for path generation. The number of neurons in the hidden layer of the neural network is $N = 100$.

The neural network will be trained using the multi-stage neural network approach explained in the previous section. Our stopping criterion is occurrence of either reaching a point with $x \leq 0$ or proceeding for 50 steps. For a given $(x,\ y,\ \theta)$-triple in the training set, let the stopping criterion occur in the Kth step. The variables in our case are:

$$\mathbf{x}(0) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \quad n = 3, \quad m = 1, \quad N = 100 \tag{5.26}$$

$C(.)$ gives the input-output relation of the path generator neural network:

$$o = C(\mathbf{x}) = \varphi\left(w_{b1}^{o} + \sum_{i=1}^{N} w_{1i}^{o}\varphi\left(w_{bi}^{h} + \sum_{j=1}^{3} w_{ij}^{h}x_j\right)\right) \tag{5.27}$$

C' can be evaluated as:

$$C' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \dot{\varphi}(v_1)\sum_{i=1}^{N} w_{1i}^{o}\dot{\varphi}(z_i)w_{i1}^{h} & \dot{\varphi}(v_1)\sum_{i=1}^{N} w_{1i}^{o}\dot{\varphi}(z_i)w_{i2}^{h} & \dot{\varphi}(v_1)\sum_{i=1}^{N} w_{1i}^{o}\dot{\varphi}(z_i)w_{i3}^{h} \end{bmatrix} \tag{5.28}$$

The output of the neural network NN_C will be discretized by using a 3-state hardlimiter of the form given in Figure 4.7 to give one of the three inputs -1, 0 and 1. Effect of this discretization process is taken into account while evaluating C', and in this evaluation, the 3-state hardlimiter is approximated by the smooth function

$$\frac{1}{2}\left(\varphi_D(x-0.3) + \varphi_D(x+0.3)\right)$$

where $\varphi_D(x) = \tanh(\frac{\lambda_D}{2}x)$ with $\lambda_D = 100$ is a sharp sigmoid, which is emulating a bipolar hardlimiter.

Next, we should find a smooth function $D(.)$ to implement the dynamics giving the next position and direction of the snake's head, given the current values, and the output of the path generator, which indicates the type of the curve piece to be tracked. Hence, for $\mathbf{x} = [x\ y\ \theta]^T$, $D(.)$ should satisfy:

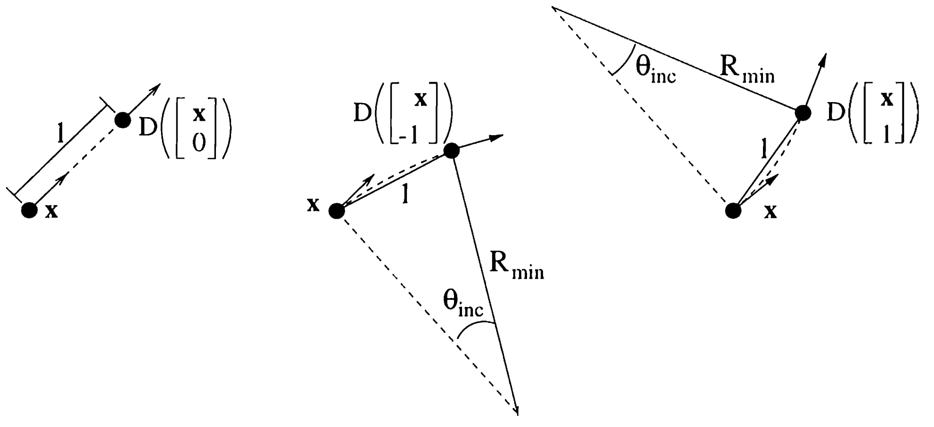


Figure 5.3: $D(.)$ for the path generation problem.

$$D\left(\begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix}\right) = \begin{bmatrix} x + \cos\theta \\ y + \sin\theta \\ \theta \end{bmatrix} \tag{5.29}$$

$$D\left(\begin{bmatrix} \mathbf{x} \\ -1 \end{bmatrix}\right) = \begin{bmatrix} x + R_{min}(\sin\theta - \sin(\theta - \theta_{inc})) \\ y + R_{min}(\cos(\theta - \theta_{inc}) - \cos\theta) \\ (\theta - \theta_{inc})_{2\pi} \end{bmatrix} \tag{5.30}$$

$$D\left(\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}\right) = \begin{bmatrix} x + R_{min}(\sin(\theta + \theta_{inc}) - \sin\theta) \\ y + R_{min}(\cos\theta - \cos(\theta + \theta_{inc})) \\ (\theta + \theta_{inc})_{2\pi} \end{bmatrix} \tag{5.31}$$

where

$$\theta_{inc} = 2\arcsin\left(\frac{1}{2\ R_{min}}\right) \tag{5.32}$$

A smooth candidate for $D(.)$ satisfying these conditions is:

$$D\left(\begin{bmatrix} \mathrm{x} \\ o \end{bmatrix}\right) = \begin{bmatrix} x + R_{min}o(\sin(\theta + o\theta_{inc}) - \sin\theta) + \cos(o\frac{\pi}{2})\cos\theta \\ y + R_{min}o(\cos\theta - \cos(\theta + o\theta_{inc})) + \cos(o\frac{\pi}{2})\sin\theta \\ (\theta + o\theta_{inc})_{2\pi} \end{bmatrix} \tag{5.33}$$

To implement $(.)_{2\pi}$, NN_{mod} pictured in Figure 3.11(b) is used. For smoothness, instead of the hardlimiter of this neural network, the sharp sigmoid $\varphi_D(x)$ defined above will be used in the emulator. So for the smooth emulator:

$$D\left(\begin{bmatrix} \mathrm{x} \\ o \end{bmatrix}\right) = \begin{bmatrix} x + R_{min}o(\sin(\theta + o\theta_{inc}) - \sin\theta) + \cos(o\frac{\pi}{2})\cos\theta \\ y + R_{min}o(\cos\theta - \cos(\theta + o\theta_{inc})) + \cos(o\frac{\pi}{2})\sin\theta \\ \theta + o\theta_{inc} - \pi - \pi\varphi_D(\theta + o\theta_{inc} - 2\pi) \end{bmatrix} \tag{5.34}$$

and D' is derived as:

$$D' = \begin{bmatrix} 1 & 0 & R_{min}o(c_{\theta+o\theta_{inc}} - c_\theta) - c_{o\frac{\pi}{2}}s_\theta & R_{min}(s_{\theta+o\theta_{inc}} - s_\theta + o\theta_{inc}c_{\theta+o\theta_{inc}}) - \frac{\pi}{2}s_{o\frac{\pi}{2}}c_\theta \\ 0 & 1 & R_{min}o(s\theta + o\theta_{inc} - s_\theta) + c_{o\frac{\pi}{2}}c_\theta & R_{min}(c_\theta - c_{\theta+o\theta_{inc}} + o\theta_{inc}s_{\theta+o\theta_{inc}}) - \frac{\pi}{2}s_{o\frac{\pi}{2}}s_\theta \\ 0 & 0 & 1 - \pi\varphi'_D(\theta + o\theta_{inc} - 2\pi) & \theta_{inc} - \theta_{inc}\pi\varphi'_D(\theta + o\theta_{inc} - 2\pi) \end{bmatrix} \tag{5.35}$$

where c_ϕ denotes $\cos\phi$ and s_ϕ denotes $\sin\phi$.

