NEAR EAST UNIVERSITY

Faculty of Engineering

Depatment of Computer Engineering

COMPUTER – AIDED CONTROL OF ROBOT MANIPULATORS

Graduation Project
COM – 400

Student : Nazım Kansel

Supervisor : Asst. Prof. Dr. Rahib Abiyev

Nicosia - 2001

ACKNOWLEDGMENT

First of all, I would like to acknowledge as well my parents Keziban - Soner KANSEL. What ever I can write or say, I can not reserved them.

I would like to thank my supervisor Asst. Prof. Dr. Rahib Abiyev who give me desire and corporation to present my project in this manner.

Also I would like to thank for all our staff of Faculty of Engineering .

Thanks to all our friends specially Ismail Muratoglu who helped me to make my graduation project. Finally, special thanks for all people that make me stronger and stronger to finish my university.

ABSTRACT

One of the actual field of application of Artificial Intelligence ideas is control of robots manipulator, which allow me to solve complicated control problems in uncertainty of environment , fuzzyness of information . The project is devoted a computer aided control of robot manipulator .

To solve this problem the control problem of robots are consider. The mathematical model of kinematics and dynamics of robot manipulators are descrabed, the interface organization between computer and robot is given.

The aplications of artifical intelegence ideas to the robot kinematics and dynamics nowday is very actual problem of control systems. In the proect the aplication of neural technology to control robot kinematics and dynamics are given. The neural control algorithm of force of robot-hand are described. Also identification and adaptive control the dynamics of robot are considered.

The structure of computer – aided control of robot manipulator is given , the function of its main blocks are given . The program to transmits signal from computer to robot manipulatior is developed in assembler .

The obtained experimental results show the efficieny of used methodology .

TABLE OF CONTENTS

INTRODUCTION

One of the major cost factors involved in robotic applications is the development of robot control. Especially the Use of advanced sensor systems and the existence of strong requirements with respect to the robot's flexibility ask for very skillful programmers and sophisticated programming environments. These circumstances let the interest in a new programming paradigm, namely Robot Programming by Demonstration (RPD) (Heise, 1989), (Munch *et al.,* 1994) grow rapidly. RPD is an intuitive method to program a robot. The programmer shows how a particular task is performed, using an interface device that allows the measurement and recording of the human's motions and the data simultaneously perceived by the robot's sensors. A natural .application of the PbD paradigm in the area of robotics concerns the acquisition of elementary skills (Asada and Yang, 1989), (Delson and West, 1993), (Kaiser, 1994). These elementary skills must be provided by the robot for the execution *of* elementary operations. They represent the interface between the planning and the control level in the robot's architecture and usually provide a direct coupling between the robot's sensors and its actuators.

The use of neural networks for robot control tasks constitutes an example of a very powerful approach to nonlinear process control: the observation- or experience-based learning of the controllers, as opposed to the model-based design of classical controllers. In this context, learning in the neural networks can efficiently replace the detailed mathematical modelling of some complex control processes, as has been shown in t.he literature in the last few years (see e.g. Narendra, 1990; Cembrano et al., 1992; Hunt et, al., 1993). Robot control is one of the fields where extensive research is being performed along this line, especially oriented towards achieving greater degrees of autonomy in robot operation. The aim of this paper is to present a brief review of the state of the art in the application of neural networks to the solution of control problems in robotics.

CHAPTER 1. CONTROL PROBLEMS OF ROBOT MANIPULATOR

1.1. Robot Control Systems and Components

A robot is a mechanical system that must be controlled in order to accomplish a useful task. The task involves the movement of the manipulator arm, so the primary function of the robot control system is to position and orient the wrist (and end effector) with a specified speed and precision.

When studying a mechanical system we are concerned with the response of the system to certain inputs. These inputs include commands to drive the system and disturbances from the environment. The system can be divided into five major components:

The input (or inputs) or the systems.

The controller and actuating devices.

The plant (the mechanism or process being controlled)

The output (the controlled variable)

Feedback elements (sensors)

By analysing the effects that the controller and the plant have on the inputs, we can predict what the outputs will be under certain conditions. In order to do this analysis we must be able to model the system mathematically.

1.2. Mathematical Models

As an example ,let us formulate the model for a familiar mechanical system: the spring-mass-damper system.

The system is illustrated in Fig. 1.1 and consists of a block with a certain mass suspended from a fixed wall by a dashpot. The mass is connected to a spring, the other end of which is given some displacement from its equilibrium position. We will use the following symbols to represent the various parameters of system behavior:

y = the displacement of the mass

M = the mass of the lock

Ks = the spring constant

Kd = the damping coefficient of the dashpot

x = the displacement of the end of the spring

Summing all of the forces we get

$$M\frac{d^2y}{dt^2}+K_d\frac{dy}{dt}+K_sy=K_sx \qquad (1.1)$$

In this system the input is x, representing the displacement of the end of the spring, and the system output is y, representing the displacement of the block. The system has been described by a second-order linear differential equation which relates the input and the output. This mathematical description of the system allows us to analyze its behaviour. Before beginning the analysis, however, we will develop other useful tools for building the models.

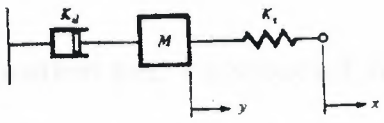

Figure1-1 Schematic diagram of a spring mass damper system

By using s with Laplace transforms, linear differential equations can be converted to equivalent expressions which are functions of s. (It is assumed that the reader is familiar with the Laplace transform and its s operator for linear systems analysis.) Using s, Eq. (1.1) can be written as

$$Ms^2\,Y(s)+K_dsY+K_sY(s)=K_sX(s) \qquad 1.2$$

The transfer function relates the output of the system to an input. The spring-mass-damper transfer function can be derived by rewriting Eq. (1.3) as

$$\frac{Y(s)}{X(s)}=\frac{K_s}{Ms^2+K_ds+K_s} \qquad 1.3$$

1.3. Block Diagrams

Figure 1.2 illustrates a general block diagram of the control system components for one joint of a robot manipulator. The input command is the defined position (and possibly speed) to which the joint is directed to move. The output variable is the actual position (and speed) of the joint. In nearly all robots today, most of the computational

functions of the joint controller are carried out by a microprocessor as portrayed in Fig. 1.2.

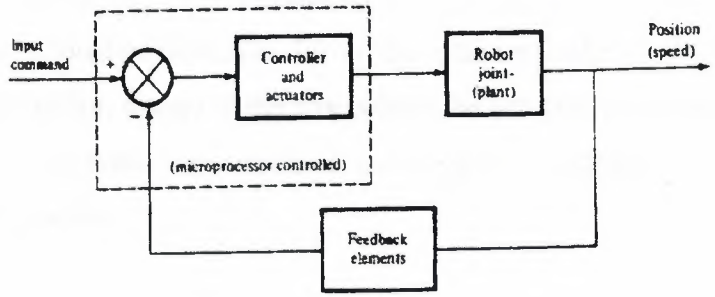

Figure 1.2 Typical block diagram configuration of a control system for a robot joint.

1.4. Robot Actuation and Feedback Components

Control of the robot manipulator requires the application of the preceding control theory to a mechanical system. In this and the following sections we discuss some of the various types of devices commonly used as components of robot control systems. We classify these devices into four categories:

- Position sensors
- Velocity sensors
- Actuators
- Power transmission devices

Position and velocity sensors are used in robotics as feedback devices, while actuators and power transmission devices are used to accomplish the control actions indicated by the controller. Position sensors provide the necessary means for determining whether the joints have moved to correct linear or rotational locations in order to achieve the required position and orientation of the end effector. The speed with which the manipulator is moved is another performance feature which must be regulated. Many robots utilize a feedback system to ensure proper speed control. This is especially important as sophisticated control systems are being developed to fine tune the dynamic performance of the manipulator during acceleration and deceleration as it

moves between points in the workspace.

Actuators and power transmission devices provide the muscle to move the robot arm. Actuators include hydraulic, electric, and pneumatic devices corresponding to the three basic robot drive systems described in Chapter Two. The power developed by these actuators must be transmitted from the actuator to the robot joint via a power transmission device, except in the case where the actuator is directly coupled to the robot joint. Transmission mechanisms, such as pulley systems, gears, and screws, are used for this purpose.

1.5. Electric Motors

As their capabilities improve, electric motors are becoming more and more the actuator of choice in the design of robots. They provide excellent controllability with a minimum of maintenance required. There are a variety of types of motors in use in robots; the most common are dc servomotors, stepper motors, and ac servomotors.

As the motor velocity increases, and the back-emf voltage increases accordingly, the current available to the armature decreases. The decreasing current reduces the torque generated by the rotor. As the torque decreases the acceleration of the rotor decreases as well. At the point at which *eb* = Vin the motor maintains a steady-state velocity (assuming there are no external disturbances on the motor). The block diagram in Fig. 1.3 illustrates the effects of the torque constant and back-emf on the model of a motor. Note that this simplified model discounts such effects as friction or inductance of the armature windings.

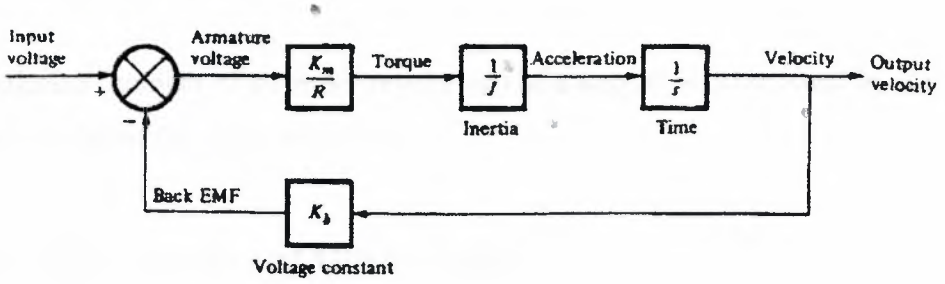

Figure 1.3 DC motor block diagram.

1.6. Stepper Motors

Stepper motors (also called stepping motors) are a unique type of actuator and have been used mostly in computer peripherals. A stepper motor provides output in the form of discrete angular motion increments. It is actuated by a series of discrete electrical pulses. For every electrical impulse there is a single-step rotation of the motor shaft. In robotics, stepper motors are used for relatively light duty applications. Also, stepper motors are typically used in open-loop systems rather than the closed-loop system.

Figure 1.4 provides a schematic representation of one type of stepper motor. The stator is made up of four electromagnetic poles and the rotor is a two-pole permanent magnet. If the electromagnetic stator poles are activated in such a way that pole 3 is N (magnetic North) and pole 1 is S then the rotor is aligned as illustrated. If the stator is excited so that pole 4 is N and pole 2 is S, the rotor makes a *900* turn in the clockwise direction. By rapidly switching the current to the stator electronically, it is possible to make the motion of the rotor appear continuous.

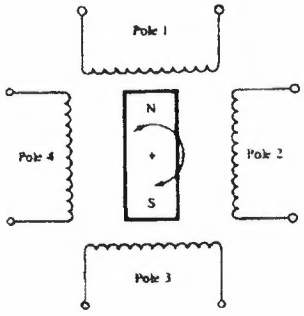

Figure 1.4 Stepper motor schematic

The resolution (number of steps per revolution) of a stepper is determined by the number of poles in the stator and rotor.

1.7. AC Servomotors and Other Types

There are numerous other aspects of electric motors which may be investigated. Recent advances in control electronics are producing ac servomotors. These motors have the advantages of being cheaper to manufacture than dc motors, they have no

brushes, and they possess a high power output. With the proper electronics package, however, their performance can be made to look very much like the performance of a dc motor.

Another type of electric motor is the brushless dc motor. It is constructed like an "inside-out" dc motor. It has a permanent magnet rotor and an electromagnetic stator. Instead of using brushes, however, commutation is performed electronically using an encoder to inform the electronics of the relative positions of the stator and rotor. Also available are linear electric motors. Their construction is similar to a dc servomotor that has been cut open and Battened out.

In almost all cases of electric motors the limiting factor on power output is heat dissipation. Some of the current used in the motor must be dissipated as heat. Two ways to increase the performance of a motor is to remove heat more quickly or to reduce the current requirements. The latter may be done by increasing the magnetic flux of the permanent magnets. Recent advances in magnetic materials are allowing for performance improvements of almost 10 times with the same power requirements.

CHAPTER 2. ROBOT MOTION ANALYSIS AND CONTROL

2.1. Introduction to Manipulator Kinematics

In order to develop a scheme for controlling the motion of a manipulator it is necessary to develop techniques for representing the position of the arm at points in time. We will define the robot manipulator using the two basic elements, joints and links. Each joint represents 1 degree of freedom. Two, the joints may involve either linear motion (joint-type L) or rotational motion (joint-types R, *T,* and *V)* between the adjacent links. The links are assumed to be the rigid structures that connect the joints.

Joints are labelled in where *n* begins with 1 at the base of the manipulator, and links are labelled Ln, again with 1 being the link closest to the base. Figure 2.1 illustrates the labelling system for two different robot arms, each possessing 2 degrees of freedom.

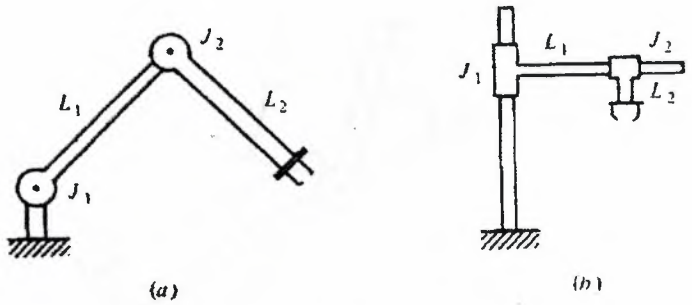

Figure 2.1 Two different 2-jointed manipulators.(a) two rotational joints (RR) , (b) two linear joints (LL)

The manipulator in Fig. 2.1*(a)* has an RR notation and the manipulator in Fig. 2.1*(b)* has an LL notation.

We will also use the symbol Ln to indicate the length of the link in some of our equation derivations early in the chapter. Later in the chapter, we define a "standard" notation system used for computing joint-link transformations. This notation system uses the symbol *an* to denote the length of a manipulator link.

2.2. Position Representation

The kinematics of the RR robot are more difficult to analyze than the LL robot, and we will make frequent use of this configuration (and extensions of it) throughout the chapter. Figure 2.2 illustrates the geometric form of the RR manipulator. For the present discussion, our analysis will be limited to the two-dimensional case. The position of the end of the arm may be represented in a number of ways. One way is to utilize the two joint angles θ_1 and θ_2. This is known as the representation in "joint" space and we may define it as

$$P_j = (\theta_1, \theta_2)$$

Another way to define the arm position is in "world" space. This involves the use of a cartesian coordinate system that is external to the robot. The origin of

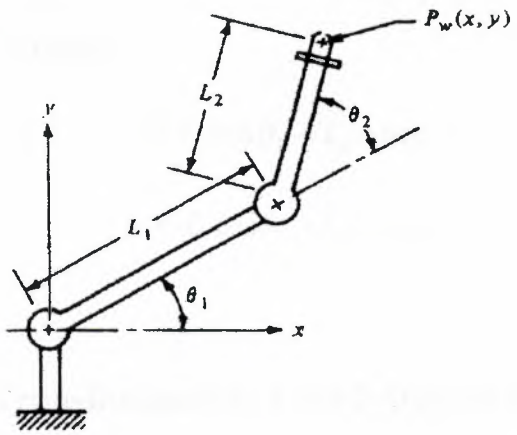

Figure 2.2 A two-dimensional 2 degree-of freedom manipulator (type RR)

the cartesian axis system is often located in the robot's base. The end-of-arm position would be defined in world space as

$$P_w = (x, y)$$

This concept of a point definition in world space can readily be extended to three dimensions, that is, $P_w = (x, y, z)$. Representing an arm's position in world space is useful when the robot must communicate with other machines. These other machines

may not have a detailed understanding of the robot's kinematics and so a "neutral" representation such as the world space must be used. In order to use both representations we must be able to transform from one to the other. Going from joint space to world space is called the forward transformation and going from world space to joint space is called the reverse transformation.

2.3. Forward Teansformation of a 2-Deeree of Freedom Arm

We can determine the position of the end of the arm in world space by defining a vector for link 1 and another for link 2.

$$r_1 = [L_1 \cos\theta_1, L_1 \sin\theta_1] \qquad (2.1)$$

$$r_2 = [L_2 \cos(\theta_1 + \theta_2), L_2 \sin(\theta_1 + \theta_2)] \qquad (2.2)$$

Vector addition of (2.1) and (2.2) yields the coordinates *x* and y of the end of the arm (point *Pw)* in world space

$$x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \qquad (2.3)$$

$$y = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \qquad (2.4)$$

2.4. Reverse Transformation of the 2-Degree of Freedom Arm

In many cases it is more important to be able to derive the joint angles given the end-of-arm position in world space, The typical situation is where the robot's controller must compute the joint angles required to move its end-of arm to a point in space defined by the point's coordinates. For the two-link manipulator we have developed, there are two possible configurations for reaching the point *(x, y),* as shown in Fig, 2.3. Some strategy must be developed to select the appropriate configuration. One approach is that employed in the control system of the Unimate PUMA robot. In the PUMA's control language, VAL, there is a set of commands called ABOVE and BELOW that determines whether the elbow is to make an angle θ_2 that is greater than or less than

zero, as illustrated in Fig. 2.3. For our example, let us assume the θ_2 is positive as shown in Fig. 2.3. Using the trigonometric identities,

$$\cos(A+B) = \cos A \cos B - \sin A \sin B$$
$$\sin(A+B) = \sin A \cos B + \sin B \cos A$$

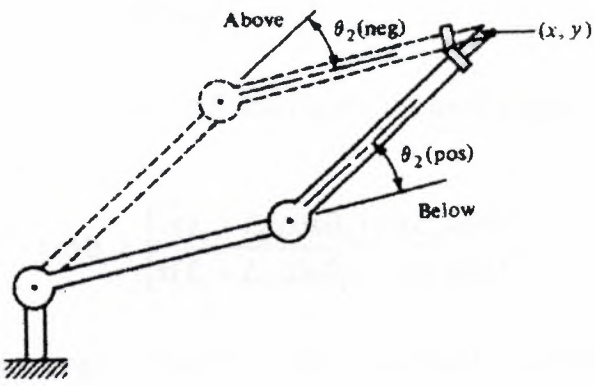

Figure 2.3 The arm at point P(x,y),indicating two possible configurations to achive the position.

We can rewrite Eqs. (2.3) and (2.4) as

$$x = L_1 \cos\theta_1 + L_2 \cos\theta_1 \cos\theta_2 - L_2 \sin\theta_1 \sin\theta_2$$

$$y = L_1 \cos\theta_1 + L_2 \cos\theta_1 \cos\theta_2 - L_2 \sin\theta_1 \sin\theta_2$$

Squaring both sides and adding the two equations yields

$$\cos\theta_2 = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2} \tag{2.5}$$

Defining α and β as in Fig. 2.4 we get

$$\tan\alpha = \frac{L_2 \sin\theta_2}{L_2 \cos\theta_2 + L_1}$$

$$\tan\beta = \frac{y}{x} \tag{2.6}$$

Using the trigonometric identity

$$\tan(A-B) = \frac{\tan A - \tan B}{1 + \tan A \tan B}$$

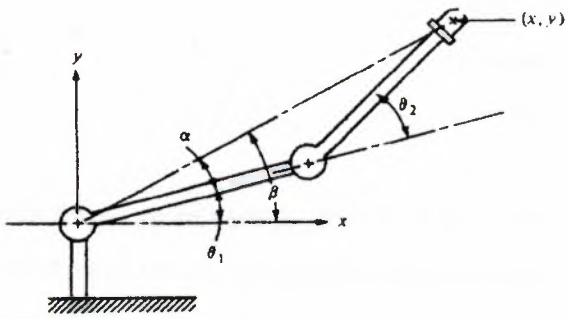

Figure 2.4 Solving for the joint angles.

We get

$$\tan\theta_1 = \frac{[y(L_1 + L_2\cos\theta_2) - xL_2\sin\theta_2]}{[x(L_1 + L_2\cos\theta_2) + yL_2\sin\theta_2]}$$

Knowing the link lengths L1 and --we are now able to calculate the required joint angles to place the arm at a position *(x,* y) in world space.

