17

Soft Computing Based Mobile Manipulator Controller Design

Abdessemed Foudil and Benmahammed Khier

1. Introduction

During the last decades, numerous papers have been written on how to apply neuronal networks, fuzzy (multi-valued) logic, genetic algorithms and related ideas of learning from data and embedding structured human knowledge. These concepts and associated algorithms form the field of soft computing. They have been recognized as attractive alternatives to the standard, well established hard computing (conventional) paradigms. Traditional hard computing methods are often too cumbersome for today's problems. They always require a precisely stated analytical model and often a lot of computation time. Soft computing techniques which emphasize gains in understanding system behaviour in exchange for unnecessary accuracy have proved to be important practical tools for many real world problems. Because they are universal approximators of any multivariate function, the neuronal networks and fuzzy logic are of particular interest for modelling highly nonlinear, unknown or partial known complex systems. Due to their strong learning and cognitive ability and good tolerance to uncertainties and imprecision, soft computing techniques have found wide applications in robotic systems control. According to Zadeh (Zadeh, 1994), the basic premises of soft computing are

- The real world is pervasively imprecise and uncertain.
- Precision and certainty carry a cost.

And the guiding principle of soft computing, which follows from these premises, is exploit tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, and low solution costs.
Both the premises and the guiding principle differ strongly from those in classical hard computing, which require precision, certainty, and rigor. However, since precision and certainty carry a cost, the soft computing approach to computation, reasoning, and decision making should exploit the tolerance for imprecision (inherent in human reasoning) when necessary. A long standing tradition in science gives more respect to theories that are quantitative, formal, and precise than those that are qualitative, informal, and approximate. Many

Source: Industrial-Robotics-Theory-Modelling-Control, ISBN 3-86611-285-8, pp. 964, ARS/plV, Germany, December 2006, Edited by: Sam Cubero

contemporary problems do not lend themselves to precise solutions such as the mobile robot coordination.

Usually, learning implies acquiring knowledge about a previously unknown or partially known system. Learning from experimental data (statistical learning) and fuzzy logic are the most important constituents of soft computing. In nowadays, the soft computing is considered as a discipline that includes an emerging and more or less established family of problem stating and solving methods that attempt to imitate the intelligence found in nature. Very often, the devices and algorithms that can learn from data are characterized as intelligent. With the increasing complexity of industrial processes, the link among ambiguity, robustness and performance of these systems has become increasingly evident. This may explain the dominant role of emerging intelligent systems. The human mental abilities of learning, generalizing, memorizing and predicting should be the foundations of any intelligent system. The intelligent system is supposed to possess human like expertise within specific domain, adapts itself and learns to do better in changing environments and explains how it makes decisions and takes actions. It should be capable to deal with the large amount of data coming from different sensors, to plan under large uncertainties, to set the hierarchy of priorities, and to coordinate many different tasks simultaneously.

The behaviour coordination architectures can be divided into two categories: arbitration and command fusion schemes. In arbitration, the selected dominant behaviour controls the robot until the next decision cycle, whereas the motor commands of the suppressed behaviours are completely ignored. The command fusion approaches aggregate the control actions of multiple concurrently active behaviors into a consensual decision. Fuzzy rule based hierarchical architectures offer an alternative approach to robotic behaviour coordination. A set of primitive, self contained behaviours is encoded by fuzzy rule bases that map perceptions to motor commands. Reactive behaviours in isolation are incapable of performing autonomous navigation in complex environments. However, more complex tasks can be accomplished through combination and cooperation among primitive behaviours. A composite behaviour is implemented as a supervisory controller that activates and deactivates the underlying primitive behaviours according to the current robot context and goals. A fuzzy coordination offers the advantage that behaviours are active to a certain degree, rather than being either switched on or off. The weight with which a behavior contributes to the overall decision depends on its current applicability and desirability.

The goal of this chapter is to present the main role of the soft computing and the contribution it can bring in the control of the complicated systems such as robotic systems. By this, we meant only a brief overview of the subject.

2. Problems and Principle of Robot Control

Industrial Robotics includes mechanical systems that are highly non-linear, ill defined and subject to a variety of unknown disturbances. The control of such systems is facing challenges in order to meet the requirements that can be of different natures. A lot of effort has been devoted to capitalizing on the advances in mathematical control theory resulting in several techniques appeared to tackle this kind of mechanical systems. The navigational planning for mobile robot is a search problem, where the robot has to plan a path from a given initial position to goal position. The robot must move without
hitting an obstacle in its environment. So, the obstacles in the robot workspace act as constraints to the navigational planning problem. A genetic algorithm can solve the problem, by choosing an appropriate fitness function that takes into account the distance of the planned path segments from the obstacles, the length of the planned path and the linearity of the path as practicable. Furthermore, the learning process is constrained by the three mutually compromising constraints complexity of the task, number of training examples and prior knowledge. Optimisation of one or two of these objectives often results in a sacrifice of the third. Learning a complex behaviour in an unstructured environment without prior knowledge requires a long exploration and training phase and therefore creates a serious problem to robotic applications. Today's robots are faced with imprecise, uncertain, and randomly changing environments. The desire to deal with these environments leads to the basic premises and the guiding principles of soft computing.
Robot control is predominately motion control using classical servomechanism control theory. Due to the nonlinearity of the manipulator motion, a wide variety of control schemes have been derived. Classical schemes include computed torque, resolved motion, PID decoupled model control, reference adaptive and resolved motion adaptive control (Whitney, 1969), (Begczy, 1974), (Dubowsky & DesForges, 1979). These schemes can be very complicated and require intensive computer resources. For instance, the computer torque technique uses the Lagrange–Euler or Newton–Euler equations of motion of the manipulator to determine the required torque to servo each joint in real time to track the desired trajectory as closely as possible. However, since there are always uncertainties in the robot dynamic model, the ideal error response cannot be achieved and the performance could be well degraded. This problem led people to using adaptive control approaches to solve these problems and relatively good results were obtained (Craig et al, 1987), (Spong & Ortega, 1988). The problem is complicated if we think to enlarge the workspace of the manipulator by mounting over it a mobile platform, resulting on a new system called a mobile manipulator. Researches to investigate the capabilities of mobile platforms with onboard manipulators are devoting considerable effort to come up with solutions to this complicated system (Yamamoto & Yun, 1994).

Now, since the first control a plication of Mamdani (Mamdani & Assilian 1974) and his team, a lot of efforts have been devoted to capitalizing on the advances of fuzzy logic theory. Many fuzzy control approaches appeared. In fact, fuzzy logic provides tools that are of potential interest to control systems. Fuzzy controllers are a convenient choice when an analytical model of the system to be controlled cannot be obtained. They have shown a good degree of robustness in face of large variability and uncertainty in the parameters, and they lend themselves to efficient implementations, including hardware solutions. These characteristics fit well the needs to precision motion control of mobile manipulators. However, the main difficulty in designing a fuzzy logic controller is the efficient formul tion of the fuzzy **If-Then** rules. It is well known that it is easy to produce the antecedent parts of a fuzzy control rules, but it is very difficult to produce the consequent parts without expert knowledge. The derivation of such rules is often based on the e perience of skilled operators, or using heuristic thinking (Zadeh, L.A.1973), (Mamdani, E.H. 1974). In recent years and due to the availability of powerful computer platform, the theory of evolutionary algorithms starts to become popular to the problem of parameter optimization. Genetic algorithm as one approach to the implementation of evolutionary algorithms was used by Karr, (Karr, C.L. (1991) to generating the rules of the cart-pole balancing fuzzy logic controller. In this work, we investigate the problem of the motion control of a mobile manipulator using fuzzy control schemes. The m chanical system is split into two subsystems where the mobile platform and the manipulator constitute the parts. Appropriate fuzzy controllers are used to control each of these two subsystems. A genetic algorithm generates the rules of the fuzzy controllers letting the system turning around an optimal solution. The motion of the platform and that of the manipulator are coordinated by a Neural like network, wich is a sort of adaptive graph of operations, designed from the kinematics model of the system. A learning paradigm is used to produce the required reference variables for each of the mobile platform and the robot manipulator for an overall coordinate behaviour.

3. Robot Model

3.1 A mobile manipulator overview architecture

A mobile manipulator system is a robotic manipulator mounted on mobile platform.This combination allows manipulation tasks over unlimited working space. However, since the platform and the manipulator have independent movement, a particular point in the workspace may be reached in multiple configurations, resulting in a system with redundancy (Lee, J. K., & Cho, H. S.

1997). This can be helpful when it is desirable to perform tasks in a cluttered environment, or to optimally configure the system (Brock, O., Khatib, O. & Viji, S. 2002). Our o jective in this work, is to devise a controller for each of the mobile base and the manipulator separately, then we implement a sort of adaptive graph of operations to generate trajectory in the joint space. The network provides reference output values of the desired motion to the mobile manipulator system. The mechanical system is made up of the non-holonomic platform upon which is mounted a robot manipulator with 3 rotational degrees of freedom as it is shown in Figure 1. The accomplishment of the task is the result of the perm nent movement of the two structures for which the success is based on the satisfaction of the tracking error. If we consider Figure 1 where the four principal coordinate frames are shown: World frame O_W, platform frame O_P, manipulator base frame O_B, and the end effector frame O_E. Then, the manipulator's end effector position/orientation with respect to O_W is given by:

$$T_E^W = T_P^W T_B^P T_E^B$$

Such that the matrix T_P^W is determined by a certain *A(q)* matrix, T_B^W is a fixed matrix and T_E^B is determined by the joint variable vector $\theta = \left[\theta_1, \theta_2, ..., \theta_{n_m}\right]^T$, n_m represents the degree of freedom of the arm manipulator.