5.3 Results

For the selection of the training sets, different methods can be used. One method is to choose the training set randomly in a small area near the origin at first, and then to enlarge the domain as training of the neural network progresses.

As another method, for a fixed number n of steps, an initial triple in the training set can be determined by moving n steps back from the desired final state randomly. For each step there are three choices to move back, one for each of the three piece types defined. Similar to the first method, the number n can be increased as the training progresses.

To have successful results using these methods, learning constant η should be adjusted during the training process carefully according to the possible number of steps to be taken for a triple in the training set. This is because of the accumulation of the multiplicative terms in Equations 5.17, 5.18, 5.24 and 5.25.

Using a fixed learning constant, we obtained the best results by choosing the training triples in the set $0 < x \leq 4$, $-2 \leq y \leq 2$, $\frac{3\pi}{4} \leq \theta \leq \frac{5\pi}{4}$. We have composed a training set by 200 such triples. The neural network is trained for 5000 epoches. The average error during the training process is plotted in 5.4.

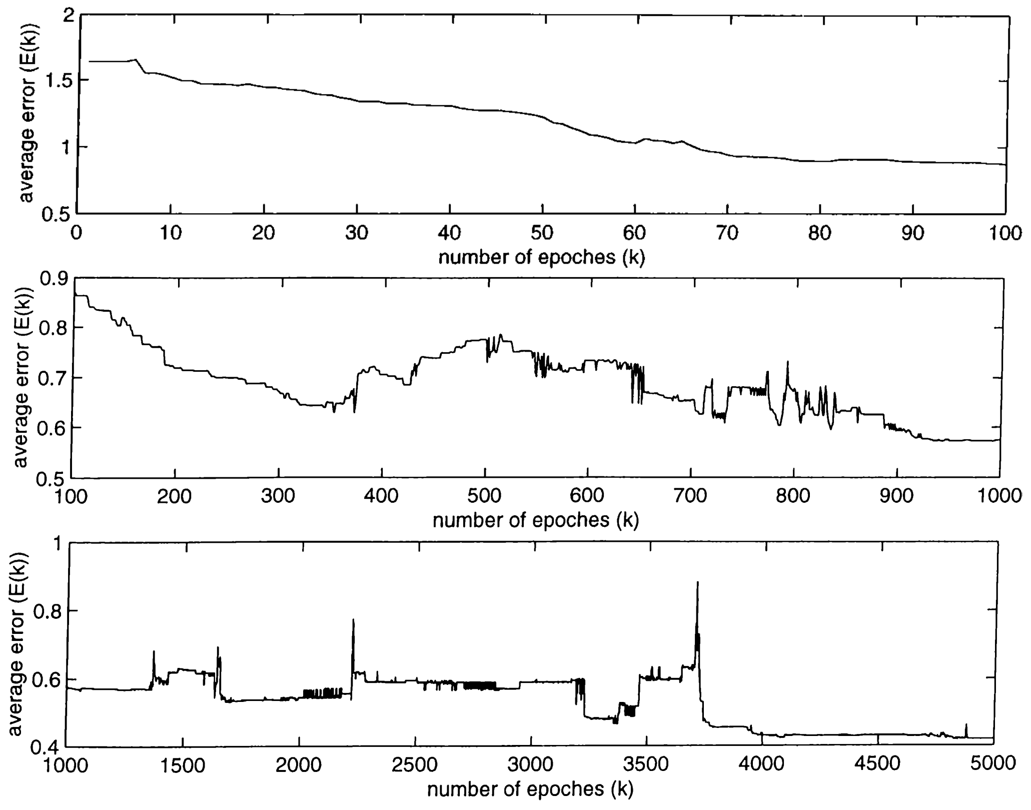


Figure 5.4: Average error during the training process.

