Learning manipulative skills using an Artificial Intelligence approach

Paweł Chmielarczyk

A thesis submitted in partial fulfilment of the requirements of
Sheffield Hallam University
for the degree of Doctor of Philosophy

December 2006

Abstract

The aim of this research was to design a non-linear controller based on an Artificial Neural Network and Reinforcement Learning algorithms implementation, which is able to perform an intelligent robotic assembly of mechanical components.

Different information was applied and combined to develop a fully unsupervised, intelligent controller. In the author's design no class labelling or geometry feature pre-training takes place. Only force and torque signals together with the direction of insertion were supplied to the controller.

A unique sandwich structure of the intelligent controller was proposed. It featured two major layers, a State Recognition module where the detection and localisation of the contact points were performed, and the Decision Making subsystem where the decision about the next action took place.

All the algorithms were implemented and tested on simulated data before being applied to the real-life peg-in-hole insertion. The results are presented in the form of graphs and tables.

Evaluation of the environmental uncertainty was accomplished. The signal from the force and torque sensor was acquired under controlled conditions. All the data was collected to establish the area and level of uncertainty (e.g. signal errors) the artificial controller would need to learn to cope with and compensate for.

The empirical part of the thesis includes the investigation into the effects of different learning methods applied on the same geometry. The influence of action-selection methods on AI agent performance was analysed. The proposed controller was applied to a set of real life peg-and-hole experiments. Both circular and square peg geometries were used, and insertions into chamfered and non-chamfered holes were performed. Materials with different friction factors were used for mating parts.

Fast and stable knowledge acquisition was clearly present in all the cases investigated. A significant reduction in contact force value during the initial stage of the learning process was recorded. The force was usually reduced to one tenth of the initial value. Some fluctuations were recorded but when the cylindrical peg was considered the value of contact forces never exceeded 0.5 N during the steady state.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank all those who helped me with this study.

Firstly, I extend my thanks to Dr Martin Howarth for providing the opportunity of carrying out this study, and for his encouragement and advise throughout. Special thanks to Richard Wainwright for his steadfast assistance, especially during robot breakdowns. Thanks also to my friend Glenys Rogerson for her patience and endurance in guiding me through the nuances of the English Language.

Finally I am grateful to my parents and friends for their continuous support throughout my studies.

List of abbreviations

The abbreviations are explained in the thesis at the point of their first reference. However they reproduced here for the benefit of the reader.

AI	Artificial Intelligence
ANN	Artificial Neural Networks
ANSI	American National Standards Institute
ART	Adaptive Resonance Theory
CAD	Computer Aided Design
CIM	Computer Integrated Manufacturing
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CRT	Cathode Ray Tube
DDCMP	Digital Data Communications Message Protocol
DOF	Degree Of Freedom
ES	Expert Systems
FL	Fuzzy Logic
FMC	Flexible Manufacturing Cell
F/T	Force and Torque
GNU	Gnu's Not Unix
I/O	Input - Output
ISO	International Standards Organisation
LKM	Loadable Kernel Modules
LTM	Long Term Memory
MLP	Multi Layer Perceptron
PKB	Primitive Knowledge Base
POSIX	Portable Operating System Interface for uniX
PUMA	Programmable Universal Machine for Assembly
RAM	Random Access Memory
RL	Reinforcement Learning
SCARA	Selective Compliant Assembly Robot Arm
SCS	Supervisory Computer System
SOM	Self Organising Map
SRV	Stochastic Reinforcement Value
STM	Short Term Memory
TD	Temporal Difference
UNECE	United Nations Economic Commission for Europe
VAL	Variable Assembly Language

List of symbols

The symbols are explained in the thesis at the point of their first reference. However they reproduced here for the benefit of the reader.

$\vec{f}$, F	force vector
$\vec{m}$, M	torque vector
$\vec{v}$	linear velocity vector
ω	angular velocity
α	RL learning ratio
γ	RL discount factor
ϵ	probability of exploratory action
μ	exploration bias
β	Gibbs distribution ratio
ρ	ART2 vigilance parameter
Q	transition function, mapping each state-action pair to a successor state
n	episode number
a	set of actions an agent can perform
s	set of states an agent can experience
r	reward from environment
t	discrete time step
P'_n	activation patterns of SOM
F1, F2	ART2 main layers
δ_x, δ_y, δ_z	linear movement correction
$\delta\theta_x$, $\delta\theta_y$, $\delta\theta_z$	rotational movement correction
O, A, T	angles of robot's TOOL coordinate system orientation
X, Y, Z	main coordinate system axes

Glossary of terms

In this section a brief glossary of domain specific terms is presented.

Artificial Neural Networks - a type of massively parallel computing architectures based on brain-like information encoding and processing models and, as such, they can exhibit brain-like behaviours such as: learning, association, categorisation, generalisation, feature extraction, pattern recognition or optimisation.

Adaptive Resonance Theory - a type of ANN where the information in the form of processing element outputs reverberate back and forth between layers. If the proper patterns develop, a stable oscillation occurs. This is the neural-network equivalent of resonance. Only during this stage can learning take place

Reinforcement learning - a goal oriented method of knowledge acquisition where the decision making process is based on value functions not on the immediate reward.

Temporal-Difference - learning algorithms to make long-term predictions about dynamic systems.

Q-learning - an off-policy, Temporal-Difference learning method. The action-value function Q directly approximates the optimal action-value function Q^*, independent of the policy being followed.

SARSA - an on-policy, Temporal-Difference algorithm. Similarly to q-learning SARSA gathers knowledge from the state-action pairs transitions but it learns the policy directly.

Value functions - a function which maps the actions (or state-action pairs) onto the response from the environment (the reward).

Reward - a primary feedback signal which provides the immediate response of the environment.

Learning policy - definition of relationship between the states and the actions.

Chapter 1

Introduction

Manufacturing could be described as a set of operations and activities performed in order to make a final product. It involves a product design, planning, production, assembly, inspection and marketing.

Computers have been used in the manufacturing process for many years. It started in the late 1960's when they were applied to direct control of groups of machine tools [25]. Later the concept of the Flexible Manufacturing Cell (FMC) was introduced. The FMC defines a group of semi-independent workstations linked by the material handling system. This design allows a production of a variety of different items automatically. The increase in computational speed of modern CPUs caused a rapid advancement in development of Computer Integrated Manufacturing (CIM) systems. This approach allows a fast and integrated flow of manufacturing activities. The process of making a part has become fully automated which has contributed to improvement in quality and efficiency.

There has been a significant technological advancement since the introduction of the first commercial robot for automated assembly in the 1970s. The modern manipulators are faster, more reliable, and cost effective. High performance and a considerable drop in prices has contributed to a significant increase in industrial application of robots. According to the United Nations Economic Commission for Europe (UNECE) report, year 2001 was a record year for robotic investments in Europe [38]. Currently there are about 838,400 units working in industry worldwide [39] and the number of installations is quickly increasing (see Figure 1.1).

The industrial robots have been successfully applied at almost every stage of the manufacturing process. Tidd in his paper [37] classifies the industrial robots according to their application into three major groups:

- handling robots - holding the components, loading or unloading the machines, working in hazardous environments (e.g. nuclear industry),
- process robots - including painting, deburring, grinding, polishing, spot and arc welding robots,
- assembly robots - involving an assembly of the final component.

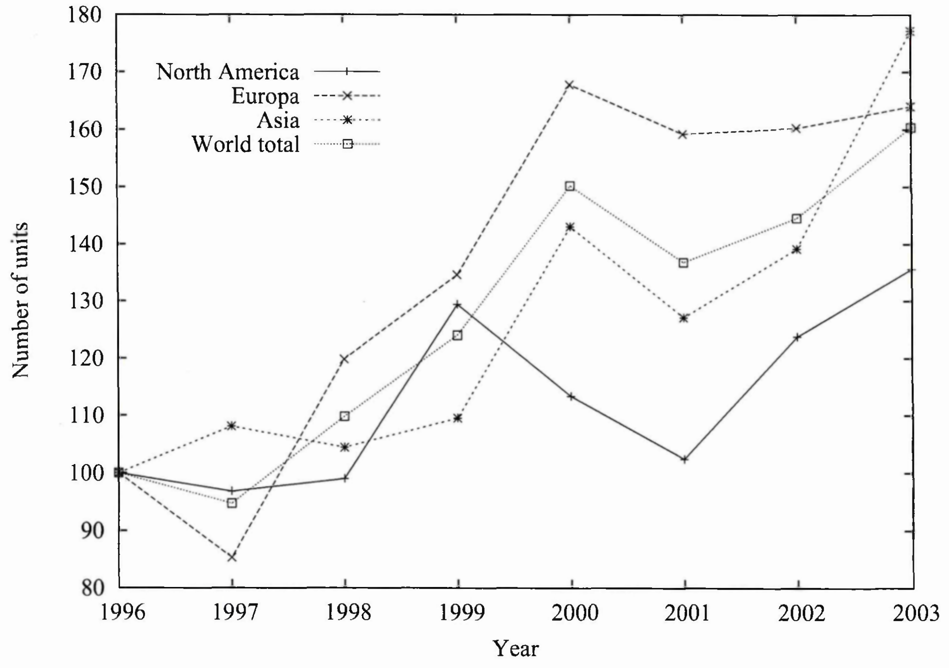


Figure 1.1: Orders placed for industrial robots by region [38, 39].

Assembly operations are amongst the most common and complicated in the manufacturing process. It usually requires an accurate motion control guided by a vision system since the miniaturisation and complexity of modern components make them impossible to be assembled by a human (e.g. electronics industry). With the introduction of CIM the speed, repeatability, and flexibility of robots is also very important.

The early applications of robots in automated assembly techniques have relied on passive accommodation methods, simple sensing systems and the manipulator's programming language. These approaches show many restrictions that make them unable to deal with a complex geometry of the components. The advantage of a passive system lies in its stability while interacting with the environment during an assembly task.

In a typical force guided assembly task, the robot's trajectory corrections are calculated and applied, according to predefined control laws. The more sophisticated force control methods involve modification of trajectory based on continuous force feedback from the system and a task description. Several methods of force control were researched. Amongst them:

- damping control [27,41], where the sensed forces cause velocity modification,
- stiffness control [43], where the force is calculated from actual and desired arm positions,
- impedance control [8, 14, 23], combination of two methods above, the dynamic relation between force error and position error is controlled,
- hybrid (position/force) control [32,41,45], in this method the force is controlled in certain directions and position control is applied in the complementary directions of the end effector's axes

Recently a new way of programming has been proposed. The intelligent control embraces substantially different techniques that include application of knowledge based Expert Systems (ES), Artificial Neural Networks (ANN), Fuzzy Logic (FL) or Reinforcement Learning (RL) algorithms. All these algorithms are different in principles and based on different theories but their applications share similar methodology: an attempt to create an effective mapping from sensory state space to the action domain.

The use of adaptive and learning capabilities in the assembly process simplifies the implementation and improves the reliability of these systems. The main advantage of an AI approach is the ability to solve problems without a detailed or explicit algorithm available for the solution. This has a tremendous impact on dealing with complex parts geometry or noisy sensory signals during the assembly process as well as the development of automatic error recovery methods.

In a complex environment, hard coding of every possible state-action mapping would result in much larger source code than in the case of self adapting methods application. In some tasks, with an infinite number of states, it is virtually impossible to implement the relationships using traditional methods. With the ability of ANNs to generalise, the complex 3-dimensional state space could be clustered and classified.

The unsupervised neural and reinforcement learning controllers show the ability to self-improve by using the experience gained from the previous tasks. This simplifies tremendously the tedious task of robot programming, and enables the controllers to learn the best action for a given state without any supervisory influence.

The unsupervised, self-organising algorithms have revolutionised the modern intelligent assembly methods and are state of the art in robot force control.

Early research work by Whitney [41, 42] and Simunovic [35] concluded that most common assembly operations could be modelled as simple peg-in-hole insertions. This represents a large number of part mating operations carried out in industry. Whitney [42] defined 4 stages during peg-in-hole assembly operation. He also defines the failures associated with insertions namely: jamming and wedging of the peg.

The peg-in-hole model, its dynamics, and acting forces have been widely studied by researchers [1,6,13,18,24,31].

The most successful industry-orientated approach was an application of Remote Centre Compliance (RCC) techniques. The passive compliance devise, proposed by Whitney et al [44], consist of springs and dampers to reduce the end effector stiffness. During the assembly, the passive component of the robot's wrist deflects under the contact forces correcting the misalignment of mating parts.

Many assembly operations involve insertions of the components with tolerances much finer than the resolution and accuracy of most available robots. This leads to the need for force feedback, experience and intelligent control methods to solve the non-linear compliance problems associated with those operations. Simons et al [34] pioneered the research and defined the need for learning and adaptivity in robotic assembly.

1.1 Aims and objectives

There are many areas where performance of intelligent controller could be improved. Most recent research has been focused on increasing the AI controller's learning speed. The research presented in this thesis aimed at the design, implementation and test of the unsupervised, intelligent system being able to perform a robotic assembly of mechanical components.

The main aims of this research include:

- Investigation of the different information sources and their employment in the peg-in-hole insertion problem. Amongst these, one source is sensory (signal from force and torque sensor) and descriptive (geometry of parts and coded task description and assembly direction).

- Investigation of automatic state recognition and clustering. This involves the analysis of contact states as well as design and implementation of a module for geometry classification.

- Investigation of unsupervised motion generation. The intelligent decision making agent, able to learn the state-action map, should be implemented, tested and applied.

- Design a stable and flexible system able to support several different AI architectures. For that reason the code portability and re-usability should be considered as an important issue during the design process,

- Test and validation of the controller using a range of mechanical components. The system should be applied on real-life, 3-dimensional peg-in-hole assembly tasks. The different features of the mating parts should be considered.

The controller should be totally unsupervised and gain all the knowledge from experience. The approach will be implemented and tested using a Puma 560 robotic arm connected to a supervisory PC with Pentium 200MHz processor under GNU/Linux operating system. The JR3 F/T sensor will be used to establish force feedback from the system.

1.2 Original contribution

As stated in the previous section, the main aim of this study was to develop, implement and validate an intelligent controller capable of learning basic manipulative skills during the peg-in-hole assembly. The complexity of geometry, nonlinearity of the insertion task and noisy feedback signals from the sensors are the main factors making the task very difficult to deal with. Previous researches on the subject show many limitations of AI controllers for automated assembly. The training speed is an important issue. To speed up the learning process different methods of contact state classification were developed. Most of them show a good on-line performance but require *a priori* knowledge about the environment.

In contrast to other researches, different information sources will be analysed and combined to develop a fully unsupervised, intelligent controller. In the author's design, no class labelling or geometry feature pre-training takes place. First, the controller uses a self organising ANN to learn the geometry of the parts. In the next step an unsupervised decision about the appropriate action is taken. The only descriptive information feed to the system is the assembly direction.

According to the author's best knowledge, the "sandwich" structure of the controller and the choice of algorithms are unique and have not been investigated before. The proposed design and selection of information sources proved to be able to cope in an environment with high level of uncertainty without the loss of on-line performance.

The improvement in system design is also an important part of this research. The modular form of the software and robust method of robot-computer connection were developed to fulfil that requirement.

1.3 Thesis structure

Research presented in this thesis is organised in the following manner:

Chapter 1: In the introduction the reasons for research on the application of Artificial Intelligence to robot control are presented. Some statistical data showing that the number of robotics units applied in industry is still increasing was included. The different methods of robot control for automated assembly were described followed by an overview of applied AI methods. The chapter was concluded by defining the aims and objectives for this research project.

Chapter 2: In this chapter the previous research on the intelligent robotic assembly is presented. This also includes the definitions of the admittance mapping and both linear and non-linear compliances. The last section embodies a discussion of the advantages and disadvantages of the presented methods. The previous research presentation was an inspiration to build the improved version of the controller.

Chapter 3: This part of the thesis embodies the detailed description of design of the system. It begins from the presentation of each element of the setup. First the Puma 560 robotics arm is described. Then the implemented methods of robot-computer interconnection and the communication protocols are analysed and compared. The F/T sensor and its Linux driver design are then introduced to the reader. The last section includes the overview of the experimental specimens.

Chapter 4: Here the applied methodology is presented. This includes the description of the controller's structure and utilised algorithms. In the last section the results from test simulations on a grid-world maze are presented.

Chapter 5: In this chapter the methodology and plan of experimental validation is included. The analysis of environment's uncertainty is also presented here.

Chapter 6: This chapter consists of the summary of empirical results. This includes the results from the real-life peg-and-hole insertions. Different geometries and algorithms are analysed and comment.

Chapter 7: Discussion and conclusions are the main part of this chapter. It concludes by the author's suggestions for future work to further improve the controller's performance.

The results in the form of tables and graphs are included in the thesis' appendices. Each main part of the thesis is preceded by a short introduction describing its content. The discussion and author's comments are included in the last section of each chapter.

Chapter 2

Intelligent motion control

In this chapter the previous research on intelligent robotic assembly will be presented. This also includes the definitions of the admittance mapping and both linear and non-linear compliances. The last section embodies a discussion of the advantages and disadvantages of the presented methods. According to the author's best knowledge the research results enclosed in this chapter represent the state of the art of the intelligent assembly process. The presented work was an inspiration to build the improved version of the controller.

2.1 Admittance mapping

Admittance (A) is the ratio defined as follows:

$$\vec{v} = A(\vec{f}) \tag{2.1}$$

It describes the relationship between forces ($\vec{f}$) acting on the end-effector and resulting velocity ($\vec{v}$). Thus the admittance matrix could be used to correct the velocities based on acting forces.

Asada in his paper [1] considered both linear and non-linear admittances. First the box palletizing task was analysed as an example of linear compliance. The aim was to place the rectangular box into a corner of the large box. As shown on the picture (see

Figure 2.1), for each path the reacting forces and so desired actions are different. In that case the mapping is assumed to be linear and the problem could be easily solved using conventional control methods.