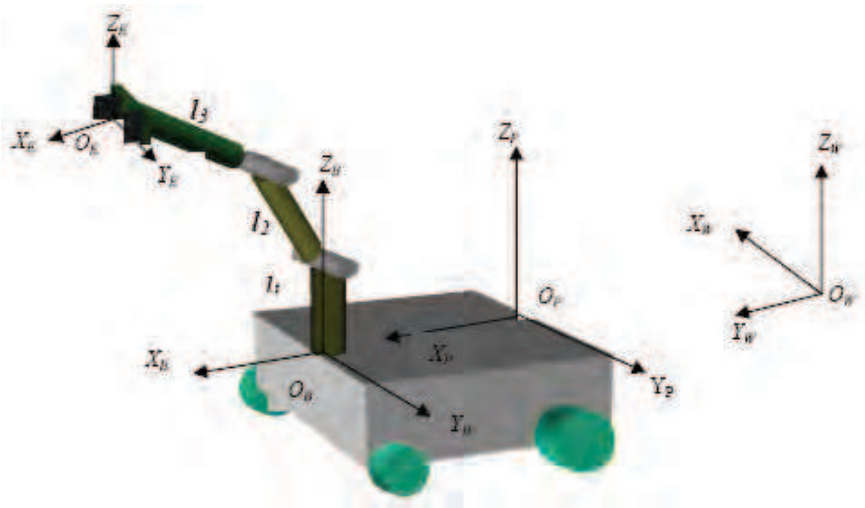


Figure 1. Mobile manipulator configuration

The vector of position of the end effector x_E^W is a non-linear function of the configuration vector , $q = \left[p^T , \theta^T \right]^T \in \Re^n , (n = 3 + n_m)$. The joint coordinates of the manipulator are $\theta = [\theta_1 , \theta_2 , \theta_3]^T$ (thus $(n_m = 3)$. Therefore the generalized coordinates of the mechanical system are:

$$q = (q_1 , q_2 , ..., q_6)^T = (x_B , y_B , z_B , \theta_1 , \theta_2 , \theta_3)^T$$

Hence, the generalized space dimension of the mechanical system is equal to λ=*6*. Now, for a given mechanical configuration system ***q***, its structure imposes to its end effector η position and orientation constraints. In our case, only the end effector position is considered. Therefore, the number of constraints is r duced to σ=3. On the other hand, we can observe that the system is non holonomic, and taking into account the constraint of the non-holonomy of the mobile platform, we can deduce the order of redundancy, which is equal to *(*λ σ*-1) = 2*. This redundancy helps increasing the manipulator dexterity, prevents the arm from singular configurations, and the let the system away from obstacles while completing a given task. On the other hand, the control of such mechanisms becomes much harder.

If we refer to Figure 1 and following the *D-H* parameterization, the outputs of the neural like network are given by equations (1), which designates the Cartesian coordinates of the task variable *E*, with respect to the world frame {***W***}. In a closed form this can be written as $X_E(t) = F(q(t));$ where ***F*** represents the direct kinematic mapping from the joint space to the task space and $X_E(t) = (x_E^W , x_E^W , x_E^W)^T$.

$$\begin{aligned} x_E^W &= x_B^W + \cos(\theta).\ [l_2 \cos(\theta_2) + l_3 \cos(\theta_2 + \theta_3)] \\ y_E^W &= y_B^W + \sin(\theta)\ .[l_2 \cos(\theta_2) + l_3 \cos(\theta_2 + \theta_3)] \\ z_E^W &= z_B^W + l_1 - l_2 \sin(\theta_2) - l_3 \sin(\theta_2 + \theta_3) \end{aligned} \quad (1)$$

Such that,

$$\theta = \theta_1 + \varphi \quad (2)$$

Where φ is the heading angle of the mobile platform, and l_1, l_2 and l_3 are the lengths of the three links composing the manipulator arm. x_B^W , y_B^W ,z_B^W , are the coordinates of the point ***B*** located in the front of the mobile platform with respect to the world frame $\{W\}$. In the sequel, we consider z_B^W equals zero for

simplicity. The goal is to find the generalized trajectory $\boldsymbol{q}(t)$ for a given task space trajectory $X_E(t)$ such that $\boldsymbol{F(q(t))} = \boldsymbol{X_E(t)}$ is satisfied. Since the system is redundant, the number of solutions is expected to be infinite. To realize a generalized task of the mechanical system, one has to derive the set of the λ generalized coordinates. In this context, an approach is suggested to investigate and solve this problem when we make a complete motion of the end effector resulting from a combined operation of the two subsystems that work in a coordinate manner.

3.2 The dynamic model of the manipulator

Two main approaches are used by most researchers to systematically derive the dynamic model of a manipulator, the Lagrange-Euler and the Newton-Euler formulations. The dynamic equations of motion are highly nonlinear and consist of inertia loading, coupling reaction forces between joints and gravity loading effects. For an *n*-link rigid manipulator the vector dynamic equation is given by:

$$\tau = M(\theta)\ddot{\theta} + B(\theta,\dot{\theta})\dot{\theta} + F(\theta,\dot{\theta}) + G(\theta) \tag{3}$$

where $\theta \in R^n$ is a vector of joint displacements, $\tau \in R^n$ is a vector of applied joint torques, $M(\theta): R^n \to R^{n\times n}$ is a symmetric positive definite manipulator inertia matrix, $B(\theta,\dot{\theta}): R^n \times R^n \times R^n \to R^n$ is a vector of centrifugal and Coriolis terms, $F(\theta,\dot{\theta}): R^n \times R^n \to R^n$ is a vector of frictional torques, and $G(\theta): R^n \to R^n$ is a vector of gravitational torques. The control of the robot manipulator is especially challenging due to the generic high nonlinearity existing in its dynamic model. Although the equations of motion (3) are complex and nonlinear for all but simple robots, they have several fundamental properties, which can be exploited to facilitate control system design.

1. **Property 1**. The inertia matrix $M(\theta)$ is symmetric, positive definite and both $M(\theta)$ and $M^{-1}(\theta)$ are uniformly bounded as a function θ of R^n.
2. **Property 2**. There is an independent control input for each degree of freedom
3. **Property 3**. The Lagrange-Euler equations for the robot are linear in the parameters.

Most feedback control laws, where a PD or PID controllers are used, are based on simplified dynamic equations. However the approach works well only at slow speeds of movement. At high speeds of movement the Coriolis and cen-

trifugal forces are major components of the dynamic equations and consequently the error can not corrected. A lot of effort has been devoted to capitalizing on the advances in mathematical control theory resulting in several techniques appeared to tackle this kind of mechanical systems. May be the most famous and which is considered, as the basic approach becoming very popular is the model based computed torque method. However, since there are always uncertainties in the robot dynamic model and the disturbances possibly arising from the actual running of the actuator or some other causes, the ideal error response cannot be achieved and the performance could be well degraded. Now, since the reliability of the PID controller has been field proven besides the application of fuzzy logic theory to process control, we propose in the next section a combination of the two to make a robust controller for robot manipulators.

3.3 The Kinematic model of the mobile platform

In this section, a kinematic description of a mobile robot is given. The vehicle has two driving wheels at the rear corners and two passive supporting wheels at the front corners. Two DC motors independently drive the two rear wheels. The vehicle presents however two constraints: It is non-holonomic, which means that it must move in the direction of the axis of symmetry, i.e.

$$\dot{y}_A - \dot{x}_A \tan\phi = 0 \tag{4}$$

ϕ is the heading angle of the vehicle from the *X*-axis of the world coordinates as it is depicted in Figure 2.

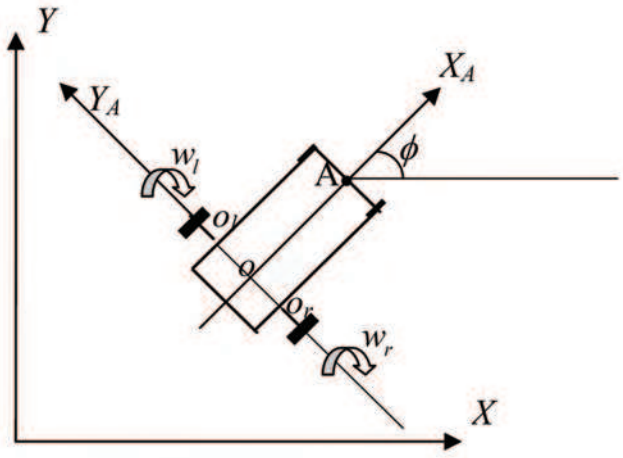


Figure 2. Mobile robot schematic

Writing the classical relationships between the velocity point O, and those of points O_l and O_r, we can easily determine the linear velocity $\vec{v}_o$, and the instantaneous angular velocity $\vec{\omega}$ of the mobile robot:

$$\vec{v}_o = \vec{v}_{or} + o\vec{o}_r \wedge \vec{\omega} \tag{5}$$

$$\vec{v}_o = \vec{v}_{ol} + o\vec{o}_l \wedge \vec{\omega} \tag{6}$$

$$\vec{\omega} = \dot{\phi}.\hat{k} \tag{7}$$

$\hat{k}$ is the unit vector along the Z_A axis; and $\vec{v}_{ol}$ and $\vec{v}_{or}$ are the linear velocities of the mobile robot points O_l and O_r respectively. When projecting expressions (5) and (6) on the X-axis and the Z-axis, we get the expressions of v_o and $\dot{\phi}$ as follows:

$$v_o = \frac{r}{2}(\omega_r + \omega_l) \tag{8}$$

$$\dot{\phi} = \frac{r}{2R}(\omega_r - \omega_l) \tag{9}$$

r and R are respectively the radius of the wheels and the width of the vehicle as it is shown in Figure 3. It has been proven by Samsung and Abderrahim (Samsung, C. & Abderrahim, K.A. 1990), that the vehicle converges better to its reference when controlling a point located in front of the rear wheel axis.