2.5. Homogeneous Transformations and Robot Kinematics

The approach used in the previous section becomes quite cumbersome when a manipulator with many joints must be analyzed. Another, more general method for solving the kinematic equations of a robot arm makes use of homogeneous transformations. We describe this technique in this section, assuming the reader has at least some familiarity with the mathematics of vectors and matrices. Let us begin by defining the notation to be used.

A point vector, v = ai + bj + ck can be represented in three-dimensional space by the column matrix

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

where $a = x/w$, $b = y/w$, $c = z/w$, and w is a scaling factor. For example, any of the following matrices can be used to represent the vector v = 25i +10j + 20k.

$$\begin{bmatrix}25\\10\\20\\1\end{bmatrix} \quad \text{or} \quad \begin{bmatrix}50\\20\\40\\2\end{bmatrix} \quad \text{or} \quad \begin{bmatrix}12.5\\5.0\\10.0\\0.5\end{bmatrix}$$

Vectors of the above form can be used to define the end-of-arm position for a robot manipulator. (If w = 0, then the vector represents direction only.)

A vector can be translated or rotated in space by means of a transformation. The transformation is accomplished by a 4 x 4 matrix H. For instance the vector v is transformed into the vector u by the following matrix

operation:

$$u = Hv$$

The transformation to accomplish a translation of a vector in space by a distance a in the x direction, b in the y direction, and c in the z direction is given by

$$H = Trans(a,b,c) = \begin{bmatrix}100a\\010b\\001c\\0001\end{bmatrix}$$

Example 4.1 For the vector v =25i+ 1Oj+20k, perform a translation by a distance of 8 in the x direction, 5 in the y direction, and 0 in the z direction. The translation transformation would be

$$H = trans(a,b,c) = \begin{bmatrix}1008\\0105\\0010\\0001\end{bmatrix}$$

The translated vector would be

$$Hv = \begin{bmatrix}1008\\0105\\0010\\0001\end{bmatrix}\begin{bmatrix}25\\10\\20\\1\end{bmatrix} = \begin{bmatrix}33\\15\\20\\1\end{bmatrix}$$

Rotations of a vector about each of the three axes by an angle θ can be accomplished by

rotation transformations. About the *x* axis, the rotation transformation is

$$\text{Rot}(x,\theta)= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

About the y axis,

$$\text{Rot}(y,\theta)= \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

About the z axis,

$$\text{Rot}(y,\theta)= \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In concept it is possible to develop the rotation transformation about any ector K, where K is not one of the major axes *x,* y, or z of the coordinate system.

2.6. Robot Arm Dynamics

The topic of robot dynamics is concerned with the analysis of the torques and forces due to acceleration and deceleration. Torques experienced by the joints due to acceleration of the links, as well as forces experienced by the links due to torques applied by the joints, are included within the scope of dynamic analysis. Solving for the accelerations of the links is difficult due to a number of factors. For one, the acceleration is dependent on the inertia of the arm. However, the inertia is dependent on the configuration of the arm, and this is continually changing as the joints are moved. An additional factor that influences the inertia is the mass of the payload and its position with respect to the joints. This also changes as the joints are moved. Figure 2.5 shows the two-link arm in the maximum and minimum inertia configurations.

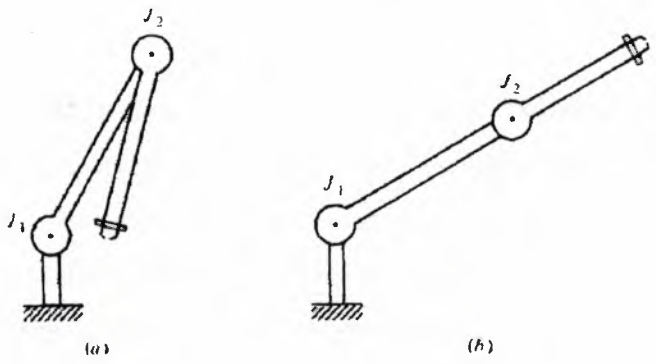

Figure 2.5 Arm inertias (a) Minimum inertia about (b) Maximum inertia about J1

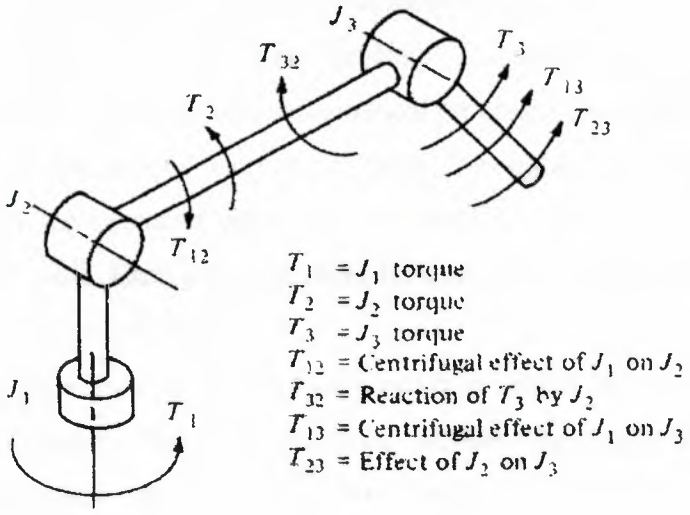

Figure 2.6 Dynamic forces and torques for a TRR robot

The torques required to drive the robot arm are not only determined by the static and dynamic forces described above; each joint must also react to the torques of other joints in the manipulator, and the effects of these reactions must be included in the analysis. Also, if the arm moves at a relatively high speed, the centrifugal effects may be significant enough to consider. The various torques applied to the two-jointed manipulator are illustrated in Figure 2.6. The picture becomes substantially more complicated as the number of joints is increased.

A detailed analysis of manipulator dynamics is beyond the scope and purpose of this chapter, and we leave it to the interested reader to pursue the subject in some of the references listed at the end of the chapter.

2.7. Configuration of a Robot Controller

The elements needed in the robot controller include: joint servo controllers, joint power amplifiers, mathematical processor, executive processor, program memory, and input device. The number of joint servo controllers and joint power amplifiers would correspond to the number of joints in the manipulator. These elements might be organized in the robot controller as shown in Fig. 2.7.

Motion commands are executed by the controller from two possible sources: operator input or program memory. Either an operator inputs commands to the system using an input device such as a teach pendant or a CRT terminal, or the commands are downloaded to the system from program memory under control of the executive processor. In the second case, the set of commands have been previously programmed into memory using the operator input device(s). For each motion command, the executive processor informs the mathematical processor of the coordinate transformation calculations that must be made. When the transformation computations are completed, the executive processor downloads the results to the joint controllers as position commands. Each joint controller then drives its corresponding joint actuator by means of the power amplifier.

Microprocessors are typically utilized in several of the components of a modern robot controller. These components include the mathematical processor, the executive processor, the servocontrollers, and the input device. Each of the control boards makes use of a common data buss and address buss. The microprocessors communicate with each other by sending messages into common areas in the system memory. This architecture provides several advantages in the design of the controller. These advantages include commonality of components, expansion of the system to more joints, and information flow between joint control elements. For example, the control configuration would permit the sharing of feedback information among the various joints, thus providing the opportunity to develop algorithms for improving the individual joint dynamics

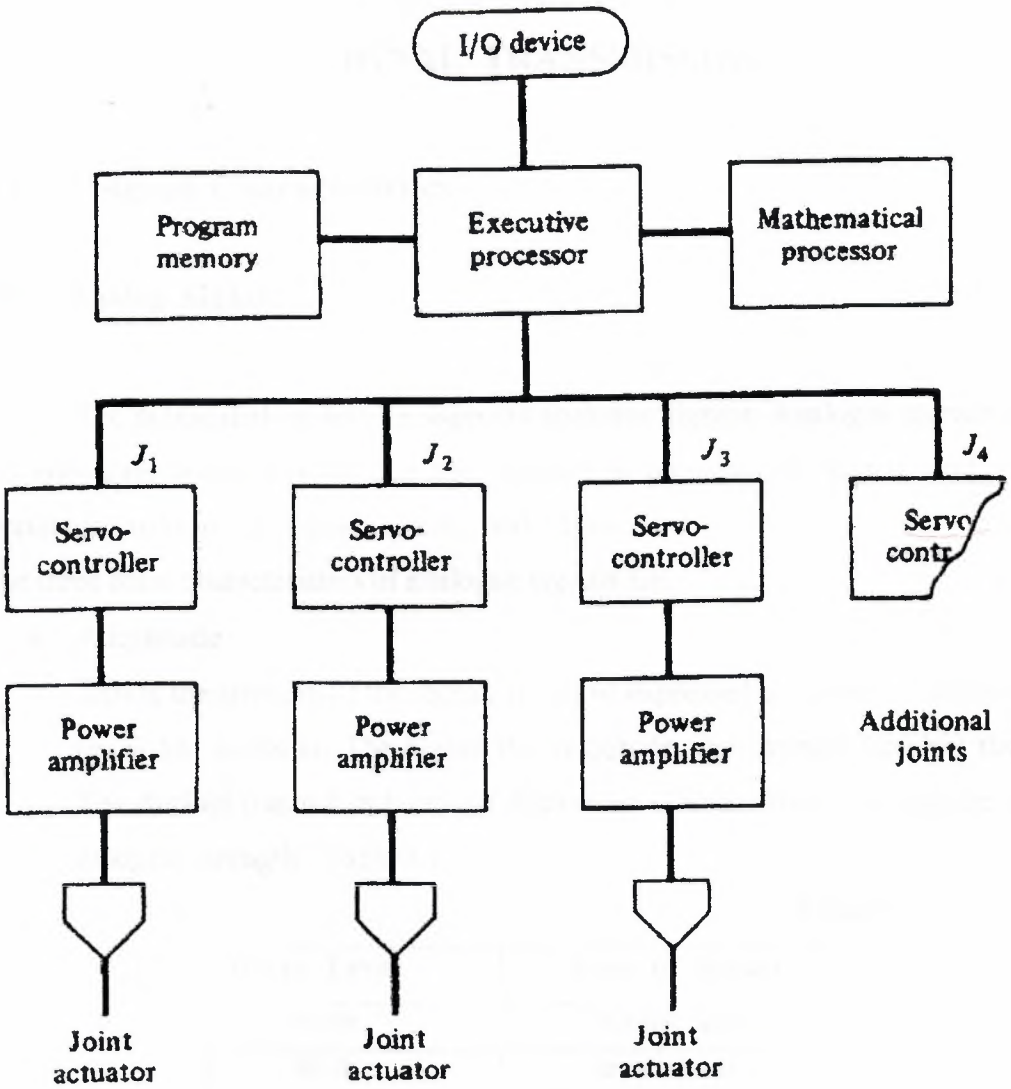

Figure 2.7 General robot controller elements.

CHAPTER 3. STRUCTURE OF INTERFACE AND SIGNAL TRANSMISSION

3.1. Signal Characteristics

3.1.1. Analog Signals

The public dial-up service supports analogue signals. Analogue signals are what we encounter every day of our life. Speech is an analogue signal, and varies in amplitude (volume), frequency (pitch), and phase.

The three main characteristics of analogue signals are,

- **Amplitude**

  This is the strength of the signal. It can be expressed a number of different ways (as volts, decibels). The higher the amplitude, the stronger (louder) the signal. The **decibel** (named in honor of Alexander Graham Bell) is a popular measure of signal strength . Table 3.1

Table 3.1

Sound Level	Type of Sound
40 db	Normal Speech
90 db	lawn mowers
110 db	shotgun blast
120 db	jet engine taking off
120 db +	rock concerts

It has been discovered that exposure to sounds greater than 90db for a period exceeding 15 minutes causes permanent damage to hearing. Our ability to hear high notes is affected. As young babies, we have the ability to hear quite high frequencies. This ability reduces as the aging process occurs. It can also be affected by too much noise over sustained periods. **Ringing** in the ears after being exposed to loud noise is an indication that hearing loss may be occurring.

The figure 3.1 shows a single signal of various amplitudes. The base line indicates a steady state, in this example, the signal amplitude rises both above and below the steady state. The measurement of the two extremes is called the **peak to peak** measurement

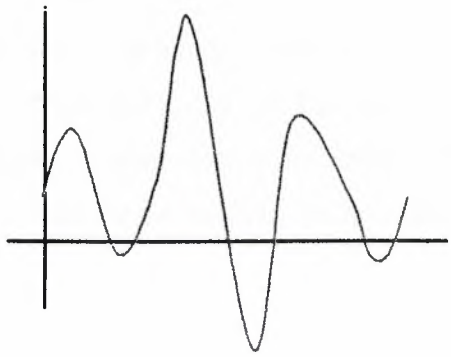

Figure 3.1. Signal Amplitude

- **Frequency**

  This is the rate of change the signal undergoes every second, expressed in Hertz (Hz), or cycles per second. A 30Hz signal changes thirty times a second. In speech, we also refer to it as the number of vibrations per second. As we speak, the air is forced out of our mouths, being vibrated by our voice box. Men, on average, tend to vibrate the air at a lower rate than women , thus tend to have deeper voices .

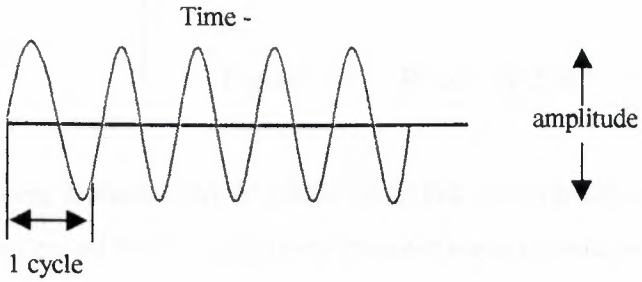

Figure 3.2. Cycling

A cycle is one complete movement of the wave, from its original start position and back to the same point again. The number of cycles (or waves) within a one second time interval is called cycles-per-second, or **Hertz.** (Figure 3.2)

An example is sitting on the beach, counting the waves as they come in on the shore. If we counted the number of waves that crashes on the beach over a minute period, then divided that number by 60 (because there are 60 seconds in one minute), we would have the number of waves per second... (i.e. frequency!).

Humans can hear from reasonably low frequency tones (about 100Hz) all the way up to about 12KHz. As we get older, our ability to hear the higher notes is lessened, due to the aging process making the bones in the ear harder and less able to vibrate. In addition, human speech contains a great deal of redundant information, and can be compacted within the range of 300Hz to 3400Hz whilst still retaining approximately 80% of its content.

- **Phase**

This is the rate at which the signal changes its relationship to time, expressed as degrees. One complete cycle of a wave begins at a certain point, and continues till the same point is reached again. Phase shift occurs when the cycle does not complete, and a new cycle begins before the previous one has fully interconnected . (Figure 3.3)

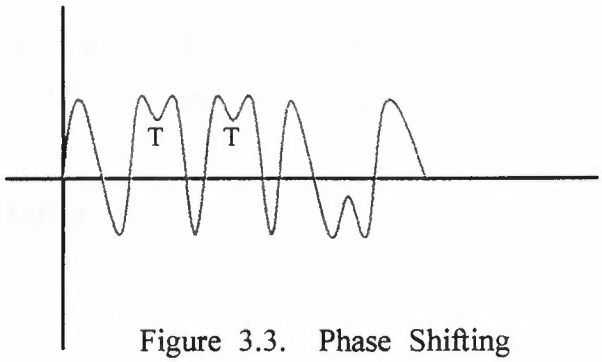

Figure 3.3. Phase Shifting

The human ear is insensitive to phase shift, but data signals are severely affected by it. Phase shift is caused by imperfections in cable media, such as joins and imperfect terminations.

In a practical sense, imagine you are in the bath. If you drop the soap, it forms ripples where the waves travel outwards towards the edge of the bath. When the wave reaches the edge, it hits the wall of the bath and bounces back. This is like phase shift, which is an abrupt change in the signals relationship.

Analogue signals are sent via the PTSN. Digital signals cannot be sent via the PTSN without being first converted to analogue.

3.1.2. Digital Signals

Digital signals are the language of modern day computers. Digital signals comprise

only two states. These are expressed as ON or OFF, 1 or 0 respectively. Figure 3.4 Examples of devices having TWO states in the home are,

- Light Switches: Either ON or OFF
- Doors: Either OPEN or CLOSED

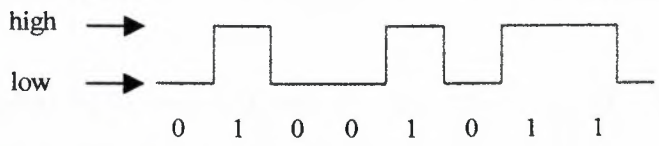

Figure 3.4. A Digital Signal

Digital signals require greater bandwidth capacity than analogue signals, thus are more expensive to communicate. The figure shows a digital signal.

3.1.3. Some Common Terms

- Baud Rate

Baud rate is the reciprocal of the shortest signal element (a measure of the number of line changes which occur every second). For a binary signal of 20Hz, this is equivalent to 20 baud (there are 20 changes per second). For telephone cables, the limiting factor in speed is the number of line changes per second. A line change is defined as switching from one state to another, for instance, switching from a 1 to a 0, or from a 0 to 1 for a digital signal. If the number of line changes per second are exceeded, errors occur and the signal at the receiving end cannot be reliably reconstructed.

- Bits Per Second

This is an expression of the number of bits per second. Where a binary signal is being used, this is the same as the baud rate. When the signal is changed to another form, it will not be equal to the baud rate, as each line change can represent more than one bit (either two or four bits).

Digital signals sent via the PSTN need to be converted to analogue first (by using a device called a modem). Digital signals can be sent via the ISDN unmodified.

- Bandwidth

Bandwidth is the frequency range of a channel, measured as the difference between the highest and lowest frequencies that the channel supports. The maximum transmission speed is dependant upon the available bandwidth. The larger the bandwidth, the higher the transmission speed. A nominal voice channel has a bandwidth of 3.1KHz. In reality this equates to about 1200bps maximum for a binary digital signal.

3.1.4. Problems of using Voice Channels for Digital Transmission

A digital signal is comprised of a number of signals. Specifically, the signal is represented as follows,signal = f + f3 + f5 +f7 +f9 +f11 +f13 ....f(infinity)

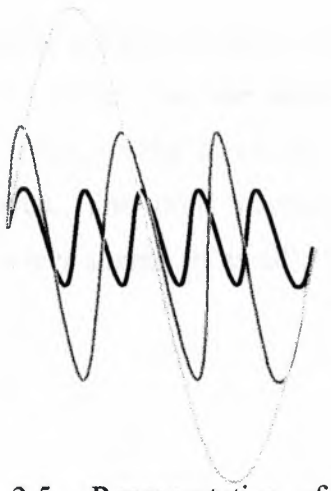

Figure 3.5. Representation of square wave as f + 3f + 5f

In figure 3.5 we can see that a digital signal has a base frequency, plus another at three times the base frequency, plus another at five times the base frequency etc. f3 is called the third harmonic, f5 the fifth harmonic and so on.

The third harmonic is one third of the amplitude of the base frequency (called the fundamental frequency), the fifth harmonic is one fifth the amplitude of the fundamental and so on.

In order to send a digital signal across a voice channel, the bandwidth of the channel must allow the fundamental plus third and fifth harmonic to pass without affecting them too much. (figure 3.6)

As can be seen, this is what such a signal looks like, and is the minimum required to be correctly detected as a digital signal by the receiver. (figure 3.7)

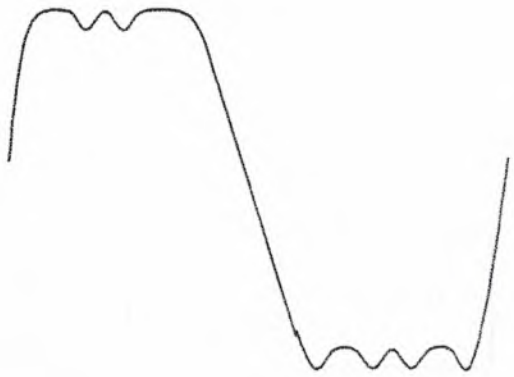

Figure 3.6. Square Wave of f + 3f + 5f

Lets consider sending a 2400bps binary digital signal down a voice channel rated with a bandwidth of 3.1KHz. The base frequency of the digital signal is 1200Hz (it is always half the bit rate), so the fundamental frequency will pass through the channel relatively unaltered. The third harmonic is 3600Hz, which will suffer attenuation and arrive severely altered (if at all). The fifth harmonic has no chance of passing the channel.