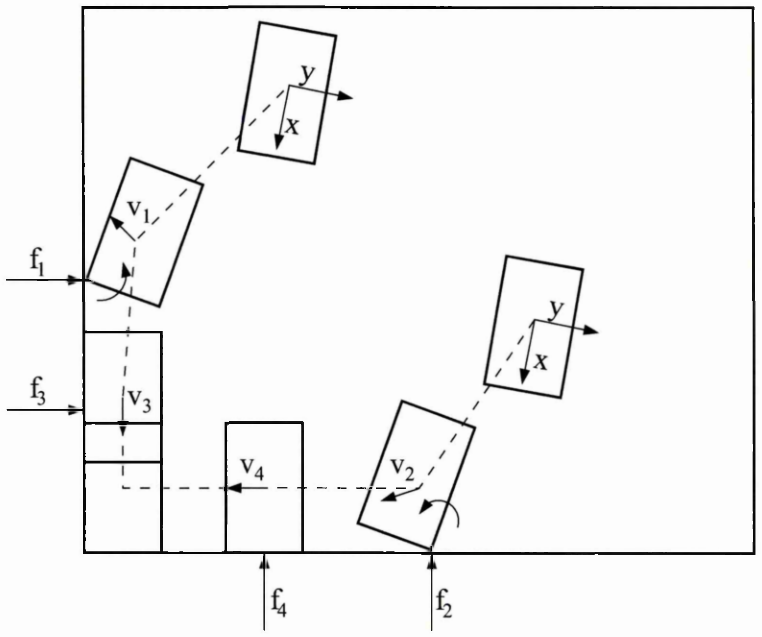


Figure 2.1: Box palletizing task (after Asada [1]).

As the assembly task becomes more complicated the force-velocity data is no longer linearly mappable. In that case applying a linear compliance unavoidably causes the error. Asada [1,2] used a chamfer-less peg-in-hole insertion task to illustrate this problem.

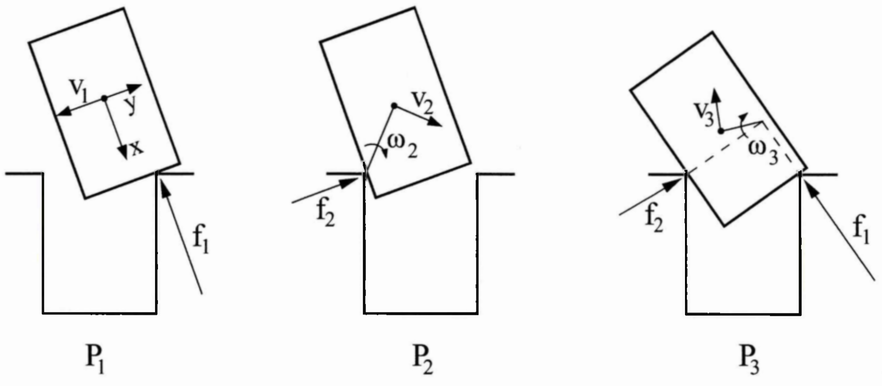


Figure 2.2: Contact states between peg and hole (after Asada [1,2]).

In the case of contact state "P_3" the force and torque vector $\vec{f_3}$ is formed by a vectorial sum of $\vec{f_1}$ and $\vec{f_2}$ from the cases "P_1" and "P_2" (see Figure 2.2).

$$\vec{f_3} = \vec{f_1} + \vec{f_2} \tag{2.2}$$

Assuming linear compliance and so to satisfy equation (2.1) the resulting velocity vector $\vec{v}_3$ should also be a vectorial sum of vectors $\vec{v}_1$ and $\vec{v}_2$. However, this is not the case. Analysing the velocities (Figure 2.2) we can write:

$$\vec{v}_1 + \vec{v}_2 = [v_{x2}, v_{y2} - v_{y1}, -\omega_2]^T \neq \vec{v}_3 \tag{2.3}$$

This indicates that in the case of the chamfer-less peg-in-hole insertion the force to velocity mapping is against the principle of superposition which is the one of most fundamental properties of linear mapping. Based on that this task should be considered nonlinear.

2.2 Nonlinear compliance control with ANNs

Compliance control is a control strategy for correcting a planned motion based on the force measured in the process [1]. There were two main methodologies in previous studies on the subject, the logical branching [3, 15], and the feedback gain method [21, 41].

In the logical branching approach the corrections of motions are described by IF-THEN-ELSE expressions. The appropriate actions are selected in accordance with the sensor state. The feedback gain method represents the strategy for correcting the motions in terms of feedback gains that relate the measured force to the change in displacements and velocities [1].

The logical branching method is relatively slow and actions are intermittent because the controller has to perform a set of discrete operations including reading and evaluating the sensor signal, logical branching to selected actions followed by reexamination of the sensor state. The advantage of this approach over the feedback gain method is that it allows the representation of highly nonlinear strategies of complex assembly. The feedback gain method, on the other hand, is founded by servo control and provides smooth, continuous actions. Due to the nature of feedback gains which are linear and invariant the method cannot be use to sufficiently describe task complexity and control strategy. That representation is required to perform nonlinear control actions and logical decision making. To overcome these problems some research has been done on geometric models of the environment [5–7, 12, 13] and complex task analysis [24]. A different approach, proposed by Asada, is to teach the compliance without relying on the part's geometry description and analysis of the complex assembly process.

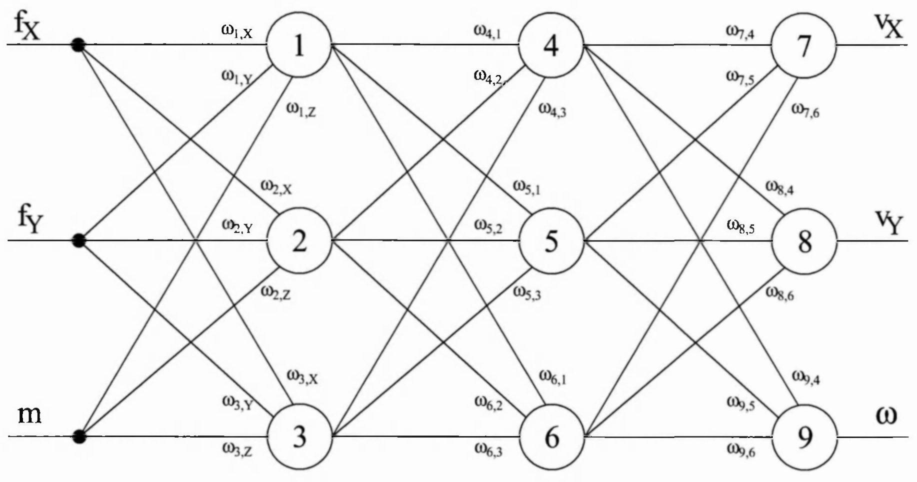


Figure 2.3: ANN structure for 2D chamfer-less insertion problem (after Asada [1]).

In his research Asada [1,2,28] proposed a new way of solving the nonlinear compliant motion control problem. He developed a method of generating control law through the measurement of an operator's motion in which the force was monitored along with the position of the end-effector. He also proposed a method of teaching the nonlinear compliance strategy. The peg-in-hole insertion problem was analysed using supervised ANN. The ANN learns not only linear compliance in terms of stiffness and damping matrices but also nonlinear compliance. Asada's strategy is based on guiding the peg by making contacts with referencing surfaces. In the most general case, where the peg's motion is considered in all of its 6 DOF, Asada's ANN uses 3 layers of neurons, featuring 6 inputs in the form of contact information and 6 outputs - matching velocities. The first layer is designed to detect the occurrence of contact point at the edge and surface forming the hole. The second layer in Asada's ANN architecture consist of units that each indicate the occurrence of a specific contact state. The third layer performs the mapping from the output of the second one onto the velocities. Despite uncertainties in the environment using this approach the peg can be led to the hole. This method is capable of performing nonlinear operations allowing the system fast smooth actions.

According to Asada the basic requirements for valid peg velocities are:

- conformity to the geometric constraints due to the contacts,
- maintaining the contacts with the referencing surface,
- guidance towards the hole.

Asada illustrates the ability of the trained feed forward network to discriminate among the contact states and to provide appropriate velocity commands in a series of simulated experiments involving two dimensional peg-in-hole insertions.

Howarth [24] developed a framework for task level programming assembly systems in which the realisation of self learning manipulative skills plays a key role in ensuring autonomous operation. The approach to the control of peg-in-hole insertions is presented based on a multi-layer perceptron (MLP) network trained with error back-propagation. The applicability of the technique to the assembly problem in 4 DOF is first successfully evaluated by training the network using supervised learning; this required the prior establishment of desired input-output pairs. The proposed ANN controller's layout consists of a three layered MLP with 5 inputs including three forces acting on the peg F_x, F_y, F_z, the torque around the peg's axis M_x and the required direction of insertion a. The output layer features 4 nodes representing velocity commands for linear motions along the peg axes (δ_x, δ_y, δ_z) and rotation around the direction of insertion ($\delta\theta z$).

Lopez-Juarez [29–31] proposed an intelligent assembly controller based on the Fuzzy ART-MAP network to control peg-in-hole insertions in 6 DOF. The Fuzzy ART-MAP algorithm represents one of the many varieties of the Adaptive Resonance Theory (ART) model initially developed by Carpenter and Grossberg [9,17], and shares with the original model the ability to produce stable and fast classification of input patterns. The structure of the network consists of two FuzzyART modules that simultaneously process patterns deriving from two separate families a and b. The two modules operate the unsupervised classification of two input patterns according to the ART mechanics, grouping them into a finite number of categories. Unlike the original ART algorithm, the FuzzyART module has been designed for handling real valued patterns. The ART modules are then linked by an associative learning network (the inter-ART module or MAP field) which learns the association between the patterns of each family, therefore producing an active mapping between heterogeneous sources of information.

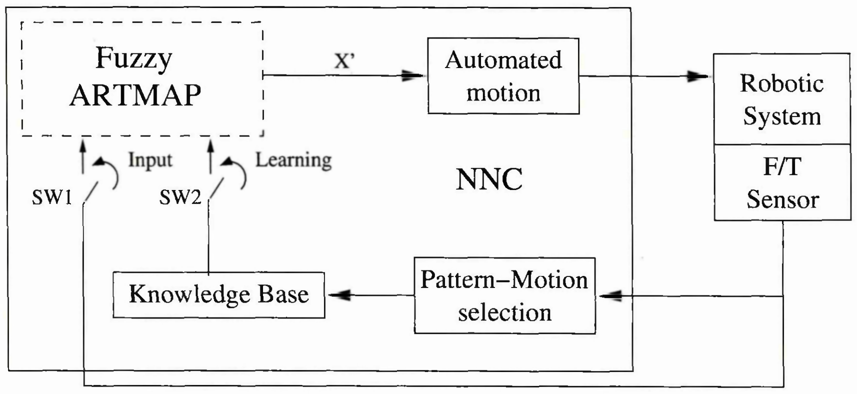


Figure 2.4: Controller structure (after Lopez-Juarez [31]).

The proposed controller consists of two subsystems: adaptation and decision modules (see Figure 2.4). The adaptation stage has an initial section where the F/T signal pre-processing takes place. The vectors are normalised to produce n-dimension input vector. The next section performs the classification of F/T signal. Learning is supervised, as a training set containing pairs of examples is required for the formation of the classification categories in each ART module and the MAP field.

Lopez-Juarez and Howarth [30] proposed the application of this architecture to force guided assembly of circular, square and semi-square pegs in 6 DOF considering both chamfered and chamfer-less matching holes. Because learning is supervised, *a priori* knowledge is provided to the network during the training stage in the form of Primitive Knowledge Base (PKB). The PKB consists of an ordered list of 6-dimensional force and torque patterns paired with appropriate motion commands in the form of a symbolic variable indicating an incremental step motion. A total of 12 motion commands are considered: positive/negative linear motions along the X, Y, Z axes of the peg and positive/negative rotations about the same axes. The associations in the training set are predefined and designed to achieve an appropriate motion strategy in response to the F/T vectors generated by the peg contacting the environment. Generally speaking, the induced behaviour is aimed at the minimisation of such contact forces. During the operation, when contact occurs, the F/T pattern x_a is presented to the ART$_a$ module for recognition; the closest matching a category is associated through the MAP field to a b category which in turn points to a robot action as previously learnt.

2.3 Learning contact states with ANNs

In more complex, real world situations which involve motions in contact, the analytical model of environment may be very difficult to construct. Errors are typically caused by uncertainty of the position and orientation of components and noisy feed back signals. In these situations the main problem is to relate the complex force and torque signals to corresponding contact states.

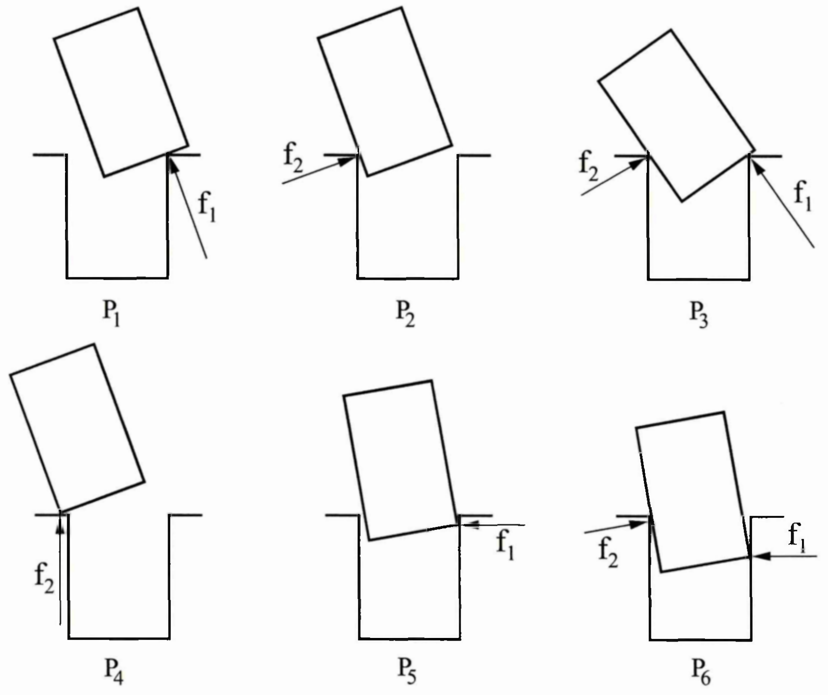


Figure 2.5: Contact states between peg and hole (after Cervera et al [13]).

Cervera et al [10, 11, 13] proposed an unsupervised scheme to deal with uncertainty. He investigated a real application of a flexible manufacturing system. The cell consists a matching centre which works with several types of tools. The tools were picked from the robot vehicle by a robotic arm. The aim was to use a force and torque sensor to detect error conditions which led to incorrect insertions.

Cervera's 2-dimensional peg-in-hole model includes the presence of clearance between parts and extended Asada's design by three additional contact states (see Figure 2.5). The constraint equation set was used to define a partition of the configuration space (2D in this case). The friction forces between matching parts were omitted. To calculate the complex relationship between force magnitude and contact states the Self Organising Map (SOM) ANN was applied. The main advantage of this approach lies in its ability for unsupervised learning (self reorganising). Due to efficiency and topological advantages the Kohonen's algorithm was used. The difference from other neural networks models is that after learning SOM's responses are arranged in the map. The map's reorganisation takes place automatically without external supervision. It is based only on internal relations in the structure of the input signal.

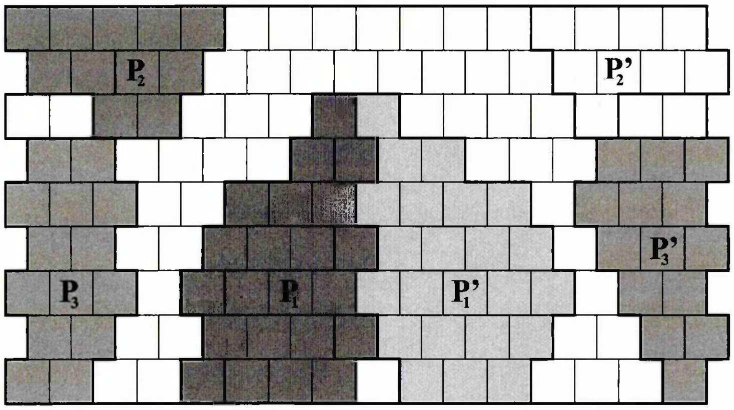


Figure 2.6: Activation patterns for the three contact states case (after Cervera et al [13]).

First, for each contact state, the known pattern was introduced to the network. Then the label was assigned to the resulting cluster of cells that were activated for this particular state (see Figure 2.6). Once the map was labelled for all known states the network was considered trained and unknown sensor signal could be applied and successfully classified. The resulting label of activated cells provides a description of the contact state corresponding to the given input. It is clear from the Figure 2.6 that SOM was able to successfully classify the three states from Asada's model (P_1, P_2, P_3) and their symmetric ones (P'_1, P'_2, P'_3).

In Cervera's model the ANN's input signal was built of two force signal and one torque value. The simulations showed that due to high similarity in reacting forces the network was not able to distinguish between some of the states (class overlapping). To overcome that, another source of information was added to the system - the orientation of the peg.

The model was successfully extended to 3-dimensional in a described early application of a flexible manufacturing system. In that case the ANN's input signal was built of three force and three torque signals.

More recently, Brignone et al [5–7] have developed a novel approach based on the interpretation of the contact conditions using a FuzzyART unsupervised classifier to merge on-line force information with a description of the components geometry. In his work Brignone combined component's geometry with the fast learning of Lopez's algorithm.

The contact between peg and hole, can be modelled with the Newton equations for the rigid body. Furthermore representing the insertion as a quasi static transition of states greatly simplifies the relations removing them from the time domain.

These observations are at the foundation of Brignone's proposed insertion architecture. Linear compliance can be achieved by matching the Euclidean normalised force signal with the orientation of the surfaces forming the hole (localisation of contact) and then selecting a motion that satisfies the geometric constraints.

2.4 Admittance mapping with Reinforcement Learning

In a complex environment it is very difficult to learn an effective control strategy. Amongst the other methods the reinforcement learning algorithms have been applied to learn non-linear control law. The controller improves its performance by repeatedly interacting with the environment and so it is able to learn the appropriate admittance mapping on-line. The principles of these methods will be described later in this thesis. In this section the author would like to focus on the application of different reinforcement learning algorithms to the automated robotic assembly.

Yang and Asada [46] proposed a novel approach for a robot to automatically generate compliance control gains through practice based on adaptive reinforcement learning algorithms. In this method, the robot repeatedly attempts to perform the task with initial simple control knowledge, and gradually improves the performance through the trials. To demonstrate the validity this method was applied to learn damping control parameters in a simple ball-aligning task. The objective was to align the ball to the corner with the minimum reaction forces from the walls. At the same time the controller was required to follow the nominal trajectory until the ball made contact with a wall. Based on these assumptions Yang and Asada defined a performance index, called reinforcement.