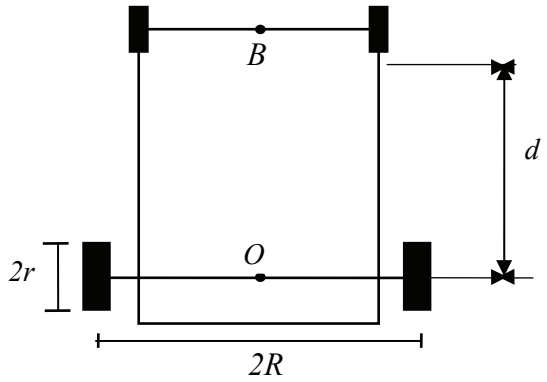


Figure 3. Geometric characteristic of the mobile robot

In this paper point B, as it is obvious from Figure 3, which is located at a distance ***d*** from point ***O*****,** has been chosen to be the position control of the vehicle such that:

$$x_B = x_o + d.\cos\phi \tag{10}$$

$$y_B = y_o + d.\sin\phi \tag{11}$$

where:

$$x_o(t+1) = x_o(t) + \Delta D.\cos(\phi + \frac{\Delta\phi}{2}) \tag{12}$$

$$y_o(t+1) = y_o(t) + \Delta D.\sin(\phi + \frac{\Delta\phi}{2}) \tag{13}$$

Such that:

$$\Delta D = \frac{r}{2}(\Delta q_r + \Delta q_l) \tag{14}$$

$$\Delta\phi = \frac{r}{2R}(\Delta q_r - \Delta q_l) \tag{15}$$

Where, (x_o, y_o) and (x_B, y_B) denote the coordinates of points ***O*** and ***A*** respectively, whereas Δq_r and Δq_l are the angular steps of the right and left wheels respectively.

4. Robot Control

The control strategy combines the mobile base behaviour and the manipulator behaviour to produce an integrated system that performs a coordinated motion and manipulation. We propose in this section the two layer robot controller and the genetic algorithm to determine the solution that gives the optimum rule base for a precompensator which is associated with the PID controller (Abdessemed, F. & Benmahammed, K. 2001).

4.1 The two layer robot controller design

Control inputs to the joints are composed of both feedback PID control and precompensator sub-systems components, (Fig. 4). The output of the precom-

pensator is considered as a new reference input to the PID-plant system. The introduction of the precompensator is justified by the fact that when the system evolves toward an abnormal mode, it is necessary to anticipate this evolution rather than to wait to arrive to this mode in order to avoid its consequences especially if it is dangerous. The dynamics of the precompensator-PID controller is explained as follows: The two inputs to the PID controller are $e_i^{'}(k)$ and $\dot{e}(k)$;

Where:

$$e_i^{'}(k) = \theta_i^c(k) - \theta_i(k) \tag{16}$$

$$\dot{e}_i(k) = \dot{\theta}_i^d(k) - \dot{\theta}_i(k); \quad \text{i=1,2 ,3 refer to the ith link.} \tag{17}$$

Note that the desired angular position is not directly compared to the measured one, but passes first through the precompensator to be transformed to a new reference angular value for the PID-plant system. Thus, one writes:

where:

$$\theta_i^c(k) = \theta_i^d(k) + m_i(k); \quad i=1,2 \text{ and } 3 \tag{18}$$

$$m_i = F_i\,(e_i\,,\, \Delta e_i); \quad i=1,2 \text{ and } 3. \tag{19}$$

$e(k)$ and $\Delta e(k)$ are inputs to the map F, and $m_i(k)$ is the output of the i-th joint; such that:

$$e_i(k) = \theta_i^d(k) - \theta_i(k); \quad i=1,\ 2 \text{ and } 3 \tag{20}$$

$$\Delta e_i(k) = e_i(k) - e_i(k-1); \quad i=1,2 \text{ and } 3 \tag{21}$$

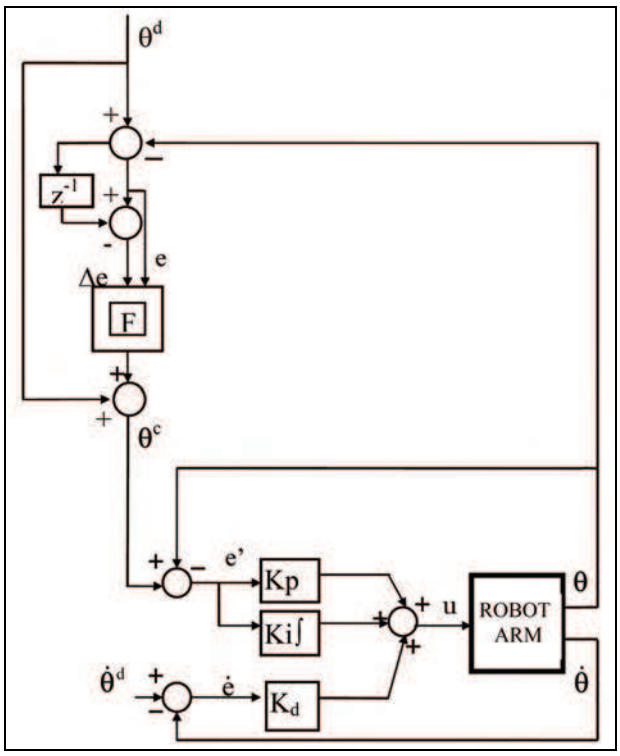


Figure 4. Diagram simulation of one-link robot control

4.1.1 The PID controller parameters determination

As a first attempt to regulate the robot manipulator, we consider the proportional-integral-derivative (PID) control law given by

$$\tau = K_P e(t) + K_D \dot{e}(t) + K_I \int e(t)dt \tag{22}$$

Where $e(t) = \theta^d - \theta$ and θ^d and θ are the desired reference and the measured trajectories respectively. The required controller parameters are found based on simplified dynamic equations; i.e, the manipulator is supposed evolving at slow speeds of movement. Consequently, the Coriolis and centrifugal forces are of the dynamic equations are neglected, thus

$$\tau = M(\theta)\ddot{\theta} \tag{23}$$

Considering equations (22) and (23), the following transfer function is obtained:

$$\frac{\theta_i}{\theta_i^d} = \frac{k_{Di}s^2 + k_{Pi}s + k_{Ii}}{J_i s^3 + k_{Di}s^2 + k_{Pi}s + k_{Ii}} \tag{24}$$

Thus, the characteristic equation is written as:

$$\Delta(s) = s^3 + \frac{k_{Di}}{J_i} s^2 + \frac{k_{P\,i}}{J_i} s + \frac{K_{Ii}}{J_i} = 0 \tag{25}$$

J_i is taken to be the fixed term of the J_{ii} element of the inertial matrix J. In robotic control, the gains are typically chosen so that the two poles are double and real negative. This gives a damping coefficient of one and by consequence a fast response without oscillations. The remaining pole is then placed on the real axis far away from the first two. Thus:

$$\Delta(s)=(s+\beta)2(s+n\beta). \tag{26}$$

The desired gains are given by the following relationships:

$$\begin{aligned} k_{Pi} &= m_i(2n+1)\beta^2, \\ k_{Di} &= m_i(2+n)\beta, \\ k_{Ii} &= m_i n\beta^3 \end{aligned} \tag{27}$$

with: $\beta>0$, $n>1$. The choice of β depends on the sampling frequency as well as to possible saturation effects.