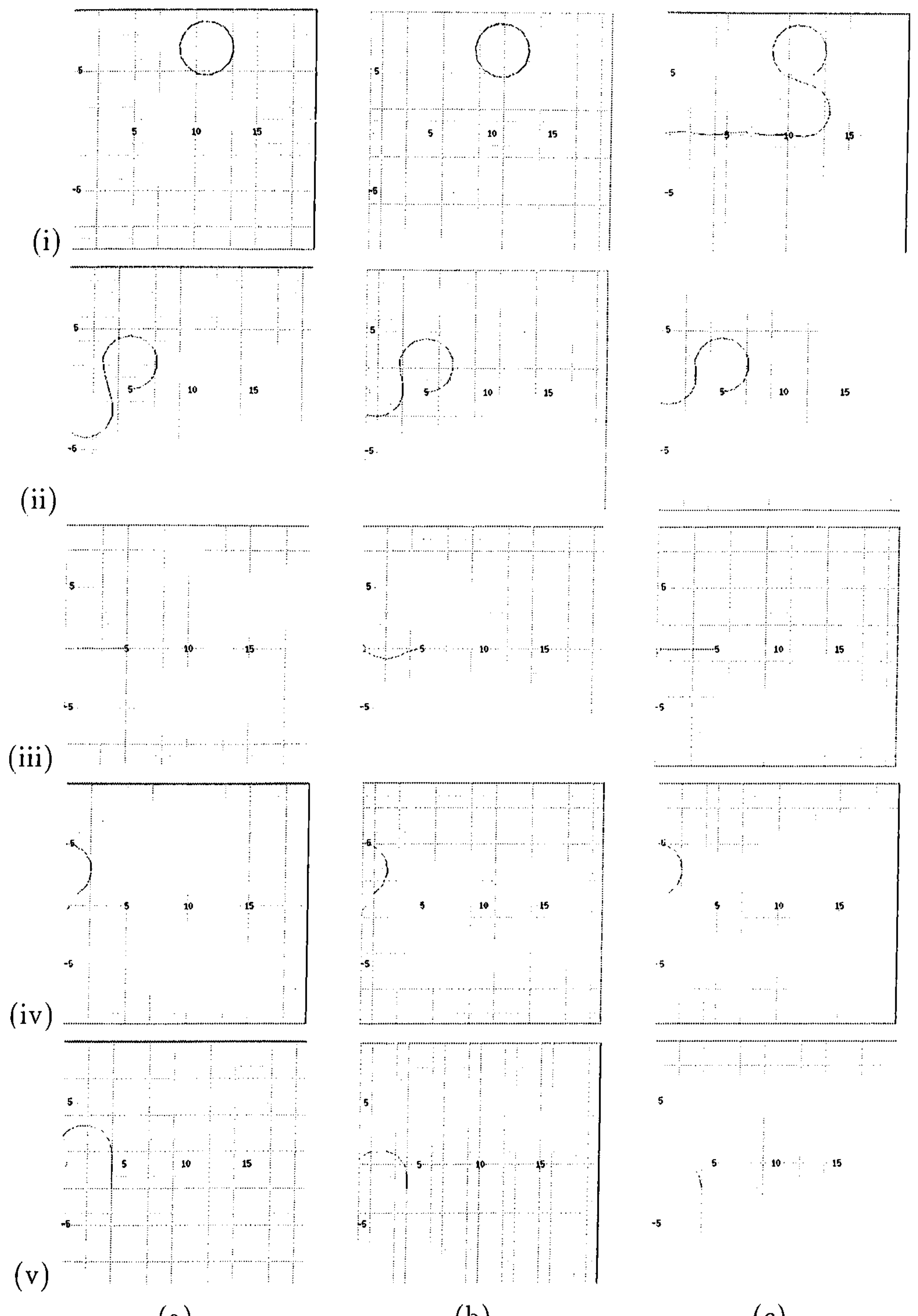


Figure 5.5: Outputs for the neural network weights of epoches (a) 100, (b) 1000 and (c) 5000 for the initial triples (i) $x = 12$, $y = 5$, $\theta = 30°$, (ii) $x = 5$, $y = 0$, $\theta = 0°$, (iii) $x = 5$, $y = 0$, $\theta = 180°$, (iv) $x = 1$, $y = 1$, $\theta = 30°$, (v) $x = 4$, $y = -2$, $\theta = 90°$.

As in Chapter 4, we have checked the path generation performance of the neural network for the weights of epoches 100, 1000, 5000, using the code *pathnnm.cpp*. The outputs obtained using these weights in the neural network for the same five sample initial triples are pictured in Figure 5.5. As can be seen on the results after epoch 5000, the neural network has a quite good generalization in the entire domain of interest. So the performance of the neural network is good enough for training with a fixed learning constant. This performance can be improved, by using the first two methods described above with adaptive selection of the learning constant.

Chapter 6

CONCLUSION

In this work, we have built up a neural network, which is composed of only weights, adders and simple activation units and which contains no software, to control a highly nonlinear task, motion planning of a mechanical snake composed of eight links. This neural network is constructed by cascading two separate neural networks, one for path generation and the other for tracking the generated path.

Three different approaches are used to construct these neural networks. To construct the neural network for tracking the generated path, a simple algorithm is developed to implement the tracking task efficiently, after fixing the form of the paths to be generated. Later, structure of the neural network is determined and the weights in this structure are evaluated directly to implement the developed algorithm.

Construction of the neural network for path generation is approached in two different ways. In the first approach, an efficient algorithm to find minimal smooth paths with fixed maximum curvature lying in the right half plane, beginning at an arbitrary point with an arbitrary tangent direction and ending at the origin with a specified tangent direction is developed. Each minimal path is approximated by a sequence of three fixed types of smooth curve pieces with smooth connections. A 3-layer neural network is trained using Back Propagation with Momentum to give the type of the next piece to be tracked in such a sequence for a given point and direction.

The average error in giving the type of the piece to be tracked decreases as the training process advances, but we have obtained the best performance in path generation for sample initial points by using the weights of epoch 9000, although the neural network is trained for 45000 epoches. This is because the approximation to discretize the minimal path (so that it can be tracked by the mechanical snake) is not robust. Unexpected subpaths can be seen especially for the points close to the boundary of regions R_{-1} and R_1, which are defined in Chapter 4. However, the complete neural network having the weights obtained after epoch 9000 for the path generator part gives acceptable results, which are very close to the desired outputs in general.

In the second approach to the construction of the path generator, multi-stage neural networks are used. We have first formulated the intuitive approach of Nguyen and Widrow to use Back Propagation in multi-stage neural networks [6, 7], and then used this formulation to train a 2-layer neural network in a small area close to the origin using a fixed learning constant. We have obtained a quite good generalization in the entire domain of interest. In spite of some problematic cases, the results of the second approach are also acceptable in general.

We have proposed two different methods to improve the results in the second approach: Enlarging the domain of the training set elements as the training process progresses, and determining the elements of the training set by moving n random steps back from the desired final state, where n is increased as training advances. Because of the accumulation of the multiplicative terms in the equations of Back Propagation for multi-stage neural networks, fixing a suitable learning constant to apply Back Propagation is not feasible for these two methods. A topic of further research is the development of the application of the multi-stage neural network approach using an adaptive learning constant.

APPENDIX A

Back Propagation with Momentum

The weight-update equation of Back Propagation with Momentum for Step n of a training process in general is given by

$$\Delta w(n) \;=\; -\eta\frac{\partial E(n)}{\partial w}+\alpha\Delta w(n-1) \qquad (A.1)$$

where $\eta > 0$ and $0 < \alpha < 1$ are constants, and $\Delta w(0) = 0$. In this appendix, the explicit form of this equation for 2-layer and 3-layer neural networks will be derived.

A.1 2-Layer Neural Networks

For the 2-layer neural network given in Figure 2.2(a), the error to be back propagated is

$$E = \frac{1}{2}\,\|\vec{d}-\vec{o}\|^2 = \frac{1}{2}\sum_{i=1}^{m}(d_i - o_i)^2 \qquad (A.2)$$

The partial derivatives of E with respect to weights of the neural network can be computed as follows:

$$\begin{aligned}
\frac{\partial E(n)}{\partial w^o_{ij}} &= -(d_i(n)-o_i(n))\dot{\varphi}(v_i(n))y_j(n) \ \text{ for } i=1,2,...,m;\ j=1,2,...,N \\
\frac{\partial E(n)}{\partial w^o_{bi}} &= -(d_i(n)-o_i(n))\dot{\varphi}(v_i(n)) \ \text{ for } i=1,2,...,m \\
\frac{\partial E(n)}{\partial w^h_{ij}} &= \sum_{l=1}^{m}\frac{\partial E(n)}{\partial v_l}\frac{\partial v_l}{\partial y_i}\frac{\partial y_i}{\partial z_i}\frac{\partial z_i}{\partial w^h_{ij}} \\
&= -\sum_{l=1}^{m}(d_l(n)-o_l(n))\dot{\varphi}(v_l(n))w^o_{li}(n)\dot{\varphi}(z_i(n))x_j(n) \\
&\quad \text{for } i=1,2,...,N;\ j=1,2,...,k \\
\frac{\partial E(n)}{\partial w^h_{bi}} &= -\sum_{l=1}^{m}(d_l(n)-o_l(n))\dot{\varphi}(v_l(n))w^o_{li}(n)\dot{\varphi}(z_i(n)) \ \text{ for } i=1,2,...,N
\end{aligned}$$