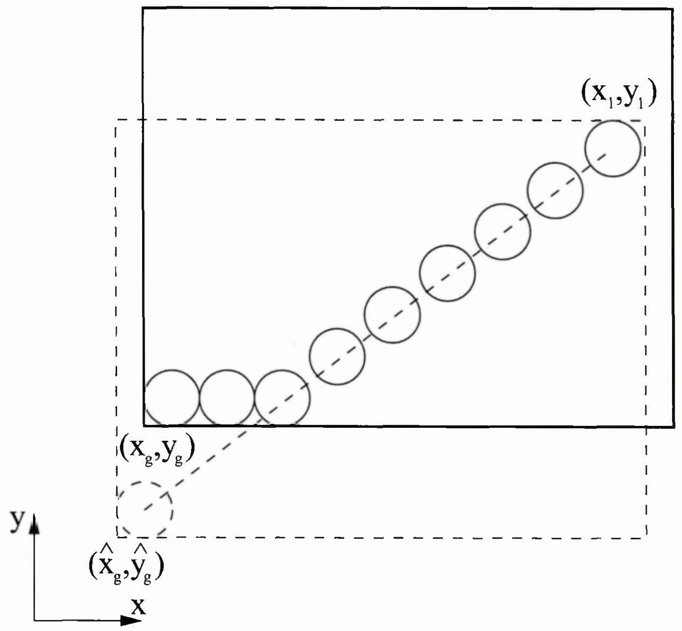


Figure 2.7: Ball-aligning task (after Yang and Asada [46]).

The objective of learning was to find the optimal compliance control law, which maximises the reinforcement, from a series of trials.

Gullapalli et al in their research [18–20] developed an ANN controller based on backpropagation units, whose outputs are Stochastic Reinforcement Value (SRV) learning units.

SRV computes its output (o) at time (t) as function of his activation (a):

$$o_t = a(\mu_t, \sigma_t) \tag{2.4}$$

The activation is a random number extracted from a Gaussian probability distribution. Learning takes place by adjusting the mean $\mu_t = \theta_t^T x_t$, and the standard deviation $\sigma_t =$

$s(\hat{r}_t)$, where $\hat{r}_t = \phi_t^T x_t$, and x_t is an input vector. The function $s()$ is a monotonically decreasing, nonnegative function of $\hat{r}_t$ with $s(1.0) = 0.0$.

The reinforcement $r(o_t, x_t)$ is used to adjust future outputs by updating the internal parameter vectors:

$$\theta_{t+1} = \theta_t + \alpha (r_t - \hat{r}_t) \left(\frac{o_t - \mu_t}{\sigma_t} \right) x_t \tag{2.5}$$

and:

$$\phi_{t+1} = \phi_t + \beta (r_t - \hat{r}_t) x_t \tag{2.6}$$

where α and β are the learning parameters.

Gullapalli used a Zebra robot and performed a number of successful peg-in-hole insertions. The outputs of the neural controller were the desired motion values. The feedback (F/T signal) to the controller was supplied by a sensor mounted on the robot's arm. In this approach the location of the hole needs to be known since it is used to compute position error required to calculate the reinforcement signal. Results show that the algorithm performs well despite uncertainty in the peg location and noise of sensor readings.

Cervera and del Pobil in their paper [12] proposed a different method for selecting the actions and achieve the goal in the minimum number of steps. The algorithm learns the relationships between sensed states and actions. The controller consists of three main components (see Figure 2.8). The guarded motion subsystem is responsible for stopping the movement when the force value exceeds the fixed threshold. The compliance motion takes over after the contact is achieved. Initially random movements are performed but the system gradually learns the state-actions relationships. During this stage the motion is restricted to the surface. The q-learning algorithm was applied to learn the best action for a given state.

The state is built from two sensory signals: position and torque. Position and orientation were obtained from the robot joint's servo encoders. The torque signal was acquired by the sensor mounted on manipulator's wrist. Kohonen's SOM were used for extracting the feature information. Three torque signals were used as an input to the network. According to authors due to the strong correlation between torques and forces the latter ones will not add new information to the system.

They investigated insertions of three different peg shapes: with circular, square and triangular cross-sections. The components were made from wood (peg) and synthetic resin (the platform with holes). In the case of circular pegs the SOM had 24 units implemented. The network was trained off-line with 70,000 data vectors from previous trials. The state description was built combining the winner from the map and relative position of the peg with respect to its initial location. The total number of states was 216. The action domain consisted of eight fixed step translations in different directions of the XY-planes. Due to complexity of geometry the number of states for the square peg increased and was set to 648. The network structure and learning process was similar to the one used in the cylindrical peg case. Two new actions in form of rotations around the normal axis to the surface were added to the system. The measurement of change in angle was taken into consideration during the agent's orientation description. The same state representation and actions were applied to the triangular peg insertion problem.

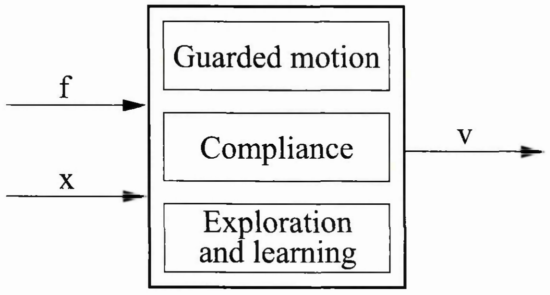


Figure 2.8: Controller architecture (after Cervera and del Pobil [12]).

The algorithm was tested on real 3D insertion tasks. The results show the ability of the system to learn to insert cylindrical and non-cylindrical components. The proposed system exhibited good generalisation capabilities for different geometries and locations of assembled parts.

As described earlier, Howarth [24] developed the method of interpretation of sensory data to minimise the contact forces and guide the peg towards the assembly goal. The major disadvantage of this technique is the fact that MLP network was trained with the supervised learning algorithm. This requires an input-output training set to be determined in advance. To overcome this limitation a reinforcement learning algorithm was implemented [24]. This allowed the ANN to learn state-action mapping without *a priori* knowledge of the environment. The system was tested and validated using an IBM 7547 SCARA robot during circular and square peg-in-hole insertions. The minimal task description was applied to the controller namely assembly direction and force data. Presented results [24] show that reinforcement learning of an assembly operation can be successfully learnt by a MLP network.

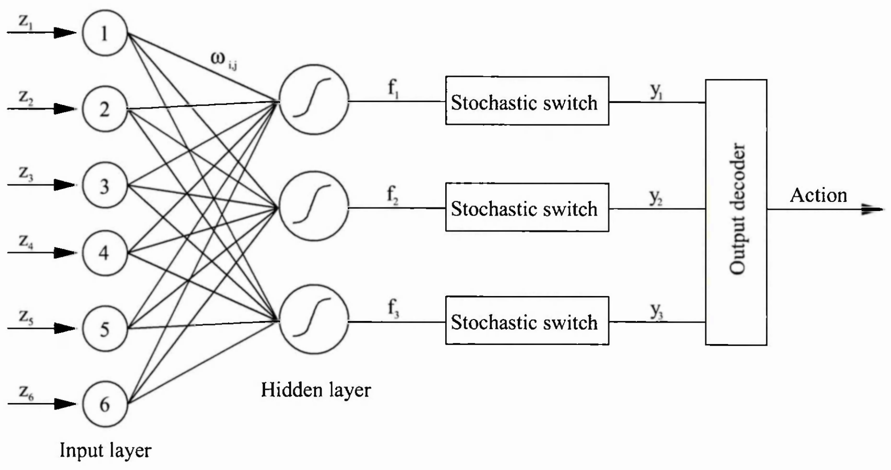


Figure 2.9: Learning agent architecture (after Brignone [4]).

Later, Brignone in his research [4] proposed an unsupervised learning method to correct angular misalignments between peg and the hole (see Figure 2.9). The controller (called by the author ART-R) consists of FuzzyART module which performs a state classification and additional output layer of reinforcement units for computing the state-action association. The symbolic representation of the environment produced by ANN was propagated to the hidden layer through a matrix of connective weights. The activation function of each node is computed in a standard manner as a weighted sum of input nodes. The selection of the output is determined by stochastic switch architecture. This forms a binary vector which represents the appropriate action. The learning algorithm used by Brignone is an implementation of the theory of Statistical Gradient Ascent performed by Reinforcement Algorithm.

The proposed method was applied and tested on a Puma 760 robotic arm. According to the author [4], the controller was able to learn autonomously what rotation to apply to the peg in order to minimise the torque load.

A large number of movements need to be performed to allow the controller to develop the optimal state-action policy (interactive learning). To speed up this process a training rig was designed. The controller performed a random rotation around of any of the three axes. This caused a torque load which was later minimised by the number of correction movements suggested by the learning agent.

2.5 Discussion

In this chapter the previous research on the intelligent robotic assembly is presented. According to the author's best knowledge, the described controllers represent the state of the art of modern automated assembly process. The systems differ in choice of the algorithms, methodologies and implementation but they share the same principles. The main aim is to develop a successful mapping between action and state domains. The learning speed is an important issue as well as choice of information feed to the system.

Asada et al [1, 2, 21, 28, 46] laid the foundation for the research on the intelligent assembly process. As stated in the chapter above he analysed and defined the linear and non-linear compliances. His 2-dimensional controller, although simple, was a major break through in research on intelligent peg-in-hole insertion tasks. The main disadvantages of proposed design was its slowness and supervised method of learning. This involves the need for *a priori* knowledge about the environment.

The researchers tried to overcome some of the limitations present in Asada's AI controller. Howarth [24, 29] used the supervised ANN algorithm. However his controller was implemented, tested and validated on a real life SCARA robot. His application of task level programming to the insertion problem in combination with the neural controller proved to be effective and able to learn the insertion task. The the high level of supervision and relatively slow learning algorithm are the main limitations.

Cervera et al [10–13] and later Brignone et al [4–7] used a geometrical approach to locate the contact states. Both methods proved to be fast and reliable but again a high level of supervision was needed. In Cervera's design, despite the use of self organising learning methods, the class labelling takes part. This involves an additional, "expert's" information supplied to the system. Similarly Brignone's controller, although using an unsupervised ANN algorithm had to be trained off-line before being applied to the real system.

The other major problem in automated assembly tasks is the automated motion generation. Different approaches were encountered in the literature. First Asada developed a frame work for the application of reinforcement learning to robot control [46]. Gullapalli et al [18–20] applied his stochastic reinforcement method on a real life robot. The learning speed was slow but it is one of the few fully unsupervised designs. Its limitation lies in a need for the peg's position and orientation signal to be acquired from the manipulator's joint encoders. According to the author this information does not aid the implementation of of an autonomous system and is not necessary to successfully accomplish the insertion task. Howarth [24] also used the reinforcement learning algorithm for automated motion generation purposes. To speed up the learning time he used data from off-line simulations.

To correct the angular misalignment between mating parts Brignone [4] applied a stochastic reinforcement algorithm to analyse torque patterns. However the experimental validation was not well documented, the controller showed the ability to correct the insertion path using rotations and accomplish an insertion task. The limitation of the design is the need for off-line pre-training using the designated test rig.

Cervera and del Pobil in their recent research [12] also applied a reinforcement learning algorithm for automated motion generation. Their controller design shares a lot of similarities with the "sandwich" structure proposed in this thesis. The author wants to emphasise the fact that both researches were being undertaken in parallel and independently. It is clear from work described in this chapter, that researchers investigating the automated assembly tasks using a peg-in-hole model face a number of complicated problems. Generally they could be divided into two sections namely:

- State recognition problem - to increase the speed of learning the complex 3-dimensional state space needs to be clustered and classified. A fast, reliable algorithm should be applied for that purpose.

- Automated motion generation problem - here the logical decision about the next action is taken. The algorithm with the ability to reorganise quickly and obtain knowledge without supervision would be the ideal solution.

Most recent designs [4, 12, 24, 30] tried to combine the different methods. In all cases presented a significant amount of supervision had to be applied in the form of off-line pre-training or class labelling. This involves the need for a supervisor with an initial knowledge of the environment. It should be stated clearly, however that the off-line training is generally aimed at speeding up a learning process, it does not limit the controller's future performance. The controller designs, presented early in this chapter, proved to be capable of gaining the necessary knowledge autonomously.

To save on development time the usage of virtual peg-in-hole simulation could be considered. The 3-dimensional model could be analysed with contact forces derived using Final Elements Method (FEM). This would help to focus purely on testing different AI methods without the need for time consuming implementation of robot-computer communication protocols. The fully tested and evacuated in "synthesised" word algorithms should be applied on real life application for further investigation.

The main aim of the work described in this thesis is to design an intelligent controller with minimal interference from the supervisor during the learning process. The author claims that using modern AI algorithms the off-line pre-training process can be omitted without the loss of on-line performance of the system.

Chapter 3

System design

In this chapter each major part of the experimental hardware will be presented. It begins from the presentation of the Puma 560 robotics arm. This includes its specification and parameters. Then the method of implementing the robot-computer interconnection and the communication protocols are analysed and compared. Later, the F/T sensor and its Linux driver design is introduced to the reader. The last section embodies the overview of the experimental specimens.

The whole system was designed and built by the author. Some improvement to other researchers [5,31] were applied. This includes the elimination of master-slave architecture replacing it with a robust terminal emulation connection method.

The robotic cell was tested, and proved its reliability during extensive experiments involving iterative peg-in-hole assembly simulation.

3.1 Experimental setup

The robotic arm and controller are the main parts of the system. The *supervisory* computer is connected to the controller via a serial port. The sensor was mounted on the robot's wrist (see Figure 3.1). The F/T signal is transmitted to the computer and then used as an input to the AI controller. The new arm position and orientation values are calculated. The incremental motion request is sent to the controller using *supervisory* or *terminal* mode.

The proposed design guarantees a stable communication and fast, bi-directional information flow between the devices. These characteristics are very important since the experimental setup is expected to work continuously for several days undertaking numerous peg-in-hole insertions.

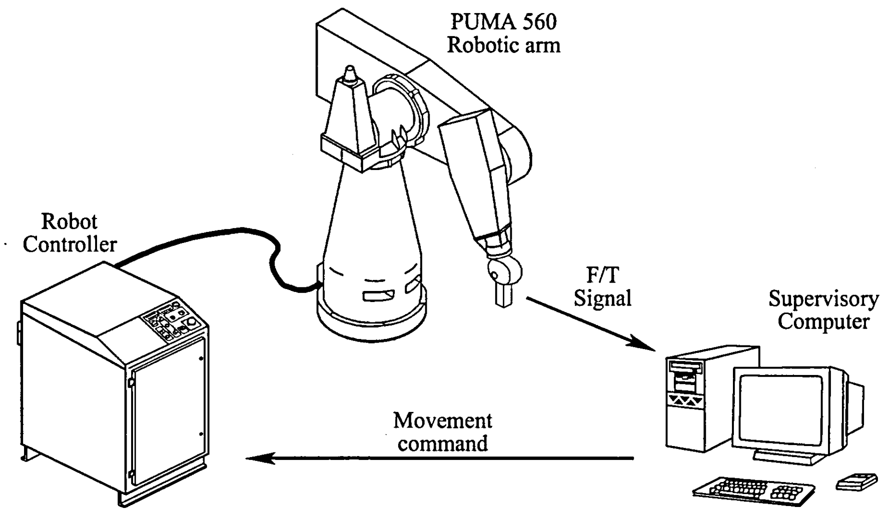


Figure 3.1: Experimental setup.

In the sections below each part of the system is described. The technical issues and problems are also presented in the Appendix E.

3.2 The robot

The Unimate Puma Mark III series 500 are amongst the most popular robots for educational purposes. The flexibility, repeatability and easy programing make them first choice manipulators for assembly tasks. The 500 series includes two models: the 5-axis Puma 552, and the 6-axis Puma 562. Both models could be set up for either a 4 or 2.5 kg payload based on performance capability. The Puma 562 was used for the purpose of this project. It consists of the manipulator arm and control unit.

3.2.1 Robotic arm

The arm is a serial, kinematic chain of components connected to each other at 1 DOF revolute joints (see Figure 3.2). Its design is similar to the human arm and consists of seven rigid bodies namely: Trunk, Shoulder, Upper Arm, Forearm, and Wrist (built of three elements) and Mechanical Components (gripper). The Shoulder rotates about the Trunk's centre, vertical axis. The Upper and Fore Arms rotate about two horizontal axes (arm joint and the elbow). The Wrist consists of the remaining three rigid bodies.

The arm members are driven by a set of permanent magnet, DC servo motors and drive trains. The brakes are built in the servomotors. They were designed to support and lock the arm in a fixed position. The arm is also equipped with a dynamic braking system which slows down the motors in an emergency condition. In that condition the power is disconnected from the motor and its terminal is bridged with a resistor to dissipate the energy.

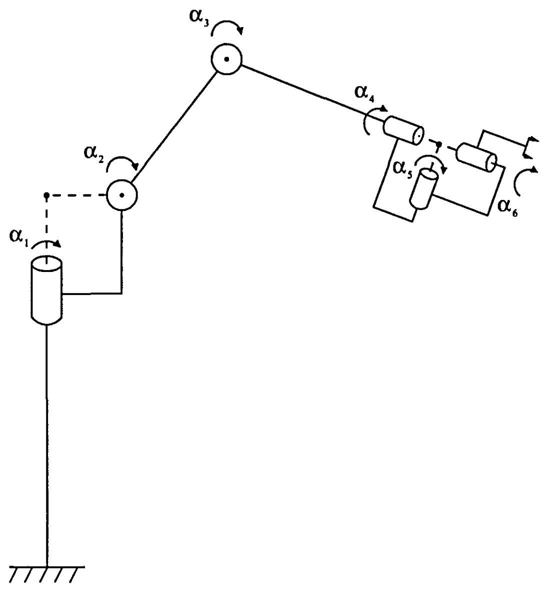


Figure 3.2: Schema of Puma 562 robotic arm.

The arm position and orientation could be controlled in two different modes namely WORLD and TOOL. The names correspond to the coordinate systems used as references and are defined as follows:

- WORLD mode - where the reference of coordinates are fixed in the robot arm base. When the WORLD mode is chosen the gripper is to move parallel to, and rotate about, those axes.

- TOOL mode - where the reference coordinates are fixed in the gripper. In this mode the gripper is to move parallel to any of TOOL axis and rotate about those axes.

The Orientation (O), Altitude (A) and Tool (T) are the angles formed by the TOOL coordinate system axes related to WORLD coordinates. The angles are defined as follows (after the robot manual [40]):

- O - a measurement of the angle formed between the WORLD X axis and a projection of the TOOL Z on the WORLD XY plane.
- A - a measurement of the angle formed between the TOOL Z and a plane parallel to the WORLD XY plane.
- T - a measurement of the angle formed between the TOOL Y and a plane parallel to the WORLD XY plane.