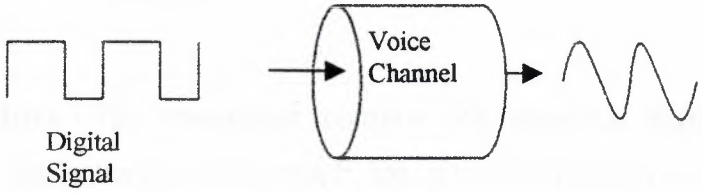

Figure 3.7. The effect of the voice channel on data signals

In this case it can be seen that only the base frequency will arrive at the end of the channel. This means the receiver will not be able to reconstruct the digital signal properly, as it will require f3 and f5 for proper reconstruction.
This results in errors in the detection process by the receiver.

3.2. Structure and Elements of Communication System

Electrical Communication Systems are designed to send messages or information from

a source that generates the messages to one or more destinations . In general, a communication system can be represented by the functional block diagram shown in figure 3.8.

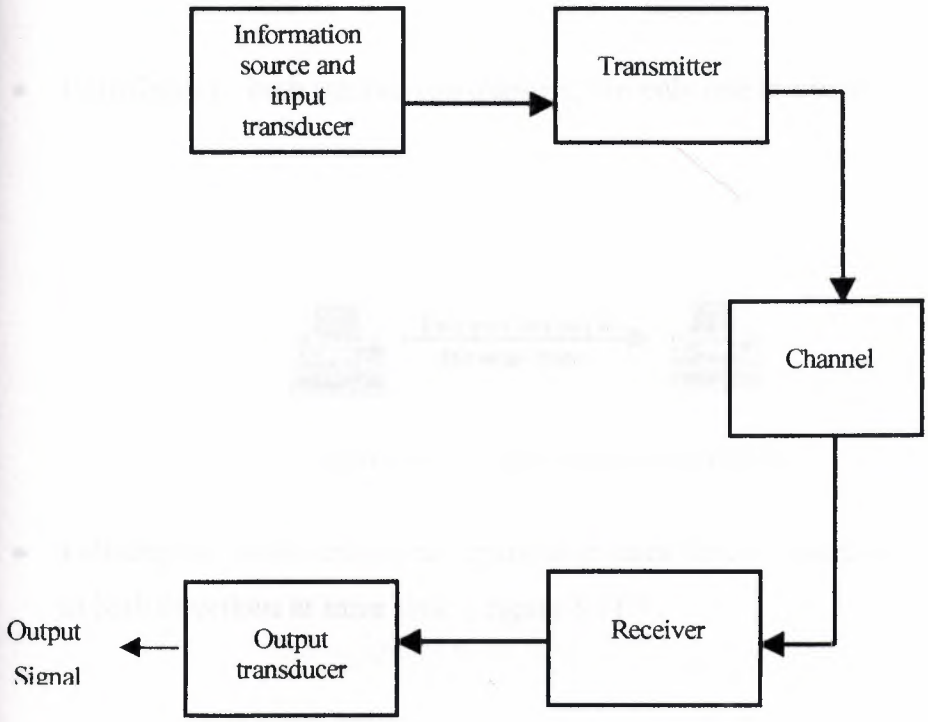

Figure 3.8. Functional Block Diagram of a Communication Systems

Transmitter : The transmitter converts the electrical signal into a form that is suitable for transmission through the physical channel or transmission medium . The transmitter must translate the information signal to be transmitted into the appropriate frequency range that matches the frequency allocation assigned the transmitter.

Transmission can be:

- Simplex - signals only in one direction; one station transmitter, other receiver; e.g. radio (figure 3.9)

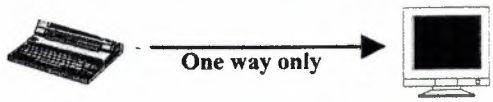

Figure 3.9. Simplex Channel Operation

- Half-Duplex - both stations can transmit, but only one at a time

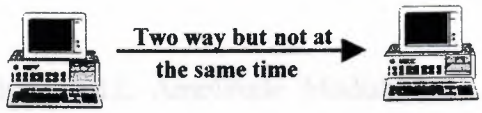

Figure 3.10. Half Duplex Operation

- Full-Duplex - both stations can transmit at same time i.e. medium is carrying signals in both directions at same time (figure 3.11)

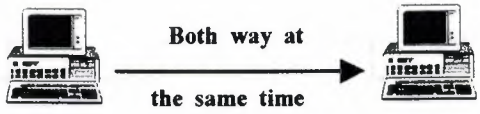

Figure 3.11. Full Duplex Operation

In general , the transmitter performs the matching of the message signal to the channel by a process called modulation .

Modulation: systematic variation of some attribute of carrier signal - e.g. amplitude, phase or frequency.

- *Amplitude Modulation (AM) :* It is sensitive to power fluctuations .For example In AM Radio broadcast , the information signal that is transmitted is contained in the amplitude variations of the sinusoidal

carrier , which is the centre frequency in the frequency band allocated to the radio transmitting station . (figure 3.12)

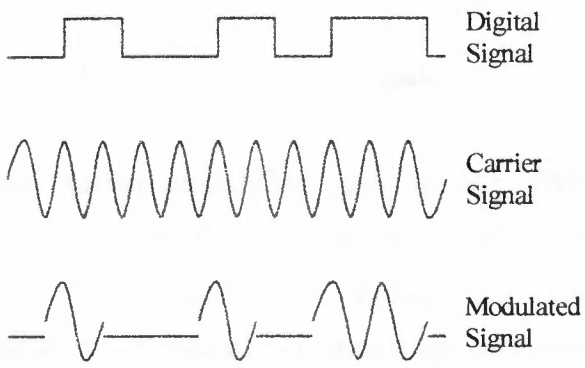

Figure 3.12. Amplitude Modulation

- *Frequency Modulation (FM) :* It allows *full duplex* communication . It needs four frequencies . In FM Radio broadcast , the information signal that is transmitted is contained in the frequency variations of the sinusoidal carrier . (figure 3.13)

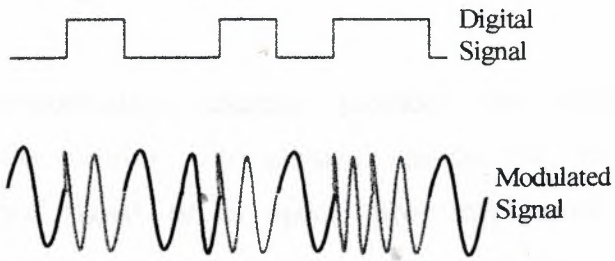

Figure 3.13. Frequency Modulation

- *Phase Modulation (PM)* : It is yet a third method for impressing the information signal on a sinusoidal carrier . It is more sophisticated . (Table 3.2)

Table 3.2

Bit Pattern	Degrees Phase Shift
00	45^0
01	135^0
10	315^0
11	225^0

In general , carrier modulation such as AM , FM and PM is performed at the transmitter to convert the information signal to a form that matches the characteristics of the channel . Thus through the process of modulation , the information signal is translated in frequency to match the allocation of the channel . The choice of the type of modulation is based on several factors , such as the amount of bandwidth allocated , the types of noise and interference that the signal encounters in transmission over the channel , and the electronic devices that are available for signal amplification prior to transmission .

In addition to modulation , other functions that are usually performed at the transmitter filtering of the information – bearing signal , amplification of the modulated signal , and in the case of wireless transmission , radiation of the signal by means of a transmitting antenna .

3.3. Channel Organizations

The communication channel provides the connection between the transmitter and the receiver . The physical channel may be a pair of wires that carry the electrical signal , or an optical fiber that carries the information on a modulated light beam , or an underwater ocean channel in which the information is transmitted acoustically , or free space over which the information – bearing signal is radiated by use of antenna.

One common problem in signal transmission through any channel is additive noise .In general , additive noise is generated internally by components such as resistors and solid – state devices used to implement the communication system . This is sometimes called thermal noise .Other sources of noise and

interference may arise externally to the system , such as interference from other users of the channel .

The effects of noise may be minimized by increasing the power in the transmitted signal . However , equipment and other practical constraints limit the power level in the transmitted signal . Another basic limitation is the available channel bandwidth . A bandwidth constraint is usually due to the physical limitations of the medium and the electronic components used to implement the transmitter and the receiver . These two limitations result in constraining the amount of data that can be transmitted reliably over any communication channel .

Data may be transmitted between two points in two different ways. Lets consider sending 8 bits of digital data (1 byte). These bits may be sent all at once (in parallel), or one after the other (serial).

- **Parallel**

  Each bit uses a separate wire. If there is eight bits sent at a time, this will require 8 wires, one for each data bit. The organisation looks like,

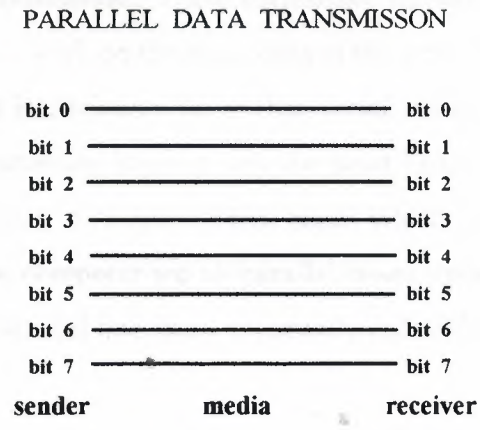

Figure 3.14. Parallel Communication

To transfer data on a parallel link, a separate line is used as a clock signal. This serves to inform the receiver when data is available. In addition, another line may be used by the receiver to inform the sender that the data has been used, and its ready for the next data.

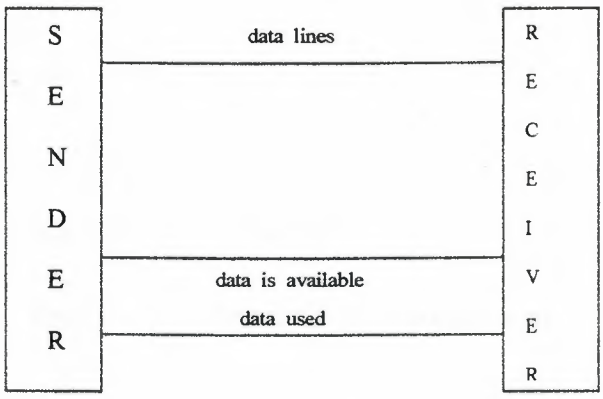

Figure 3.15.

In the figure 3.15, the sender places the data on the data lines, then signals the receiver that data is available by asserting (sending a pulse) on the **DA** (Data is available) line. After reading the state of the data lines, the receiver signals back to the sender that it has processed the data and is now ready for some more data by asserting the **DU** (Data used) line. The sender, upon getting the DU signal, removes the data and sends the next data element in the same manner.

This exchange of signals such as DA and DU between the sender and the receiver is called a **handshake**. These handshake signals allow the sender and receiver to keep synchronised (work on the same data at the same time in the proper sequence). Parallel transmission is obviously faster than serial, because more than one bit is sent at a time. Parallel transmission is good only for short links, and examples are found in all computers. The address, data and control buses which interface the processor to other peripherals inside the computer are all parallel buses. In addition, most printers on PC's (LPT1/LPT2) use a parallel interface, commonly called the Centronics Interface.

- Serial

  Each bit is sent over a single wire, one after the after. The organisation looks like,

No signal lines are used to convey clock (timing information) and handshake signals. There are two methods (asynchronous and synchronous) in which timing information is encoded with the signal so that the sender and receiver are synchronised (working on the same data at the same time). If no clock information was sent, the receiver can mis-interpret the arriving data (due to bits being lost, going too slow). In asynchronous, each character is synchronised using a start and stop signal. In synchronous, each group or block of characters is synchronised using a synchronise flag.

Figure 3.16. Serial Data Transmisison

3.3.1. Characteristics of Communication Channels

Wireline Channels :

The telephone network makes extensive use of wire lines for voice signal transmissions , as well as data and video transmission . Twisted Pair & Coaxial Cable are basically guided electromagnetic channels which provide relatively modest bandwidth .

Signals transmitted through such channels are distoted in both amplitude and phase and further corrupted by additive noise . Twisted – pair wireline channels are also prone to crosstalk interference from physically adjacent channels . Because wireline channels carry a large percentage of our daily communications around the country and the world ..

Fiber Optic Channels :

Optical fibers offer the communications system designer a channel bandwidth that is several orders of magnitude larget than coaxial cable channels . The transmitter or modulator in a fiber optic communication system is a light source , either a light – emmitting diode (LED) or a laser . Information is transmitted by varying (modulating) the intensity of the light source with the message signal . The light propogates through the fiber as a light wave and is amplified periodically along the transmission path to compensate for signal attenuation .

Wireless Electromagnetic Channels :

In wireless communication systems , electromagnetic energy is coupled to the propagation medium by an antenna which serves as the radiator . The physical size and the configuration of the antenna depend primarily on the frequency of operation . To obtain efficient radiation of electromagnetic energy ,

the antenna must be longer than 1 / 10 of the wavelength . The mode of propogation of electromagnetic waves in atmosphere and in free space may be subdivided into three categories , namely , ground – wave propagation , sky – wave propagation and line-of-sight propagation (LOS) .

Ground – wave propagation (figure 3.19) is the dominant mode of propagation for frequencies in the MF band (0.3 to 3 MHz.) . This is the frequency band used for AM broadcasting and maritime radio broadcasting .

Sky – wave propagation (figure 3.20) results from transmitted signals being reflected from the Ionospere , which consists of several layers of charged particles ranging in altitude from 30 to 250 miles above the surface of the earth . During the day – time hours , the heating of the lower atmosphere by the sun causes the formation of the lower layers at altitudes below 75 miles . However , during the night – time hours , the electron density in the lower layers of ionosphere drops sharply and the frequency absorption that occurs during the day – time is significantly reduced .

Frequencies above 30 MHz. Propogate through the ionosphere with relatively little loss and make satellite and extraterresterial communications possible . Hence , at frequencies in the VHF band and higher , the dominant mode of electromagnetic propagation is line – of – sight (LOS) propogation . For terresterial communications systems , this means that the transmitter and receiver antennas must be in direct LOS with relatively little or no obstruction . For this reason , television stations transmitting in the VHF and UHF frequency bands mount their antennas on high tower to achieve a broad coverage area .In general, the coverage area for LOS propagation is limited by the curvature of the earth .

Receiver : The function of the receiver is to recover the message signal contained in the received signal . If the message signal is transmitted by carrier modulation , the recevier performs carrier modulation to extract the message from the sinusoidal carrier . Since the signal demodulation is performed in the presence of additive noise and possibly other signal distortion , the demodulated message signal is generally degraded to some extent by the presence of these distortions in the received signal .

Besides performing the primary function of signal demodulation , the recevier also performs a number of peripheral functions , including signal filtering and noise suppression .

3.4. Interface by Parallel Ports

The Parallel Port is the most commonly used port for interfacing home made projects. This port will allow the input of up to 9 bits or the output of 12 bits at any one given time , thus requiring minimal external circuitry to implement many simpler tasks. The port is composed of 4 control lines, 5 status lines and 8 data lines . It's found commonly on the back of your PC as a D-Type 25 Pin female connector. There may also be a D-Type 25 pin male connector. This will be a serial RS-232 port and thus, is a totally incompatible port.

Newer Parallel Port's are standardised under the IEEE 1284 standard first released in 1994. This standard defines 5 modes of operation which are as follows,

1. Compatibility Mode.
2. Nibble Mode. (Protocol not Described in this Document)
3. Byte Mode. (Protocol not Described in this Document)
4. EPP Mode (Enhanced Parallel Port).
5. ECP Mode (Extended Capabilities Mode).

The aim was to design new drivers and devices which were compatible with each other and also backwards compatible with the Standard Parallel Port (SPP). Compatibility, Nibble & Byte modes use just the standard hardware available on the original Parallel Port cards while EPP & ECP modes require additional hardware which can run at faster speeds, while still being downwards compatible with the Standard Parallel Port.

Compatibility mode or "Centronics Mode" as it is commonly known, can only send data in the forward direction at a typical speed of 50 kbytes per second but can be as high as 150+ kbytes a second. In order to receive data, you must change the mode to either Nibble or Byte mode. Nibble mode can input a nibble (4 bits) in the reverse direction. E.g. from device to computer. Byte mode uses the Parallel's bi-directional feature (found only on some cards) to input a byte (8 bits) of data in the reverse direction.

Extended and Enhanced Parallel Ports use additional hardware to generate and manage handshaking. To output a byte to a printer (or anything in that matter) using compatibility mode, the software must,

1. Write the byte to the Data Port.

2. Check to see is the printer is busy. If the printer is busy, it will not accept any data, thus any data which is written will be lost.
3. Take the Strobe (Pin 1) low. This tells the printer that there is the correct data on the data lines. (Pins 2-9)
4. Put the strobe high again after waiting approximately 5n microseconds after putting the strobe low. (Step 3)

This limits the speed at which the port can run at. The EPP & ECP ports get around this by letting the hardware check to see if the printer is busy and generate a strobe and /or appropriate handshaking. This means only one I/O instruction need to be performed, thus increasing the speed. These ports can output at around 1-2 megabytes per second. The ECP port also has the advantage of using DMA channels and FIFO buffers, thus data can be shifted around without using I/O instructions.

3.4.1. Hardware Properties

Below is a table of the "Pin Outs" of the D-Type 25 Pin connector and the Centronics 34 Pin connector. The D-Type 25 pin connector is the most common connector found on the Parallel Port of the computer, while the Centronics Connector is commonly found on printers. The IEEE 1284 standard however specifies 3 different connectors for use with the Parallel Port. The first one, 1284 Type A is the D-Type 25 connector found on the back of most computers. The 2nd is the 1284 Type B, which is the 36 pin Centronics Connector, found on most printers.

IEEE 1284 Type C however, is a 36 conductor connector like the Centronics, but smaller. This connector is claimed to have a better clip latch, better electrical properties and is easier to assemble. It also contains two more pins for signals, which can be used to see whether the other device connected, has power. 1284 Type C connectors are recommended for new designs, so we can look forward on seeing these new connectors in the near future.

The table 2.3 table uses "n" in front of the signal name to denote that the signal is active low. e.g. Error. If the printer has occurred an error then this line is low. This line normally is high, should the printer be functioning correctly. The "Hardware Inverted" means the signal is inverted by the Parallel card's hardware. Such an example is the Busy line. If +5v (Logic 1) was applied to this pin and the status register read, it would return back a 0 in Bit 7 of the Status Register.

Table 3.3 Pin Assignments of the D-Type 25 pin Parallel Port Connector.

Pin No (D-Type 25)	**Pin No (Centronics)**	**SPP Signal**	**Direction In/out**	**Register**	**Hard ware Invert ed**
1	1	nStrobe	In/Out	Control	Yes
2	2	Data 0	Out	Data	
3	3	Data 1	Out	Data	
4	4	Data 2	Out	Data	
5	5	Data 3	Out	Data	
6	6	Data 4	Out	Data	
7	7	Data 5	Out	Data	
8	8	Data 6	Out	Data	
9	9	Data 7	Out	Data	
10	10	nAck	In	Status	
11	11	Busy	In	Status	Yes
12	12	Paper-Out / Paper-End	In	Status	
13	13	Select	In	Status	
14	14	nAuto-Linefeed	In/Out	Control	Yes
15	32	nError / nFault	In	Status	
16	31	nInitialize	In/Out	Control	
17	36	nSelect-Printer / nSelect-In	In/Out	Control	Yes
18 - 25	19-30	Ground	Gnd		

The output of the Parallel Port is normally TTL logic levels. The voltage levels are the easy part. The current you can sink and source varies from port to port. Most Parallel Ports implemented in ASIC, can sink and source around 12mA. However these are just some of the figures taken from Data sheets, Sink/Source 6mA, Source 12mA/Sink 20mA, Sink 16mA/Source 4mA, Sink/Source 12mA. As you can see they vary quite a bit. The best bet is to use a buffer, so the least current is drawn from the Parallel Port.