Using these equations, we obtain the explicit weight-update equations as follows:

$$\begin{aligned}
\delta^o_i(n) &= (d_i(n)-o_i(n))\dot{\varphi}(v_i(n)) \ \text{ for } i=1,2,...,m && (A.3) \\
\Delta w^o_{ij}(n) &= \eta\delta^o_i(n)y_j(n)+\alpha\Delta w^o_{ij}(n-1) && \\
&\quad \text{for } i=1,2,...,m;\ j=1,2,...,N && (A.4) \\
\Delta w^o_{bi}(n) &= \eta\delta^o_i(n)+\alpha\Delta w^o_{bi}(n-1) \ \text{ for } i=1,2,...,m && (A.5) \\
\delta^h_i(n) &= \sum_{l=1}^{m}\delta^o_l(n)w^o_{li}(n)\dot{\varphi}(z_i(n)) \ \text{ for } i=1,2,...,N && (A.6) \\
\Delta w^h_{ij}(n) &= \eta\delta^h_i(n)x_j(n)+\alpha\Delta w^h_{ij}(n-1) && \\
&\quad \text{for } i=1,2,...,N;\ j=1,2,...,k && (A.7) \\
\Delta w^h_{bi}(n) &= \eta\delta^h_i(n)+\alpha\Delta w^h_{bi}(n-1) \ \text{ for } i=1,2,...,N && (A.8)
\end{aligned}$$

A.2 3-Layer Neural Networks

The error to be back propagated is of the same form as that for the 2-layer neural network. The partial derivatives of this error with respect to the weights of the neural network in Figure 2.2(b) are as follows:

$$\begin{aligned}
\frac{\partial E(n)}{\partial w^o_{ij}} &= -(d_i(n) - o_i(n))\dot{\varphi}(v_i(n))y_{2j}(n) \text{ for } i = 1,2,...,m;\ j = 1,2,...,N_2 \\
\frac{\partial E(n)}{\partial w^o_{bi}} &= -(d_i(n) - o_i(n))\dot{\varphi}(v_i(n)) \text{ for } i = 1,2,...,m \\
\frac{\partial E(n)}{\partial w^{h2}_{ij}} &= \sum_{l=1}^{m} \frac{\partial E(n)}{\partial v_l}\frac{\partial v_l}{\partial y_{2i}}\frac{\partial y_{2i}}{\partial z_{2i}}\frac{\partial z_{2i}}{\partial w^{h2}_{ij}} \\
&= -\sum_{l=1}^{m}(d_l(n) - o_l(n))\dot{\varphi}(v_l(n))w^o_{li}(n)\dot{\varphi}(z_{2i}(n))y_{1j}(n) \\
&\quad \text{for } i = 1,2,...,N_2;\ j = 1,2,...,N_1 \\
\frac{\partial E(n)}{\partial w^{h2}_{bi}} &= -\sum_{l=1}^{m}(d_l(n) - o_l(n))\dot{\varphi}(v_l(n))w^o_{li}(n)\dot{\varphi}(z_{2i}(n)) \text{ for } i = 1,2,...,N_2 \\
\frac{\partial E(n)}{\partial w^{h1}_{ij}} &= \sum_{p=1}^{m}\left(\frac{\partial E(n)}{\partial v_p}\sum_{q=1}^{N_2}\frac{\partial v_p}{\partial y_{2q}}\frac{\partial y_{2q}}{\partial z_{2q}}\frac{\partial z_{2q}}{\partial y_{1i}}\frac{\partial y_{1i}}{\partial z_{1i}}\frac{\partial z_{1i}}{\partial w^{h1}_{ij}}\right) \\
&= -\sum_{p=1}^{m}\left((d_p(n) - o_p(n))\dot{\varphi}(v_p(n))\sum_{q=1}^{N_2} w^o_{pq}(n)\dot{\varphi}(z_{2q}(n))w^{h2}_{qi}(n)\dot{\varphi}(z_{1i}(n))x_j(n)\right) \\
&\quad \text{for } i = 1,2,...,N_1;\ j = 1,2,...,k \\
\frac{\partial E(n)}{\partial w^{h1}_{bi}} &= -\sum_{p=1}^{m}\left((d_p(n) - o_p(n))\dot{\varphi}(v_p(n))\sum_{q=1}^{N_2} w^o_{pq}(n)\dot{\varphi}(z_{2q}(n))w^{h2}_{qi}(n)\dot{\varphi}(z_{1i}(n))\right) \\
&\quad \text{for } i = 1,2,...,N_1
\end{aligned}$$

Using these equations, we obtain the explicit weight-update equations as follows:

$$\begin{aligned}
\delta^o_i(n) &= (d_i(n) - o_i(n))\dot{\varphi}(v_i(n)) \text{ for } i = 1,2,...,m \qquad (A.9) \\
\Delta w^o_{ij}(n) &= \eta\delta^o_i(n)y_{2j}(n) + \alpha\Delta w^o_{ij}(n-1)
\end{aligned}$$

$$\text{for } i = 1, 2, ..., m;\ j = 1, 2, ..., N_2 \tag{A.10}$$

$$\Delta w_{bi}^{o}(n) = \eta \delta_i^o(n) + \alpha \Delta w_{bi}^{o}(n-1) \quad \text{for } i = 1, 2, ..., m \tag{A.11}$$

$$\delta_i^{h2}(n) = \sum_{l=1}^{m} \delta_l^o(n) w_{li}^o(n) \dot{\varphi}(z_{2i}(n)) \quad \text{for } i = 1, 2, ..., N_2 \tag{A.12}$$

$$\Delta w_{ij}^{h2}(n) = \eta \delta_i^{h2}(n) y_{1j}(n) + \alpha \Delta w_{ij}^{h2}(n-1)$$

$$\text{for } i = 1, 2, ..., N_2;\ j = 1, 2, ..., N_1 \tag{A.13}$$

$$\Delta w_{bi}^{h2}(n) = \eta \delta_i^{h2}(n) + \alpha \Delta w_{bi}^{h2}(n-1) \quad \text{for } i = 1, 2, ..., N_2 \tag{A.14}$$

$$\delta_i^{h1}(n) = \sum_{l=1}^{N_2} \delta_l^{h2}(n) w_{li}^{h2}(n) \dot{\varphi}(z_{1i}(n)) \quad \text{for } i = 1, 2, ..., N_1 \tag{A.15}$$

$$\Delta w_{ij}^{h1}(n) = \eta \delta_i^{h1}(n) x_j(n) + \alpha \Delta w_{ij}^{h1}(n-1)$$

$$\text{for } i = 1, 2, ..., N_1;\ j = 1, 2, ..., k \tag{A.16}$$

$$\Delta w_{bi}^{h1}(n) = \eta \delta_i^{h1}(n) + \alpha \Delta w_{bi}^{h1}(n-1) \quad \text{for } i = 1, 2, ..., N_1 \tag{A.17}$$