4.1.2 The precompensator design

The precompensator is a fuzzy controller. Its main goal is to enter into action whenever it is needed to reinforce the conventional controller to provide the necessary commands that allow the end effector to track the desired trajectory with minimum error. A fuzzy controller is a system, which use a rule-based expert in the form of **If Then** statements. The input state with respect to a certain universe of discourse constitutes the premise of the rule, whereas the output state constitutes the consequence of the rule. We can distinguish among the steps of the rule treatment, the following procedures: *Fuzzification*, *Fuzzy inference* and *Defuzzification.* The main difficulty in designing a fuzzy logic controller is the efficient formulation of the fuzzy **If-Then** rules. It is well known that it is easy to produce the antecedent parts of a fuzzy control rules, but it is very difficult to produce the consequent parts without expert knowledge. In this work, the fuzzy rule base of the precompensator designed is found by using a genetic algorithm that search for the solution that gives the optimum rule base for the precompensator. If we assume that the error and its derivative are partitioned into K and L subsets then, the the number of possible combinations is LxK, which represent the number of rules per output. A single rule is defined as:

Rule R_{ij} *:* *if e is* A_i^e *and* Δe *is* $A_j^{\Delta e}$ *then* m_{ij} *is* A_p^m *i=1, 2, ... 7; j=1, 2, ..., 7; p=1, 2, ..., 7.*

R_{ij} is the label of he fuzzy if-then rule, A_i^e and $A_j^{\Delta e}$ are fuzzy subsets on the interval *[-1, 1]*, and A_p^m is the consequent fuzzy subset. The aim is to sort out the appropriate optimised rules adequate for the system by employing an evolving genetic algorithm. The system being evolved is encoded into a long valued string called a chromosome. Initially a random population of these strings is generated. Each string is then evaluated according to a given performance criterion and assigned a fitness score. The strings with the best score are used in the reproduction phase to give the next generation. Here, the genetic algorithm is presented as a seven-tuple entity and is abstracted in the following encapsulated form:

$$GA=\{M(t),\ l,\ D,\ \Phi_F,\ sel,\ pcross,\ pmut\} \tag{28}$$

where:

- $M(t) = \{m_0, \ldots, m_6\}^l$ encoding chromosome ($-1 \le m_i \in \Re \le 1$).
- $l \in \aleph$ length of chromosome.
- $D \in \aleph$ population size.
- $\Phi_F : M \rightarrow \Re$ fitness function.
- $sel : Crom^D \rightarrow Crom$ parent selection operation
- $pcross : Crom^2 \rightarrow Crom^2$ crossover operation.
- $pmut : Crom \rightarrow Crom$ mutation operation.

Each individual chromosome represents a complete rule base solution formulated as a set *M* of the generated *KxL* fuzzy if-then rules such that:

$$M^i=\{m_i^j \mid i=1, \ldots, D;\ j=1, \ldots, l\} \tag{29}$$

The set of all the individuals represents a population. If we denote by *P(t)* a population at a time *t*, then we can write:

$$P(t)=\left\{M^1(t),\ M^2(t),\ \ldots, M^D(t)\right\} \tag{30}$$

Where $m_i^j \in S_m \in \Re$, are the *j-th* consequent parts of the fuzzy rules of the *i-th* individual. It takes its value from the set S_m*={-1, -0.66, -0.33, 0.0, 0.33, 0.66, 1}*. These values correspond to the projection of the peaks of the membership functions on the normalized universe of discourse of the action.

4.2 Mobile platform fuzzy control design

If we consider the vehicle moving in a free obstacle environment, then the optimal trajectory from its current position to its end configuration is naturally a line joining these two extreme points as it is shown for instance by Figure 5, where θ is the angle between the symmetric axis of robot and the line that joins the control point of the robot to its final point.

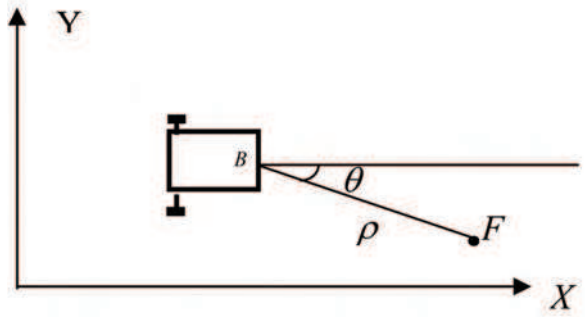


Figure 5. Example of situation " Reaching a point "

If we link the points with segments, then the goal is to control the driving point *A* of the autonomous robot with respect to these segments and to come the closest possible to the end point. The distance ρ becomes zero when the vehicle stabilizes at its final configuration. Figure 6 gives a schematic block diagram of this architecture. From this figure one can notice that the inputs to the fuzzy controller are ρ and θ, and its output is the steering angle γ. (Abdessemed, F. & Benmahammed, K. 2004)

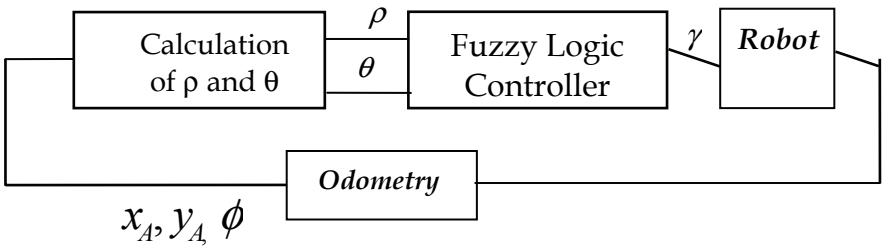


Figure 6. Block diagram of the controlled system

The best fuzzy system is implemented with five and eight triangular membership functions for the controller input variable ρ and θ respectively. The second step in designing an *FLC* is the fuzzy inference mechanism. For instance, the knowledge base of the system consists of rules in the form:

Rule $\boldsymbol{R_{ij}}$ *:* *IF* $\boldsymbol{\rho}$ *is S AND* $\boldsymbol{\theta}$ *is NM THEN* $\boldsymbol{\gamma}$ *is* n_p *i=1, 2, ... N; j=1, 2, ..., K; p=1, 2, ..., L.*

Such that:

$$\gamma(k)=F[\rho(k),\theta(k)] \tag{31}$$

$\rho(k)$ and $\theta(k)$ are inputs to the map F, and the output $\gamma(k)$ denotes a numerical value within the interval $[-90°, +90°]$, characterizing the relative variation in the direction that should be taken by the vehicle to reach the final point. The rules could be defined by using the human-like description of the vehicle's movement behaviour. But, this approximate human reasoning may lead to certain unsatisfactory rules. Furthermore, the global behaviour of the vehicle may result from the combination of several basic behaviours for instance, the trajectory tracking, the vehicle speed, and the obstacle avoidance. As a matter of fact, it is not obvious to define the respective rules. Therefore, and following the same approach described in the last section, we propose an evolutionary algorithm as an efficient solution for the extraction of the consequent part of the rules. A $(\mu+\lambda)$-evolutionary programming is described by an octuple entity defined by the following format:

$$\boldsymbol{EP}=\{I(t), L, \mu, \lambda, sel, pmut, f, g\} \tag{32}$$

For which the components are defined as follows:

- $I= [a_1, a_2, \ldots, a_{2L}]$ Encoding chromosome
- $2L \in \aleph$ Length of chromosome
- $\mu \in \aleph$ Population size
- $\lambda \in \aleph$ Number of offspring = μ
- $pmut: I \rightarrow I$ mutation operator
- $f: \Re\, L \rightarrow \Re$ fitness function
- $g: \Re\, L \rightarrow \Re$ set of constraints

The design of the *EP* is based mainly on three mechanisms:

- The representation of individuals,
- Implication of the variation operators,
- The generation procedure.

Each individual chromosome represents a complete rule base solution. The components: $(a_1 = m_1,\ a_2 = m_2,\ \ldots,\ a_L = m_L)$ determine the consequent part of the fuzzy rules and the remaining components, $(a_{L+1} = \sigma_1,\ a_{L+2} = \sigma_2,\ \ldots,\ a_{2L} = \sigma_L)$ contain the standard deviation, which controls the mutation process. A complete string of chromosome could be written in the following way: $a_1 a_2 \ldots\ a_L a_{L+1} a_{L+2} \ldots\ a_{2L}$, representing one individual. The set of all the individuals represent a population. If we denote by *P(k)* a population at a time *k*, then we can write:

$$P(k) = \bigcup_{i=1\ldots\mu} I^i(k) \text{ with } I^i = \left\{ a^i_j,\ \text{i}=1,\ldots,\mu,\ j=1,\ldots,2L \right\} \tag{33}$$

Where I^i designates the *i-th* individual in which the components a_j describe the consequent parts of the rules and the standard deviations. In this application, we have rather chosen the floating point encoding instead of the binary code. The algorithm seeks many local optima and increases the likelihood of finding the global optimum, representing the problem goal.

5. Motion Control of a Mobile Manipulator

The control strategy combines the mobile base behavior and the manipulator behavior to produce an integrated system that performs a coordinated motion and manipulation. If we refer to the arm manipulator by *agent1* and the mobile platform by *agent2*, then the architecture shown by Figure 7 illustrates the actions on the environment by the two defined agents.