3.4.2. Port Addresses

The Parallel Port has three commonly used base addresses. These are listed in table 2, below. The 3BCh base address was originally introduced used for Parallel Ports on early Video Cards. This address then disappeared for a while, when Parallel Ports were later removed from Video Cards. They has now reappeared as an option for Parallel Ports integrated onto motherboards, upon which their configuration can be changed using BIOS.

LPT1 is normally assigned base address 378h, while LPT2 is assigned 278h. However this may not always be the case as explained later. 378h & 278h have always been commonly used for Parallel Ports. The lower case h denotes that it is in hexadecimal. These addresses may change from machine to machine.

When the computer is first turned on, BIOS (Basic Input/Output System) will determine the number of ports you have and assign device labels LPT1, LPT2 & LPT3 to them. BIOS first looks at address 3BCh. If a Parallel Port is found here, it is assigned as LPT1, then it searches at location 378h. If a Parallel card is found there, it is assigned the next free device label. This would be LPT1 if a card wasn't found at 3BCh or LPT2 if a card was found at 3BCh. The last port of call, is 278h and follows the same procedure than the other two ports. Therefore it is possible to have a LPT2 which is at 378h and not at the expected address 278h.

What can make this even confusing, is that some manufacturers of Parallel Port Cards, have jumpers which allow you to set your Port to LPT1, LPT2, LPT3. Now what address is LPT1? - On the majority of cards LPT1 is 378h, and LPT2, 278h, but some will use 3BCh as LPT1, 378h as LPT1 and 278h as LPT2. Life wasn't meant to be easy.

Table 3.4 Port Addresses

Address	Notes:
3BCh - 3BFh	Used for Parallel Ports which were incorporated on to Video Cards - Doesn't support ECP addresses
378h - 37Fh	Usual Address For LPT 1
278h - 27Fh	Usual Address For LPT 2

The assigned devices LPT1, LPT2 & LPT3 should not be a worry to people wishing to interface devices to their PC's. Most of the time the base address is used to interface the port rather than LPT1 etc. However should you want to find the address of LPT1 or any of the Line PrinTer Devices, you can use a lookup table provided by BIOS. When BIOS assigns addresses to your printer devices, it stores the address at specific locations in memory, so we can find them.

Table 3.5 - LPT Addresses in the BIOS Data Area;

Start Address	Function
0000:0408	LPT1's Base Address
0000:040A	LPT2's Base Address
0000:040C	LPT3's Base Address
0000:040E	LPT4's Base Address (Note 1)

Note 1 : Address 0000:040E in the BIOS Data Area may be used as the Extended Bios Data Area in PS/2 and newer Bioses.

The above table, table 2.5, shows the address at which we can find the Printer Port's addresses in the BIOS Data Area. Each address will take up 2 bytes.

3.4.3. Software Registers - Standard Parallel Port (SPP)

Note 1 : If the Port is Bi-Directional then Read and Write Operations can be performed on the Data Register.

The base address usually called the Data Port or Data Register is simply used for outputting data on the Parallel Port's data lines (Pins 2-9). This register is normally a

write only port. If you read from the port, you should get the last byte sent. However if your port is bi-directional, you can receive data on this address. See Bi-directional Ports for more detail.

Table 3.6 Data Port

Offset	Name	Read/Write	Bit No.	Properties
Base + 0	Data Port	Write (Note-1)	Bit 7	Data 7
			Bit 6	Data 6
			Bit 5	Data 5
			Bit 4	Data 4
			Bit 3	Data 3
			Bit 2	Data 2
			Bit 1	Data 1
			Bit 0	Data 0

Table 3.7 Status Port

Offset	Name	Read/Write	Bit No.	Properties
Base + 1	Status Port	Read Only	Bit 7	Busy
			Bit 6	Ack
			Bit 5	Paper Out
			Bit 4	Select In
			Bit 3	Error
			Bit 2	IRQ (Not)
			Bit 1	Reserved
			Bit 0	Reserved

Table 3.8 Control Port

Offset	Name	Read/Write	Bit No.	Properties
Base + 2	Control Port	Read/Write	Bit 7	Unused
			Bit 6	Unused
			Bit 5	Enable Bi-Directional Port
			Bit 4	Enable IRQ Via Ack Line
			Bit 3	Select Printer
			Bit 2	Initialize Printer (Reset)
			Bit 1	Auto Linefeed
			Bit 0	Strobe

The Status Port (base address + 1) is a read only port. Any data written to this port will be ignored. The Status Port is made up of 5 input lines (Pins 10,11,12,13 & 15), a IRQ status register and two reserved bits. Please note that Bit 7 (Busy) is a active low input. E.g. If bit 7 happens to show a logic 0, this means that there is +5v at pin 11. Likewise with Bit 2. (nIRQ) If this bit shows a '1' then an interrupt has **not** occurred.

The Control Port (base address + 2) was intended as a write only port. When a printer is attached to the Parallel Port, four "controls" are used. These are Strobe, Auto Linefeed, Initialise and Select Printer, all of which are inverted except Initialise.

The printer would not send a signal to initialise the computer, nor would it tell the computer to use auto linefeed. However these four outputs can also be used for inputs. If the computer has placed a pin high (e.g. +5v) and your device wanted to take it low, you would effectively short out the port, causing a conflict on that pin. Therefore these lines are "open collector" outputs (or open drain for CMOS devices). This means that it has two states. A low state (0v) and a high impedance state (open circuit).

Normally the Printer Card will have internal pull-up resistors, but as you would expect, not all will. Some may just have open collector outputs, while others may even

have normal totem pole outputs. In order to make your device work correctly on as many Printer Ports as possible, you can use an external resistor as well. Should you already have an internal resistor, then it will act in Parallel with it, or if you have Totem pole outputs, the resistor will act as a load.

An external 4.7 resistor can be used to pull the pin high. I wouldn't use anything lower, just in case you do have an internal pull up resistor, as the external resistor would act in parallel giving effectively, a lower value pull up resistor. When in high impedance state the pin on the Parallel Port is high (+5v). When in this state, your external device can pull the pin low and have the control port change read a different value. This way the 4 pins of the Control Port can be used for bi-directional data transfer. However the Control Port must be set to xxxx0100 to be able to read data, that is all pins to be +5v at the port so that you can pull it down to GND (logic 0).

Bits 4 & 5 are internal controls. Bit four will enable the IRQ (See Using the Parallel Ports IRQ) and Bit 5 will enable the bi-directional port meaning that you can input 8 bits using (DATA0-7). This mode is only possible if your card supports it. Bits 6 & 7 are reserved. Any writes to these two bits will be ignored.

3.4.4. Using The Parallel Port to Input 8 Bits.

If your Parallel Port doesn't support bi-directional mode, don't despair. You can input a maximum of 9 bits at any one given time. To do this you can use the 5 input lines of the Status Port and the 4 inputs (open collector) lines of the Control Port.

The inputs to the Parallel Port has be chosen as such, to make life easier for us. Busy just happens to be the MSB (Bit 7) of the Status Port, then in ascending order comes Ack, Paper Out and Select, making up the most significant nibble of the Control Port. The Bars are used to represent which inputs are Hardware inverted, i.e. +5v will read 0 from the register, while GND will read 1. The Status Port only has one inverted input.

The Control port is used to read the least significant nibble. As described before, the control port has open collector outputs, i.e. two possible states, high impedance and GND. If we connect our inputs directly to the port (For example an ADC0804 with totem pole outputs), a conflict will result if the input is high and the port is trying to pull it down. Therefore we use open collector inverters.

However this is not always entirely necessary. If we were connecting single pole switches to the port with a pull up resistor, then there is no need to bother with this protection. Also if your software initialises the control port with xxxx0100 so that all the pins on the control port are high, then it may be unnecessary. If however you don't bother and your device is connected to the Parallel Port before your software has a chance to initialise then you may encounter problems.

Another problem to be aware of is the pull up resistors on the control port. The average pull-up resistor is 4.7k. In order to pull the line low, your device will need to sink 1mA, which some low powered devices may struggle to do. Now what happens if I suggest that some ports have 1K pull up resistors? Yes, there are such cards. Your device now has to sink 5mA. More reason to use the open collector inverters.

Open collector inverters were chosen over open collector buffers as they are more popular, and thus easier to obtain. There is no reason, however why you can't use them. Another possibility is to use transistors.

The input, D3 is connected via the inverter to Select Printer. Select Printer just happens to be bit 3 of the control port. D2, D1 & D0 are connected to Init, Auto linefeed and strobe, respectively to make up the lower nibble. Now this is done, all we have to do is assemble the byte using software. The first thing we must do is to write xxxx0100 to the Control Port. This places all the control port lines high, so they can be pulled down to input data.

```
outportb(CONTROL, inportb(CONTROL) & 0xF0 | 0x04);
```

Now that this is done, we can read the most significant nibble. This just happens to be the most significant nibble of the status port. As we are only interested in the MSnibble we will AND the results with 0xF0, so that the LSnibble is clear. Busy is hardware inverted, but we won't worry about it now. Once the two bytes are constructed, we can kill two birds with one stone by toggling Busy and Init at the same time.

```
a = (inportb(STATUS) & 0xF0); /* Read MSnibble */
```

We can now read the LSnibble. This just happens to be LSnibble of the control port - How convenient! This time we are not interested with the MSnibble of the port, thus we AND the result with 0x0F to clear the MSnibble. Once this is done, it is time to

combine the two bytes together. This is done by OR'ing the two bytes. This now leaves us with one byte, however we are not finished yet. Bits 2 and 7 are inverted. This is overcome by XOR'ing the byte with 0x84, which toggles the two bits.

```
a = a |(inportb(CONTROL) & 0x0F); /* Read LSnibble */
a = a ^ 0x84; /* Toggle Bit 2 & 7 */
```

Note: Some control ports are not open collector, but have totem pole outputs. This is also the case with EPP and ECP Ports. Normally when you place a Parallel Port in ECP or EPP mode, the control port becomes totem pole outputs only. Now what happens if you connect your device to the Parallel Port in this mode? Therefore, in the interest of portability I recommend using the next circuit, reading a nibble at a time.

3.4.5. Parallel Port Modes in BIOS

Today, most Parallel Ports are multimode ports. They are normally software configurable to one of many modes from BIOS. The typical modes are,

Printer Mode (Sometimes called Default or Normal Modes)
Standard & Bi-directional (SPP) Mode
EPP1.7 and SPP Mode
EPP1.9 and SPP Mode
ECP Mode
ECP and EPP1.7 Mode
ECP and EPP1.9 Mode

Parallel Port Modes and the ECP's Extended Control Register

It is better to set the Parallel Port to ECP and EPP1.9 Mode and use the ECP's Extended Control Register to select different modes of operation. The ECP Registers are standardised under Microsoft's Extended Capabilities Port Protocol and ISA Interface Standard, thus we don't have that problem of every vendor having their own register set.

When set to ECP Mode, a new set of registers become available at Base + 0x400h. Here we are only interested in the Extended Control Register (ECR) which is mapped at Base + 0x402h. It should be stated that the ECP's registers are not available for port's with a base address of 0x3BCh.

Table 3.9 - Extended Control Register (ECR)

Bit	Function	
7:5	Selects Current Mode of Operation	
	000	Standard Mode
	001	Byte Mode
	010	Parallel Port FIFO Mode
	011	ECP FIFO Mode
	100	EPP Mode
	101	Reserved
	110	FIFO Test Mode
	111	Configuration Mode
4	ECP Interrupt Bit	
3	DMA Enable Bit	
2	ECP Service Bit	
1	FIFO Full	
0	FIFO Empty	

The table 3.9 is of the Extended Control Register. We are only interested in the three MSB of the Extended Control Register which selects the mode of operation. There are 7 possible modes of operation, but not all ports will support all modes. The EPP mode is one such example, not being available on some ports.

3.5. Interface by Serial / RS232 Port

The Serial Port is harder to interface than the Parallel Port. In most cases, any device you connect to the serial port will need the serial transmission converted back to parallel so that it can be used. This can be done using a UART. On the software side of things, there are many more registers that you have to attend to than on a Standard Parallel Port. (SPP)

Advantages of using serial data transfer rather than parallel :

Serial Cables can be longer than Parallel cables. The serial port transmits a '1' as -3 to -25 volts and a '0' as +3 to +25 volts where as a parallel port transmits a '0' as 0v and a '1' as 5v. Therefore the serial port can have a maximum swing of 50V compared to the parallel port which has a maximum swing of 5 Volts. Therefore cable loss is not going to be as much of a problem for serial cables than they are for parallel.

You don't need as many wires than parallel transmission. If your device needs to be mounted a far distance away from the computer then 3 core cable (Null Modem Configuration) is going to be a lot cheaper that running 19 or 25 core cable. However you must take into account the cost of the interfacing at each end.

Infra Red devices have proven quite popular recently. You may of seen many electronic diaries and palm top computers which have infra red capabilities build in. However could you imagine transmitting 8 bits of data at the one time across the room and being able to (from the devices point of view) decipher which bits are which? Therefore serial transmission is used where one bit is sent at a time. IrDA-1 (The first infra red specifications) was capable of 115.2k baud and was interfaced into a UART. The pulse length however was cut down to $3/16^{th}$ of a RS232 bit length to conserve power considering these devices are mainly used on diaries, laptops and palmtops.

Microcontroller's have also proven to be quite popular recently. Many of these have in built SCI (Serial Communications Interfaces) which can be used to talk to the outside world. Serial Communication reduces the pin count of these MPU's. Only two pins are commonly used, Transmit Data (TXD) and Receive Data (RXD) compared with at least 8 pins if you use a 8 bit Parallel method (You may also require a Strobe).

3.5.1. Hardware Properties

Devices which use serial cables for their communication are split into two categories. These are DCE (Data Communications Equipment) and DTE (Data Terminal Equipment.) Data Communications Equipment are devices such as your modem, TA adaptor, plotter etc while Data Terminal Equipment is your Computer or Terminal .

The electrical specifications of the serial port is contained in the EIA (Electronics Industry Association) RS232C standard. It states many parameters such as –

1. A "Space" (logic 0) will be between +3 and +25 Volts.
2. A "Mark" (Logic 1) will be between -3 and -25 Volts.

3. The region between +3 and -3 volts is undefined.
4. An open circuit voltage should never exceed 25 volts. (In Reference to GND)
5. A short circuit current should not exceed 500mA. The driver should be able to handle this without damage. (Take note of this one!)

Above is no where near a complete list of the EIA standard. Line Capacitance, Maximum Baud Rates etc are also included. For more information please consult the EIA RS232-C standard. It is interesting to note however, that the RS232C standard specifies a maximum baud rate of 20,000 BPS!, which is rather slow by today's standards. A new standard, RS-232D has been recently released.

Serial Ports come in two "sizes", There are the D-Type 25 pin connector and the D-Type 9 pin connector both of which are male on the back of the PC, thus you will require a female connector on your device. Below is a table of pin connections for the 9 pin and 25 pin D-Type connectors.

The EIA RS232-C Standard

Specifies a 25 pin connector as the standard interface in data communication networks, with lettering pin designations for ground, data, control and timing circuits. The table 3.9 shows the designations for each of the 25 pins of the standard.

Table 3.9

INTERCHANGE	CIRCUIT NO.	PIN NO.	DESCRIPTION
AA	101	1	Projective Ground
BA	103	2	Transmit Data
BB	104	3	Receive Data
CA	105	4	Request to Send
CB	106	5	Clear to Send
CC	107	6	Data set ready
AB	102	7	Signal Ground
CF	109	8	Receive Line Signal
			Detect / Carrier Detect
---	---	9	Reserved
---	---	10	Reserved
---	---	11	Unassigned
SCF	122	12	Secondary RLSD
SCB	121	13	Secondary CTS
SBA	118	14	Secondary TD
DB	114	15	Transmitter Signal

			Element Timing
SBB	119	16	Secondary RD
DD	115	17	Receiver Signal
			Timing Element
---	---	18	Unassigned
SCA	120	19	Secondary RTS
CD	108.2	20	Data Terminal Ready
CG	110	21	Signal Quality
			Detector
CE	125	22	Ring Indicator
CH / CI	111 / 112	23	Data Signal Rate
			Detector
DA	113	24	Transmit Signal
			Element Timing
---	---	25	Unassigned

DTE / DCE Speeds

We have already talked briefly about DTE & DCE. A typical Data Terminal Device is a computer and a typical Data Communications Device is a Modem. Often people will talk about DTE to DCE or DCE to DCE speeds. DTE to DCE is the speed between your modem and computer, sometimes referred to as your terminal speed. This should run at faster speeds than the DCE to DCE speed. DCE to DCE is the link between modems, sometimes called the line speed.

Most people today will have 28.8K or 33.6K modems. Therefore we should expect the DCE to DCE speed to be either 28.8K or 33.6K. Considering the high speed of the modem we should expect the DTE to DCE speed to be about 115,200 BPS.(Maximum Speed of the 16550a UART) This is where some people often fall into a trap. The communications program which they use have settings for DCE to DTE speeds. However they see 9.6 KBPS, 14.4 KBPS etc.

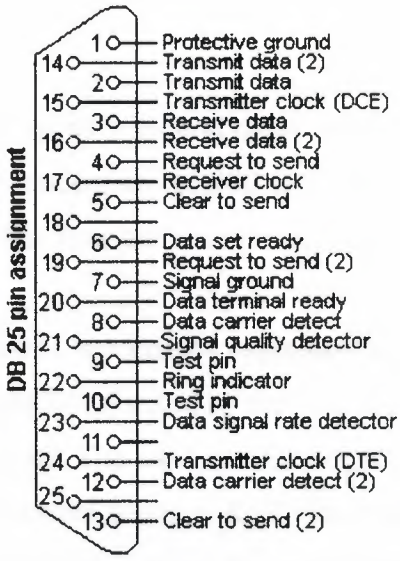

Figure 3.17. RS232 female connector

Today's Modems should have Data Compression build into them. This is very much like PK-ZIP but the software in your modem compresses and decompresses the data. When set up correctly you can expect compression ratios of 1:4 or even higher. 1 to 4 compression would be typical of a text file. If we were transferring that text file at 28.8K (DCE-DCE), then when the modem compresses it you are actually transferring 115.2 KBPS between computers and thus have a DCE-DTE speed of 115.2 KBPS. Thus this is why the DCE-DTE should be much higher than your modem's connection speed.

Some modem manufacturers quote a maximum compression ratio as 1:8. Lets say for example its on a new 33.6 KBPS modem then we may get a maximum 268,800 BPS transfer between modem and UART. If you only have a 16550a which can do 115,200 BPS tops, then you would be missing out on a extra bit of performance. Buying a 16C650 should fix your problem with a maximum transfer rate of 230,400 BPS.

These are MAXIMUM compression ratios. In some instances if you try to send a already compressed file, your modem can spend more time trying the compress it, thus you get a transmission speed less than your modem's connection speed. If this occurs try turning off your data compression. This should be fixed on newer modems. Some files compress easier than others thus any file which compresses easier is naturally going to have a higher compression ratio.

3.5.2. Flow Control

If our DTE to DCE speed is several times faster than our DCE to DCE speed the PC can send data to your modem at 115,200 BPS. Sooner or later data is going to get lost as buffers overflow, thus flow control is used. Flow control has two basic varieties, Hardware or Software.

Software flow control, sometimes expressed as Xon/Xoff uses two characters Xon and Xoff. Xon is normally indicated by the ASCII 17 character where as the ASCII 19 character is used for Xoff. The modem will only have a small buffer so when the computer fills it up the modem sends a Xoff character to tell the computer to stop sending data. Once the modem has room for more data it then sends a Xon character and the computer sends more data. This type of flow control has the advantage that it doesn't require any more wires as the characters are sent via the TD/RD lines. However on slow links each character requires 10 bits which can slow communications down.

Hardware flow control is also known as RTS/CTS flow control. It uses two wires in your serial cable rather than extra characters transmitted in your data lines. Thus hardware flow control will not slow down transmission times like Xon-Xoff does. When the computer wishes to send data it takes active the Request to Send line. If the modem has room for this data, then the modem will reply by taking active the Clear to Send line and the computer starts sending data. If the modem does not have the room then it will not send a Clear to Send.