APPENDIX B

Proofs of the Facts on Tracking the Generated Path

B.1 Proof of Fact 3.1

In Figure B.1:

$$\sin\beta = \frac{1}{2R_{min}} \Rightarrow \beta = \arcsin\left(\frac{1}{2R_{min}}\right) \tag{B.1}$$

and

$$\gamma + 2\beta = 180° \tag{B.2}$$

Since $K_{-1}L$ and LM_{-1} are type -1, K_0L and LM_0 are type 0, and K_1L and LM_1 are type 1 curve pieces, where the direction for a curve piece AB is taken to be from A to B; the couple $(P_i,\ P_{i+1})$ coincides with the couple $(\overrightarrow{M_{P_i}L},\ \overrightarrow{LK_{P_{i+1}}})$. So ϕ_i is given by the directional angle from $\overrightarrow{M_{P_i}L}$ to $\overrightarrow{LK_{P_{i+1}}}$. So we can obtain the following table from the figure using Equations B.1 and

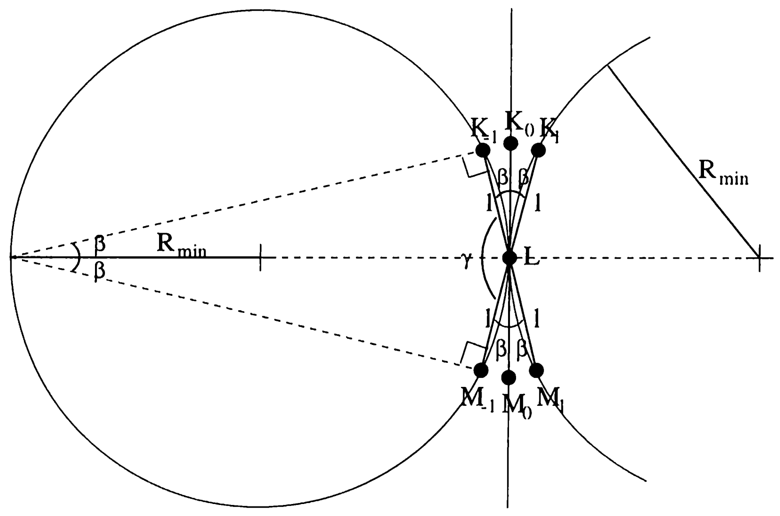


Figure B.1: Fact 3.1.

B.2:

P_i	-1	-1	-1	0	0	0	1	1	1
P_{i+1}	-1	0	1	-1	0	1	-1	0	1
ϕ_i	2β	β	0	β	0	$-\beta$	0	$-\beta$	-2β

If we summarize this table in closed form, we get the result we are looking for:

$$\phi_i = -\beta(P_i + P_{i+1}) \tag{B.3}$$

□

B.2 Proof of Fact 3.2

Using Fact 3.1, we get:

$$\phi_k = -\beta(P_k + P_{k+1}) \tag{B.4}$$

$$\phi_{k+1} = -\beta(P_{k+1} + P_{k+2}) \tag{B.5}$$

$$\phi_{k+2} = -\beta(P_{k+2} + P_{k+3}) \tag{B.6}$$

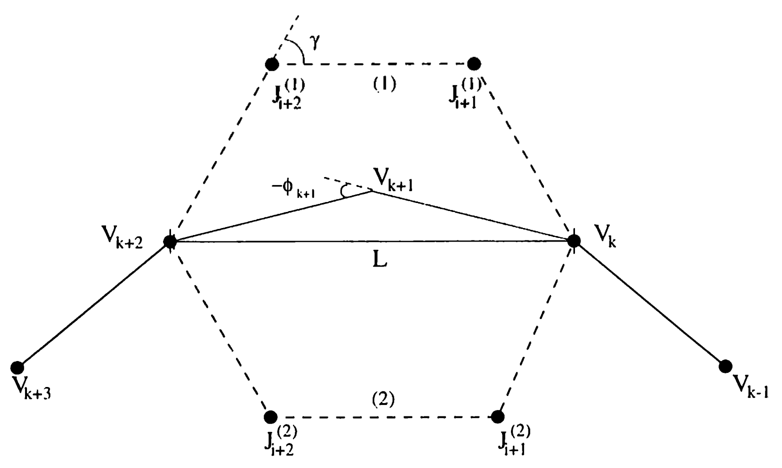


Figure B.2: Fact 3.2.

In Figure B.2:

$$L = 2\cos\left(\frac{|\phi_{k+1}|}{2}\right) \tag{B.7}$$

and

$$\gamma = \arccos\left(\frac{L-1}{2}\right) = \arccos\left(\cos\left(\frac{|\phi_{k+1}|}{2}\right) - \frac{1}{2}\right) \tag{B.8}$$

Using Equation B.5:

$$\gamma = \arccos\left(\cos\left(\frac{\beta|P_{k+1} + P_{k+2}|}{2}\right) - \frac{1}{2}\right) \tag{B.9}$$

So, defining α_j as

$$\alpha_j = \arccos\left(\cos\left(\frac{\beta j}{2}\right) - \frac{1}{2}\right) \tag{B.10}$$

we get

$$\gamma = \alpha_{|P_{k+1}+P_{k+2}|} \tag{B.11}$$

There are two candidates for the configuration of the sequence Segment $i+1$, Segment $i+2$, Segment $i+3$ as pictured in Figure B.2. For Candidate (1):

$$\theta_{i+1} = \theta_{i+2} = \gamma = \alpha_{|P_{k+1}+P_{k+2}|} \tag{B.12}$$

$$\begin{aligned} \theta_i &= \phi_k + \angle V_{k+1}V_kV_{k+2} - \gamma \\ &= \phi_k + \frac{\phi_{k+1}}{2} - \gamma \end{aligned} \tag{B.13}$$

Using Equations B.4 and B.5, we get:

$$\theta_i = -\beta(P_k + \frac{3}{2}P_{k+1} + \frac{1}{2}P_{k+2}) - \alpha_{|P_{k+1}+P_{k+2}|} \tag{B.14}$$