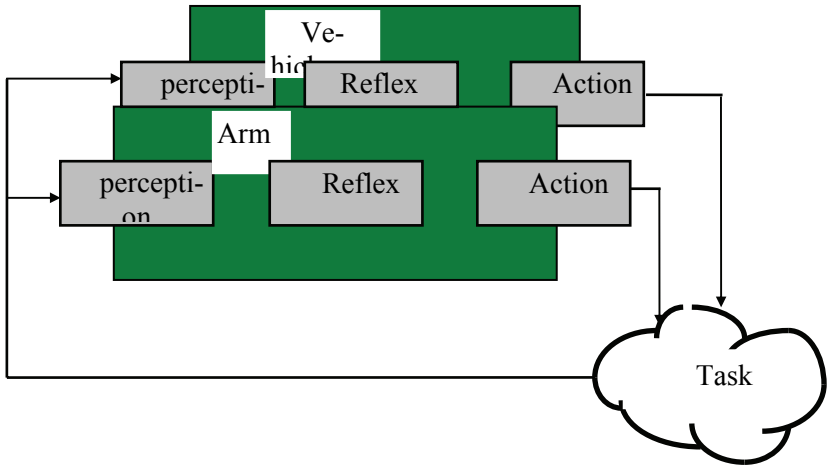


Figure 7. Configuration of the coordinated motion and manipulation of the robotic system architecture

To provide a solution to the mobile manipulation motion, we have arranged the direct geometric model equations (1) into a sort of adaptive graph of operation made up of three layers, and which we have called a neural-like-network (Fig. 8). Each layer has a number of transfer functions each of which is defined by the expressions given by equations (34). This neural-like-network is the kernel of our proposal; it is very interesting and uncommon in robot trajectory generation (F. Abdessemed, *et al*. 2006). In this case, the two mechanical stru tures are considered as a unique entity. This arrangement will facilitate the implementation of the back propagation algorithm as a learning rule to adapt the weights so that the output values of the neural-like-network come close to the desired reference values describing the task space trajectory. The accomplishment of the task is the result of the permanent movement of the two structures for which the success is based on the satisfaction of the tracking error. Figure.8 illustrates the model architecture of this combined structure. For convenience we define *x31*, *x32*, *x33* as the outputs of the network, and which designates the Cartesian coordinates of the task variable $E: x_E^W, y_E^W$ and z_E^W respectively. Where:

$$
\begin{aligned}
&f_{11}(\theta,\ w_{11}) = x_{11} = \cos(w_{11}\theta)\\
&f_{12}(\theta_2,\ w_{22}) = x_{12} = \cos(w_{22}\theta_2)\\
&f_{13}(x_{12}) = x_{13} = \cos^{-1}(x_{12})\\
&f_{14}(x_{13}, \theta_3,\ w_{33}) = x_{14} = (x_{13} + w_{33}\theta_3)\\
&f_{21}(x_{11}) = x_{21} = \sin(\cos^{-1}(x_{11}))\\
&f_{22}(x_{12},\ x_{14}) = x_{22} = l_3\cos(x_{14}) + l_2 x_{12}\\
&f_{23}(x_{13}) = x_{23} = l_2\sin(x_{13})\\
&f_{24}(x_{14}) = x_{24} = l_3\sin(x_{14})\\
&f_{31}(x_{11},\ x_{22}) = x_{31} = x_{11}.x_{22} + b_1 x_B\\
&f_{32}(x_{21},\ x_{22}) = x_{32} = x_{21}.x_{22} + b_2 y_B\\
&f_{33}(x_{23},\ x_{24}) = x_{33} = l_1 - x_{23} - x_{24}
\end{aligned}
\qquad (34)
$$

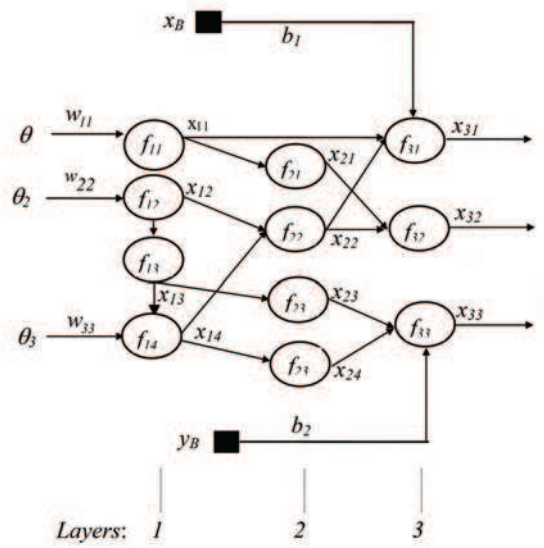


Figure 8. The adaptive graph of operations

For convenience we define x_{ij} as the outputs of the network, and which x_{31}, x_{32}, x_{33} are the outputs of the network, and wich designates the Cartesian coordinates of the task variable $E.(x_E^W, y_E^W z_E^W)$. Let $\boldsymbol{q}(k)$ be the input vector, such that $\boldsymbol{q}^T(k) = [\theta, \theta_2, \theta_3, x_B, y_B]$ and ${}^m X_E^W$ the measured output vector such that $X_E^W(k) = [{}^m x_E^W, {}^m y_E^W, {}^m z_E^W]^T$,, and the weighting vector $\boldsymbol{W}$ such that $W^T(k) = [w_{11}, w_{22}, w_{33}]$. If we set the criterion E_p be the tracking error given by equation (35), then the control objective is to design a control law, which guaranties: E_p to go zero when k goes to infinity,, k is the running time.

$$E_p = \sum_{k=1}^{N(3)} (x_{3k} - r_k)^2 = \left[(x_{31} - r_1)^2 + (x_{32} - r_2)^2 + (x_{33} - r_3)^2 \right] \tag{35}$$

Where

${}^d X_E^W = (r_1, r_2, r_3)^T = ({}^d x_E^W, {}^d y_E^W, {}^d z_E^W)^T$, represent the desired operational coordinates and ${}^m X_E^W = (x_{31}, x_{32}, x_{33})^T = ({}^m x_E^W, {}^m y_E^W, {}^m z_E^W)^T$ the operational measured coordinates in the world frame. The effect of adjusting the weighting vector $\boldsymbol{W}$ to the error $\boldsymbol{Ep}$ is determined by the ordered derivatives $\partial^+ E_p / \partial W(k)$ (Werbos, 1974). Now, we apply the back-propagation learning rule to generate the appropriate parameter-weighting vector $\boldsymbol{W}(k)$ (Rumelhart *et al*, 1986). Once determined, the weights are used to update the input vector $\boldsymbol{q}$. The elements of this vector will serve as inputs to the low level controllers of the two agents as

illustrated by the block diagram of Fig. 9. The reference states of the plant at time *k+1* are functions of the reference states at time *k* and the computed weights at time *k+1*, and can be expressed symbolically as

$$q(k+1) = \Psi(q(k), W(k+1)) \tag{36}$$

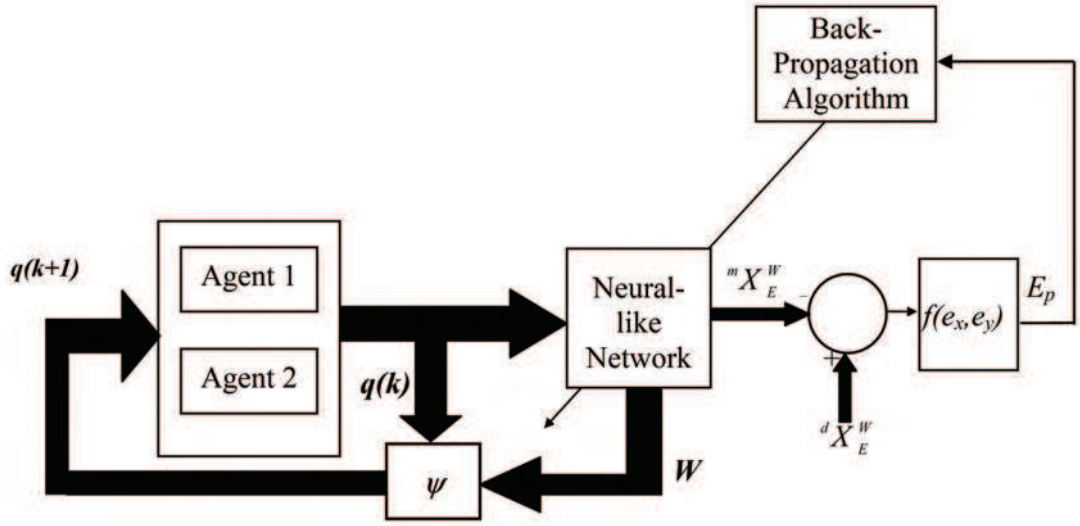


Figure 9. Configuration of the controlled system including the adaptive graph of operations

6. Back-Propagation Learning Rule

6.1 Output Layer

The error signal for the *j-th* output node can be calculated directly:

$$\varepsilon_{3,i} = \frac{\partial^{+} E_p}{\partial x_{3,i}} = \frac{\partial E_p}{\partial x_{3,i}} \tag{37}$$

Therefore

$$\varepsilon_{31} = 2(x_e^d - x_{31}),\ \varepsilon_{32} = 2(y_e^d - x_{32}),\ \varepsilon_{33} = 2(z_e^d - x_{33})$$

6.2 Internal Layers

The error signals of these internal nodes at the *j-th* position are calculated using the following equation

$$\varepsilon_{l,i}=\underbrace{\frac{\partial^{+}E_p}{\partial x_{l,i}}}_{\substack{\text{Error}\\ \text{Signal of}\\ \text{layer l}}}=\sum_{m=1}^{N(l+1)}\underbrace{\frac{\partial^{+}E_p}{\partial x_{l+1,m}}}_{\substack{\text{Error}\\ \text{Signal of}\\ \text{layer l+1}}}\times\frac{\partial f_{l+1,m}}{\partial x_{l,i}} \tag{38}$$

$$\varepsilon_{l,i}=\sum_{m=1}^{N(l+1)}\varepsilon_{l+1,m}\frac{\partial f_{l+1,m}}{\partial x_{l,i}} \tag{39}$$