RS-232 Signal Descriptions: The interface transfers data between the computer and the modem via the TD and RD lines. The other signals are essentially used for FLOW CONTROL, in that they either grant or deny requests for the transfer of information between a DTE and a DCE. Data cannot be transferred unless the appropriate flow control line are first asserted.

The interface can send data either way(DTE to DCE, or, DCE to DTE) independently at the same time. This is called FULL DUPLEX operation. Some systems may utilize software codes so that information may only be transmitted in one direction at a time (HALF DUPLEX), and requires software codes to switch from one direction to another (i.e., from a transmit to receive state).

The following is a list of common RS232 signals.

- Request To Send (RTS): This signal line is asserted by the computer (DTE) to inform the modem (DCE) that it wants to transmit data. If the modem decides this is okay, it will assert the CTS line. Typically, once the computer asserts RTS, it will wait for the modem to assert CTS. When CTS is asserted by the modem, the computer will begin to transmit data.
- Clear To Send (CTS): Asserted by the modem after receiving a RTS signal, indicating that the computer can now transmit.
- Data Terminal Ready (DTR): This signal line is asserted by the computer, and informs the modem that the computer is ready to receive data.
- Data Set Ready (DSR): This signal line is asserted by the modem in response to a DTR signal from the computer. The computer will monitor the state of this line after asserting DTR to detect if the modem is turned on.
- Receive Signal Line Detect (RSLD): This control line is asserted by the modem, informing the computer that it has established a physical connection to another modem. It is sometimes known as Carrier Detect (CD). It would be pointless a computer transmitting information to a modem if this signal line was not asserted. If the physical connection is broken, this signal line will change state.
- Transmit Data (TD): is the line where the data is transmitted, a bit at a time.
- Receive Data (RD): is the line where data is received, a bit at a time.

A lot of signals work in pairs. Some signals are generated by the DTE, and some signals are generated by the DCE. If you were measuring the signals on a computer which was NOT connected to a modem, you could only expect to see those signals that the DTE can generate.

Exchanging information between a DCE and DTE :

Now, lets look at the sequence that occurs when data is transferred between a DTE and a DCE. The data can only be transferred after the correct sequence of signals is followed, for instance, there is no point sending data if the modem is turned off. Lets go through each of the steps involved (i.e., signal line assertions required) to transmit and receive characters across the RS232 interface.

TRANSMITTING DATA (DTE to DCE)

- 1: Assert DTR and RTS
- 2: Wait for DSR
- 3: Wait for CTS

- 4: Transmit the data

Step 1 and 2 are essential to ensure that the modem is on-line and connected to another modem. Waiting for DSR checks that the modem is on-line.

RECEIVING DATA (DCE to DTE)

- 1: Assert DTR
- 2: Wait for DSR
- 3: Receive the data

3.5.3. Connection of Two DTE Devices

Often, two DTE devices need to be connected together using a serial link. This is for file transfer or printer access. The problem is that DTE devices expect to talk directly to DCE devices, not another device of the same type. DTE's cannot generate signals like DSR and CTS, so connecting two DTE's together will result in neither getting permission to send, and thinking that the modem is off-line (by not receiving DSR).

To allow the interconnection of two DTE devices without using DCE's, a special type of cable must be used. This is called a Null Modem Cable, which fools the DTE into thinking that it is connected to a DCE device. In this case, modems are not used, so the connection looks like :

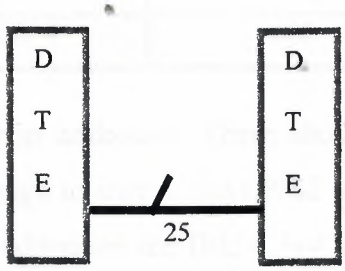

Figure 3.18.

RS232D (9 pin Connector)

The following table illustrates the 9 pin serial connector as found on most PC's today. This has all but replaced the previous 25 pin connector found on earlier PC's.

Table 3.10 Pin Descriptions of DB9

SIGNAL	PIN No.
Carrier Detect	1
Receive Data	2
Transmit Data	3
Data Terminal Ready	4
Signal Ground	5
Data Set Ready	6
Request To Send	7
Clear To Send	8
Ring Indicator	9

3.5.4. Serial Port's Registers (PC's)

Port Addresses & IRQ's

Table 3.11 Standard Port Addresses

Name	Address	IRQ
COM 1	3F8	4
COM 2	2F8	3
COM 3	3E8	4
COM 4	2E8	3

Above is the standard port addresses. These should work for most P.C's. If you just happen to be lucky enough to own a IBM P/S2 which has a micro-channel bus, then expect a different set of addresses and IRQ's. Just like the LPT ports, the base addresses for the COM ports can be read from the BIOS Data Area.

Table 3.12 COM Port Addresses in the BIOS Data Area

Start Address	Function
0000:0400	COM1's Base Address
0000:0402	COM2's Base Address
0000:0404	COM3's Base Address
0000:0406	COM4's Base Address

First In / First Out Control Register (FCR)

The FIFO register is a write only register. This register is used to control the FIFO (First In / First Out) buffers which are found on 16550's and higher.

Bit 0 enables the operation of the receive and transmit FIFO's. Writing a '0' to this bit will disable the operation of transmit and receive FIFO's, thus you will loose all data stored in these FIFO buffers.

Bit's 1 and 2 control the clearing of the transmit or receive FIFO's. Bit 1 is responsible for the receive buffer while bit 2 is responsible for the transmit buffer. Setting these bits to 1 will only clear the contents of the FIFO and will not affect the shift registers. These two bits are self resetting, thus you don't need to set the bits to '0' when finished.

Bit 3 enables the DMA mode select which is found on 16550 UARTs and higher. More on this later. Bits 4 and 5 are those easy type again, Reserved.

Table 3.13: FIFO Control Register

Bit	Notes		
Bits 6 and 7	Bit 7	Bit 6	Interrupt Trigger Level
	0	0	1 Byte
	0	1	4 Bytes
	1	0	8 Bytes
	1	1	14 Bytes
Bit 5	Enable 64 Byte FIFO (16750 only)		
Bit 4	Reserved		
Bit 3	DMA Mode Select. Change status of RXRDY & TXRDY pins from mode 1 to mode 2.		
Bit 2	Clear Transmit FIFO		
Bit 1	Clear Receive FIFO		
Bit 0	Enable FIFO's		

Bits 6 and 7 are used to set the triggering level on the Receive FIFO. For example if bit 7 was set to '1' and bit 6 was set to '0' then the trigger level is set to 8 bytes. When there is 8 bytes of data in the receive FIFO then the Received Data Available interrupt is set.

The line status register is a read only register. Bit 7 is the error in received FIFO bit. This bit is high when at least one break, parity or framing error has occurred on a byte which is contained in the FIFO.

When bit 6 is set, both the transmitter holding register and the shift register are empty. The UART's holding register holds the next byte of data to be sent in parallel fashion. The shift register is used to convert the byte to serial, so that it can be transmitted over one line. When bit 5 is set, only the transmitter holding register is empty. So what's the difference between the two? When bit 6, the transmitter holding and shift registers are empty, no serial conversions are taking place so there should be no activity on the transmit data line. When bit 5 is set, the transmitter holding register is empty, thus another byte can be sent to the data port, but a serial conversion using the shift register may be taking place.

The break interrupt (Bit 4) occurs when the received data line is held in a logic state '0' (Space) for more than the time it takes to send a full word. That includes the time for the start bit, data bits, parity bits and stop bits.

A framing error (Bit 3) occurs when the last bit is not a stop bit. This may occur due to a timing error. You will most commonly encounter a framing error when using a null modem linking two computers or a protocol analyser when the speed at which the data is being sent is different to that of what you have the UART set to receive it at.

A overrun error normally occurs when your program can't read from the port fast enough. If you don't get an incoming byte out of the register fast enough, and another byte just happens to be received, then the last byte will be lost and a overrun error will result.

Bit 0 shows data ready, which means that a byte has been received by the UART and is at the receiver buffer ready to be read.

3.6. DAC & ADC Conversion

We have a certain number of bits whose logic levels are characterised with electric levels, say logic zero = 0 volts and logic one = V volts. This bit set can be interpreted as a number comprised between zero and 2n - 1, and so it can represent the

levels of an analog signal. The analog signal can be extracted directly from the digital signals if the electric levels of all of the bits are summed up with certain weights. The most significant bit is not weighted (that is, it has weight = 1), the bit next to it has a weight of 1/2, the following bit has a weight of 1/4 and so on. The least significant bit will have a weight of 1/2n - 1. This way the resulting output signal is analog (because it is not restricted to two values, 0 or V volts) and its level is proportional to the logic value represented by the bit set.

In the practice the DAC is applied to the parallel port of the PC. In this case we have 8 bits (but it could be more if we use not only the data pins, but also the control pins) whose electric levels are 0 volts (for logic zero) and approximately +5 volts (for logic one). The summing and weighting of the bits are made through resistors. The simplest way is to connect the pin of the bit number j with a resistor whose value is 2j x R. This would require 8 resistors, each twice the previous one, something which is difficult to achieve in practice. That's why a more elaborated circuit was designed. In the figure you can see that the DAC in made with a chain of

resistors which only needs two different kinds of resistors. This makes the circuit stabler and easier to build in practice. The output signal will be approximately comprised between zero and +5 volts with 28 = 256 different levels, that is, the resolution of the DAC is 8 bits.

The main problem was in the design of a simple threshold detector. It has to be sensible to one step of the DAC, which is 5/256 = 20 mV. The conversion of a small step of voltage into a logic signal (for example, 5 volts if the step exceeded threshold and 0 volts otherwise) is typically done through a chain of transistors, so that they convert a step of 20 mV into a step of 5 V (gain = 5/20x10-3 = 250). As long as I wanted to keep the design as simple as possible I tried to use only one transistor, then the transistor has to be carefully chosen and polarized. My knowledge of transistors was not enough to do it this way, so I tried another approach.

The specifications of the logic levels say basically that there are 3 situations when an input signal is applied. If the signal is below some level, say 1 volt, it will be interpreted as a logic 0. If the signal is above a certain level, say 4 volts, it will be interpreted as a logic 1. If the signal is between both level then nothing can be said, the manufacturer of the equipment cannot predict the behaviour, it will be randomly interpreted as a logic 0 or a logic 1. My bet was that the level in which the

interpretation of the input signal changes from logic 0 to logic 1 is fixed and stable enough, though it will change randomly from computer to computer.

The input signal to be digitalised is added to the DAC signal and applied to the base of a transistor. The collector of the transistor is connected to an input of the parallel port and the emitter to a variable resistor. The ends of the variable resistor are connected to ground and to a negative voltage source. I took the serial port as a source for this negative voltage, but any other source can be used. The serial port uses two electric levels, -12 and +12 volts approximately. There should not be risk of damage because the power consumption is low, besides, by definition of the RS-232C standard, any pin can be short-circuited with any other pin (including ground) without damage, otherwise your port is not a real RS-232C. When the signal in the base of the transistor grows, the voltage of the collector decreases, that's how the input of the parallel port is set from 1 to 0. The variable resistor has to be adjusted so that in absence of signal the digitalisation results in the level 128 of 256. The volume control of the sound source (microphone, cassette, radio, CD, etc.) has to be adjusted so that the extreme levels (0 and 255) are almost never reached (to avoid saturation).

After the first trials I was surprised with the stability of the device. My bet was right. Once the variable resistor has been correctly set, the fluctuation is +/- 1 LSB, that is, like professional devices!

According to Nyquist's theorem, the maximum frequency that can be successfully digitalised is half of the sampling frequency. With my 386/40 MHz running a C program I scanned at a rate of 14 kHz. This sets the limit in 7 kHz, which is enough for human voice but not for hi-fi music. With a faster computer and programming directly in assembler the limit should be raised. To eliminate frequencies incorrectly digitalised, so they appear as noise, I added the low pass filter that can be found in the design. If your computer is faster than mine maybe you'll have to change the filter's constant to be optimum with your setup.

The digitalisation process requires a lot of memory, at a sampling rate of 14 kHz, the data flow is 14 kb per second. As long as I don't know how to manage extended memory, my particular solution was to create a RAM disk and use it to store temporarily the data.

The design is very experimental and is intended to serve as an introduction to basic electronics, interfacing PCs with your own devices and its programming (you have to

take into account that now really good sound cards have become quite cheap). I'm currently working on other devices so I don't plan to improve this design, but it can be done in many ways. Try it. The filter can be better, the extra source (for which I took the serial port) might be removed, the manual switch that changes from ADC mode to DAC mode can be substituted by an automatic switch (transistor, relay) controlled with the parallel port itself. Also the software is very naive, I didn't care about timing, so now one have to try several times to play the data at the right speed. More efficient code could be written to use extended memory, to make some sound enhancement, data compression, etc.

As a closing remark, it is interesting to notice that there are several sound cards which use the parallel port, like Disney Sound. They are basically a DAC as described here. Therefore it is very likely that you can play the example found in the preceding page with the software there, in any of these cards. Conversely it's also true, you may use the design of this page with any program that allows sound cards attached to the parallel port.

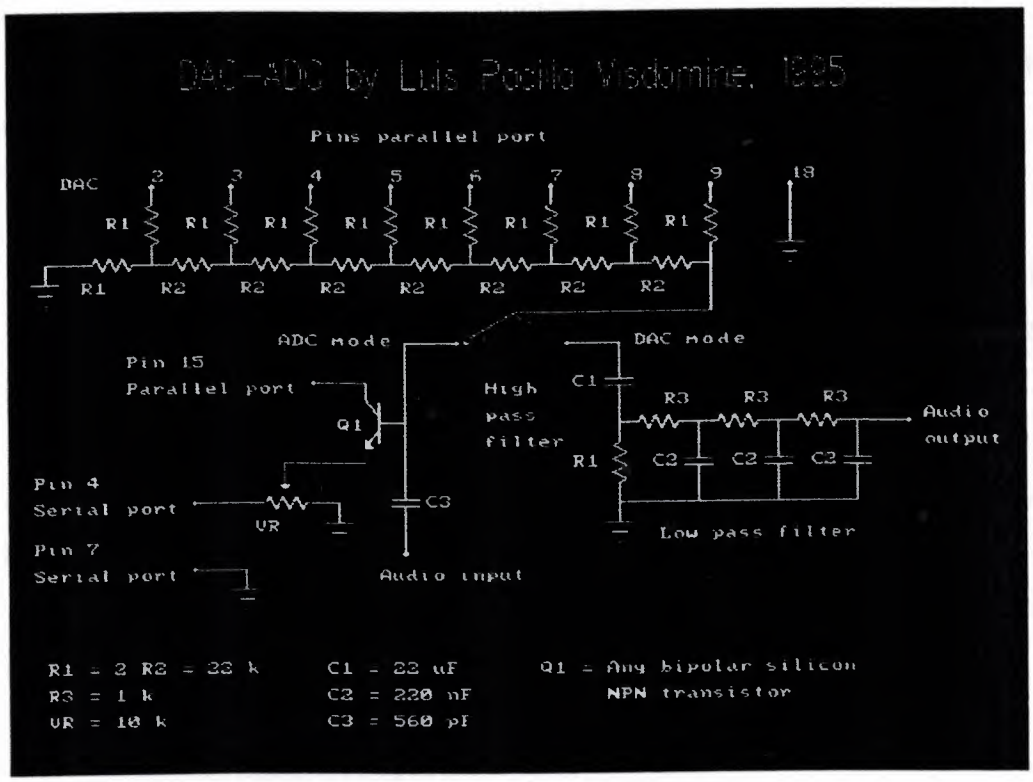

Figure 3.19.

The analog-to-digital converter (ADC) is used to convert an analog voltage to a digital number (figure 3.20).

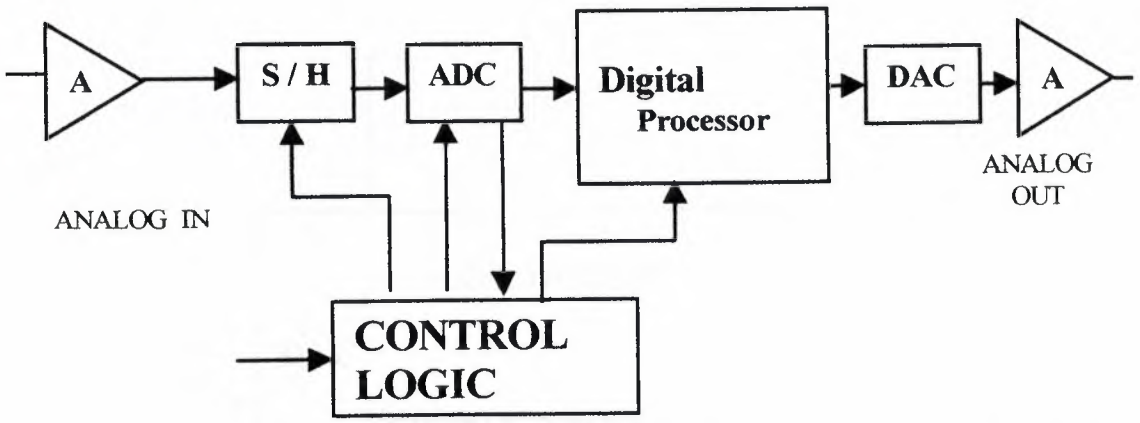

Figure 3.20 A generalised hybrid and digital circuit by which input analog data can be transmitted, stored, delayed, or otherwise processed as a digital number before re-conversion back to an analog output.

The parallel-encoding or flash ADC design provides the fastest operation at the expense of high component count and high cost (figure).

The resistor network sets discrete thresholds for a number of comparators. All comparators with thresholds above the input signal go false while those below go true. Then digital encoding logic converts the result to a digital number.

The successive-approximation ADC is the most commonly used design (figure). This design requires only a single comparator and will be only as good as the DAC used in the circuit.

The analog output of a high-speed DAC is compared against the analog input signal. The digital result of the comparison is used to control the contents of a digital buffer that both drives the DAC and provides the digital output word.

The successive-approximation ADC uses fast control logic which requires only n comparisons for an n-bit binary result (figure 3.21).