Similarly:

$$\begin{aligned} \theta_{i+3} &= \phi_{k+2} + \angle V_{k+1}V_{k+2}V_k - \gamma \\ &= \phi_{k+2} + \frac{\phi_{k+1}}{2} - \gamma \\ &= -\beta(\frac{1}{2}P_{k+1} + \frac{3}{2}P_{k+2} + P_{k+3}) - \alpha_{|P_{k+1}+P_{k+2}|} \end{aligned} \tag{B.15}$$

We can proceed in the same fashion for Candidate (2):

$$\begin{aligned} \theta_{i+1} = \theta_{i+2} &= -\gamma = -\alpha_{|P_{k+1}+P_{k+2}|} \\ \theta_i &= \phi_k + \angle V_{k+1}V_kV_{k+2} + \gamma \end{aligned} \tag{B.16}$$

$$
\begin{aligned}
&= -\beta(P_k + \frac{3}{2}P_{k+1} + \frac{1}{2}P_{k+2}) \;+\; \alpha_{|P_{k+1}+P_{k+2}|} && \text{(B.17)} \\
\theta_{i+3} &= \phi_{k+2} + \angle V_{k+1} V_{k+2} V_k + \gamma \\
&= -\beta(\frac{1}{2}P_{k+1} + \frac{3}{2}P_{k+2} + P_{k+3}) \;+\; \alpha_{|P_{k+1}+P_{k+2}|} && \text{(B.18)}
\end{aligned}
$$

□

APPENDIX C

Determination of κ_{max}

To determine a specific value for the maximum curvature, κ_{max} (minimum radius of curvature, R_{min}), we will use the following constraints:

1. The interior joint angles are bounded by 63°: $|\theta_i| \leq 63°$ for $i = 2, 3, 4, 5, 6$.
2. $|J_iJ_{i+3}| \geq 2.8$ for any sequence J_i, J_{i+1}, J_{i+2}, J_{i+3} of joints all of which are on the path to be tracked.
3. $|J_iJ_{i+3}| \leq 2.2$ for any sequence J_i, J_{i+1}, J_{i+2}, J_{i+3} of joints in which J_{i+1} and J_{i+2} are not on the path to be tracked.

To analyze the effect of the second constraint, let us consider the possible case pictured in Figure C.1, in which J_i, J_{i+1}, J_{i+2}, J_{i+3} are on a circular arc of radius R_{min}. To satisfy the constraint:

$$|J_iJ_{i+3}| \geq 2.8 \quad \text{(C.1)}$$
$$\Rightarrow |J_iK| = |LJ_{i+3}| \geq 0.9 \quad \text{(C.2)}$$
$$\Rightarrow \alpha \leq \arccos(0.9) \quad \text{(C.3)}$$

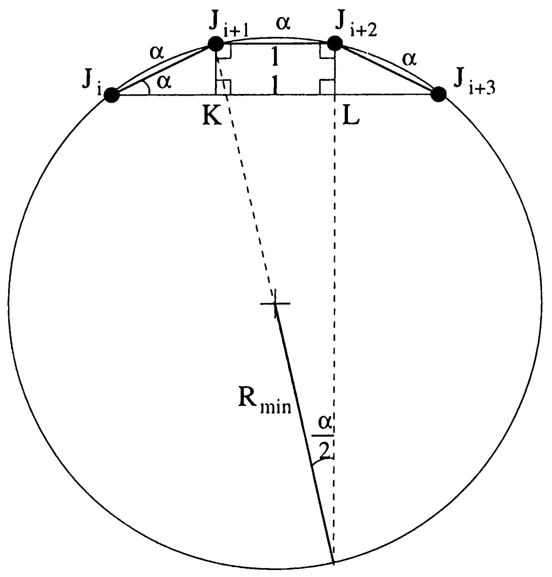


Figure C.1: J_i, J_{i+1}, J_{i+2}, J_{i+3} on a circular arc of radius R_{min}.

$$\Rightarrow \quad 2\arcsin\left(\frac{1}{2R_{min}}\right) \leq \arccos(0.9) \qquad (C.4)$$

$$\Rightarrow \quad R_{min} \geq \frac{0.5}{\sin(0.5\arccos(0.9))} \qquad (C.5)$$

$$\Rightarrow \quad R_{min} \geq 2.236 \quad (\kappa_{max} \leq 0.447) \qquad (C.6)$$

So, we have found a lower boundary for R_{min}. If we can show that $R_{min} = 2.236$, the minimum possible value of R_{min}, satisfies the second constraint for all other possible cases and that it does not violate the other two constraints as well, R_{min} will be determined as this specific value. For $R_{min} = 2.236$:

To complete the analysis of the second constraint:

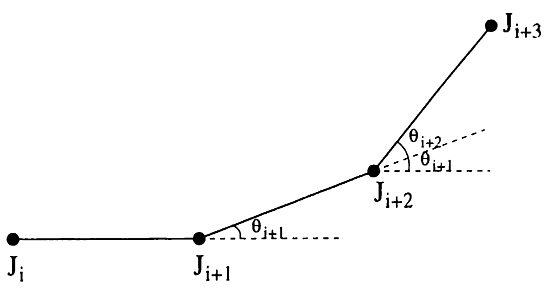


Figure C.2: Computation of $|J_iJ_{i+3}|$.

$$\begin{aligned}
|J_iJ_{i+3}|^2 &= (1+\cos\theta_{i+1}+\cos(\theta_{i+1}+\theta_{i+2}))^2+(\sin\theta_{i+1}+\sin(\theta_{i+1}+\theta_{i+2}))^2 \\
&= 1+\cos^2\theta_{i+1}+\cos^2(\theta_{i+1}+\theta_{i+2}) \\
&\quad +2\,(\cos\theta_{i+1}+\cos(\theta_{i+1}+\theta_{i+2})+\cos\theta_{i+1}\cos(\theta_{i+1}+\theta_{i+2})) \\
&\quad +\sin^2\theta_{i+1}+\sin^2(\theta_{i+1}+\theta_{i+2})+2\sin\theta_{i+1}\sin(\theta_{i+1}+\theta_{i+2}) \\
&= 3+2\,(\cos\theta_{i+1}+\cos\theta_{i+2}+\cos(\theta_{i+1}+\theta_{i+2})) \qquad \text{(C.7)}
\end{aligned}$$

Because of Corollary 3.1,

$$-2\beta \;\le\; \theta_{i+1},\ \theta_{i+1} \;\le\; 2\beta \qquad \text{(C.8)}$$

where

$$\beta \;=\; \arcsin\left(\frac{1}{2R_{min}}\right) \;=\; 12.921^\circ \qquad \text{(C.9)}$$

in our case. So absolute values of θ_{i+1}, θ_{i+2} and $\theta_{i+1}+\theta_{i+2}$ are bounded by 51.92°, which is smaller than 90°. So using Equation C.7:

$$|J_iJ_{i+3}|^2 \;\ge\; 3+2\,(\cos(2\beta)+\cos(2\beta)+\cos(4\beta)) \qquad \text{(C.10)}$$

Noting that the summation on the right gives $|J_iJ_{i+3}|^2$ for the case in which J_i, J_{i+1}, J_{i+2}, J_{i+3} are on a circular arc of radius R_{min}, that is this summation is equal to 2.8^2, we get

$$|J_iJ_{i+3}| \;\ge\; 2.8 \qquad \text{(C.11)}$$

for every possible sequence J_i, J_{i+1}, J_{i+2}, J_{i+3}. So the second constraint is satisfied.