Item	Specifications
Position Repeatability	0.004 in. (0.1 mm)
Payload	8.8 lbs. (4.0 kg) or 5.5 lbs. (2.5 kg)
Straight Line Velocity	19.2 in./sec. max (0.5 m/sec. max)
I/O Capacity	32/32
Arm Weight	140 lbs. (63.0 kg)
Power Requirements	110/208/240 VAC, 1500 W
Ambient Operating Temperature	50 - 120F (10 - 50C)
Number of Axes	6
Drive	Electric DC servos
Gripper Control	4-way pneumatic solenoid

Table 3.1: Puma 562 arm specification (after robot manual [40]).

The output from the AI controller is in form of δ_x, δ_y, δ_z values, which represent movement correction and 3 values $\delta\theta_x$, $\delta\theta_y$, $\delta\theta_z$ for rotation correction. Since the definition of these differ from the definition of O, A, T angles, the transformation between these values had to be found. An appropriate algorithm was developed and implemented (see Appendix C) but after close investigation it became clear that the angles were the components of Euler Z-Y-Z orientation description. The appropriate software was written to calculate the constant rotation increment value.

To request an incremental movement the `DO MOVE HERE:TRANS(X,Y,Z,O,A,T)` command was sent to VAL II. The X, Y, Z values represent the increment on each axis respectively and O, A, T represent the components of Euler Z-Y-Z orientation description.

Some of specifications for the Puma 562 robotic arm are given in Table 3.1.

3.2.2 Control unit

The main part of the system is the arm's control module. The system uses an LSI-11/73 as a central processing unit and communicates with individual joint processors for robot arm motions. All the information, including the signals from position encoders pass through the module. It performs a real-time calculation for position and velocity control using the incremental optical encoder signals. During power up the calibration of the arm needs to be performed. Using coarse potentiometer reading the exact position of high precision encoder's index mark is recorded.

The movement commands are interpreted by a software present in the memory. The motors are driven according to the control law and required trajectory. The movement is monitored by an incremental encoders signal which forms the closed-loop control system. The basic specifications of the Puma 562 control system are presented in the Table 3.2.

Item	Specifications
Controller	System computer
Central Processing Unit	LSI-II/73
Teaching Method	Teach Pendant, computer terminal or intelligent external controller
Programing Language	VAL II
External Program Storage	Double density floppy disk
VAL software storage memory	45 k words
User memory	19 k words (expandable)

Table 3.2: Puma 562 control unit specification (after robot manual [40]).

The control module used for the purpose of this research was equipped with a set of control cards namely:

- LS-11/73 Central Processing Unit (CPU) board - the core of the system which process all programming instructions.
- 64 k words CMOS Random Access Memory board - provides a storage for user software.
- DL-11/J Quad Serial Boards - an interface that links the CPU to the external systems like terminal, Teach Pendant, SUPERVISOR or ALTER communication standards.
- I/O board - provides an interface for 32 inputs and 32 output for communication with CPU.
- "A" and "B" interface boards - which provide respectively a parallel communication link between the servo boards and connects the control section of control card set to each digital servo.
- six digital servo boards - each equipped with separate microprocessor to calculate and send the analog drive signals for DC motors.

Variable Assembly Language (VAL) was developed in Unimation Inc. as language for industrial manipulators. It is permanently stored as a part of the robot's operating system. VAL-II combines a sophisticated, easy-to-use programming capability with advanced servo control methods. It provides a full set of instructions for software development including editing and file-system maintenance facilities.

3.3 Robot-computer connection

The first important task during the design of the connection was to decide which transmission protocol to use. A number different approaches have been implemented and tested by other researchers.

The ideal system would involve only one *supervisory* computer with intelligent agent and sensor driver installed. This PC should also send the incremental movement commands to the robot controller. Thus a decision about the transmission type and communication protocol had to be made.

The considered solutions included:

- ALTER Mode - used by Lopez et al during their research [29–31]. This proved to be complicated solution and require an additional *slave* computer to maintain connection.
- Teach pendant mode simulation - used by Brignone et al [5–7]. This solution proved to work well but for similar reasons to the case of ALTER mode, a *slave* computer needs to be added to the system to help maintain the connection.
- Supervisory Computer System (SCS) mode. The main disadvantage of this solution lies in the complexity of protocol and data exchange rules. Because of existence of DDCMP frames it increases time overhead in the serial communication.
- Terminal emulation. This solution involves disconnecting the robot CRT terminal and connecting PC via serial port instead.

The last two modes were implemented and tested by the author. The Supervisory Computer System (SCS) offers the capability of communicating with an external computer using rigorous communication protocol. This assures the integrity of information transferred between the VAL II system and the external computer.

Various levels of communication had to be implemented. Communications systems are customarily described in terms of functional layers (see Table 3.3). Each layer in this model supplies a specific function to the communication message as it transmits the system (the ISO model is included for reference only).

The ISO model assumes the computer communication network support is modularised into functional units corresponding to each layer. Each support module is responsible for providing the specified network services to the module above it.

The supervisory communications system in VAL II is comprised of four layers. These layers correspond roughly to ISO Application/Presentation, ISO Session/Transport, ISO Network/Datalink, and ISO Physical layers respectively.

To establish a stable connection with the robot controller the RS-232, DDCMP, and data exchange with VAL II Network Manager and VAL II Logical Units had to be implemented. Therefore, only those will be described below. Detailed information about other communication layers can be found in the robot manual [40].

ISO reference model	VAL II functional layers
Communication User	VAL II User Program
Application	VAL II Logical Units
Presentation	
Session	VAL II Network Manager
Transport	
Network	DDCMP
Data Link	
Physical	RS-232C

Table 3.3: Communication layers (after robot manual [40]).

3.3.1 Physical link - RS-232C

The physical communication link for VAL II supervisory communication is an EIA RS-423 serial link. This is compatible with common EIA RS-232C standard. The main differences between the two standards are maximum data rate and electronic design. RS-423 uses lower voltages and differential signals to allow cable lengths up to about 300 meters.

The serial interface between the robot controller and the computer was made using RS-232C standard. Therefore, the RS-423 original standard in the asynchronous serial line board had to be changed. A number of resistors were removed from the board to allow single-ended voltage input. No changes had to be made to the computer since its serial port was already using RS-232C standard.

Serial link was implemented using POrtable Standard for UNIX (POSIX) terminal control functions should work, with a few modifications, under Linux, IRIX, HP-UX, SunOS, Solaris, Digital UNIX, and most other UNIX operating systems. The 'C' code was written and tested connecting the two computers via serial link.

Both serial ports `/dev/ttyS0` and `/dev/ttyS1` (COM1 and COM2 respectively) were used for the communication.

3.3.2 Link Control - DDCMP

The Link Control of the VAL II supervisory interface implements a subset of the Digital Data Communications Message Protocol (DDCMP) specification developed by the Digital Equipment Corporation (DEC). The layer of the supervisory communication protocol can accept a transmission request from the network manager while simultaneously reading a message from the communication hardware. On one side the link control layer communicates with the Network Manager layer and on the other side it deals directly with the actual hardware which is performing the communication. The most important task of the data link layer is to ensure that each message transmitted or received is 100% error free.

DDCMP provides the rules for message framing, sequencing, and acknowledgement. These rules assure message integrity. DDCMP performs these functions through the means of special data sequence which envelopes the VAL II message data.

Figure 3.3 shows the format of DDCMP frames sent and received over the communication link [40]. It includes the envelope added by DDCMP to the VAL II data message.

DDCMP requires that each transmitted frame be acknowledged. This data link level acknowledgement is completely independent of any application level acknowledgement between each logical unit and its corresponding application.

The Data Message Portion format (see Figure 3.3) is used for records transmitted by VAL II and for records transmitted by the supervisory system. Data messages sent over the communication link between VAL II and a supervisory system have the general form of messages sent between VAL II and the operator at the robot system terminal.

The only difference is that the messages sent to a supervisory computer have a routing and identification information preceding them. This allows the supervisory computer to process them.

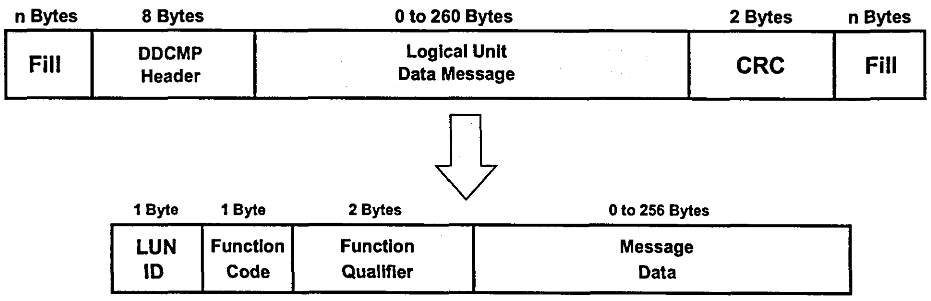


Figure 3.3: DDCMP frame types used by VAL II.

A set of tasks like connection initialisation and data exchange were also implemented. Additionally the Cyclic Redundancy Code (CRC) algorithm was implemented since DDCMP requires a CRC-16 check sum on each data portion.

The CRC algorithm performs a mathematical calculation on a block of data and returns a number that represents the content and organisation of that data. The idea is to have the CRC return a number that uniquely identifies the data.

Before tests on the real robot, the connection stability and data integrity was carefully checked using two computers. To read data from the serial port and analyse DDCMP frames the *gkermit-1.0* package was used. It offers medium-independent terminal session and file transfer.

Tested and debugged software was finally installed on the PC directly connected to the robot. The successful connection was established and tested. A more detailed description can be found in Appendix E.

3.3.3 CRT terminal mode connection

The existence of DDCMP significantly increases the amount of data to be transfered over the serial link. To overcome that the terminal emulation mode connection type was implemented and applied onto the system.

To work efficiently the emulator software has to meet certain requirements, namely:

- the VALII operating system needs to be loaded from the hard drive at the beginning of the session,
- full terminal emulation has to be implemented to allow communication with the robot's operating system,
- for external storage the floppy drive emulator needs to be designed.

Two separate pieces of software were written. For VALII communication a simple dumb terminal was designed. The program called *pterm* was successfully compiled and tested on the system. It allows the user to release VALII commands from PC without the need of the controller's CRT terminal.

Due to the controller's design the VALII operating system is stored on the floppy disk and needs to be loaded into the robot's memory during the boot up process. To simplify the design a drive emulator (*pfloppy*) was implemented and run on the PC. It works as a client responding to the robot's requests for the data. This approach gives the advantage of using the computer's hard drive to store the robot's programs. It also simplifies the overall design since both *pterm* and *pfloppy* applications are running on the same computer which offers a full replacement of the old CRT terminal and the floppy drive.

The full floppy drive protocol was implemented including basic file-system operations like file delete, copy, move or load. The standard VALII disk maintenance commands are applied and all the operations could be learnt from the robot's manual.

3.4 The sensor

The feedback signal to the *intelligent agent* was provided by Force and Torque (F/T) sensor mounted on the robot's wrist.

The JR3 sensor with internal electronic and receiver card for ISA (IBM-AT) bus was chosen since it provides force and torque data with very low noise. The sensor was pre-calibrated with maximum loads 15 lbs on X and Y axes and 30 lbs on Z axis.

It is built as a monolithic aluminium device containing analogue and digital electronic systems in the main body. Foil strain gauges sense the loads imposed on the sensor. The strain gauge signals are amplified and combined to become analogue representations of the force loads on the three axes. In the applied model the analogue data is converted to digital form by electronics contained within the main body of the sensor. The data from all six axes is sent to the receiver at a rate of 8 kHz. The data stream also includes feedback monitoring the sensor's power supply voltage and information about sensor characteristics. Sending a digital stream of data instead of analogue one has the main advantage of being able to use long cables without causing damage to the signal quality.

The JR3 receiver board architecture consists of dual-ported RAM, to which the host and JR3 can both read and write. This RAM allows the host to read data from the sensor with little overhead. It also allows the sensor to be reconfigured on the fly by writing configuration commands to the RAM.

3.4.1 JR3 sensor driver design

The manufacturer does not provide a device driver so an appropriate program in 'C' language had to be written. Two different approaches were implemented and compared. First a separate program was created to provide a communication interface between the controller and sensor circuit board. This solution was relatively easy to implement but, due to operating system security policy, the access to I/O memory ports from user space is limited. Therefore the driver and controller had to be run with computer administrator's permissions.

Later the Loadable Kernel Module (LKM) driver was developed. This more complicated approach requires a lot of development time and resources but is far more flexible and provides the end-user with a standard communication interface to the device.

Traditionally the device drivers were built as an integral part of the UNIX kernels. This "monolithic" structure proved to be very rigid and resource demanding since all the interfaces are loaded into the memory and exist throughout the operating system runtime. When a new device was added to the system the whole kernel had to be set up and later recompiled. Under Linux the kernel features can be expanded dynamically. The piece of code called a module could be added and removed during operating system runtime (without the reboot). This approach also proved to be memory efficient since LKMs can be loaded automatically when they are needed and removed from the memory after being idle for some time. It is also much easer to debug and maintain the code during the development time.

The LKMs are not only the device drivers. The kernel could also be extended by:

- file-system drivers - implements the interface for different file-systems access e.g. MS-DOS, FAT, NFS ext2, or ext3,
- network drivers - implements a set of network protocols e.g. IPX, TCP/IP,
- system calls - implements the interconnection between user space process and a kernel e.g. read, write the file,
- TTY line disciplines - implements a set of drivers for terminal devices,
- executable interpreters - implements an interface for different formats of executables run under Linux.

The architecture of the receiver board from ISA bus perspective is two 16-bit wide registers. To read data values from the DSP's address space, first the address of the desired data was written to the address port, then the data was read from the data port. Writing data was done in an analogous manner.

The receiver board was placed in the supervisory computer ISA bus. The base I/O address was set to 0x260 because the default settings were causing conflict with devices already installed. The basic features of the sensor were implemented allowing the user to:

- change the position and orientation of the sensor's X, Y, and Z axes,
- read the filtered or raw data from all 6 sensor's channels,
- read scaled or unscaled data.
- zero the offsets on each axis of the sensor.

All of these operation can be performed during application runtime but in author's implementation of the system all sensor settings are set during the controller's initialisation process.

3.5 Experimental specimens

As stated in Chapter 2 many different part geometries were tested and analysed during the peg-in-hole insertion. Asada [1] simulated the pegs with the circular cross-section and chamfer-less hole, Gullapalli [18] used square pegs. Cervera [12], Howarth [24] and Juarez [29] investigated both circular and square geometries. Brignone [4] in his research also analysed triangular pegs with chamfered holes.

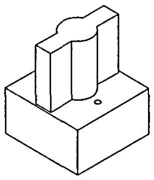
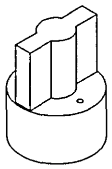
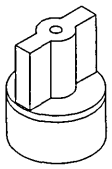
Square peg	
Clearance: 0.2 mm Length: 24.8 mm Width: 24.8 mm Height: 13.3 mm	
Cylindrical peg	
Clearance: 0.5 mm Diameter 24.8 mm Height: 14.7 mm	
Cylindrical peg with rubber end	
Clearance: 0.5 mm Diameter: 24.8 mm Height: 14.7 mm	

Table 3.4: Experimental specimens.

In this research, during the initial tests only the circular peg was used. This geometry, due to its simplicity, makes tuning and debugging of the software a much easer task. Later cylindrical peg with rubber end and square pegs were applied. The drawings and basic dimensions the reader can find in the Table 3.4. The pegs were manufactured from aluminium and featured a special flange profile to maintain better grasp quality. The robot gripper was also redesigned for this purpose.

In some cases, due to the small clearance between mating parts, the closed chamber with compressed air underneath the peg could prevent the successful insertion. To avoid the pressure build up a small vent hole was drilled through the peg.

The controller validation methodology (explained in detail in Chapter 4) requires the simulation of the insertion of parts with different friction coefficients. For that purpose, an additional circular peg was designed and manufactured. It consists of two parts, namely: the aluminium flange profiled to fit the robot gripper and the main cylindrical section made from the rubber (see Table 3.4) for details.

In Chapter 2 the difference between linear and non-linear compliances was explained. In this work the controller needs to be tested and validated on both insertion cases. To simulate linear and non-linear compliance chamfered (45^o) and chamfer-less holes were used respectively. All the hole parts are manufactured from the aluminium blocks. They are firmly screwed into the robot's working table to prevent sliding during the insertion process.

3.6 Discussion

The hardware used for the purpose of this project was chosen based on two major factors, applicability and availability. The JR3 F/T sensor was an attractive choice both because the manufacturer offers a version which can fit the robot wrist with no modifications and also has clear and easy to programme interface.

All elements of the system were designed and implemented by the author during the first part of the research. This include:

- supervisory communication system implementation (including DDCMP protocol),
- implementation of robot floppy disc drive emulator (*pfloppy*),
- implementation of robot CRT terminal (*pterm*),

- JR3 F/T sensor loadable Module Kernel driver implementation,

The software was written in ANSI C language under GNU/Linux operating system. As stated before, after initial tests the robot-computer communication mode was entirely redesigned. Due to the existence of the DDCMP protocol frame the Supervisory connection system turned out to be slow and difficult to implement fully. In the case of prolonged peg-in-hole experiments a simple terminal emulator run on the PC computer proved to be a much more flexible and robust tool.

The major improvement to other researchers [4, 31] was to eliminate the presence of master-slave architecture. In the author's design it was replaced by a single master computer running the communication software, sensor driver and AI controller for position correction calculations.

The three different geometries of the parts were applied to evaluate the controller's performance. The major improvement was to use different material for the peg. The material was chosen to test the controller under different friction conditions.

A new approach to the software design was also proposed. The AI controller was built in a modular form (as a plug-in). This increases its flexibility enabling different architectures to be developed and tested. The main disadvantage lies in an increase of complexity of the software. The overall structure is shown in the Figure 3.4.

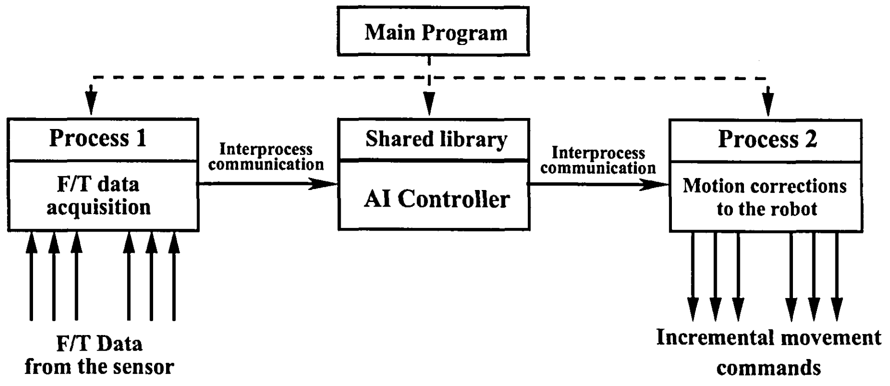

Main Program
Process 1
F/T data acquisition
Interprocess communication
Shared library
AI Controller
Interprocess communication
Process 2
Motion corrections to the robot
F/T Data from the sensor
Incremental movement commands

Figure 3.4: Software design.