Such that: $0 \le l \le L-1$

$$\varepsilon_{2,j}=\sum_{m=l}^{2}\varepsilon_{3,m}\frac{\partial f_{3,m}}{\partial x_{2,j}}. \quad j=1,2. \tag{40}$$

Therefore the error signals of the nodes at the internal layer are as follows:

$$\varepsilon_{2,1}=\varepsilon_{3,2}.x_{2,2},\ \varepsilon_{2,2}=\varepsilon_{3,1}.x_{1,1}+\varepsilon_{3,2}.x_{2,1},\ \varepsilon_{2,3}=-\varepsilon_{33},\ \varepsilon_{2,4}=-\varepsilon_{3,3}$$

6.3 Input Layer

The first layer contains four neurons arranged in the way presented by Figure 8. The general form for the error signal is given by the following equation:

$$\varepsilon_{1,i}=\sum_{m=1}^{4}\varepsilon_{2,m}\frac{\partial f_{2,m}}{\partial x_{1,i}} \tag{41}$$

Explicitly the error signals are found to be

$$\varepsilon_{1,1}=-\varepsilon_{2,1}\frac{x_{1,1}}{\sqrt{1-x_{1,1}^2}}+\varepsilon_{3,1}.x_{2,2} \tag{42}$$

$$\varepsilon_{1,2}=-\varepsilon_{2,2}.l_2-\varepsilon_{1,3}\frac{1}{\sqrt{1-x_{1,2}^2}} \tag{43}$$

$$\varepsilon_{1,3}=\varepsilon_{2,3}.l_2\cos(x_{1,3})+\varepsilon_{1,4} \tag{44}$$

$$\varepsilon_{1,4}=l_3\left[\varepsilon_{2,4}.\cos(x_{1,4})\ -\ \varepsilon_{2,2}\sin(x_{1,4})\right] \tag{45}$$

6.4 Weight adjustment

To adjust the weights we make use of the following known equation:

$$w_{ij}^{l}(k+I)=w_{ij}^{l}(k)-\mu\frac{\partial E_p}{\partial w_{ij}^{l}(k)}\bigg|_{w_{ij}^{l}(k)} \tag{46}$$

Where

$$\frac{\partial^{+}E_p}{\partial w}=\frac{\partial^{+}E_p}{\partial x_{l,i}}\frac{\partial^{+}f_{l,i}}{\partial w}=\varepsilon_{l,i}\frac{\partial f_{l,i}}{\partial w} \tag{47}$$

Therefore the weights are updated according to the following resulting equations:

$$w_{11}(k+1)=w_{11}(k)+\eta\varepsilon_{1,1}.\ \theta.\ \sin(w_{11}.\theta) \tag{48}$$

$$w_{22}(k+1)=w_{22}(k)+\eta\varepsilon_{1,2}.\theta_2\sin(w_{22}\theta_2) \tag{49}$$

$$w_{33}(k+1)=w_{33}(k)-\eta\theta_3\varepsilon_{1,4} \tag{50}$$

$$b_1(k+1)=b_1(k)-\eta x_B\varepsilon_{3,1} \tag{51}$$

$$b_2(k+1)=b_2(k)-\eta y_B\varepsilon_{3,2} \tag{52}$$

Where, the two last equations represent the updates of the biases b_1 and b_2. Nevertheless, the steepest descent algorithm is slower for on-line applications. For that reason, it is rather advisable to use the Levenberg-Marquardt algorithm, which has proved to be an effective way to accelerate the convergence rate. Its principal advantage is that it uses information about the first and second derivatives and does not need to invert the Hessian matrix.

$$x_{k+1}=x_k-\left[J^TJ+\mu I\right]^{-1}J^T\varepsilon \tag{53}$$

The weights are updated on each iteration by the following found expressions:

$$
\begin{aligned}
w_{11}(k+1) &= w_{11}(k) - \frac{j_{11}}{j_{11}^2 + \mu}\varepsilon_{1,1} \\
w_{22}(k+1) &= w_{22}(k) - \frac{j_{22}}{j_{22}^2 + \mu}\varepsilon_{1,2} \\
w_{33}(k+1) &= w_{33}(k) - \frac{j_{33}}{j_{33}^2 + \mu}\varepsilon_{1,4} \\
b_1(k+1) &= b_1(k) - \frac{j_{44}}{j_{44}^2 + \mu}\varepsilon_{3,1} \\
b_2(k+1) &= b_2(k) - \frac{j_{55}}{j_{55}^2 + \mu}\varepsilon_{3,2}
\end{aligned}
\tag{54}
$$

Where

$$
\begin{aligned}
j_{11} &= -(\theta_1 + \varphi)\, sin(w_{11}(\theta_1 + \varphi)) \\
j_{22} &= -\theta_2\, sin(w_{22}\theta_2) \\
j_{33} &= \theta_3 \\
j_{44} &= x_B \\
j_{55} &= y_B
\end{aligned}
\tag{55}
$$

Where, *I* is the identity matrix. At this end stage, we give the reference state variables at time *k+1* and which are represented by the following expressions:

$$
\begin{aligned}
\theta(k+1) &= w_{11}(k+1)\theta(k) \\
\theta_2(k+1) &= w_{22}(k+1)\theta_2(k) \\
\theta_3(k+1) &= w_{33}(k+1)\theta_3(k) \\
\theta_1 &= \theta - \varphi \\
x_B(k+1) &= b_1(k+1)x_B(k) \\
y_B(k+1) &= b_2(k+1)y_B(k)
\end{aligned}
\tag{56}
$$

7. Simulation Results

Simulation examples are carried out in order to evaluate the developed approach. It is desirable to move the end effector from its initial position P_1 *(1, 1, 0.2)* to its final position P_2 *(5,5,0.5)*, by tracking instantaneously a linear specified trajectory of the end effector generated by a uniform Cartesian movement. Neural-like-network learns the desired values presented and adjusts the weights appropriately in order to present to the system the corresponding reference state variables. The results of the simulation are shown in Figures 12 to 15 and indicate how successfully the Cartesian coordinates of the endeffector track their corresponding reference values very closely. We notice that the small departures from the reference trajectories are due to the cumulated tolerable errors from the learning process. The learning algorithm was run by using a learning rate μ=*0.05* for a laps of time not exceeding real time control. All the weights have been initialised to the value of one. At each step, the learning rate is updated depending on the behaviour obtained. If the overall error is improved, then the learning rate is increased by the value μ=μ*μ_*inc;* otherwise, it is deceased by the value μ=μ*μ_*dec,* knowing that μ_*inc* and μ_*dec* take the values of *1.05*, and *0.95* respectively. Figures 16 to 19 show the plots of the manipulator angular values as well as the orientation of the mobile platform, and Figure 20 clearly shows the trajectories of the end effector and the mobile platform in the *xyz* space. Figure 21 depicts a 3D perspective of the simulation environment.