To analyze the first constraint:

For $j = 2, 3, 4, 5, 6$, J_j can be either a joint connecting two segments which lie on two successive pieces of 7-piece-representation of the snake or one of the joints J_i, J_{i+1}, J_{i+2}, J_{i+3} of the case of Fact 3.2.

For the first case, because of Corollary 3.1:

$$|\theta_j| \leq 2\beta = 25.84° < 63° \tag{C.12}$$

For the second case, if J_j corresponds to J_{i+1} or J_{i+2}, because of Fact 3.2:

$$|\theta_j| \leq \max(\alpha_0, \alpha_1, \alpha_2) \tag{C.13}$$

If J_j corresponds to J_i or J_{i+3}, since θ_j is an interior angle, the choice among the candidates given in Fact 3.2 will be so that the sign in front of the α-term is opposite to the sign of the rest of the expression for that angle. So the inequality given in C.13 is valid for this situation as well.

Having

$$\begin{aligned}
\alpha_0 &= \arccos(\cos(0°) - 0.5) = 60° && \text{(C.14)}\\
\alpha_1 &= \arccos(\cos(\beta/2) - 0.5) = 60.42° && \text{(C.15)}\\
\alpha_2 &= \arccos(\cos(\beta) - 0.5) = 61.66° && \text{(C.16)}
\end{aligned}$$

we get

$$|\theta_j| \leq 61.66° < 63° \tag{C.17}$$

for the second case. Thus the first constraint is also satisfied for all possible cases.

The third constraint will be eventually satisfied, since J_i and J_{i+3} are always at the ends of two consecutive pieces of the 7-piece-representation in our design, and the distance $|J_iJ_{i+3}|$ can be at most 2.

Since all of the three constraints are satisfied we can fix our parameters as:

$$R_{min} = 2.236 \quad \text{and} \quad \kappa_{max} = 0.447 \tag{C.18}$$

APPENDIX D

C^{++} Codes

This appendix includes the C^{++} codes used to implement the training algorithms in the thesis, to show the generated paths and to simulate motion of the mechanical snake. The codes are given on the attached disk. Descriptions of these codes are as follows:

basicnn.h In this header file, basic constants used in the neural networks are defined. Bipolar, unipolar and 3-state hardlimiters are implemented. Implementation of the first hardlimiter is done by directly defining its function, and the other two hardlimiters are implemented as simple neural networks with bipolar hardlimiters as activators.

constants.h General mathematical constants and a structure for points in R^2, which are used in the other codes, are defined in this header file.

curvat.h This header file is composed of functions generating random numbers, random number sequences and paths with a given maximum curvature.

make*file* This make file compiles *file.cpp* and produces the executable file *file*.

nn3layer.cpp This file is written to realize the training process in Chapter 4 using the training set generated by *nn3trset.cpp*.

nn3layer.h This header file defines a class to represent 3-layer neural networks. Functions to train 3-layer neural networks, to show the parameters of these networks and to run them are included in this class.

nn3trset.cpp This code produces the training set for the training process in Chapter 4.

nnmult.cpp This file is written to realize the training process in Chapter 5 using the training set generated by *nnmultr.cpp*.

nnmult.h This header file defines a class to represent 2-layer neural networks. Functions to train 2-layer neural networks using multi-stage neural network approach, to show the parameters of these networks and to run them are included in this class.

nnmultr.cpp This code produces the training set for the training process in Chapter 5.

path.cpp This program is written to show the outputs of Algorithm 4.1 on the screen. Inputs `x`, `y` and `theta` (in degrees) required by the program determine the initial position and direction. `CP[`*i*`]` on the output screen gives the value of CP_i in Algorithm 4.1.

path.h This header file includes the C^{++} code of Algorithm 4.1.

pathdy.h Functions to show the paths generated by NN_{path} of Chapter 4 are collected in this header file.

pathdym.h Functions to show the paths generated by NN_{path} of Chapter 5 are collected in this header file.

pathnn.cpp This program is written to show the paths generated by NN_{path} of Chapter 4 on the screen. Paths are drawn as sequences of line segments which are either curve pieces of type 0 themselves or chords of the pieces of type -1 or 1. Inputs `x`, `y` and `theta` (in degrees) required by the program determine the initial position and direction.

pathnn.h This header file implements NN_{path} of Chapter 4 and its components.

pathnn_d.cpp This program shows the desired paths obtained using the original forms of R_{-1}, R_0 and R_1,which are defined in Chapter 4. Paths are drawn as sequences of line segments which are either curve pieces of type 0 themselves or chords of the pieces of type -1 or 1. Inputs `x`, `y` and `theta` (in degrees) required by the program determine the initial position and direction.

pathnn_d.h Functions to construct and to show the paths of program *pathnn_d.cpp* are collected in this header file.

pathnnm.cpp This program is written to show the paths generated by NN_{path} of Chapter 5 on the screen. Paths are drawn as sequences of line segments which are either curve pieces of type 0 themselves or chords of the pieces of type -1 or 1. Inputs `x`, `y` and `theta` (in degrees) required by the program determine the initial position and direction.

pathnnm.h This header file implements NN_{path} of Chapter 5.

tetanng.cpp This program shows the outputs of NN_{path} of Chapter 4 for the points in the domain of interest on a collection of constant-θ-planes. The output is -1 for red points, 0 for green points and 1 for blue points.

tetaplg.cpp This program shows the desired outputs used in Chapter 4 for the points in the domain of interest on a collection of constant-θ-planes. The output is -1 for red points, 0 for green points and 1 for blue points.

track.h Functions to control the tracking process of program *tracknn_d.cpp* are collected in this header file.

trackdy.h Functions which initialize the configuration of the snake by NN_{track} and which realize the adjustment of the initial configuration using the neural networks NN_{adj1} and NN_{adj2} are collected in this header file.