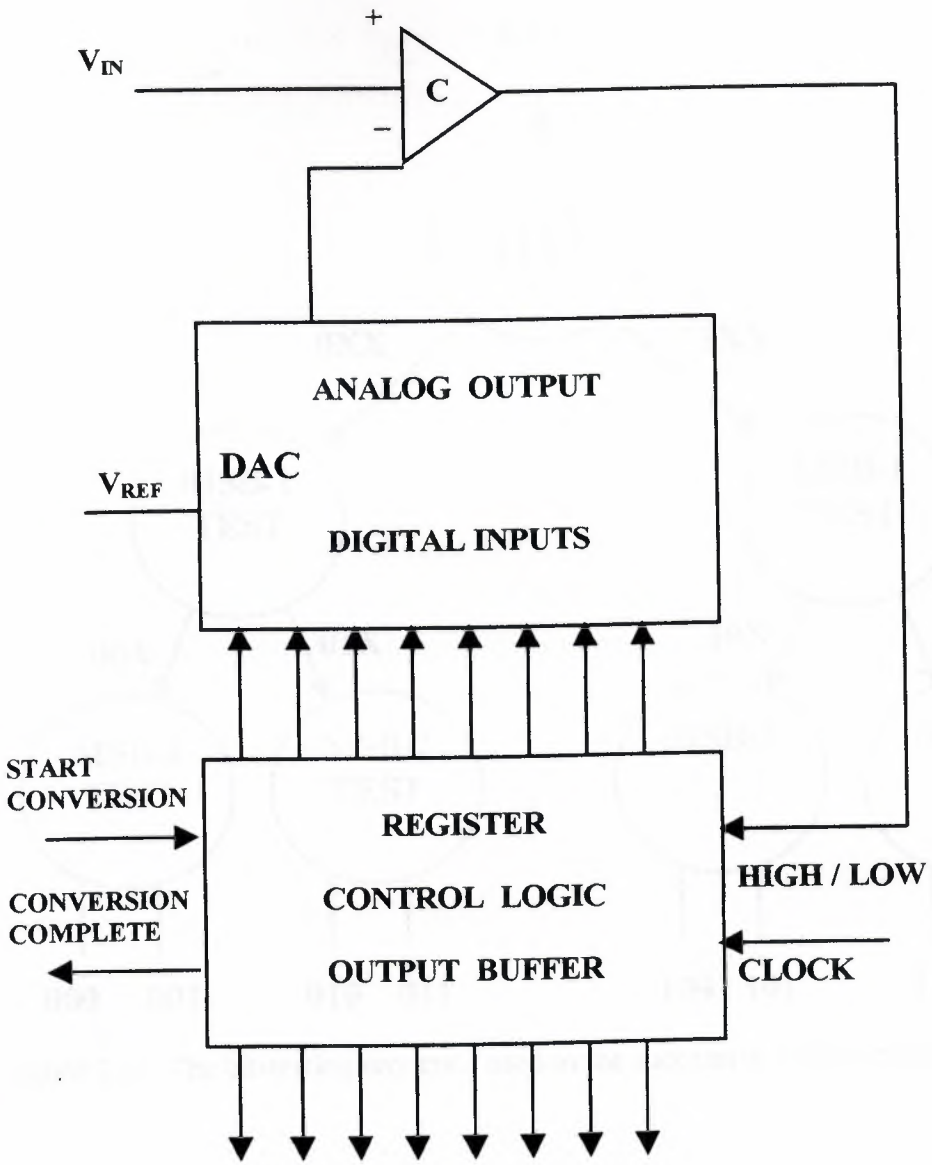

Figure 3.22. The block diagram of an 8-bit successive-approximation ADC

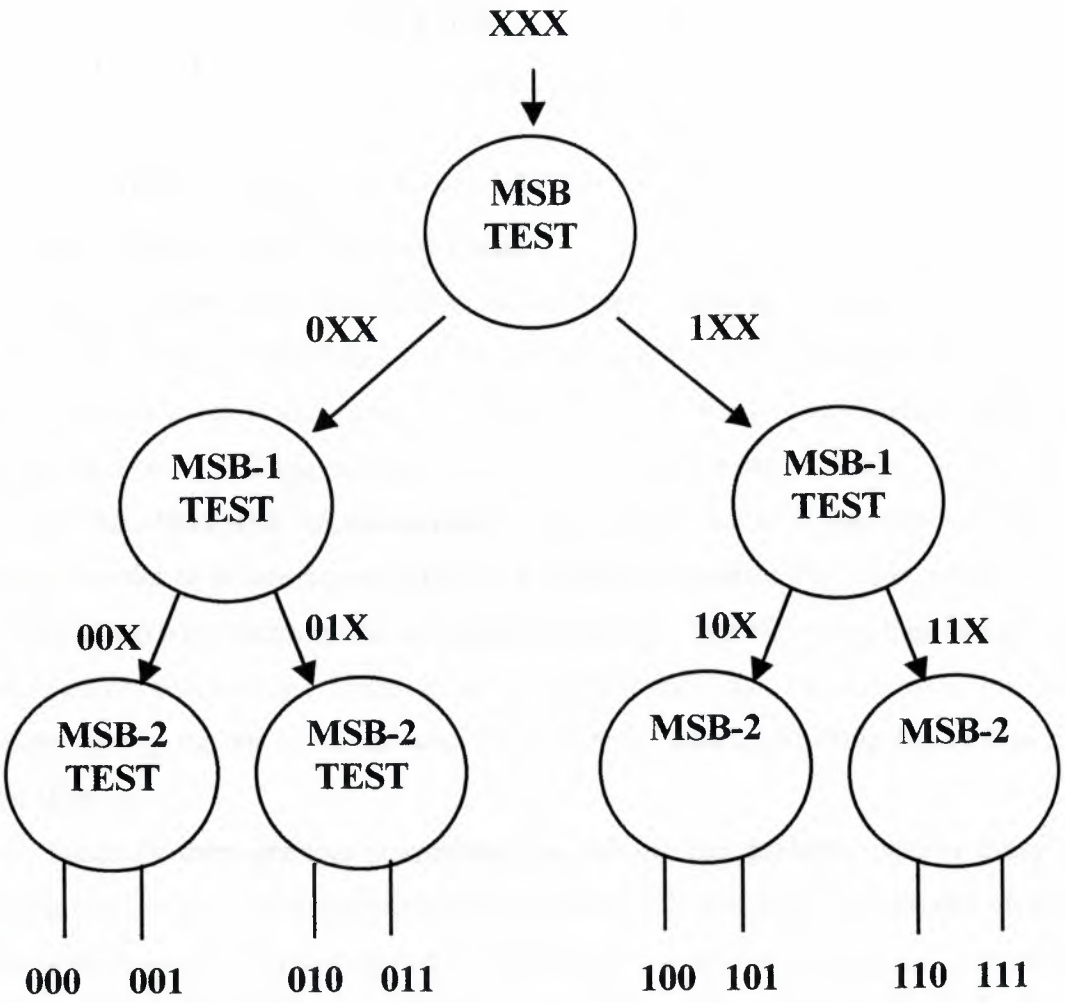

Figure 3.21. The bit-testing sequence used in the successive approximation method.

CHAPTER 4. ARTIFICAL INTELLEGNCE IN ROBOT CONTROL

4.1. Neural Control of Robot Hand

4.1.1. Force Control of Robot Hand

To manipulate an object with a multi-fingered gripper the fingers have to exert forces on the object. These forces can be divided into two classes, namely forces causing motion, called manipulation forces, and holding forces. While manipulation forces cause moving the object, holding forces keep it in rest. Manipulation forces, arising from a contact of the object with the surroundings, might cause sliding of the object inside of the gripper. In order to secure a save grip it is inevitable to control the finger forces in a way that they always compensate the manipulation forces. Therefore, holding forces can be superposed on acting finger forces in order to bring them into the interior of the friction cone, presenting the set of all allowed finger forces, without effecting the motion of die object (Fig. 4.1)

Basically there are two possibilities for solving this problem, namely grasp force planning and grasp force control. Grasp force planning is expected to determine an optimal initial force state of die system in order to secure a successful manipulation. Therefore, die force planning component has to compute finger forces mat avoid sliding of die object during die manipulation.

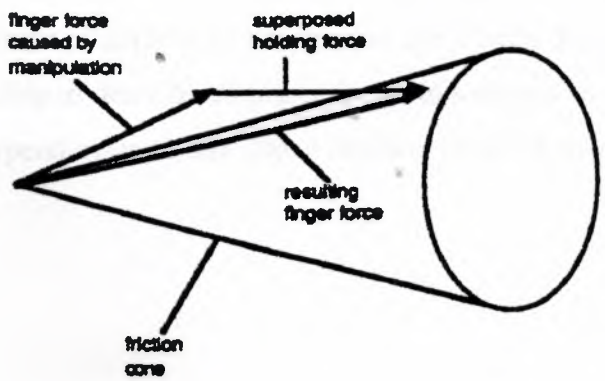

Figure 4.1. Holding and manipulation forces

This can be achieved either by determination of finger forces, compensating the whole spectrum of manipulation forces deriving from die object handling, or by only optimising the initial force state of die system followed by the on-line force adaptation. Grasp force planning is particularly based upon a precise mathematical model of die system and uses optimisation algorithms for the determination of an initial grasp force. The reason for only applying these optimisation algorithms to planning tasks is their high expense in time. This forbids their real-time application for an adaptive force control.

To solve this problem, other control structures have to be developed. The basic idea is to transfer highly precise planning algorithms to neural networks in order to use their capability of massive parallel and distributed processing. That allows to achieve die desired real-time capability for on-line force controlling without loosing the precision provided by the planning algorithms. Other possible way is to use the fuzzy decision making logic, which expresses the a-priory knowledge about the holding force behaviour inside the friction cones, to produce die necessary reactions of die force control system.

The corresponding fuzzy and neural control algorithms for force planning and adaptation of a multi fingered gripper, are presented in this chapter, It is upon the following simple model of interaction between the finger tips and the surface of the object. The contact between a finger tip and the object is treated as a single point; further it is assumed, that the finger is capable of only imposing forces on the object but no moments. This model is called point contact with friction. The force adaptations are based on the following control strategy: The contact between the finger tips and the object must be saved. To grant this request the following friction condition has to be satisfied. The tangential force must be less than the normal force multiplied by the friction coefficient. For an easier treatment of this problem it is possible to describe all finger forces in a co-ordinate system, in which the x-axis is pointing perpendicular to the object surface directed towards the interior (Fig. 4.2).

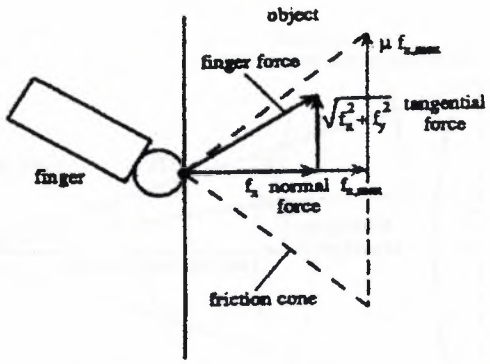

Figure 4.2. Finger-object contact

To avoid sliding, a finger force *f* has to satisfy the following condition:

$$0 \le |f_x| \le f_{\max}, \sqrt{f_y^2 + f_z^2} \le \mu . |f_x|$$

Where μ denotes the coefficient of friction, and the maximum normal force of a finger. Due to this condition, die set of all allowed finger forces, called the friction cone, is the following:

$$F := \left\{ f \mid 0 \le |f_x| \le f_{\max}, \sqrt{f_y^2 + f_z^2} \le \mu . |f_x| \right\}$$

As there is more than one possibility for compensating finger forces, a criterion for their comparison must be given in order to choose the best among them. A finger force sh9uld not lie too close to the border of the friction cone where sliding might occur. Hence, the force controller has to observe the condition under consideration that the minimal distance of the measured force from the lateral area and the bottom face of the friction cone is a maximum (Fig.3-3). If this condition is satisfied the grasp is called stable:

$$S(f) := \{\min(|f - P(f)|, f_{\max} - |f_x|), IFf \varepsilon F0$$

With *P(f)* being die projection of die finger force on the lateral area of the frictionc

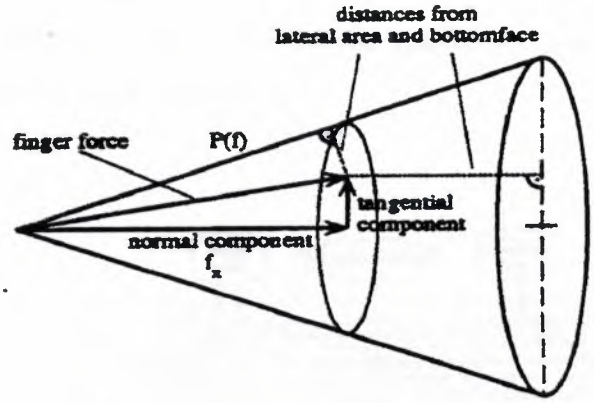

Figure 4.3. Stability of a finger force

Because a grip is only as stable as the smallest stability value of all participating finger forces, the grip stability *St* for a gripper, consisting of n fingers, is defined as:

$$S_t = \min_{i=1}^{n}(S(f_i))$$

If a grip is optimised by maximizing its stability. all finger forces have a safety distance to the border of the cone. Manipulation forces, changing during the manipulation, do in general not cause the finger forces to leave the corresponding friction cones immediately.

4.1.2. Multi-Level Control System

The concept of the multi-level control system architecture for a multifingered robot hand combines the advantages of neural networks and a fuzzy logic approach. An important property of advanced control systems is the fine manipulation control of a robot hand requiring force planning and real-time force computation capabilities. Therefore, the system configuration for fine manipulation control (Fig. 4.4) is realized as a hierarchical multi-level organization based on a neuro adaptation level and a fuzzy correction level that are co-coordinated at the task interpretation level.

The determination of relevant grasp parameters is .performed with the help of a planning component. The object location continually being supervised by the optical and

tactile sensors. Using the information on the object location and that from the world model and knowledge base. the planning level of the system successively determines me grasp points. me grip matrix and the grasp force.

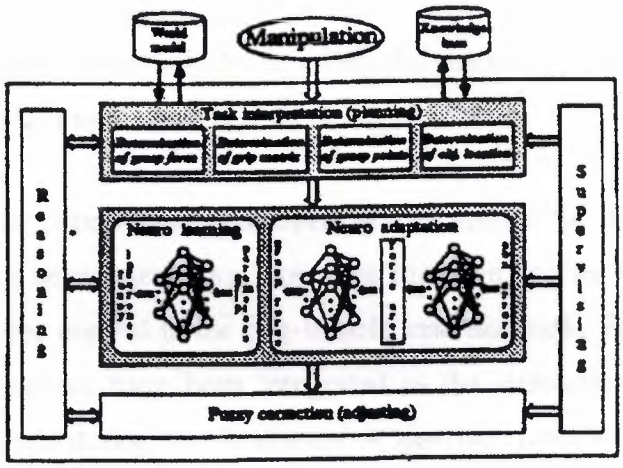

Fig.4.4.Multi-level control system architecture

The neural controller that will be described in detail below performs the real-time adaptation of grasp forces. Furthermore, the neural component of the system is continually learning the input-output behaviour of the underlying fuzzy controller to replicate it. After the learning is complete, the fuzzy controller can be replaced by die neural network that provides die better real-time properties of the control system. The main task of the fuzzy control level is to perform a short-term local correction of the grasp forces within die corresponding friction cones. The fuzzy controller interacts with the underlying conventional PID controller, which receives as input the force values, which must be applied. During the manipulation, the reasoning process coordinates the interactions of all system levels according to the current process state.

4.1.3. Neural Force Control

For the development of grasp force controllers for multi-fingered robot hands two main problems have to be solved: As force adaptation requires anon-linear control, the family of conventional P, PI, PID-controllers is not suitable. The second problem arises from the real time conditions the controllers have to fulfil. For both of these problems an

approach using neural networks seems suitable. not only because of their ability for non-linear control but also for the reduction of computation time by exploiting the characteristics of neural nets i.e. their massive parallel and distributed processing.

A main issue in the development of the neural force controllers was to achieve the preciseness of conventional planning algorithms and fuzzy controllers with an improvement in computation time. in order to satisfy the strict real time conditions required for the peg-in-hole-insertion task.

The idea for the two first neural force controllers was to imitate the control behaviours of existing algorithms with neural networks. This idea bas been applied to conventional and fuzzy algorithms for force control in the peg-in-hole insertion task. For the generation of training data both algorithms have been integrated in the simulation module described below and their in-/outputs during the simulation of insertion tasks were sampled on disc for a later training of the neural networks (see Fig.4.5).

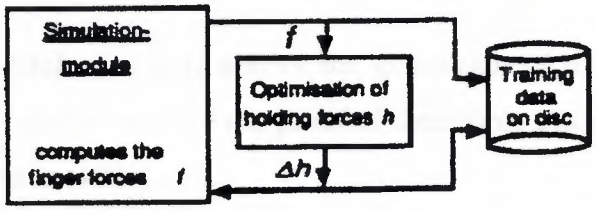

Figure 4.5. Generation of training data for the neural nets using the simulation module and conventional algorithms for force adaptation (optimisation of holding forces)

The third approach for designing a neural controller exploits neural networks capability of generalization. The neural controller hasn't been developed for imitating other algorithm's control behaviours but for learning force control on only a few but characteristic aims of force control. All the three neural controller which will be presented have been realized by three layered back propagation neural networks. Their control behaviour will be compared by giving me maximum initial positioning errors $\Delta\otimes$ (the vertical angle error) and Δx (the horizontal positioning error) which could be tolerated by force control as also the computation time, which has been measured by 100.000 calls on a Spare 2 workstation.

4.1.4. Simulation Module

The neural grasp force controllers, which are presented in this paper have been developed for a d1ree fingered robot band performing the frequently occurring peg-in-hole insertion task During this task positioning errors Δx and Δ⊗ lead to a contact of the peg with the surroundings and cause manipulation forces m, see Fig. 4.6.

For this task a computer based simulation module bad been implemented and it found its application in the generation of training data for the neural force controllers. This simulation module computes the manipulation forces during an insertion task by stepwise lowering the peg into the hole, whereby the manipulation forces are computed by taking account of the linear relationship between the changing positioning errors and the occurring manipulation forces, this simulation approach is called stiffness-control. Special attention is given to the finger forces *f,* that can directly be computed by multiplication of the manipulation force m with the grip matrix G and adding the holding forces *h.* which are adjusted by force control, thus

$$f = m.G + h$$

This simulation module not only allows the generation of training data for the neural controllers, but also can be used for the proof of their force control ability, as it bas been discussed in Sundermann (1993).

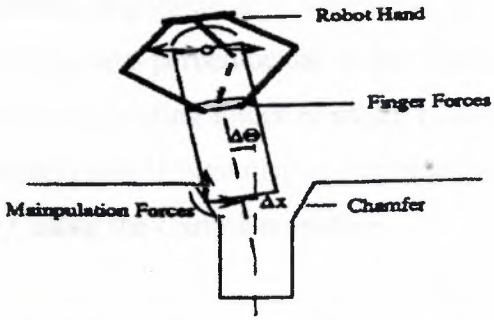

Figure 4.6. Peg-in-hole insertion task with positioning errors Δx and Δ⊗

The simulation proceeds as shown in Fig. 4.7.

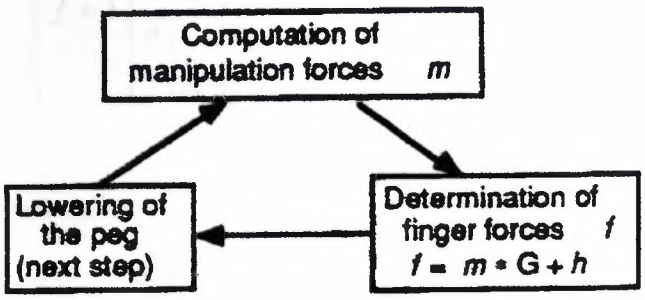

Figure 4.7. The simulation

4.1.5. Neural Control and Planning Algorithm

The aim for the first neural controller was to imitate a planning algorithm for grasp forces with an improvement in computation time for making it applicable for real-lime grasp force control.

The planning algorithm. For the planning algorithm it is suitable to describe the acting finger forces *fi* of the finger *i* in the following way:

With *fi,t* being the tangential $f_i = \begin{pmatrix} f_{i,t} \\ f_{i,n} \end{pmatrix}$ and *fi,n* the normal force component. The planning algorithm now determines for each finger force *fi* an additional holding force *hi,* which points perpendicular to the objects surface and optimises the finger forces *fi* stability *w.* As the holding forces *hi* mustn't cause a motion of the peg they have to be equal in each contact point If summing up the three 2-dimensional force vectors *fi* to a 6-dimensional vector *f,* using the Cartesian product

$$f1 \times f2 \times f3 \rightarrow f = \begin{pmatrix} f1 \\ f2 \\ f3 \end{pmatrix}$$

the problem of optimising the grip stability can be described as

$$\begin{bmatrix} R000 \\ 0R0 \\ 00R \end{bmatrix} \quad \left(f + \lambda . \begin{bmatrix} h \\ h \\ h \end{bmatrix} \right) \geq \begin{bmatrix} b \\ b \\ b \end{bmatrix} \qquad \text{(A)}$$

With

$$R = \begin{bmatrix} -1\mu \\ 01 \\ 0-1 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \\ -f_{n,\max} \end{bmatrix}, h = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

And μ the coefficient of friction. The system (A) bas to be optimised with regard to λ; that provides the optimal holding force. By normalizing each component of the finger forces to the maximum allowed normal force *fn,max* the stability *w* of a finger force *fi* with additional holding force $\lambda . h$ is given by

$$w(f_i + \lambda h) = w(\lambda) = Min\{(-(f_{i,n} + \lambda h_n)), ((f_{i,n} + \lambda h_n) - f_{i,t})\}$$

The grip stability

$$\Omega(\lambda) = \underset{i=1}{\overset{3}{Min}} \{f_i + \lambda h\}$$

must then be optimised with regard to A. what is performed by the algorithm of Hooke-Jeeves (Bronstein and Semendjajew,1988).