The software consists of the main (ANN controller) and two child processes (F/T sensor driver and computer-robot connection module) running in parallel. The controller itself is compiled as a shared library and loads up or removes from memory at application runtime.

This approach allows the user to create different AI controllers and test them without any interference to the main application code. The specification of the application interface allows researchers to focus on controller development and implementation without coding the I/O operations needed to establish connection with robot.

The C code can be compiled as a module and in a similar manner to Internet browsers' plug-in, load and execute without stopping the main application. It is also possible to create the different parts of the controller, either ANN or reinforcement agent, as separate plug-ins and switch between them at the application runtime.

The system was initially designed to work with Puma 500 series robots. However, due to mechanical failure of the manipulator, the proposed system was later transferred, with no modification, onto a Staubli RX90 robotic arm. This demonstrates the portability and flexibility of the design and methodology.

Due to the complexity of the modular design the software debugging was very difficult. For that reason most of the presented results were acquired from the experiments using a non-modular version of the software. However, the system was deployed and tested, and performed well during the initial experiments.

Chapter 4

Methodology

During this research, the author will extend the earlier work by investigating how the AI controller should combine different information sources in order to generate a successful assembly strategy. The information sources mainly concern: task description, geometry, on-line force and torque contact data analysis and information derived from previous moves (experience). The originality of this approach lies in the definition of a methodology to merge information derived from different sources to attempt an interpretation of the contact state, in terms of contact location and the peg's attitude.

4.1 Controller architecture

As stated in Chapter 2, most current intelligent controller designs share a number of common disadvantages. The poor on-line performance or need for an explicit description of the environment are amongst the major drawbacks in automated assembly. Performing complex tasks (e.g. peg-in-hole insertion) the intelligent controller often deals with infinitive, 3-dimensional Cartesian space. Therefore, for efficient on-line state-action relationship, learning the state domain should be simplified. The classification is often necessary but by no means an easy task.

The reinforcement learning methods give the advantage of unsupervised on-line knowledge acquisition but their relatively slow learning process affects the controller's overall performance. Additionally, a clear state description information from the environment is required by the decision making agent. Because of that the noisy feedback signals need to be filtered, preprocessed and classified before being fed as an input to the algorithm.

The supervised learning methods are amongst the most popular algorithms for state clustering and pattern recognition. The major disadvantage of these methods, although they are fast and reliable, is the need for input-output pairs to train the network. This is, in fact, supplying the agent with an explicit knowledge about the geometry and relationship between the mating parts. That information is often unavailable or difficult to access, especially when introducing new geometry features during the learning process.

Those limitations inspired the author to propose the novel AI controller architecture. The design objectives were set as follows:

- the controller needs to be able to work without supervision with the environments featuring complex 3-dimensional geometry - this involves the implementation of an autonomous, independent decision making agent able to learn and work in the environment with a high degree of uncertainty,

- the controller should react fast and adapt easily for rapid changes and be able to utilise the knowledge from previous insertions - this involves issues like the stability-plasticity dilemma and a combination of different information sources to speed up the learning process,

- the architecture needs to be simple and modular - this allows easy performance evaluation and application of different AI methods.

In the presented configuration (see Figure 4.1), the controller receives two signals from the environment. The state description (s) is fed to the controller alongside the reward (r) feedback. The agent influences the environment by sending a control signal in the form of incremental movement correction data to the robot.

The proposed controller architecture features a "sandwich" structure with two major layers (see Figure 4.1) namely:

- State recognition or *State Clustering* module. This layer is designed to detect and localise the contact points that the agent experiences during the assembly task, Here

3-dimensional state domain classification takes place. The applied algorithm should show the ability for fast, unsupervised, adaptive learning. It is essential to efficiently analyse the environment's features and to follow its on-line changes.

- the decision making or *Learning Agent* subsystem learns to apply the next action (a) based on the calculated reward (r) and the state description signal (s).

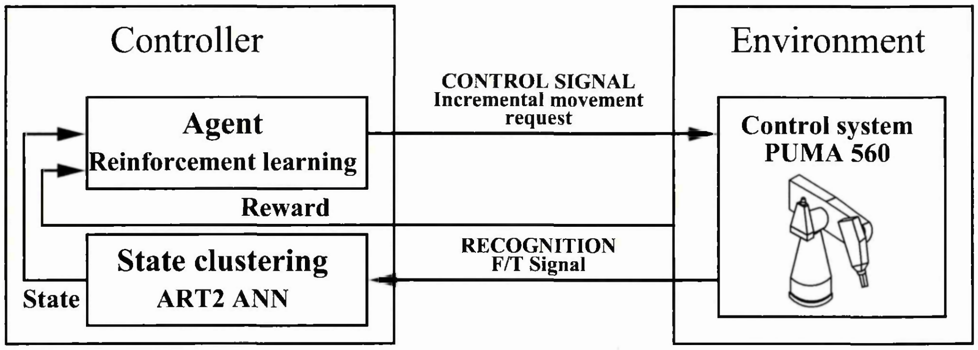


Figure 4.1: AI controller architecture.

4.1.1 The information flow within the controller

The F/T signal is acquired from the sensor and fed straight to the *State Clustering* module. This produces a number corresponding to the location of the contact point, called a *state description*. The information is sent as an input to the *Learning Agent* subsystem. In this module, the next action-selection takes place. The reward signal from the environment plays a significant role in the decision making process (see the next Section for details). This results in an action number which is later decoded to the incremental motion command understandable to the robot. The robot executes the move which causes the change in state. Then the new F/T vector is read and fed again to *State Clustering* module. This closed loop process is repeated until the stop condition occurs.

4.2 The learning agent

This is the main, decision-making part of the controller. Here the next action-selection takes place. Many different algorithms were applied in the past to learn manipulative skills during peg-in-hole insertion (see Chapter 2 for details). After careful consideration the author decided to take a reinforcement learning approach to the problem. The complete insertion can be considered as an *episode* (or play) and every single movement as a *step*. The controller will learn the state-action relationship by continuously repeating the insertions. This will result in developing the optimal insertion strategy.

Two different reinforcement learning algorithms were implemented and tested namely: SARSA and q-learning. Both methods belong to the Temporal-Difference family but differ in the learning policy (see Section 4.2.2 for details). The three different action-selection policies were also implemented and tested. All methods were evaluated with the simulated grid-world environment before being applied on a real life system.

4.2.1 Reinforcement learning

Following Sutton and Barto's description [36], Reinforcement Learning is meant to be a straightforward framing of the problem of learning from iteration to achieve the goal. It is a numerical way of finding and optimising the state-action relationship. The learning module gains knowledge about the task and makes an unsupervised decision about the next action based on the feedback signal it receives.

The learner and decision-maker is called *the agent*. The thing it interacts with, comprising everything outside the agent, is called *the environment*. The environment gives rise to *rewards*, a special signal whose values the agent tries to maximise over time. A complete specification of an environment defines a task, one instance of the reinforcement learning problem. The agent and environment interact continuously (see Figure 4.2).

During learning the adaptive system tries the actions on its environment, then it is reinforced by receiving a scalar evaluation (reward). The reinforcement learning algorithms selectively retain the outputs that maximise the received reward over time.

Reinforcement learning tasks are generally treated in discrete time steps. At each time step (t), the learning system receives some representation of the environment's state (s_t), it takes an action (a_{t+1}), and one step later it receives a scalar reward (r_{t+1}), and finds itself in a new state (s_{t+1}).

The significant feature that distinguishes the reinforcement learning approach from the other artificial intelligence algorithms is its ability to learn from experience. The idea is to capture the most significant aspects of the environment to solve the problem as opposed to supervised learning algorithms, which are mainly focused on finding the methods of acting.

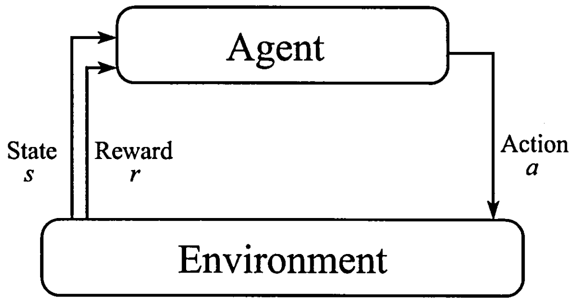


Figure 4.2: Reinforcement learning principle.

Reinforcement learning is a goal oriented method of knowledge acquisition. The decision making process is based on *value functions* (explained later in this Chapter) not on the immediate reward. This means that the agent deals with the whole problem rather than dividing it into simple sub-tasks as in the case of supervised methods. The reinforcement learning controller must interact with the environment to gain the knowledge and to achieve the goal. The supervised algorithms, on the other hand, require a set of examples to learn from. It is often impossible to obtain a good set which describes well the complex environments the agent must deal with.

Evolutionary methods are often used to solve reinforcement learning problems. The theory is based on organisms producing a skilled offspring even if they do not learn during their lifetime. The genetic algorithms do not interact with an environment ignoring the fact that learning policy is a function of state to action pairs (function values). Moreover, the information about the experienced states and action taken is also ignored and lost for future reference. These methods are very effective when the state signal is noisy or the agent is unable to interpret it accurately.

Value functions versus the immediate rewards

As mentioned before, the value functions play a significant role in reinforcement learning methods. It is a function which maps the actions (or state-action pairs) onto the response from the environment (the reward). It classifies good and bad events describing the desirability of their occurrence. The agent, during the learning process, tries to maximise the total rewards. This defines the feature of the whole problem. In the other words, the value of a state is the total amount of reward an agent can expect to acquire in the future starting from there [36]. The value functions are built to represent the prediction of rewards and to take into account the action likely to follow (future prediction). They play a significant role in the unsupervised decision making process. The system chooses the actions based on the function values not the rewards. The choices with the highest value are likely to gain the highest amount of rewards in the future.

Differently from the value functions, which describe what is good or bad in a global sense, the rewards provide information about the immediate response of the environment. This primary feedback signal directly affects the policy changing it to choose the appropriate action in the future. It is easy to obtain since it is given directly by the environment the agent deals with. The value functions, on the other hand, have to be estimated from sequences of interactions (the agent's lifetime experience). These methods of estimation play a key part in every reinforcement learning algorithm [36].

The genetic algorithms (mentioned earlier in this chapter) do not use a value function, instead they rely on immediate rewards to search through the policy space for the best actions. These methods can produce very quick and effective results working in environments featuring small or easily classified policies.

Learning policy

The policy is the core of every reinforcement learning algorithm. It defines the relationship between the states and the actions.

The early reinforcement learning algorithms have used a trial-and-error method to learn the optimal policies. The modern ones try to learn the model of the environment instead. The agent's ability to emulate the environment gives the benefit of next state and reward prediction for a given state-action pair. In other words, the advanced systems with high level deliberative planning are able to consider future events during the decision making process.

Action-selection methods

Reinforcement learning estimates the action taken rather than instructing the system. This feature has been widely used for function optimisation, and is a basis of evolutionary algorithms [36]. The agent evaluates how good an action taken is, but not whether it is the correct choice. The instructive approach, on the other hand, is an action independent approach and indicates the best known solution.

The action dependent, evaluative feedback approach requires the agent to decide how far to explore the environment or to exploit the current knowledge. This decision making process is called *exploration-exploitation dilemma* and can significantly influence the on-line performance of the reinforcement learning algorithm [36]. The supervised methods implement the exploration-exploitation trade-off explicitly, relying heavily on the expert knowledge. Three different action-selection methods were implemented and tested by the author. They differ in principles and applications:

- Greedy action-selection. This method relies entirely on knowledge exploitation. The agent always chooses the action with the highest estimated value. This means that no exploration of the environment takes place. It performs poorly with the complex environment and rarely produces an optimal result. Its limitation lies in the fact that it ignores other actions which may produce better results in the long run. However, this method performs very well in non-complex environments where the system knows each action after trying it once (reward variance equal to zero). In the case of greedy action-selection the probability of greedy action (a_g) is equal to one for every episode ($P(a_g) = 1$).

- ϵ-greedy action-selection. It is a very similar approach to the greedy method. The agent still chooses the action with the highest estimated value but also allows the exploration with a small probability ϵ. The method was implemented so that the value of ϵ depends on the number of the current episode. This reduces the amount of exploration as the number of plays increases. The probability of greedy action a_g was set as follows:

$$P(a_g) = 1 - \epsilon + \frac{\epsilon}{|a|} \tag{4.1}$$

where ϵ depends on the episode number n and it is described by following equation [36]:

$$\epsilon = \frac{1}{\mu * n + 1} \tag{4.2}$$

where μ is a bias which regulates how much exploration the agent will experience throughout the plays (see Figure 4.3).

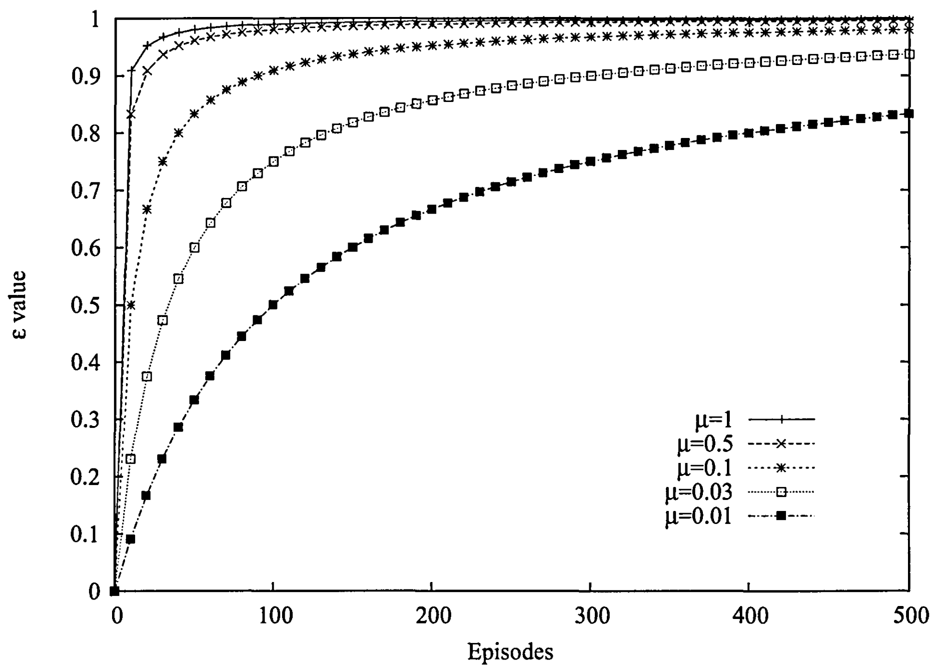


Figure 4.3: ϵ-greedy action-selection method.

The Figure 4.3 presents how the value of ϵ changes over a number of episodes. The simulation was performed on a standard ϵ-greedy algorithm using equations presented above (4.1). Five different values of μ were tested on a simulated 500 episodes problem. It is clear from Figure 4.3 that the probability of choosing a random action decreases with the increased number of plays. On the other hand if the value of μ gets smaller the agent is allowed to explore more.

The main advantage of this policy over the greedy action-selection method is the fact that with the increase of episodes, every possible action can be tried and evaluated an infinite number of times [36]. The ϵ-greedy action-selection method guarantees that during exploration time every action has an uniform chance to be chosen. This is independent of its estimated value so even the actions with the lowest values can be applied. This feature is highly undesirable in the environments where the "bad" actions could have serious implications on system safety and performance (e.g. workpiece damage, robot collision).

- Softmax action-selection. In this method the probability of each action is a function of its estimated value. The uniform choice in the form of random function present in ϵ-greedy method has been replaced with a system of weighted actions. In Softmax selection policy the probability of choosing the action is usually governed by Gibbs distribution, described as follows:

$$P(a_i) = \frac{e^{\beta * a_i}}{\sum_j e^{\beta * a_j}} \tag{4.3}$$

where β is a ratio which determines how much exploration the agent is allowed to take. When $\beta = 0$ the actions are selected randomly and for $\beta \rightarrow \infty$ the greedy policy is applied. In between the above limits the actions with the highest estimated value still have the highest probability of being chosen. Although the actions showing the lowest estimated values have a relatively small chance of being applied (see Figure: 4.4). This approach favourers the "good" actions over the "bad" ones.

It is clear from Figure 4.4 that for the $\beta = 0$ the uniform random function governs the action-selection. With the increase of β the actions with the higher estimated values have more chance of being chosen than those with low values. The sum of probabilities is of course equal to one.

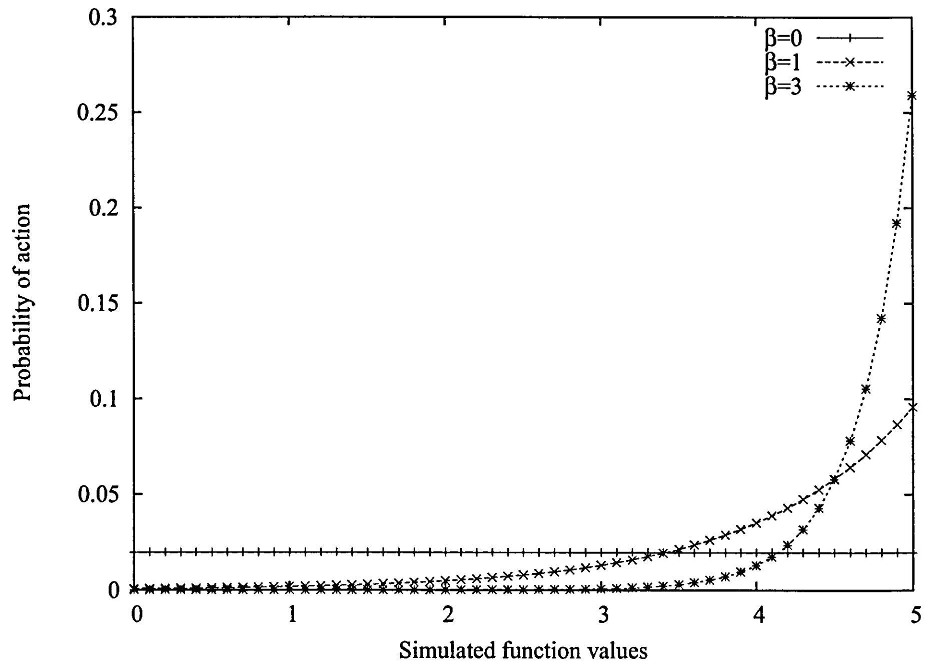


Figure 4.4: Gibbs distribution for simulated action values.