tracknn.cpp This program is written to show the processes controlled by the complete controller which has the path generator obtained in Chapter 4 on the screen. Weights of NN_G are stored in *wpath.dat*. Inputs `x`, `y` and `theta` (in degrees) required by the program determine the initial position and direction. User of the program can see the generated path, the tracking process, the path and the tracking process together, or adjustment of the mechanical snake's initial configuration by selecting the related item on the menu. Generated paths are drawn as sequences of line segments which are either curve pieces of type 0 themselves or chords of the pieces of type -1 or 1.

tracknn.h This header file implements the neural networks NN_{adj1} and NN_{adj2}.

tracknn_d.cpp This program shows the desired outputs of the complete controller using the original forms of R_{-1}, R_0 and R_1,which are defined in Chapter 4. Inputs `x`, `y` and `theta` (in degrees) required by the program determine the initial position and direction. User of the program can see the generated path, the tracking process, or the path and the tracking process together, by selecting the related item on the menu. Generated paths are drawn as sequences of line segments which are either curve pieces of type 0 themselves or chords of the pieces of type -1 or 1.

tracknnm.cpp This program is written to show the processes controlled by the complete controller which has the path generator obtained in Chapter 5 on the screen. Weights of NN_{path} are stored in *wpathm.dat*. Inputs `x`, `y` and `theta` (in degrees) required by the program determine the initial position and direction. User of the program can see the generated path, the tracking process, the path and the tracking process together, or adjustment of the mechanical snake's initial configuration by selecting the related item on the menu. Generated paths are drawn as sequences of line segments which are either curve pieces of type 0 themselves or chords of the pieces of type -1 or 1.

w3layer This directory includes the weights we have obtained for different epoches during the training with multi stage neural network approach. Files *w3l_100.dat*, *w3l_500.dat*, *w3l_n_.dat* include the weights obtained after epoches 100, 500 and n thousand respectively.

wmult This directory includes the weights we have obtained for different epoches during the training with minimal path approach. Files *wm_100.dat*, *wm_500.dat*, *wm_n_.dat* include the weights obtained after epoches 100, 500 and n thousand respectively.

wpath.dat This data file includes the weights of NN_G for *pathnn.cpp* and *tracknn.cpp*.

wpathm.dat This data file includes the weights of NN_{path} for *pathnnm.cpp* and *tracknnm.cpp*.

xvinit.h This header file is used to include the header files necessary for XView programming and to define color constants used in the programs based on XView programming.

yilan.cpp This program demonstrates the basic linear and circular motion patterns of the 8-link mechanical snake. Type of the motion can be selected by using the menu.

yilan.h This header file defines a class to represent the 8-link mechanical snake. Functions to implement motion and sensing of the mechanical snake and to show the configuration of the snake are included in this class.

REFERENCES

[1] B. Widrow and M. A. Lehr "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, pp. 1415–1442, 1990.

[2] G. A. Bekey and K. Y. Goldberg. *Neural Networks in Robotics.* Kluwer Academic Publishers, Norwell, 1993.

[3] S. Haykin. *Neural Networks.* Prentice Hall, Upper Saddle River, 1994.

[4] A. Cichocki and R. Unbehauen. *Neural Networks for Optimization and Signal Processing.* John Wiley & Sons, 1993.

[5] V. R. Vemuri. *Artificial Neural Networks: Concepts and Control Applications.* IEEE Computer Society Press, Los Alamitos, 1992.

[6] D. H. Nguyen and B. Widrow "Neural networks for self-learning control systems," *IEEE Control Systems Magazine*, pp. 18–23, April 1990.

[7] D. Nguyen and B. Widrow "The truck backer-upper," in *International Neural Network Conference*, volume 1, pp. 399–407, Paris, July 1990.

[8] M. W. Spong, F. L. Lewis, and C. T. Abdallah. *Robot Control.* IEEE Press, Piscataway, 1993.

[9] J. M. Zurada. *Introduction to Artificial Neural Systems.* PWS, Boston, 1992.

[10] C. H. Chen. *Fuzzy Logic and Neural Network Handbook.* McGraw-Hill, 1996.

[11] D. Psaltis, A. Sideris, and A. A. Yamamura "A multilayered neural network controller," *IEEE Control Systems Magazine*, pp. 17–21, April 1988.

[12] F. C. Chen "Back-propagation neural networks for nonlinear self-tuning adaptive control," *IEEE Control Systems Magazine*, pp. 44–48, April 1990.

[13] W. T. Miller, R. P. Hewes, F. H. Glanz, and L. G. Kraft "Real-time dynamic control of an industrial manipulator using a neural-network-based learning controller," *IEEE Transactions on Robotics and Automation*, vol. 6, pp. 1–9, 1990.

[14] J. D. Bošković and K. S. Narendra "Comparison of linear, nonlinear and neural-network-based adaptive controllers for a class of fed-batch fermentation processes," *Automatica*, vol. 31, pp. 817–840, 1995.

[15] A. K. Jain and K. Karu "Learning texture discrimination masks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, pp. 195–205, 1996.

[16] V. P. Roychowdhury, K. Y. Siu, and T. Kailath "Classification of linearly nonseperable patterns by linear threshold elements," *IEEE Transactions on Neural Networks*, vol. 6, pp. 318–331, 1995.

[17] S. Watanabe and M. Yoneyama "Ultrasonic visual sensor for three-dimensional object recognition using neural networks," *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 240–249, 1992.

[18] L. Jin, K. Chan, and B Xu "Off-line Chinese handwriting recognition using multi-stage neural network architecture," in *IEEE International Conference on Neural Networks - Conference Proceed.*, volume 6, pp. 3083–3088, Piscataway, NJ, 1995.

[19] P. T. Chan, M. Palaniswami, and D. Everitt "Neural network-based dynamic channel assignment for cellular mobile communication systems," *IEEE Transactions on Vehicular Technology*, vol. 43, pp. 279–287, 1994.

[20] Y. Shan and Y. Koren "Design and motion planning of a mechanical snake," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, pp. 1091–1100, 1993.

[21] G. S. Chirikjian and J. W. Burdick "A hyper-redundant manipulator," *IEEE Robotics & Automation Magazine*, pp. 22–29, December 1994.

[22] S. Hirose, E. F. Fukushima, and Tsukagoshi S. "Basic steering control methods for the articulated body mobile robot," *IEEE Control Systems Magazine*, pp. 5–14, February 1995.

[23] C. Gans "How snakes move," *Scientific American*, vol. 222, pp. 82–96, 1970.

[24] Z. Li and J. F. Canny. *Nonholonomic Motion Planning.* Kluwer Academic Press, Norwell, 1993.

[25] L. E. Dubins "On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, pp. 497–516, 1957.