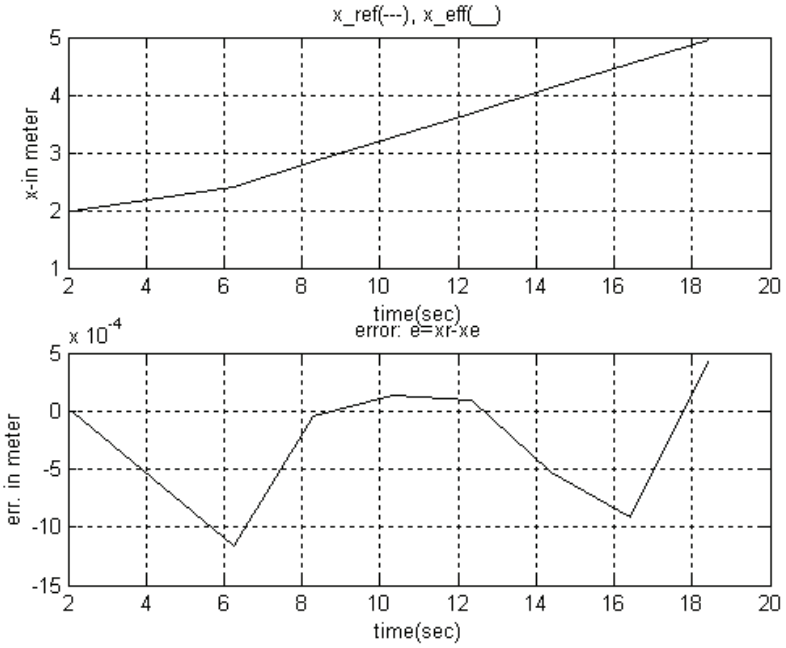


Figure 12. Desired and measured x-trajectory plots and the error resulted

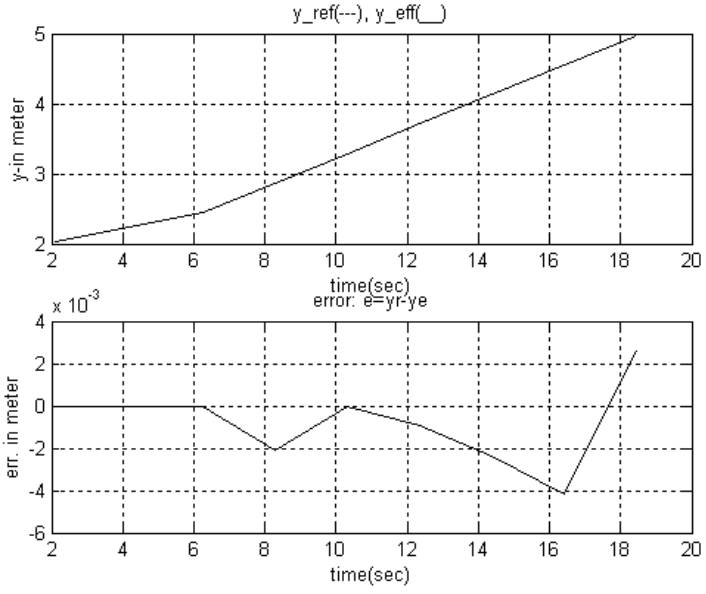


Figure 13. Desired and measured y-trajectory plots and the error resulted

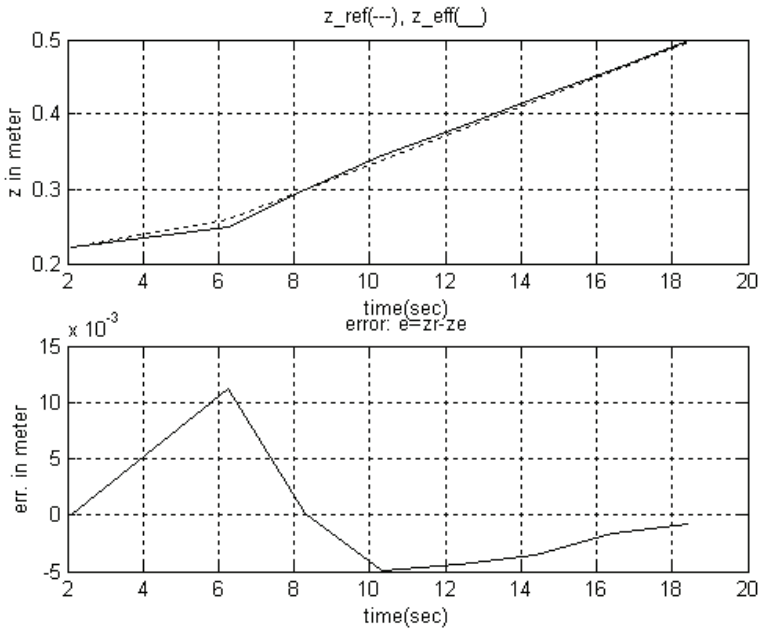


Figure 14. Desired and measured z-trajectory plots and the error resulted.