4.2.2 Temporal-Difference methods

The main idea of reinforcement learning is called *Temporal-Difference* learning. These methods are, in general, learning algorithms to make long-term predictions about dynamical systems. They are based on estimating value functions, functions of states (s) or action-states pairs ($Q(s, a)$) that estimate how good it is for the learning system to be in a given states or to take a certain action in a given state. Such value actions guide the action-selection mechanism, the policy to balance exploration and exploitation, in order to maximise reward over time, and let the learning system achieve its goal.

The Temporal-Difference methods are based on Monte Carlo and Dynamic Programming approaches (the detailed description of those methods can be found in the book [36]). Similarly to Monte Carlo the Temporal-Difference methods learn from experience without the need for a model of the environment. The major limitation of the Monte Carlo method lies in the way it updates its estimates. The agent must wait for the result of an action to calculate the value function. This means that whole episode must be completed by the agent to learn. In applications featuring very long episodes or continuous task-based problems the Monte Carlo methods would produce a low on-line performance.

To solve this problem Temporal-Difference methods combine Dynamic Programming methods into the above mentioned system. The learning takes place after each state transition by interaction in discrete time steps. This method of calculating estimates based on past estimates ("guess from the guess") was proved to converge to the correct answer (see [36] for details).

Q-learning

In this type of Temporal-Difference learning method, the learnt action-value function Q directly approximates the optimal action-value function Q^*, independent of the policy being followed (off-policy). The policy still has an effect in that it determines which state-action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated. This is a minimal requirement in the sense that any method guaranteed to find optimal behaviour in the general case must require it. Under this assumption and a variant of the usual stochastic approximation conditions on the step-size sequence, has been shown to converge with a probability of one to Q^*.

The Q function is defined as followed (after Sutton and Barto [36]):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{4.4}$$

where: t is a discrete time step at which an agent receives representation of the environment; s is a finite set of states an agent can be in; a is a finite set of actions an agent may perform; Q is a transition function, mapping each state-action pair to a successor state; r is a reward function, mapping states-action pairs to payoffs, α is a learning rate and γ is a discount factor. These last two ratios are learning parameters.

Each state-action pair is assigned the highest discounted cumulative reward possible, if state s is the agent's starting state and a is the first action performed. So function Q is based on an optimal policy.

The optimal strategy is not known in advance. But it shows that the Q function can be approximated by continuously collecting locally observed payoffs and by updating the approximation of Q, using the equation above, although the values for the successor states may have been initialised to arbitrary values.

The q-learning was implemented after Sutton and Barto [36]. The algorithm is presented in the form of a flowchart below (see Figure 4.5) where Q stands for q-function (estimate function value), s is a state, a an action, t a time. E represents a maximum number of episodes. The applied action-selection policy could be either greedy, ϵ-greedy, or Softmax.

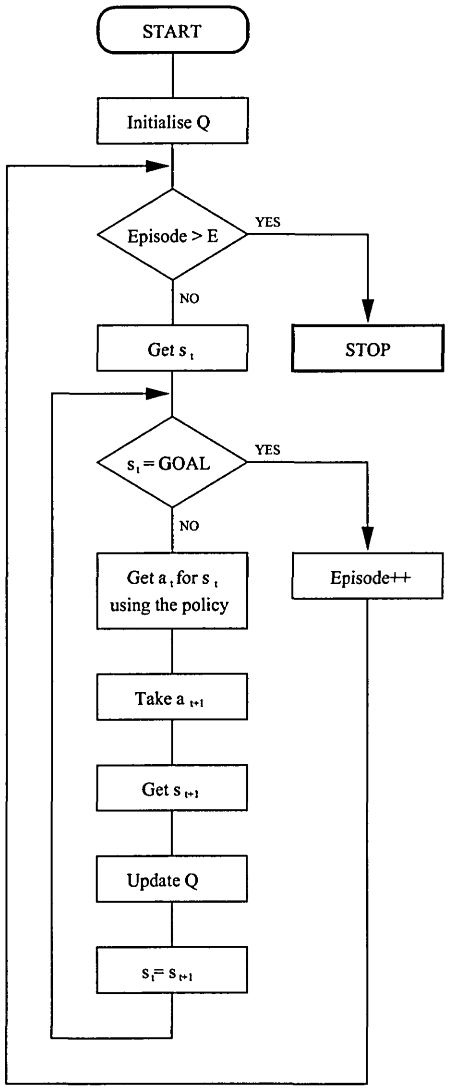


Figure 4.5: One step Q-learn algorithm flowchart.

The basic q-learning algorithm was applied. The class number was set as an input describing the actual contact forces. Using this configuration, the reinforcement algorithm needs to learn how to deal with a particular part of the component's geometry. Moreover the self-learnt geometry information includes only orientation of the given plane. Based on that the agent can be trained with a set of surfaces separately and then be able to work efficiently with an object built from these planes. So, unless the part consists of a feature which has not been learnt before, the change of geometry (e.g. from circular to square peg) should not affect the controller's performance in the early stages of insertion.

The q-learning algorithm was used with greedy, ϵ-greedy and Softmax action-selection methods. Based on research by the author the application of this method together with proposed state domain clustering has not been applied for peg-in-hole insertion before.

Q-learning learns values for optimal policy so after a training period the peg is expected to slide along the chamfer's surface to the hole. Unfortunately, in case of the ϵ-greedy action-selection method, it occasionally chooses the movement towards the wall inevitably increasing the contact forces.

SARSA

The SARSA algorithm belongs to the on-policy Temporal-Difference methods class. Similarly to q-learning SARSA gathers the knowledge from the state-action pairs transitions but it learns the policy directly. The on-line performance of the algorithm depends on the application but generally is better than in the case of q-learning. The learning of optimal policy relies on applied action-selection method. Each state-action pairs must be visited an infinitive number of times to gain the best possible solution. This could be achieved by applying an ϵ-greedy action-selection method which converges in limit to greedy policy (see section 4.2.1 for details).

The relationship that governs the learning process is described by the equation below (after Sutton and Barto [36]):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{4.5}$$

as in case of q-learning: t is a discrete time step; s is a finite set of states an agent can be in; a is a finite set of actions an agent may perform; Q is a transition function, mapping each state-action pair to a successor state; r is a reward function, mapping states-action pairs to payoffs, α is a step size and γ is a discount factor.

As mentioned in the previous section the q-learning method learns the optimal policy. This could be a disadvantage in some applications especially when particularly low rewards are assigned to some movements. In those cases, despite a pre-learnt optimal policy, due to exploratory random action choice the agent can experience an error condition (sudden increase of forces in the case of peg-and-hole insertion). The SARSA method, due to the fact that it learns a safe path, is not affected by this problem. It should be emphasised at this point that it is possible for SARSA (by applying an appropriate action-selection method) to converge to optimal policy (see Section 4.5.1 for simulation results).

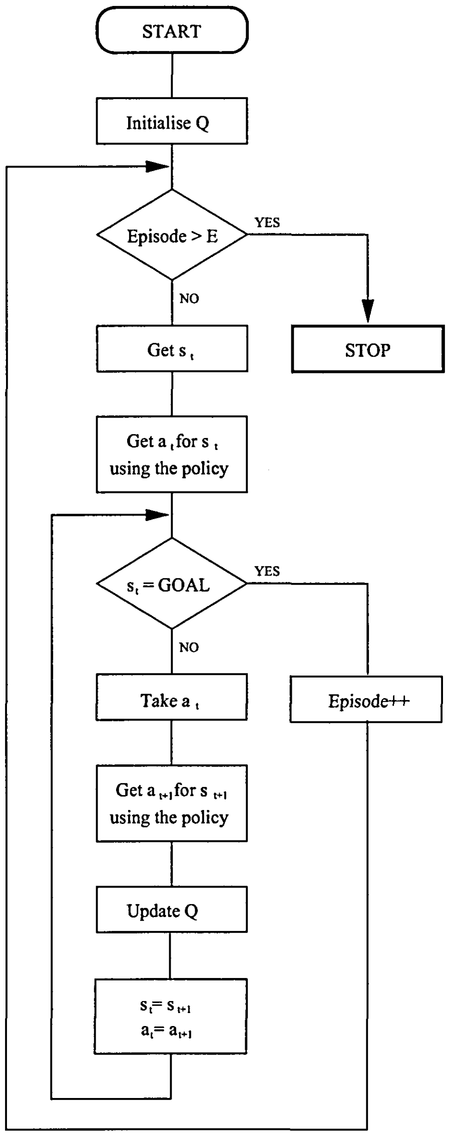


Figure 4.6: SARSA algorithm flowchart.

The SARSA algorithm was implemented, applied and tested with greedy, ϵ-greedy and Softmax action-selection methods. The implementation method in the form of a flowchart is presented in Figure 4.6. As in the case of the q-learning algorithm flowchart, Q stands for q-function (estimate function value), s is a state, a an action, t a time. E represents a maximum number of episodes.

4.3 State domain clustering module

To learn an appropriate action the reinforcement learning agent requires a clear definition of the state. This information can be constructed from almost any data acquired from the environment. In most cases the raw signals have to be preprocessed or classified and in some applications the state can be built from a combination of two or more features. According to reinforcement learning theory, the preprocessing system is nominally a part of the environment [36]. It is clear from Figure 4.1 that in the proposed design the preprocessing module is placed as a part of the controller. The reason for it is the fact that applied methods of classification are fully self-adapting and together with the reinforcement learning agent they form an intelligent controller. Although, it should be emphasised at this point that the preprocessing system called for the purpose of this thesis *state clustering module* is not a part of a reinforcement learning agent. Both subsystems work independently of each other and serve different purposes. The decision about the next action is taken by the agent without knowing how the state signal was constructed.

4.3.1 Markovian State

The choice of the best action for a particular state is not an easy task. The decision has to be made based on immediate rewards but the overall performance also needs to be taken into account. Making the choice by monitoring just the immediate sensations will significantly affect the agent's learning speed since, very often, the actions with poor immediate rewards can produce good results on the global (task) scale.

According to Sutton and Barto [36], a good state representation is built by the information sources in such a way that it concludes and compacts the past experiences. Ideally all data important for the future decision-making process should be retained. A state signal with all relevant past events encoded is called *Markovian state*. This usually involves more than just immediate rewards from the environment, but never the whole task history.

The states that feature the Markovian property play a significant role in the reinforcement learning process. The fact that past sensations are retained in the current state makes the decision-making process history independent. In the other words all that matters for the future is included so the agent can predict the next state (s_{t+1}) and reward (r_{t+1}) based on current state (s_t) and the action (a_t). This very important feature is essential for a fast, reliable decision-making process. The next action choice which is based on the state signal with Markovian property is as good as the decision based on the entire history of the task.

4.3.2 State classification using Artificial Neural Networks

The State Classification module was designed to generalise the large and continuous state space. In the author's application the only sensory signal from the environment is in the form of a force and torque signal. Because of that, the unsupervised pattern recognition or, in the other words, function approximation system needs to be implemented.

There are many ways to accomplish that including the application of neural networks or fuzzy logic systems, etc. One of the simplest approaches is to partition the continuous state space into a set of classes, and treat each class as an autonomous object.

There is no universally accepted definition of Neural Network. According to ANN FAQ [33], the Artificial Neural Network is a network of many simple processors (Processing Elements), each possibly having a small amount of local memory. The units are connected by communication channels which usually carry numeric data, encoded by any of various means. The units operate only on their local data and on the inputs they receive via the connections (see Figure 4.7).

Zaknich in his definition [47] describes Artificial Neural Networks as an engineering discipline concerned with non-programmed adaptive information processing systems that develop associations between objects in response to their environment. That is, they learn from examples. Neural Networks are a type of massively parallel computing architectures based on brain-like information encoding and processing models and, as such, they can exhibit brain-like behaviours such as: learning, association, categorisation, generalisation, feature extraction, pattern recognition or optimisation.

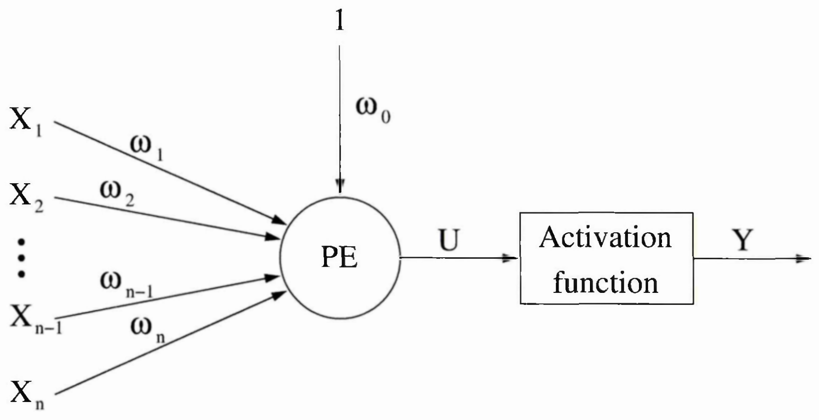


Figure 4.7: Basic Processing Element.

Given noisy sensory inputs, they build up their internal computational structures through experience rather than preprogramming according to a known algorithm.

Haykin in his book [22] gives a more formal definition: "A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- knowledge is acquired by the network through a learning process,
- inter-neuron connection strengths known as synaptic weights are used to store knowledge."

The initial intent of Artificial Neural Networks was to explore and reproduce human information processing tasks such as speech, vision, and knowledge processing. These algorithms also demonstrated their superior capability for classification and function approximation problems. This has great potential for solving complex problems such as systems control, data compression, optimisation problems, pattern recognition, and identification.

There are many different types of Neural Networks (see Figure 4.8). They differ in the kinds of data they accept (categorical and quantitative variables), in topology (feed-forward, feedback or recurrent) or in the learning approach.

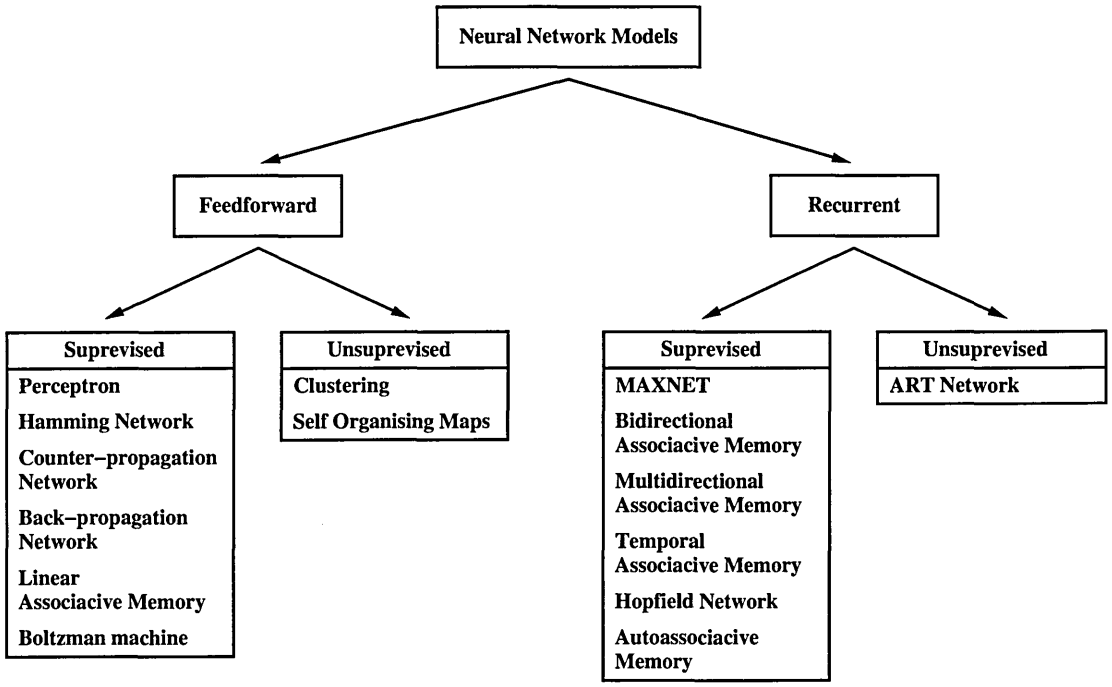


Figure 4.8: ANN's taxonomy (after Huang and Zhang [25]).

The three main types of ANN based on the learning approach are listed as follows (after Zaknich [47]):

- *Supervised learning* - this type of artificial Neural Networks are trained to perform a task by a *teacher* repeatedly showing them representative examples of the inputs they will receive, paired with the desired outputs. During each learning or training iteration the magnitude of the error between the desired and the actual network response is computed and used to make adjustments to the internal network parameters or weights according to some learning algorithm. As the learning proceeds the error is gradually reduced until it achieves a minimum, or at least an acceptably small, value.

- *Reinforcement learning* - in this type of learning the algorithm does not need to compute the exact error between the desired and the actual network response, rather for each training example the network is given a pass/fail signal by the *teacher*. If a fail is assigned, the network continues to readjust its parameters until it achieves a pass or continues for a predetermined number of tries, whichever comes first. Reinforcement learning is sometimes thought of as a special type of supervised learning and it should not be confused with the unsupervised methods described earlier in this chapter.

- *Self-organising learning* - takes examples of the inputs and forms automatic inter groupings and reorganisations or self-clusterings of the input data based on some measure of closeness or similarity. It is then sometimes possible to assign some meaning to those clusters in context with the nature of the data and problem involved.

Neural Networks have been successfully applied to many manufacturing activities spreading from the design phase through process planning, scheduling and monitoring to quality assurance. In robotics where the simplest movement requires a sophisticated calculation intelligent algorithms were used to reduce the computational complexity. Due to its massive parallelism they offer a significant increase in processing speed. This appears to be useful in solving forward and inverse kinematic problem. In robot dynamics the Neural Networks have been successfully applied to learn inverse dynamical relationships.

Adaptive Resonance Theory

Adaptive Resonance Theory (ART) was developed by Stephen Grossberg and Gail Carpenter over the period of 1976-86, during their studies of the behaviour of models of systems of neurons [9, 17]. ART was developed to solve the learning instability problem suffered by standard feedforward networks. The weights which have captured some knowledge in the past continue to change as new knowledge comes in. There is, therefore, a danger of losing the old knowledge with time.