Transformation to a neural net. The transformation of this algorithm to a neural network has been achieved by applying this optimisation method for finger forces inside of the simulation module to force control. Training data has then been generated by simulating several insertion task, with storing the in- and outputs of force control on disc (as already has been shown in Fig.2) for a later off-line training of a .three layered back propagation neural network, with the neurons distribution 6:3:1. After presenting the training data 10.000 times the neural network imitated the original algorithm with error less than 5%. The Table 1 compares the control behaviour of the neural net to the original algorithm, by presenting the maximum initial positioning errors 4x and A8, which could be tolerated by

force control and by relating the computation time of both the algorithm and the neural net to one another.

Table 1 comparison neural and algorithmic control

Controller	Δ_x	Δ_o	Computation control
Algorithm	4.7mm	12.8°	>60s
Neural control	4.7mm	12.8°	13s

This proves, that it was possible to transform the optimisation algorithm to a neural net without a loss of accuracy in force correction, but with a significant reduction in computation time. Another advantage of the neural net is that its computation time remains constant for all inputs. while the computation time of the iterative optimisation algorithm varies for different inputs, therefore the neural controller is more reliable for force control.

4.1.6. Neural and fuzzy Control

The idea for another neural controller is similar to die one used in the last approach. At first a fuzzy algorithm for force control was developed which bas then been transformed to a neural network in order to reduce its computation time.

The Fuzzy Controller The fuzzy controller has been developed for optimising each finger force *fi* separately from the others. The input of the fuzzy controller is supplied by a transformation of the normal force component *fi,tto NF:=fi,n/fn,max and TF:=fi,t/fi,n.*The controller outputs $\Delta fi,n$, which optimises the corresponding finger force. Since for symmetrically positioned fingertips the force adjustment bas to be equal for all finger forces, they all are adjusted with die maximum of the $\Delta fi,n$. Thegeneralisation to not symmetrically positioned fingertips and the fuzzy rule base have been described in DOIsam *el* 01. (1994).

Imitation of the fuzzy controller. For the transformation of the fuzzy control algorithm to a neural net the set of training data has been generated by dividing the range [0,1] of each of the two inputs NF and TF into a discrete set of 20 equidistant points (0.05,0.1,...,1). The 400 different pairs (20x20) of inputs together with their corresponding

controller outputs have then been used as the set of training data. A back propagation neural network with the neurons distribution 2:2: I then has been trained on this set The comparison between the control behaviours of the fuzzy controller and the neural net is given in Table 2. This comparison again shows that the neural network control is as precise as the fuzzy control but reduces the computation time by a factor

Table 2

Controller	Δx	Δo	Computation control
Fuzzy control	4.6mm	12.8°	17s
Neural control	4.6mm	12.9°	8 s

4.1.7. Direct Neural force Control

The last two neural force controllers have been realised by first developing other control algorithms and then imitating them by neural networks. This had the disadvantage that the algorithms had to be developed and tested first with a great cost in time before transforming them to neural networks. The approach used now took use of neural networks capability for generalisation: The neural network has been trained to only a few but characteristic control aims and has been able to generalise these aims even for unknown situations.

The presented neural controller again optimises a finger force separately from the others. The force adaptation is then performed as it has been done for the fuzzy controller, with the maximum of all Δfn, Using the same abbreviations NF and TF as for the previous fuzzy controller the requested control aims are given by Table 3.

Table 3 Requested control aims

NF	TF	Requested Δfn
0.2	0.5	0.02
0.2	1	0.05

0.2	1.5	0.17
0.5	0.2	0.005
0.5	0.5	0.01
0.5	0.8	0.03
0.5	1	0.05
0.5	1.5	0.3
0.8	0.2	0.005
0.8	0.5	0.01
0.8	0.7	0.02
0.8	0.9	0.06
0.8	1	0.1

After training a neural net with the neurons distribution 2:2:1 the net was able to perform force control even in unknown situations. This neural controller has shown the same control quality as the one described in Section 4.3 (see Table 2).1bis proves that a neural net trained to only a few examples is able to perform force control as precise as planning algorithms or fuzzy control, but with an improvement in computation time.

4.2. Neural Networks for Robot Control

4.2.1 Learning Inverse Kinematics

One of the most important problem in the robotics is the identifications of nonlinearly systems. The most relevant system identification problems are the kinematics and dynamics of robotics. There are in direct and inverse versions for kinematics and dynamics. The kinematics problems are concerned with the mapping between joints coordinates of the robot and the resulting and effectors position and orientation in the physical space. Dynamics problem deal with the relationship between joint motor commands and the resulting response of the end effecter.

The knowledge of inverse kinematics, which relates the end-effector coordinates with the required coordinates in the robot joints, is essential in any robot control application. Most commercial robots provide explicit mathematical models of their inverse kinematics, which are implemented in their execution controllers. However, robots frequently undergo mechanical changes during operation causing miscalibrations with respect to the original kinematics model. It is reasonable to expect that a robot arm with a certain degree of autonomy to be capable of relearning or recalibrating its inverse kinematic relationship. Most of the neural approaches to inverse kinematics have involved supervised learning schemes, applied to solving the non-redundant problem. The two learning rules most widely used have been LMS and back propagation.

The general approach is to optimise a performance criterion, which includes manipulability and singularity avoidance terms, through the use of the Jacobian of the inverse transformation. A different, approach to inverse kinematics is to deal with the problem from a more behavioural point of view: relating images of the end-effector position and orientation to the required coordinates in the joints. Consequently, in this approach, it is not required to know the actual Cartesian coordinates of the end-effector, but, the goal is to relate sensory information directly to joint coordinates. This approach is, in fact, very appropriate for visuomotor control and is sometimes referred to as the hand-eye coordination problem.

Neurons are arranged in a 3D lattice to match the dimensionality of physical space.

The learning process makes this lattice converge to a discrete representation of the workspace. Each neuron *i* has an associated four-dimensional vector Wi representing the retinal coordinates of a point of the workspace. The response of the network to a given input u is the vector of joint angles *Ok* and the 3x4 Jacobian matrix *Ak* associated with the winning neuron *k*.

The joint angles produced for this particular input are then obtained with the expression:

$$\theta(u) = \theta_k + A_k(u - w_k)$$

A learning cycle consists of the following four steps:

1. First, the classical Kohonen rule is applied to the weights:

$$w_i^{new} = w_i^{old} + ch_k(i)(u(t) - w_k(t))$$

Where c is the learning rate and *hk(.)* is a Gaussian function cantered at *k* used to modulate the adaptation steps as a function of the distance to the winning neuron.

2. By applying θ(u) to the real robot, the end-effector moves to position u' in camera coordinates. The difference between the desired position u and the attained one u' constitutes an error signal that permits applying an error-correction rule, in this case the LMS rule:

$$\theta^* = \theta_k + \Delta\theta = \theta_k + A_k(u - u')$$

3. By applying the correction increment A_k (u-u') to the joints of the real robot, a refined position u" in camera coordinates is obtained. Now, the LMS rule can be applied to the Jacobian matrix by using $\Delta u = (u\text{-}u'')$ as the error signal

$$A^* = A_k + (\Delta\theta - A_k\Delta_u)\frac{\Delta_{u^T}}{\|\Delta u\|^2}$$

4. Finally, the Kohonen rule is applied to the joint angles:

$$\theta_i^{new} = \theta_i^{old} + c'h'k(i)(\theta_i^* - \theta_k(t))$$

and the Jacobian matrix:

$$A_i^{new} = A_i^{old} + c'h'k(i)(A_i^* - A_k(t))$$

Where again c' is the learning rate and *hk* (.) is a Gaussian function cantered at *k* used to modulate the adaptation steps as a function of the distance to the winning neuron.

The previously cited works showed this method to converge and to self-organize into a reasonable representation of the workspace in a limited number of iterations. With appropriate modifications, the algorithm described above has been used for the purpose of inverse kinematics recalibration in a simulated space robot. Firstly, a 3-degree-of-freedom case was solved and the system is currently being extended to 6 degrees of freedom with success.

4.2.2 Controlling Inverse Dynamics With Neural Networks

The inverse dynamics problem consists of the control of the dynamic trajectory of the arm end-effector, through the motor commands at its joints, so that it follows a desired reference path, with a high degree of accuracy and precision. In state-space-representation terms, the state variables of this problem are the position, velocity and acceleration of the end effector and its control variables are the motor commands (currents or voltages) at each joint. The efficient solution of the dynamic control problem through conventional control schemes would require a deep knowledge of the system behaviour, translated into a very accurate nonlinear mathematical model.

However, in the applications described in this paper, as in many others, this type of model is not available. Furthermore, the complexity of the model makes real-time implementations computationally intensive and the addition of payloads to the system may greatly affect the overall dynamic behaviour.

A feasible approach in conventional adaptive control is the determination of an approximate piecewise-linear model of the nonlinear plant, and the synthesis of appropriate linear controllers the different operating conditions corresponding to each one of the linear models. A common procedure in this approach is the use of indirect adaptive control, where an adaptive identifier of the plant updates the parameters of the plant model on-line and the parameters of the linear controller are then computed with the estimated, plant values. Although this approach has been successful in several applications, the stability conditions for convergence of the adaptive schemes are based on a relatively slow time variation of the

plant parameters with respect to the control adaptation times. The use of linear models to approximate nonlinear processes leads, in general, to models with time-varying parameters, where the parameter variations in time may be too fast to guarantee convergence of the adaptive control algorithms.

For these reasons, the neural identification and control is very relevant to the problem of the robot dynamics. On the one hand, the ability to learn a nonlinear behaviour through appropriate examples of inputs and outputs may overcome the modelling difficulties. On the other, it is expected that, the use of neural networks will produce more efficient nonlinear approximations of the plant model, which do not require such a fast adaptation of parameters as linear approximations, thus providing more adequate conditions for stability of the closed-loop systems. A general approach to adaptive control, proposed by Narendra (1990), consists of an indirect model reference adaptive scheme with a series-parallel structure.

A different type of neural network, CMAC (Cerebellar Model Articulation Controller), originally developed by Albus (1975), has also been used in similar neural adaptive control schemes.The CMAC scheme is shown to have considerably better model-tracking capabilities in this simplified problem, as well as a much faster convergence of the learning algorithm.

4.2.3 CMAC for Dynamic Control of a Robot Arm

The CMAC network, based on the cerebellar model for neuromuscular control, is basically a nonlinear table look-up technique which maps each N-dimensional input. state-space vector to a corresponding output vector of the same or different dimension. Each input vector activates exactly c input neurons with overlapping receptive fields, where c *is* a variable parameter representing the extent of generalization within the state space. The potentially very large virtual state space is mapped to a smaller physical weight table using a fixed random hashing function. A supervised training method, resembling the Widrow-Hoff rule, is used to adjust the CMAC memory values based on these observations. The learned information is used to predict the command signals required to produce desired changes in the sensor outputs.

Miller et al. have studied a neural-based learning control system for the dynamic control of robot manipulators with multiple feedback sensors and multiple command variables. The system has been studied for the control of two and five-jointed robot arms without vision, and a five-jointed robot arm with vision. In this scheme, a neural network is used, in place of an explicit system model, to adaptively learn an approximate dynamic model of the controlled robot in appropriate regions of the system state space. The CMAC neural control module adaptively learns the unknown nonlinear mapping between .the sensor outputs and the system command variables from on-line observations of each during system operation. In these experiments, the CMAC neural controller is implemented in a closed-loop control system, where it is used to predict the actuator torques required to make the robot follow a desired trajectory. These torques are a function of the current joint positions, velocities and accelerations, and are used as feed forward terms in parallel with a fixed-gain linear feedback controller. The control signal input to the robot is the sum of the terms from the CMAC module and the feedback controller. At first, the network memory values are initialised to zero, and the feedback controller provides initial movements upon which the CMAC controller may learn and improve. The network is trained by adapting the CMAC memory values after each control cycle, based on observations of the control inputs and obtained sensor outputs. The difference between the observed sensor output and that computed by the CMAC module is used to calculate the memory adjustment for the current control cycle using a form of the Widrow-Hoff LMS training rule. After several control iterations, the network outputs approach the correct values, the error input to the fixed-gain controller decreases to zero (and hence its output also), and the CMAC module effectively takes over full control of the system.

The concepts of CMAC have been applied to the control of the OHR in simulation. The simulator of the robot dynamics is a variable structure state-space model of the telescopic harvesting arm of the OHR, which takes into account variations of inertia moments and friction coefficients with the elongation. In order to generate acceptable training data, a PID controller is added to the robot model, since it would otherwise produce unstable responses. The implementation in this project differs from those of the aforementioned works, especially in the complexity of the dynamic behaviour of the robot (variable structure), which poses serious difficulties to the learning process. Firstly, a

discrete-time state-space representation of the dynamic process was preferred to the original input-output scheme and the output error signals were used as feedback signals. Therefore, the inputs to the CMAC network include information on the current state and next desired state, as well as tapped delays of both.

Additionally, the original control scheme *is* not readily applicable in this case, since the linear controller used with the OHR requires at least a derivative component. While in the original scheme the proportional controller could be made to operate in parallel with the CMAC controller in on-line learning, this is *not* the case when the linear controller contains derivative and integrator terms. Therefore, in the adopted CMAC controller scheme, shown in Fig. 1, the outputs of the linear and the CMAC controllers are not additive. Rather, the linear controller is initially used for a first phase of training the CMAC controller; during this phase, the CMAC module is not connected to the robot. When both controllers provide control signals, which are equal, to a certain degree of accuracy, the CMAC controller is switched on, and a period of on-line training starts, where the neural controller can improve its performance with respect to the PID controller results. The linear controller is a backup system, which takes over in case of malfunction of CMAC.

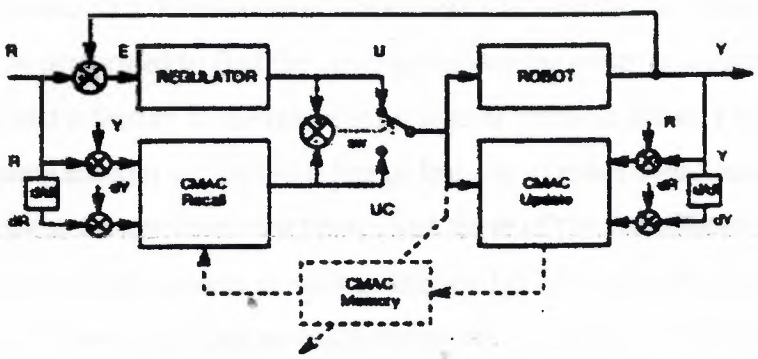

Fig. 4.13. Scheme CMAC control of robot dynamics.

The results of this implementation show that. CMAC can efficiently learn the inverse dynamics of this complex robot model and that it can improve the results of applying a classical PID controller. Moreover, it is important to take into account that the PID controller was specially designed for the mathematical model of the simulator, so that

the PID control results are the best case situation, while poorer results are to be expected in real conditions, where deviations from the model will undoubtedly appear. Conversely, the CMAC learning is not model dependent, so that the real robot conditions should not pose additional difficulties for control

4.2.4 Learning Visual Positioning

Many authors have tackled the problem of visual positioning of robot manipulators, for applications ranging from inspection, grasping and assembly of parts to docking and navigation of autonomous guided vehicles. Most of the existing works have tended to rely on simple geometrical features extracted from images, such as points, lines or circles, together with projection transformations to analytically derive the mapping from 2D image space to the robot Cartesian coordinates (Abidi, 1990; Mandel, 1987; Kabuka, 1987; Chaumett.e, 1991; Espiau, 1992). Although existing approaches based on analytical methods have produced useful results, they all depend on simple geometric features which are assumed to be always visible and extractable in the camera image. In addition, the physical relationship between all object features must be known in order to derive the projection transformations. Furthermore, since more than one cue is typically used, matching must be performed to find the correspondence between each feature in the observed image and a feature in the reference image or physical scene. Finally, in order to derive the complete transformation from image features to robot coordinates, the precise relationship between the cameras coordinates and those of the end effector must be known, requiring calibration of the camera as well as knowledge of its intrinsic parameters. Obviously, most of these assumptions place strong restrictions on the observed scene, and the operating conditions and robustness to noise of the resulting system.

The mapping between 2D image feature deviations and robot position or joint angles is a highly nonlinear relationship, which depends on the type and relative positions of the object features, the camera-robot relationship, the intrinsic camera parameters, and the robot kinematics. A neural network may be used to learn the entire transformation implicitly based on training examples, thus avoiding explicit computation of all intermediate transformations. A first work in this area was that of Hashimoto (1992) who used back

propagation networks to learn the mapping between the image deviations of 4 projected points of a viewed object with respect to a "reference" or desired image, and the corresponding joint angles of a robot with a camera mounted on its end effector. Hashimoto's work showed the capability of a neural network to perform visual positioning with a degree of accuracy similar to that obtained using analytical techniques. However, many possibilities by which neural networks can be exploited to produce more general-purpose visual positioning systems remain open for research. For example, training neural networks to map from more global image descriptors to robot commands could make visual servoing methods much less sensitive to the presence and extractability of a few simple geometric features in the observed scene. Similarly, the need for explicit feature matching may be avoided and robustness to obstructed features, noisy images, and changing reference positions may be increased.

The approach of this work was to develop more flexible and robust visual positioning methods based on neural networks. In all the experiments performed, the overall objective was to train a neural network to represent the mapping between the variations, with respect to a desired "reference" image, of several prespecified image features or descriptors as seen from a camera mounted on the robot end effector, and the 3D Cartesian position and orientation of the end effector relative to the reference pose. Unlike in earlier neural approaches (Hashimoto, 1992), inverse kinematics of the robot. were not. included in the mapping. In this way aspects related to the visual mapping problem could be studied independently from the robot kinematics and the additional benefit was achieved that the learned transformation is completely independent from the chosen reference position, and may therefore be used to correctly position the robot relative to the object regardless of their initial locations. The general scheme for image-based robot control is shown in Fig. 4.14

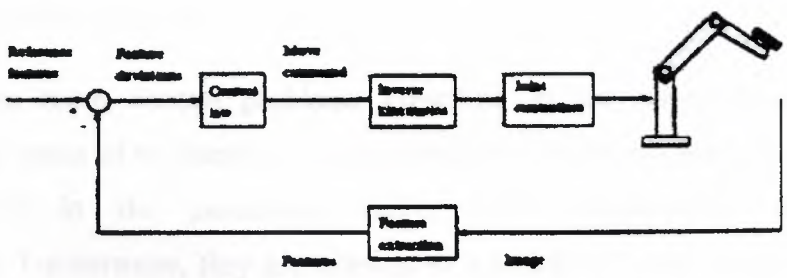

Fig. 4.14 General scheme for visual positioning

Two sets of experiments were performed, in which a neural network was used to learn the mapping between a chosen set of extracted image features and the 6D movements required to approximate the camera to a prespecified reference position (a more detailed description of this application is given in Venaille et. al., 1994). In initial experiments, four point features were used, thus allowing comparison of results with similar existing works based on analytic and neural methods. The differences between the x, y coordinates of the four points in the reference and observed images were used as inputs to the network. In the second set of experiments, the features used consisted of 32 Fourier descriptors used to encode the shape of the extracted silhouette of an observed object on a uniform background. As before, the differences between the descriptors in the reference and observed images comprised the network inputs. In both cases, training sets were constructed by moving the robot-mounted camera to 1000 random positions in the vicinity of the reference pose, and the extracted feature deviations for each image, along with the applied 6D movement, were used as training examples. Back propagation networks were trained using these data sets and tested in a closed-loop visual positioning system to test their ability to guide the camera back to the reference position from any given initial position and orientation within a limited range. In both sets of experiments, the neural visual positioning systems were capable of converging on the reference position to an accuracy of less than half a pixel in an average of 2 to 10 movements, depending on the range of initial displacements used. Even when the camera was displaced to three times the range of movements used to generate the training set, the networks generalized quite well and were able to achieve the same final positioning accuracy in just a few more approach movements. Thus, this work demonstrated the capability of using neural networks to accurately position a robot manipulator based on image information even when complex image features are used for which it is not possible to derive explicit transformation relationships or perform feature.