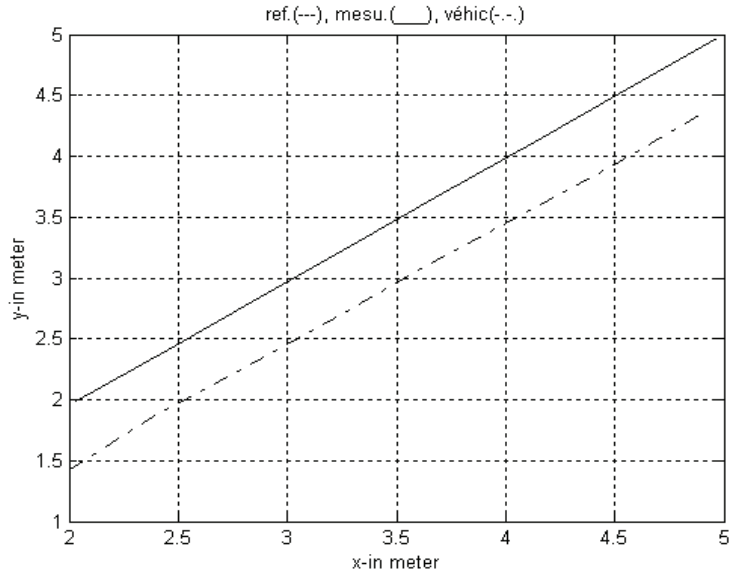


Figure 15. X-Y Plots of the end-effector an d the mobile platform trajectories

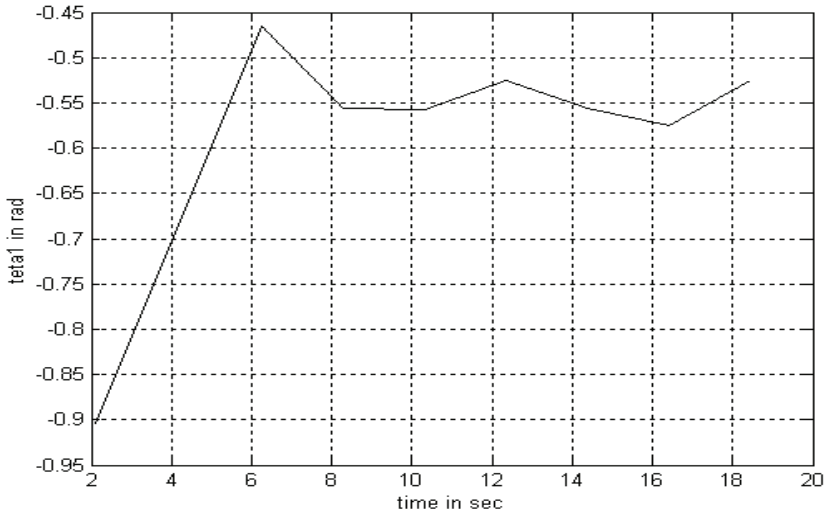


Figure 16. Angular trajectory θ_1

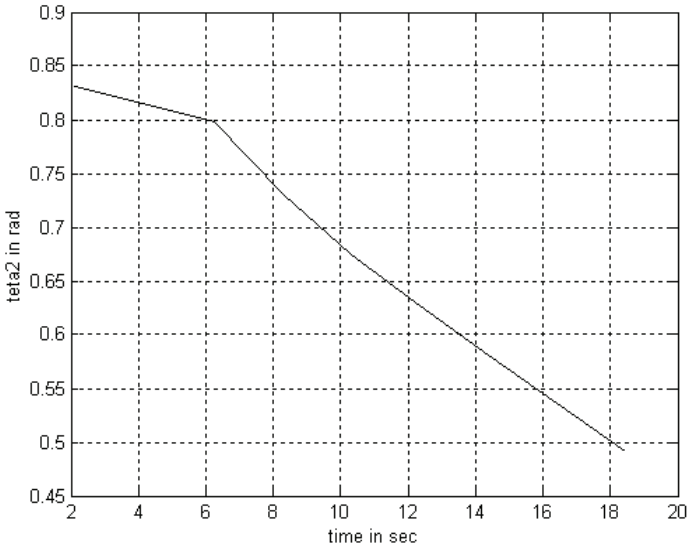


Figure 17. Angular trajectory θ_2

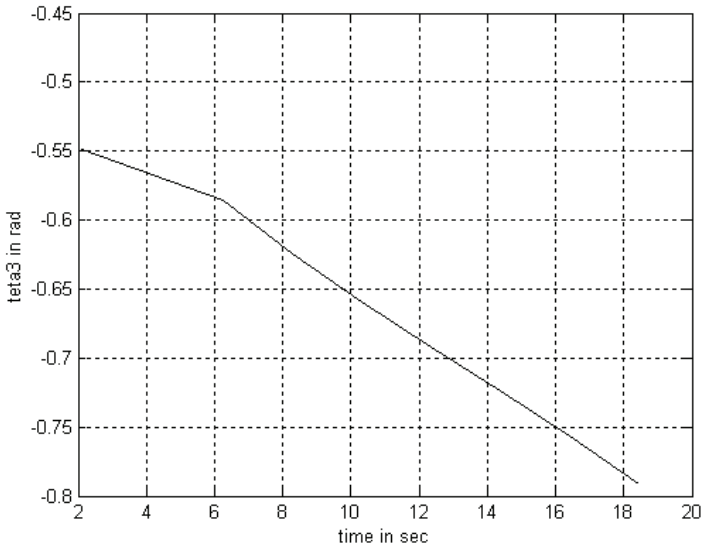


Figure 18. Angular trajectory θ_3

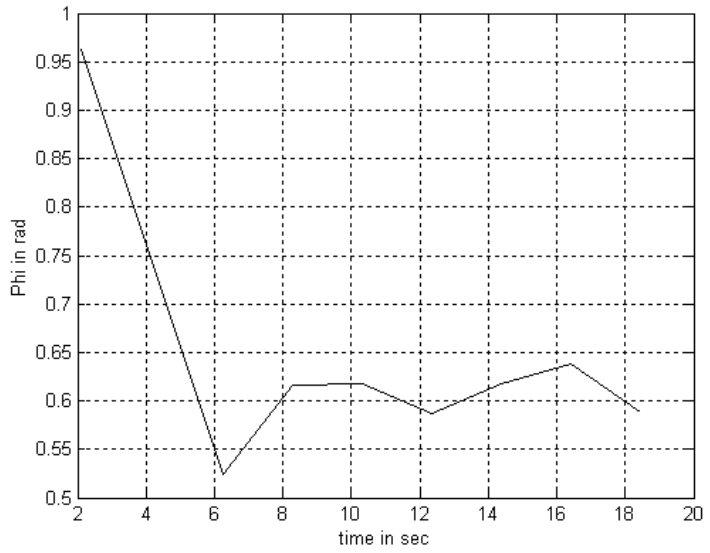


Figure 19. Angular trajectory φ

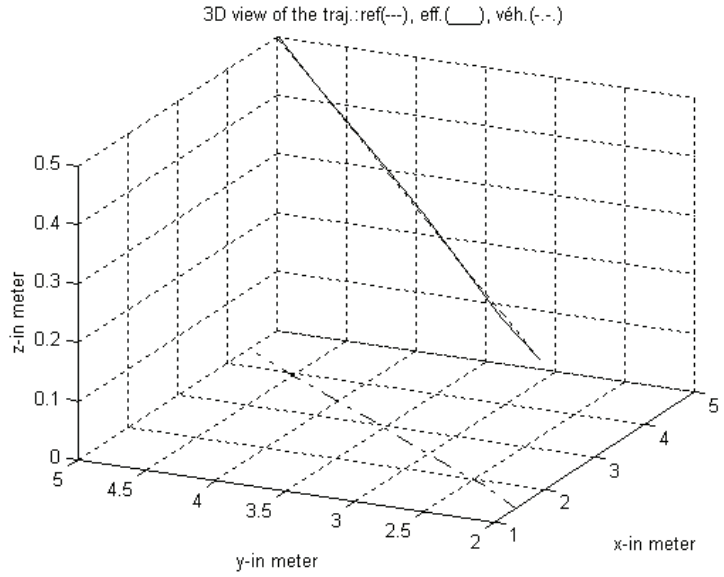


Figure 20. X-Y-Z plots of the end-effector and the mobile platform trajectories

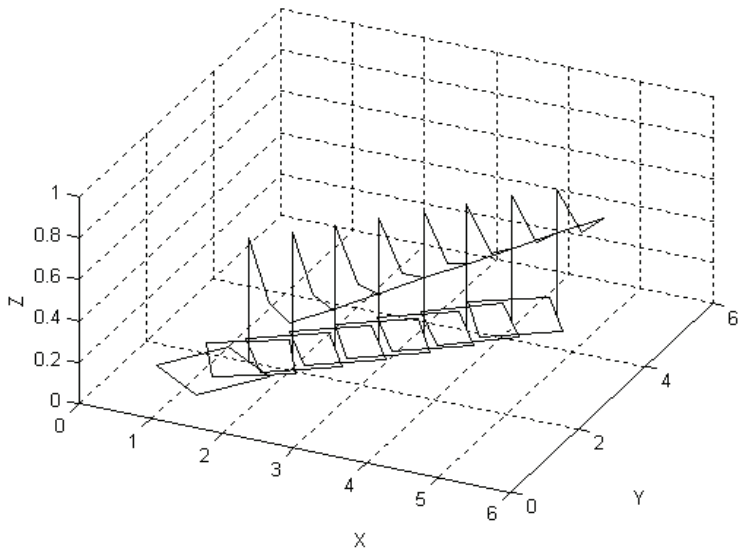


Figure 21. A 3D perspective of the simulation environment

8. Conclusion

Soft computing is an emerging field that consisting of complementary elements of fuzzy logic, neural computing, evolutionary algorithms, and machine reasoning. In this
paper we propose the use of a back propagation to train a neural-like-network to coordinate the motion of a robotic manipulator with the motion of the mobile platform over which a robot manipulator is mounted. The network pr vides reference output values of the desired motion to the mobile manipulator system. The parameter weighting vector determined is used to compute inputs to the platform and to the manipulator so that the end-effector trajectory is tracked with minimum error. Robot manipulator and platform control is predominately motion control. The mobile platform is considered as a "macro-mechanism" with coarse, slow dynamic response, and the arm is a fast and accurate "mini-device. For this reason we consider the kinematics model for the mobile platform and the dynamic model for the robot manipulator. Fuzzy controllers are used as means to control each of the two subsystems separately; and the overall system demonstrates very good control characteristics.

9. References

Abdessemed, F. & Benmahammed, K. (2001). A Two-layer robot controller design using evolutionnary algorithms, Journal of Intelligent and Robotic System 30, pp. 73-94, 2001.

Abdessemed, F. & Benmahammed, K. (2004). A Fuzzy-based reactive controller for non-holonomic mobile robot, Journal of Robotics and Autonomous Systems 47, pp. 31-46, 2004.

Bejczy, A. K., (1974). Robot Arm Dynamics and Control, *Technical Memorandum 33-669*, Jet Propulsion Laboratory, Feb. 1974.

Brock, O., Khatib, O. & Viji, S. (2002). Task-Consistent Obstacle Avoidance and Motion Behavior for Mobile Manipulator, In Proc. of the IEEE Int. Conf. on Robotics and Automation, 2002.

Craig, J.J., Hsu, P. & Sastry, S. (1987). Adaptive Control of Mechanical Manipulators, *The Inernational Journal of Robotics Research*, pp 16-28, Vol. 6. No. 2. Suymmer 1987.

Dubowsky, S., desforges, D. T., (1979). The Application of Model reference Adaptive Control to Robotic Manipulators, *Transaction of the ASME, Journal of Dynamic Systems, Measurement and control*, pp 193-200, Vol. 101, Sep. 1979.

Karr, C.L. (1991). Design of an Adaptive Fuzzy Logic Controller Using a Genetic Algorithm, in Proc. Fourth .Int..Conf. on Genetic Algorithms. pp. 450-457, San Diego, July 13-16, 1991.

Lee, J. K., & Cho, H. S. (1997). Mobile Manipulator Motion Planning for Multiple Tasks Using Global Optimization Approach, Jour. of Intell. and Robotic Systems 18, pp. 169-190, 1997.

Mamdani, E.H. & Assilian, S. (1974). Application of fuzzy algorithms for control of simple dynamic plant, Proc, Inst.Elec.Eng., pp.1585-1588, Vol. 121. 1974.

Mamdani, E.H. (1974). Application of Fuzzy Algorithms for Control of Simple Dynamic Plant, *Proc.IEE,121(12)*; pp1585-1588.-1974

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by Error Propagation, *in Parallel Distributed Processing: Exploration, in the Microstructure of Cognition, MA: MIT Press*, pp. 318-362,Vol.1, D.E.Rumelhart and James L. McClelland, Eds, Cambridge, Chap.8, 1986.

Samsung, C. & Abderrahim, K.A. (1990). Mobile robot control Part.I: Feedback control of a nonholonomic wheeled cart in cartesian space, Technical report, INRIA, France, 1990

Spong, M. W. & Ortega, R. (1988). On Adaptive Inverse Dynamic Control of Rigid robots, IEEE trans. Automat., 1988.

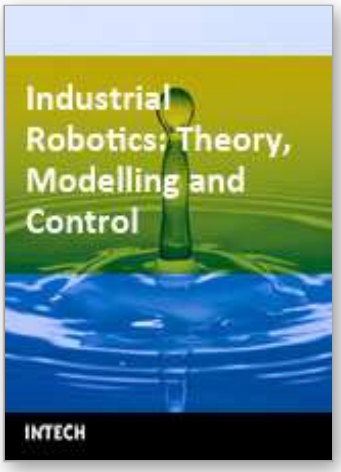
Werbos, P. (1974). Beyond Regression: New tools for Prediction and Analysis in the Behavioral Sciences, *Ph.D. Dissertation*, Harvard Univ., 1974.

Whiteney, D. E. (1969). Resolved Motion Rate Control of Manipulators and Human Protheses, *IEEE Transactions on Man Machine System*, pp. 47-53, Vol. MMS-10, No. 2, June 1969.

Yamamoto, Y. & Yun, X. (1994). Coordination locomotion and manipulation of a mobile manipulator, *IEEE Transactions on Automatic Control*, pp. 1326-1332, 39(6),1994

Zadeh, L. (1994). Fuzzy logic, Neural network, and Soft computing, commun. ACM, pp. 77-84, Vol. 37, no. 3, 1994.

Zadeh, L.A. (1973). Outline of a New Approach to the Analysis of Complex Systems and Decisions, IEEE Trans.SMG, pp. 28-44. Vol. 3, 1973



Industrial Robotics: Theory, Modelling and Control

Edited by Sam Cubero

ISBN 3-86611-285-8

Hard cover, 964 pages

Publisher Pro Literatur Verlag, Germany / ARS, Austria

Published online 01, December, 2006

Published in print edition December, 2006

This book covers a wide range of topics relating to advanced industrial robotics, sensors and automation technologies. Although being highly technical and complex in nature, the papers presented in this book represent some of the latest cutting edge technologies and advancements in industrial robotics technology. This book covers topics such as networking, properties of manipulators, forward and inverse robot arm kinematics, motion path-planning, machine vision and many other practical topics too numerous to list here. The authors and editor of this book wish to inspire people, especially young ones, to get involved with robotic and mechatronic engineering technology and to develop new and exciting practical applications, perhaps using the ideas and concepts presented herein.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821