Adaptive Resonance Theory gets its name from the particular way in which learning and recall interplay in the network. In this type of network, information in the form of processing element outputs reverberate back and forth between layers. If the proper patterns develop, a stable oscillation occurs. This is the neural-network equivalent of resonance. Only during this stage can learning take place [16].

A *resonance* can be attained in one of two ways. If the network has learnt to recognise the input pattern before then the resonant state will be achieved quickly when the input vector is presented. During this state the memory of the stored pattern will be reinforced. If the input vector is not recognised the network will look in the stored patterns database for a match. If no match is found the network will enter the resonant state and the new pattern will be stored for the first time. Thanks to this design the network is able to respond quickly to previously learnt data, yet remains able to learn new patterns.

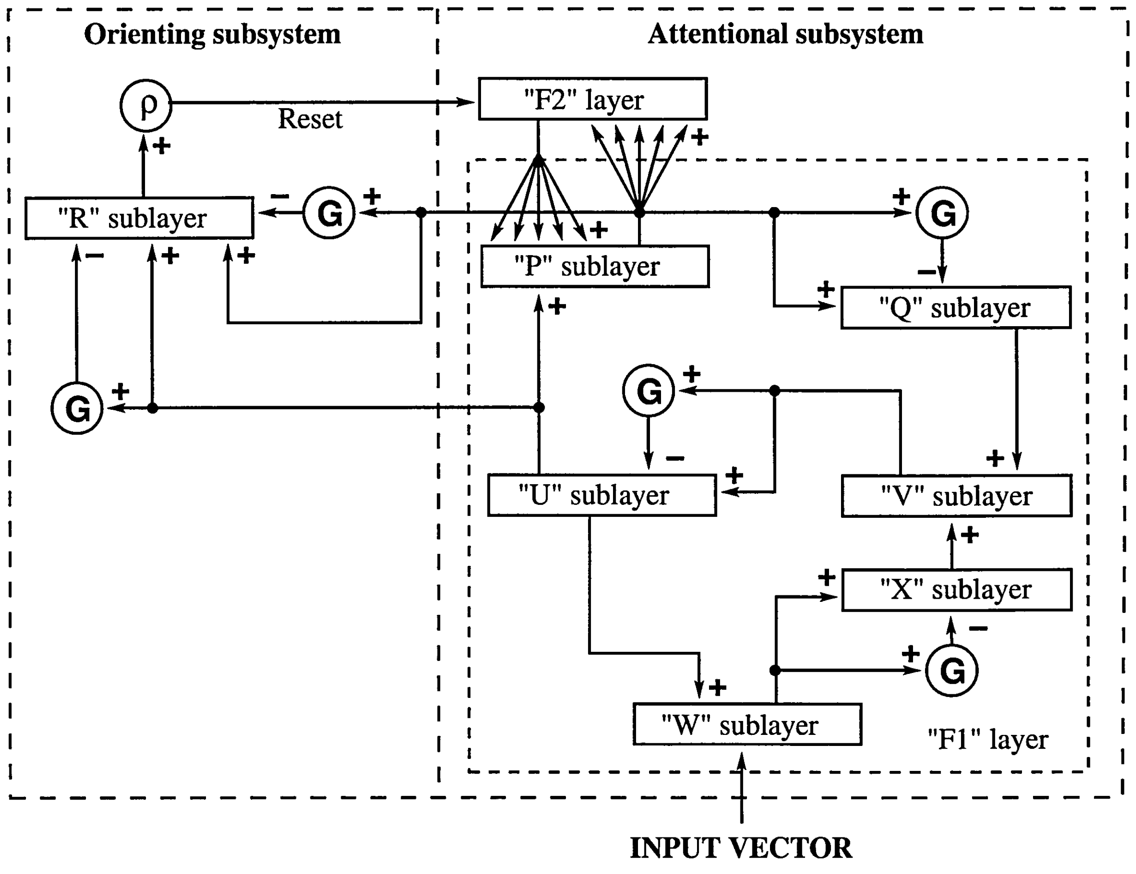


Figure 4.9: Implemented ART2 architecture (after Ferrman and Skapura [16]).

In the basic ART system two major subsystems can be distinguished. The *attentional subsystem* is represented by F_1 and F_2 layers of nodes. Patterns of activity that develop over them is called Short Term Memory (STM) traces, because they exist only with a single appearance of an input vector. The Long Term Memory (LTM) in ART architecture is represented by a set of weights associated with top-down and bottom-up connections between F_1 and F_2 layers.

The *orienting subsystem* is responsible for sensing mismatches between top-down and bottom-up patterns on the F_1 layer. Its operation can be modelled by the addition of terms to the dynamic equations that describe the activities of the F_2 processing element. The orienting subsystem could be modelled as as a single processing element with output to each unit on the F_2 layer. When a pattern mismatch occurs, the orienting subsystem should inhibit the F_2 unit that resulted in the non-matching pattern and should maintain that inhibition throughout the reminder of the matching cycle. The vigilance parameter (ρ) is associated with this subsystem. It measures the degree to which the system discriminates between different classes of input patterns. For a given set of patterns to be classified, a large value of ρ will result in finer discrimination between classes (see Section 4.5.2 for more results).

ART2 is the type of Adaptive Resonance Theory family which accepts analogue vectors components as well as binary components. It is also able to recognise the underlying similarity of identical patterns superimposed on constant backgrounds having a different level.

The price for this additional capability is an increase in complexity of the F_1 processing layer. It consists of several sub-levels and gain control systems that serve to remove noise, enhance contrast, and to normalise an analogue input pattern.

The activity of each unit on each sublayer of F_1 is governed by an equation [16]:

$$\epsilon \dot{x}_k = -Ax_k + (1 - Bx_k)J_k^+ - (C + Dx_k)J_k^- \tag{4.6}$$

where: J_k^+ is an excitatory input to the kth unit, J_k^- is an inhibitory input, x_k refer to activities on the particular layer and A, B, C, D are constants,

The equations that govern the activities on each of the six sub-layers are as follows [16]:

$$w_i = I_i + au_i \tag{4.7}$$

$$x_i = \frac{w_i}{e + \|\mathbf{w}\|} \tag{4.8}$$

$$v_i = f(x_i) + bf(q_i) \tag{4.9}$$

$$u_i = \frac{v_i}{e + \|\mathbf{v}\|} \tag{4.10}$$

$$p_i = u_i + \sum_j g(y_j) z_{ij} \tag{4.11}$$

$$q_i = \frac{p_i}{e + \|\mathbf{p}\|} \tag{4.12}$$

where w_i, u_i, x_i, v_i, p_i, q_i represent activities on the F_1 sublayers (see Figure 4.9); $g(y_j)$ is an output function from F_2 layer; $f(x)$ is a threshold linear function and e is a network parameter typically set to a positive number considerably less than 1.

Processing on F_2 of ART2 is identical to that performed on ART1. Bottom-up inputs are calculated as follows:

$$T_j = \sum_i p_i z_{ji} \tag{4.13}$$

The output function of F_2 is govern by equation:

$$g(y_j) = \begin{cases} d & T_j = \max(T_k) \forall k \\ 0 & otherwise \end{cases} \tag{4.14}$$

4.4 Software implementation and data structures

Most Artificial Neural Network algorithms share the same basic concepts of distributed and highly interconnected processing elements. The key goal at this stage of research was to develop a flexible system able to support several different AI architectures. For that reason the code portability and re-usability was an important issue during the design process.

Artificial Intelligence algorithms, due to their specific structure, are well suited for object oriented programming. C is a general purpose structural programming language. It was designed for the UNIX environment and was successfully adopted by the other operating systems. However, the quasi-objective programming is also allowed in C. By organising the variables into structures the programmer can access them in a very similar way as the classes and objects in C++. The major difference (but by no means limitation) is the fact that the "C" function cannot be a member of the structure.

For the purpose of this project the data was organised into several structures. With a few exceptions, and the author's modifications, the implementation follows Ferrman and Skapura's [16] proposal.

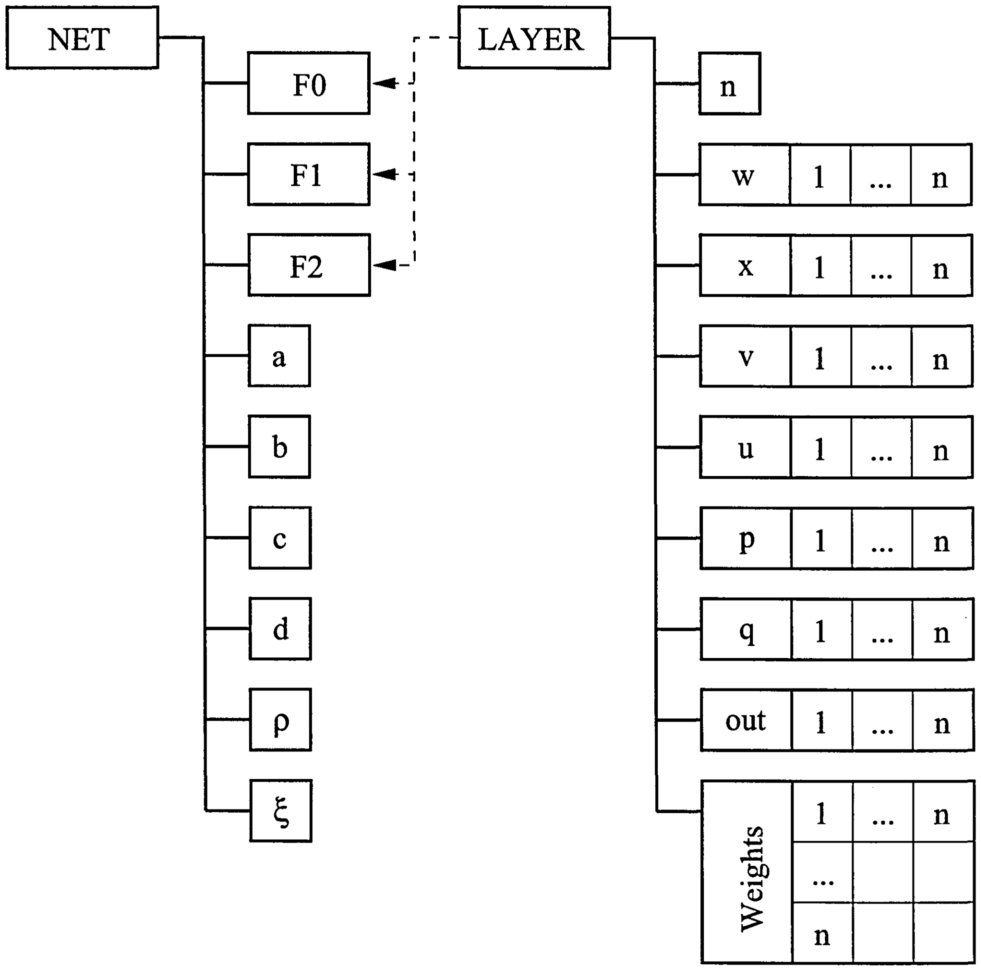


Figure 4.10: Data structure implementation for ART2 Neural Network.

The ART2 data was organised into two nested structures: `LAYER` and `NET` (see Figure 4.10). The first one defines the structure of a single layer of Processing Elements. It consists of the pointers to all sublayers described in the Section 4.3.2. The structure `NET` implements the whole topology of the ART2 network. It is built of three pointers to `LAYER` structure namely `F0`, `F1` and `F2`. The set of learning parameters including the vigilance (ρ) is also implemented there. This design creates the flexibility of quick programming of different topologies of Artificial Neural Networks. The relationship between the layers and the learning algorithm are implemented independently of those structures.

The number of layers or input and output units can vary and depends on the application. The dynamic memory allocation was used to implement the data.

The reinforcement learning agent was implemented in a similar manner to ART2 network. The structure consists of pointer to state-action q-function matrix as well as a set of learning parameters.

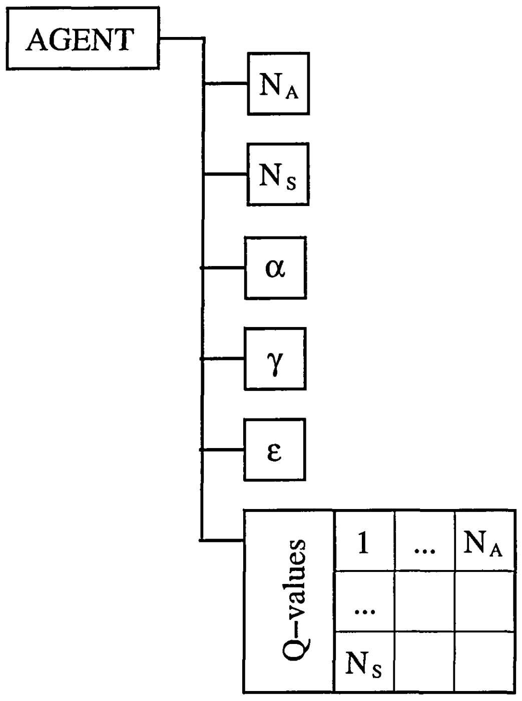


Figure 4.11: Data structure implementation for reinforcement learning agent.

The relevant fragments of C code listings can be found in Appendix E.

4.5 Algorithms simulation

All algorithms described earlier were implemented and tested on simulated data before being applied onto the real-life peg-in-hole insertion. Every part of the controller was evaluated separately. This approach gave the author an opportunity to understand the information flow within the Artificial Intelligence algorithms and to learn how the different ratios affect the learning speed and overall performance.

Simulated data was used for the purpose of the tests. Two different tests were performed. The grid-world was implemented as a representation of the environment for the agent to interact with, and set of generated force and torque arrays were used to investigate the efficiency of the State Clustering module.

The summary graphs are presented in the main body of the the thesis followed by detailed analysis. The remaining results are included in the Appendix A.

4.5.1 Learning Agent evaluation

To evaluate the performance of action-selection methods the 2-dimensional maze was designed (see Figure 4.12). It is based on a matrix of squares with seven rows and columns. The agent was programmed to move around the white area. The grey squares represent the obstacles.

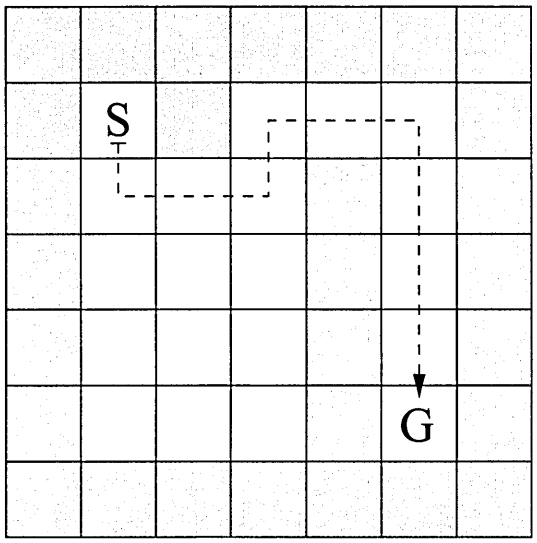


Figure 4.12: A grid-world environment to evaluate the reinforcement learning methods.

During each play the agent could experience up to 15 discrete states (see Figure 4.13). The state described by number "15" is practically impossible for the agent to achieve since it represents the lockup position. There were four available actions to chose from, namely: move North, South, East or West. The agent was allowed to take only one step at a time. The goal "G" was setup in the bottom right corner of the maze and was defined by the state number "14". Each play was started from the square described by "S". The optimal path from "S" to "G" is indicated by the dashed line (see Figure 4.12).

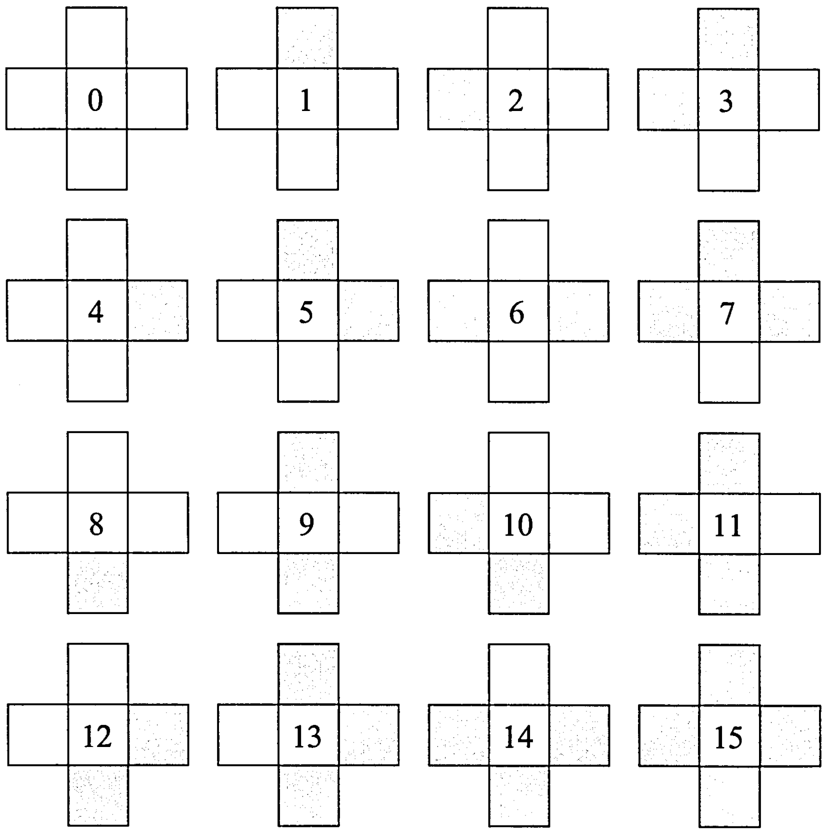


Figure 4.13: The set of states the agent can experience during each play.

To simplify learning the grid-world was designed so there is only one best action for the current state. The optimal trajectory (indicated by the dashed line) involves ten movements which represents the shortest route from the START to GOAL conditions.

Action-selection policy evaluation

The action-selection policies described early in Section 4.2.1 were implemented and applied on the grid-world from Figure 4.12. For all simulations the discount ratio γ was set to 0.9. This makes agent take future rewards into account so it becomes more farsighted. The learning ratio α was set to the value of 0.1.

First ϵ-greedy policy was evaluated using the SARSA reinforcement learning algorithm. Four different bias values μ (see Equation 4.2) were used to show how the action-selection policy affects the agent's learning speed and on-line performance.