The three robot control problems presented in the paper have challenging characteristics in terms of no linearity, unstructured ness of the environments, uncertainty or time-variation in the parameters and/or speed requirements for real-time implementations. Furthermore, they are relevant to a variety of robot control applications and they are useful to demonstrate how specific properties of neural networks can be

efficiently exploited in robotics In choosing the learning solutions presented in the paper, the authors have sought to combine the criteria of efficiency in learning and amenability for real-time implementation, with an orientation towards the most behavioural models, i.e., avoiding, as much as possible, the need for explicit coding of workspace, objects or repetitive-task representations. From the point of view of demonstrating the adequacy of neural learning in robotics, the solutions to the issues of inverse kinematics, inverse dynamics and visual positioning provide an excellent tested for different concepts in learning. While in the inverse-kinematics recalibration problem a form of unsupervised learning approach is successfully applied, the problem of visual positioning is an example of the use of supervised learning, albeit without explicit matching of image features. The solution of the dynamic control problem with a neural adaptive control scheme is illustrative of the concepts of supervised on-line learning. A number of extensions of this research work are foreseen. On the subject of inverse kinematics, the issues of redundancy and singularities will be dealt with in more detail; additionally, the problems concerned with the volume of the arm itself and its possible collisions with obstacles in the workspace will be investigated. Furthermore, a thorough study on the stability conditions and the robustness of the dynamic control with CMAC must be carried out. Similarly, the research work on the selection of global descriptors and neural network structures to optimise the convergence of the visual positioning module will be continued. Finally, the problem of learning to handle objects with force-torque control involving measurement uncertainty is also being considered for a more detailed study, since it is a very relevant problem for industrial applications. In this case, reinforcement learning techniques will be investigated, as proposed by Gullapalli (1993), Gullapalli et al.(1994) and Torras (1994).

CHAPTER 5. TUNING OF ELEMENTARY SKILLS FOR INTELLIGENT ROBOTS

5.1. Skills in Robot Programming and Control

Considering the structure of a typical robot system it must be noted that real-world problems can usually not simply be classified as being completely located on the symbolic (planning) respectively the subsymbolic (control) layer. In fact, most complex tasks require some *reflexive* skills which are usually associated with a subsymbolic component, as well as *planning* or *reasoning* capabilities, typically to be realized by means of a symbolic module (Torras, 1992).

In the robotics community the concept of skills can be found throughout a number of different applications. Skills are usually employed in a NASREM-style hierarchical architecture (Albus *et* cl.t 1981)t where they integrate the *H* t *M* t and *G* modules up to the level of action primitives. In telerobotics, robots provide the capability to autonomously execute certain operations and relieve the operator from difficult control tasks. These individual capabilities are referred to as skills the concept itself is also known as shared autonomy (Brunner *et* cl.t 1992)t (Hasegawa *et* al.t 1992). In robot programming, Skills-Oriented Robot Programming (SKORP, see (Archibald and Petriu, 1993» relies on the existence of a set of skills as the building blocks of a robot program (Dufay and Latombe 1984) (Heise, 1989), (Kaiser and Kleuziger,1994). The approaches to skill learning are mainly aiming at identifying a control function for a given task. They can be found both in the robotics (Asada and Yang, 1989) (Deleon and West 1993) (Liu and Asada, 1993) and the machine learning community (Gallipolis, 1992)
(Lin, 1993). In general, all works share the same principal view on skills as the ability of the robot to safely change the world from a given state to a defined one in the presence of uncertainty t with the individual control functions applied using only initialisation data and direct sensorial information at runtime. Considering the remarks made above, two mainstreams regarding the application of skills become evident:

1. Skills as a mean to describe the robots capabilities such that this description can be used for programming or planning purposes and hides implementation details.

2. Skills as an operational representation of the individual control function that can directly be applied on the robot and meets the robots requirements.

In order to not mix up these heterogeneous aspects of skills, a slightly different point of view is adopted which allows to distinguish between the operational descriptions (the procedure being called to perform a certain operation) on the one hand and the abstract operator describing the effects of the skill execution on the other hand.

The operational description is referred to as the skill. It is defined as a tupelo (Rt, f-s *Is)* with $R \neq \theta$ being a minimal consistent set of situation-action rules (which might be represented by a numerical function) and *Is* (the termination condition) being a situation evaluation function *(f-s: Sit-* [0, 1]), where *Sit* is the set of situations described by the perception of the available sensors. *Ms* is the set of all skills.

The learnable[1] description, the skill model. is a finite automaton $A = (z_0^A, z^A, T^A, z_T^A, z_E^A)$, with Z_0^A being the start and Z_T^A being the target state, Z^A being the set of all states existing in A,t and t^A being the set of all transitions in A. Z_E^A is an additional error state. *MA* is the set of all skill models Z_T^A The skill model Z_0^A describes how certain skills are to be employed in order to arrive at state t^A if the current state is

The two representations are integrated by the definition of elementary operations as triples *(A, S, fs)* with $A = (Z_0^A, Z^A, Z_T^A, , Z_E^A)$ being a skill model. $S = \{S_1, ..., Sn\}$ being the set of corresponding skills and $f_s : Sit^2 \times M_A \rightarrow M_s, k \geq 1, f_s(s_c, s_g, A) = (S_{n1}, ..., S_{nk}), S_{ni} \varepsilon \{1, ..., k\}$. 1. Being a selection function,

For the actual execution of an elementary operation, the selection function f_s determines the k skills that should be used in order to arrive at the goal situation s_g from the current situation s_c.

To illustrate the meaning of the formal descriptions given above, consider the peg-into-hole operation, a typical task for a manipulation robot. Even if this task is a quite simple one, it already exhibits the usefulness of following a skill-based approach. As (figure 5.1) shows, four different phases of the insertion process can be identified, namely the detection of the hole, the horizontal and vertical alignment of the peg and the hole, and,

finally, the insertion process itself. The individual skills building the whole operation correspond to these phases.

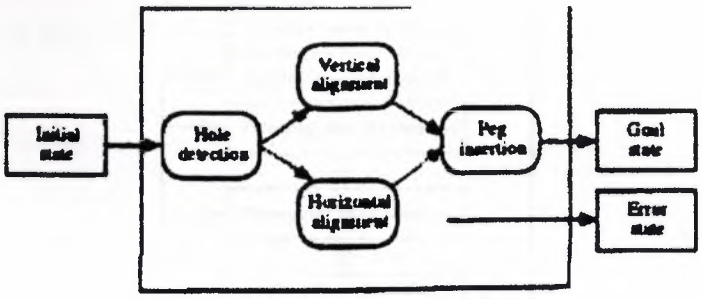

Figure 5.1. Skill model of the peg-into-hole task, comprising the skills of hole detection, alignment, and insertion.

5.2. Identification of Learning Tasks

The actual procedure to generate an operational description of a skill comprises several steps (Kaiser and Kreuziger, 1994). Some of these steps (see also figure 5.2) involve the acquisition of new or the modification of existing knowledge or its representation. Hence, to solve the learning tasks described below, it is necessary to use a representation that allows for an explicit access to and an alteration of the represented control knowledge, such as the orthogonal one described above.

The first learning task to be solved is to find the general structure of the operation that is represented by the given examples. As (figure 5.3) shows, the segmentation of a given example results in a sequence of phases. This sequence can be employed as an example to be used induce the structure of the task (see also (Dufay and Latombe, 1984)). Following the segmentation of the set of available examples, the control functions taking care of the individual segments, i.e., the skills controlling the several phases *of* the task are to be learned. This task comprises two different aspects. First, the control function itself, i.e., the mapping between sensorial input and the control output to the robot is to be learned. Second, means are necessary to detect the beginning and the end of a phase.

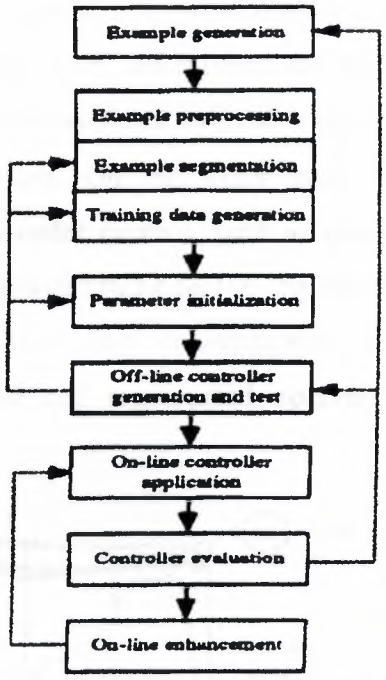

Figure 5.2. Different phases of the controller Design. Gray arrows indicate feedback loops.

Both tasks can be solved by means of function approximation techniques such as neural networks or fuzzy controllers.

Apart from knowledge about the structure of the task, the generation of a complete elementary operation requires a description of the set of situations that are valid for the application of this particular operation. Also, the goals that should be achieved by executing the operation must be described. These descriptions must exist in both an operational and a symbolic representation. The operational representation can be derived from the given examples by following the same approach used for detecting the beginning and the end of a phase. The symbolic representation, however, requires clustering and the definition of a concept describing the covered situations. Since the semantics of the performed operation and, consequently, of the situations cannot be derived from the numerical examples, is must be provided externally.

Additional knowledge that should be extracted from the given examples describes the evaluation criteria that are applied to the robot and the controller during operation. On the one hand, the sensorial input (e.g., the minima and the maximal of the values that were perceived while the example was recorded) represent boundaries for the sensorial input as it should occur during execution. On the other hand, the examples exhibit physical characteristics of the system under control, such as qualitative knowledge about the dependency of the change of sensorial input on the change of applied control signal. In order to perform efficient adaptation and enhancement of the controller on-line, these characteristics must be found and represented operationally (Schiffmann and Geffers, 1993).

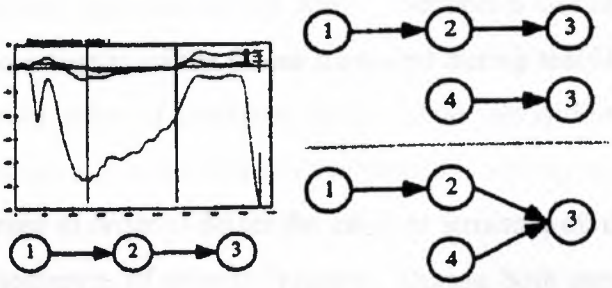

Figure 5.3. Mapping of an example to a sequence of states and induction of structure.

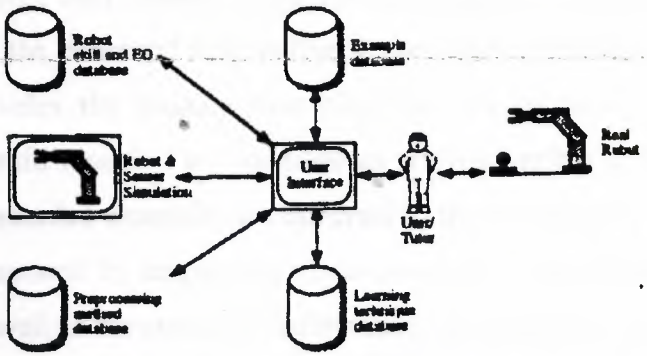

Figure 5.4. Components of the interactive programming environment.

5.3. The Interactive Programming Environment

Apart from the methodological aspects involved in the design of an operational description for an elementary operation, each of the design steps (see figure 5.2) must also be supported by the programming environment. This programming environment consists of several components (see figure 5.4) that are integrated by means of a graphical user interface providing a direct link to the real robot and, additionally, to a robot and sensor simulator.

During the example generation, the tutor/teacher must have access to the same sensorial information (and, if possible, to no other) as the robot will have during execution. With respect to compliant motion, this means that force reflection is desirable. At least, a fast, intuitively understandable display of the forces measured during teaching is necessary. The pre-processing and segmentation of examples should yield two different results. First, the examples must be normalized in order to allow comparisons among several examples. Second, they must be analysed in order to detect the inherent structure of the task that has been performed, yielding a sequence of several "phases". During both steps, the programming system must provide access to already given examples. It must also provide the tutor with a selection of methods that allow him or her to interactively support these steps.

Additionally, several examples are necessary to generate a sufficiently general set of training data for a that can be used by a learning technique (such as neural networks). This step requires to merge data sample during the same phase of a different trial to a single training file. Also, the structural information about these examples (i.e., the result of the segmentation) provides the training examples that are necessary to induce the task's structure, i.e., the skill model. Two components are responsible to provide the necessary support. First, all recorded examples are assigned to the skill and the elementary operation that have been generated by employing these examples. The example database contains these examples as well as the structural information that has been obtained during the segmentation step. Also, an informal description of the example given by the tutor is available. The pre-processing technique database provides a set of methods such as filters, statistical analysis, and segmentation algorithms that can be applied either automatically or triggered by the tutor.

For the actual generation of a controller (i.e., for a set of skills taking care of the individual "phases" of a task), several techniques such as neural networks and fuzzy controllers are available (Giordana *et 41.,* 1994). However, each of these techniques must be used in accordance with the characteristics of the given problem. The actual parameterisation of the employed technique must be supported by an analysis of the given training data yielding a set of constraints that the parameters must fulfil. Moreover, access to these parameters must take place intuitively, enabling the tutor to incorporate qualitative knowledge in the design process. The learning technique database comprises methods to generate skills and elementary operations from the pre-processed examples. The first application of the generated controller on the robot is also the final test with respect to the operationally of the controller. To minimize the risk in this phase, the programming system should provide a simulation that enables the tutor to test at least the minimum requirements that must be met for safety. The availability of a simulation is also especially useful if the robot itself cannot be used for training.

During application, the robot will finally generate more and better examples for the task that .it is performing. These examples describe the robot's characteristics much better than the initial, constrained, tutor-generated ones and should be used by the programming system to enhance its internal description of the task, and, if necessary, the tasks associated skill model. The skill and EO database contains already acquired skills and generated elementary operations in different representations. The descriptive representation of both is generated by the tutor and describes the purpose of the skill or the elementary operation in natural language. The operational representation of a skill consists of the actual network or fuzzy controller implementing the skill and/or the termination criterion. The operational representation of an elementary operation is given by the skill model plus the selection function, whereas an operator represents the elementary operation formally. Since the application of the controller on the real robot takes place under strong real-time constraints, the programming system itself is not be responsible for continuous monitoring of the controller. Instead, it evaluates the performance of the controller after the whole elementary operation has been executed and employers the same evaluation criteria as those used for on-line adaptation (see below). Depending on the result of the evaluation, a local adaptation of the controller, the incorporation of data from an additional experiment,

the change *of* the evaluation criteria, or a complete redesign *of* the controller are possible

The on-line enhancement of the controller comprises adaptation to a changing robot and environment as well as continuous improvement with respect to the given evaluation criteria. Both aims can only be achieved while the robot is operating, and the corresponding learning components are therefore not a part of the programming system but of the controller itself. However, the programming system must aid the tutor in selecting a proper technique to handle the online enhancement. Moreover, also this technique must be parameterised before it can be applied. Also, adaptation usually requires some additional knowledge about the task that cannot be obtained completely by the controller. Hence, the programming system must provide means to extract this knowledge (such as coarse rules describing the decency of the change *of* forces with respect to the change of velocity) from the existing examples and to incorporate this knowledge in the on-line learning components.

5.4. An Architecture Supporting Real-Time Control and Enhancement

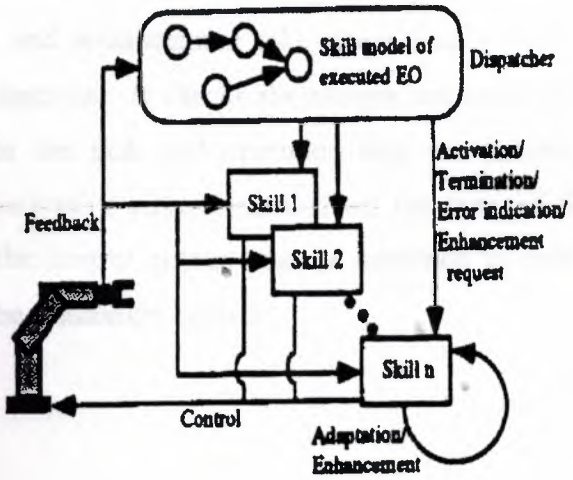

Figure 5.5. The control architecture

Table 1 The formal operator describing the peg-into-hole operation.

Preconditions	GRIPPED(ROBOT,PEG) &CONTACT(PEG ,PIECE) &NOT IN (PEG,HOLE)
Add list	IN (PEG,HOLE)
Delete list	NOT IN (PEG,HOLE)

Figure 5.5 shows the architecture as it has been developed according to the requirements defined so far. With respect to the peg-into-hole task, it is shown how the system works and how the twofold representation is employed. For purposes of understand ability, the formal operator description has been given in STRIPS-like syntax (see table 1). First, the planning system selects the peg into-hole operator as the next action to be executed. The invocation of the operator activates the dispatcher that uses the skill model to select the skill to be activated. I.e., the dispatcher will first activate the hole detection skill. After the termination function associated to an active skill has fired, it will select the vertical alignment and horizontal alignment skills for parallel activation. After both of them have terminated successfully, the peg insertion skill will be applied, which's termination also means the termination of the whole operation.

Adaptation and enhancement take place locally with respect to the skills and on request of the dispatcher. It checks the current sensorial input against the limits given by the examples for the skill and operation that are currently active. If these limits are exceeded, the qualitative rules generated on the base of the examples are employed to determine how the control output must be modified in order to improve the performance with respect to the evaluation criteria.

CONCLUSION

The control of complicated technological processes by using traditional control algorithm is not enough satisfay such characteristics of control systems, as accuray, vitality , adaptivity. In these conditions one of actual way of constructing control system is the use artificial intelligence ideas.

In the proje the computer – aided control of robot manipulators by using artificial intelligence elements are considered. The kinematics dynamics of robots are investigated. To the control of robot dynamics the neural–network technology is used. The structure and control algorithm of robot hand and robot dynamics are given. Neural identification and control problem of robots are described. As an example. The program is written in assembler. The obtained results satisfy the efficiency of application neural network technology in control of robot by using computers.

References:

[1] Industrial Robotics Technology, Programming and Applications
Mikell P. Groover, Mitchell Weiss , Roger N. Nagel , Nicholas G. Odrey

[2] Artifical Intelligence In Real – Time Control 1994
S.Fatikow, K.Sundermann
Neural – Based Learning In Grasp Force Control Of a Robot Hand.

[3] Artifical Intelligence In Real – Time Control 1994
M.Kasier, A.Giordana, M.Nuttin
Integrated Acquisition, Execution, Evaluation and Tuning of Elementary Skills for Intelligent Robots

[4] Artifical Intelligence In Real – Time Control 1994
G.Cembrano , C.Torras , G. Wells
Neural Networks for Robot Control

[5] Artifical Intelligence In Real – Time Control 1994
Th. Laengle , T.C. Lueth
Decentralized Control of Distributed Intelligent Robot and Subsystems

[6] Integrated Acquisitoin , Execution , Evaluation and Tuning of Elementary Skills for Intelligent Robots
M.Kaiser , A.Giordana , M.Nuttin

[7] Training Neurofuzzy Systems
D.J. Mills , M.Brown , C.J. Harris

[8] Artifical Intelligence In Real – Time Control 1994
Increasing a Knowledge Representation for FMS Control with Fault Detection and Error Recovery Capabilities

[9] Artificial Intelligence and Mobile Robots
David Kortenkamp , R. Peter Bonasso , Robin Murphy

[10] Telecommunications
Fahretten Sadigoglu

[11] Communication Systems Engineering
John G. Proakis , Masoud Salehi
Prentice Hall

[12] www.ieee.com

[13] www.britanica.com

[14] http://www.lvr.com/parport.htm

[15] http://dynamik.fb10.tu-berlin.de/manuals/hardware/pp-AMP-36.html

[16] http://www.stokely.com/unix.serial.port.resources/A-B-Ycablepinout.html

[17] http://www.doc.ic.ac.uk/~ih/doc/par/

[18] http://margo.student.utwente.nl/stefan/hwb/menu_Cable.html

[19] http://www.hut.fi/Misc/Electronics/circuits/power_from_pc.html

[20] http://www.lammertbies.nl/comm/cable/parallel.html

[21] http://cyclone.parad.ru/hwb/ca_ParallelPortLoopbackCheckIt.html

[22] http://www.kron.com