The agent is programmed to move around the maze until the goal is reached. After that the controller returns to the starting point to begin another episode. The reward function was implemented as follows, the value of "-1" was returned for every movement and the controller was granted "0" for achieving the goal. In this, however simple, design the agent reduces the overall number of steps by maximising the cumulative reward. This approach also gives an opportunity of easy analysis of the results since the negative feedback value directly corresponds to the number of actions taken.

The results from the simulation can be seen on the Figure 4.14.

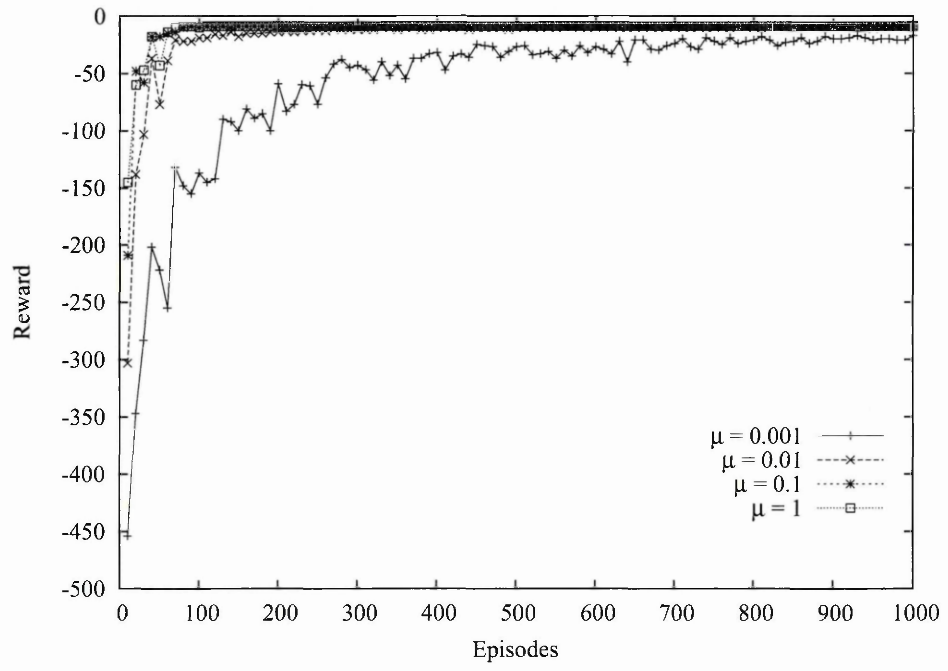


Figure 4.14: ϵ-greedy policy evaluation using SARSA algorithm.

The on-line performance of the algorithm decreases while the amount of exploration increases. Clearly applying $\mu = 0.001$ produces unacceptable results for such a simple environment. The controller explores the environment by executing random actions but its on-line performance is significantly lower in relation to the other three policies. The reason for it lies in the number of collisions with the obstacles caused by the random actions.

The Softmax action-selection policy was applied on the maze using exactly the same conditions. Again, for all simulations the discount ratio γ was set to 0.9 and the learning ratio α was set to the value of 0.1. The different values of β (see Equation 4.3) were applied namely: $\beta = 1$, $\beta = 2$ and $\beta = 5$.

Three separate simulations were performed. The results can be seen on Figure 4.15 below.

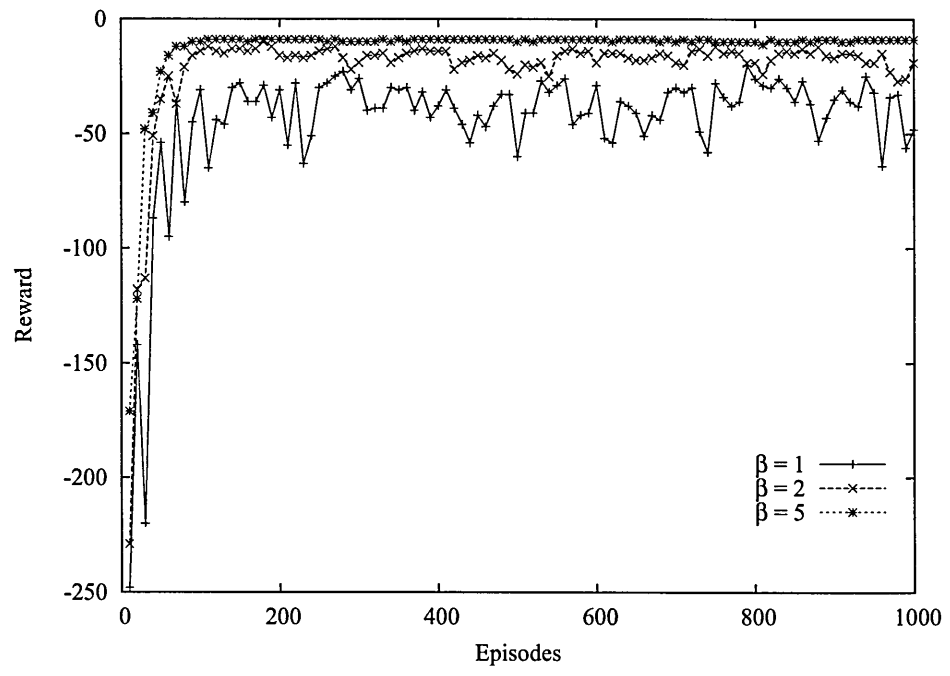


Figure 4.15: Softmax policy evaluation using SARSA algorithm.

As stated before in this chapter Softmax policy, due to its implementation, favours the the "good" actions over the "bad" ones. The value of β governs how much exploration the agent can take. It could be seen from Figure 4.15 that reducing β the number of random actions increases at the cost of on-line performance which drops. This has a similar effect on how quickly the agent can learn like the ratio of ϵ in the case of ϵ-greedy policy.

It should be emphasised at this point that the value of β was constant throughout each simulation. This is the reason that the curves stabilise at a certain level and do not converge to the optimal reward (which could be considered as the optimal trajectory). The optimal policy could be achieved by changing β from episode to episode (like in the case of ϵ).

Using this approach, and assuming enough exploration, the agent will eventually learn the optimal policy. In this case the optimal policy is defined by assigning the best action (move) to a given state. Having learnt that, the agent is able to follow the shortest trajectory from the START to GOAL (see Figure 4.12).

State	Learnt Policy							**Optimal policy**
	ϵ-greedy				Softmax			
	μ=0.001	μ=0.01	μ=0.1	μ=1	β=1	β=2	β=5	
0	E	E	E	E	E	E	E	–
1	E	E	E	E	E	E	E	**E**
2	E	E	E	E	E	E	E	**E**
3	E	E	E	E	E	E	E	**E**
4	N	N	N	N	N	N	N	**N**
5	S	S	S	S	S	S	S	**S**
6	S	S	S	S	S	S	S	**S**
7	S	S	S	S	S	S	S	**S**
8	N	W	E	E	E	E	E	–
9	E	E	E	E	E	E	E	**E**
10	N	W	N	S	E	E	E	–
11	N	N	N	N	N	N	N	–
12	N	N	E	E	N	N	N	–
13	N	N	N	N	N	N	N	–
14	**G**	**G**	**G**	**G**	**G**	**G**	**G**	**G**
15	N	N	N	N	N	N	N	–

Table 4.1: Learnt policies for different action-selection methods.

The learnt policies for both ϵ-greedy and Softmax action-selection methods are presented in Table 4.1. The actions are mapped onto set of encoded states the agent can experience walking around the maze (see Figure 4.13 for details). Letters "N", "E", "S", "W" represent the direction of movements: North, East, South and West respectively, and a letter "G" describes the GOAL.

The optimal policy was derived by the author following the shortest possible trajectory form START to GOAL states. It could be seen that both methods successfully converged to the optimal policy. Despite low on-line performance the agent managed to learn the best state-action relationship. This proves that under the right circumstances (the tight exploration-exploitation balance) the SARSA algorithm can learn the optimal policy. The tested maze represents a relatively simple task for the agent to learn. It will be shown in the next section that SARSA, as opposite to q-learning, being an on-policy method does not always guarantee convergence to such results.

The SARSA algorithm combined with Softmax action-selection method performs very well and the optimal policy is quickly learnt together with a good on-line performance. This is due to the fact that even during random action-selection non-uniform function is used and the action with better values has more chance to be chosen.

The simulations with q-learning algorithm applied produced very similar results (for the graphs see Appendix A). The reason for that lies in the design of the environment. In this particular task both methods quickly converged to the optimal policy. The maze from Figure 4.12 due to its complex shape but relatively simple optimal trajectory was a good choice to exercise the SARSA and q-learning methods with different action-selection policies. However, it was not appropriate to compare those algorithms and to clearly emphasise the differences amongst them. To serve that purpose another maze-like environment was implemented.

Reinforcement methods evaluation

Two reinforcement learning methods were implemented and compared. The SARSA on-policy method and q-learning algorithm were simulated on a cliff-walk task described in Sutton and Barto's book [36]. The main purpose of those simulations was to compare the algorithms, test the implementations and to learn the agent's responses for different values of learning ratios.

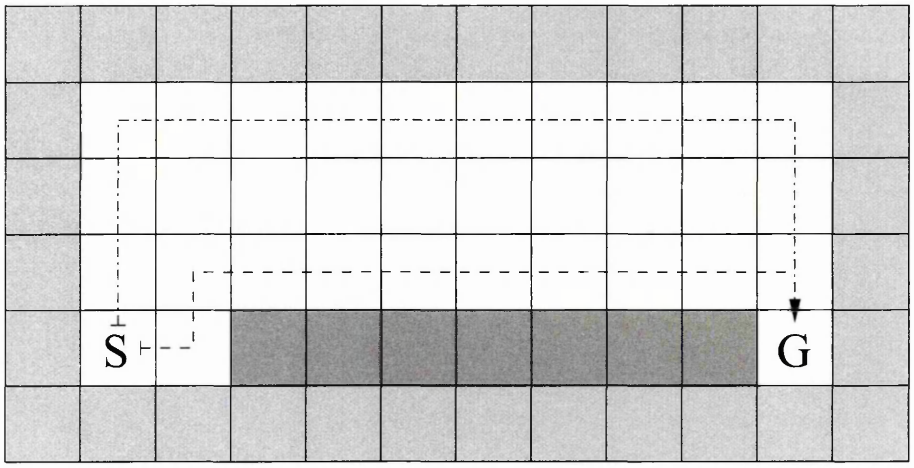


Figure 4.16: Cliff walk environment (after Sutton and Barto [36]).

The environment was defined in a similar way to the maze from the previous exercise. It consisted of a set of white, light grey and dark grey squares (see Figure 4.16). The agent was allowed to move around the white area. The light grey colour represents the walls. They mark the area within which the agent can operate. The major difference to the previous maze lies in the presence of the dark grey section. Those squares mark the edge of the cliff. The agent is not allowed to move around the light grey squares but can freely step onto the dark ones. However this will result in the "sudden death" condition followed by the reward "-100" and the agent being sent back to the starting point. For every other move the agent receives the reward of "-1" except when it reaches the goal which is granted with the reward of "0".

The existence of cliff alongside the simple geometry of the maze gives the agent an opportunity to develop two strategies to achieve its goal (see Figure 4.16). The optimal policy, represented by the dashed line, is the quickest but the most dangerous approach. The agent walking along the cliff edge takes a high risk of stepping over it during the random action-selection move. On the other hand, the safe trajectory (represented by dashed and dotted line), although longer, can still produce better results. It is because the agent has a good chance to recover after random, exploratory action.

Every episode was started from the square described by the letter "S". The goal was implemented as a class "14" and described by letter "G". To make implementation easy an additional white square was added one step to the East from the starting position. This eliminates the ambiguity of having the START and GOAL defined by the same state number. The list of states is identical to the one from the previous simulation and can be seen on Figure 4.13. The set of actions is also similar and includes one step movements in four directions: North, East, South and West.

The agent was programmed to perform 1000 episodes. For the clarity of results, the moving average window with the size of 10 was applied on the output data.

Both algorithms were simulated using the same learning parameters. The step size (α) together with discount ratio (γ) were set to the value of 0.1. The ϵ-greedy action-selection method was used with the parameter ϵ set to the constant value of 0.1. This will prevent the SARSA algorithm converging to the optimal policy (see Figure 4.14 for comparison).

The optimal and safe policies presented on the graphs below indicate the maximum value of the reward per episode. It was derived from the maze model (see Figure 4.16) counting the number of steps for each trajectory. This can be followed by the agent only when the relevant policy has been learnt and no exploration takes place (greedy action-selection method).

The figure below presents the results of the on-line performance of SARSA reinforcement learning method.

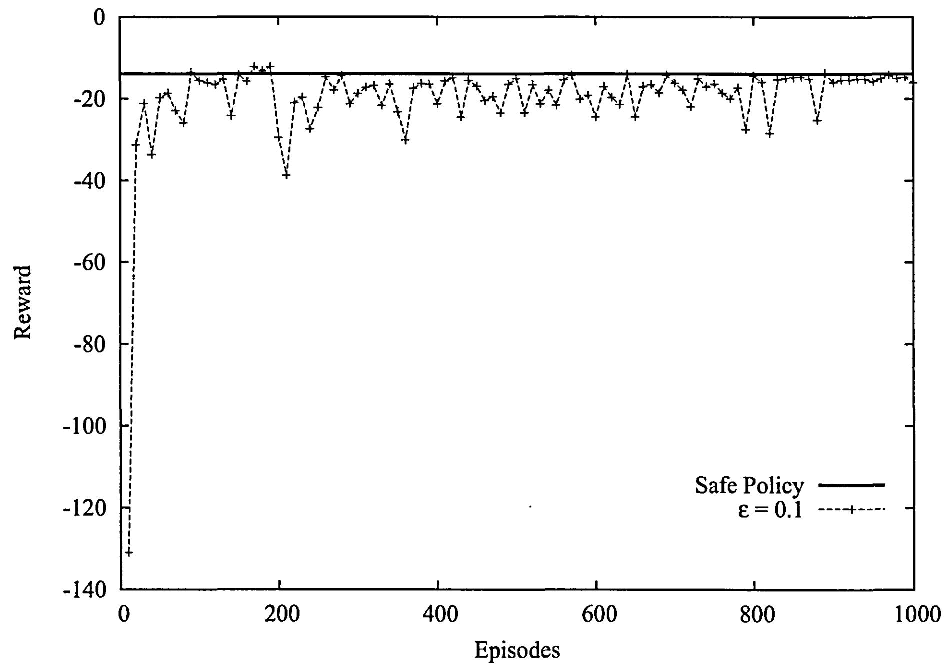


Figure 4.17: The on-line performance of SARSA method .

It can be seen from Figure 4.17 that the agent with the SARSA algorithm implemented quickly learnt (after 30 episodes) the safe policy. The good on-line performance and stable rewards against the episodes graph indicates that the number of times the agent fell off the cliff was low.

After initial transition the agent stabilised its rewards just below the value of safe policy (which was "-14" in case of environment from Figure 4.16). It is due to an action-selection policy with a constant value of ϵ which makes the exploration-exploitation balance unchanged throughout the learning time. The agent, despite of having learnt the safe policy (see Table 4.2), was executing random, exploratory actions which often caused collisions with the walls lowering the value of cumulative rewards. Those actions, although wrong, affect the overall performance less than the movements causing the agent to step over the edge of the cliff.

The amount of failures per ten episodes is shown on Figure 4.18. "Failure" is defined as an action which causes the agent to step over the edge of the cliff. This, due to high value of the reward (-100) followed by the premature end of the episode, significantly lowers the agent's on-line performance.

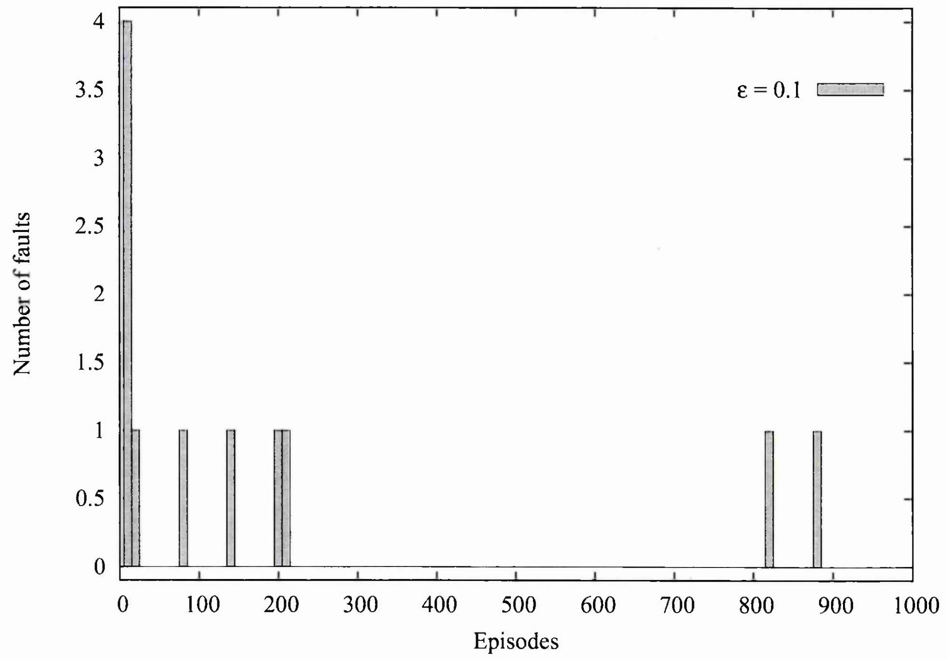


Figure 4.18: The number of the agent's failures using SARSA algorithm.

In the Figure above, each block represents the actual number of times within a 10 episodes window that the controller failed to accomplish the task. This only includes the agent stepping over the cliff. The maximum number of steps was set to the high value of 1000. The agent never exceeded this and all terminated episodes were caused by the "sudden death" condition.

After a short transition time the SARSA agent rapidly learnt to avoid falling over the cliff's edge. It is important to notice that the action-selection policy was taken into account during the learning process and the safe path from START to GOAL was chosen. This is a very important feature of the SARSA reinforcement learning algorithm. Considering the action-selection policy during the knowledge acquisition and decision-making processes makes SARSA a member of the on-policy algorithms family.

The two failures, experienced towards the end of the learning process (episodes 820 to 830 and 880 to 890), could be caused only by an exceptionally "bad" set of random actions. The agent had already developed a safe policy from START to the GOAL and only the combination of exploratory movements could distract the controller from reaching the goal.

The q-learning algorithm was implemented and tested under exactly the same conditions as in the case of the SARSA method. The learning ratios (α and γ) were set as 0.1 together with ϵ-greedy parameter (ϵ). The on-line performance after 1000 epochs was presented on Figure 4.19 below.

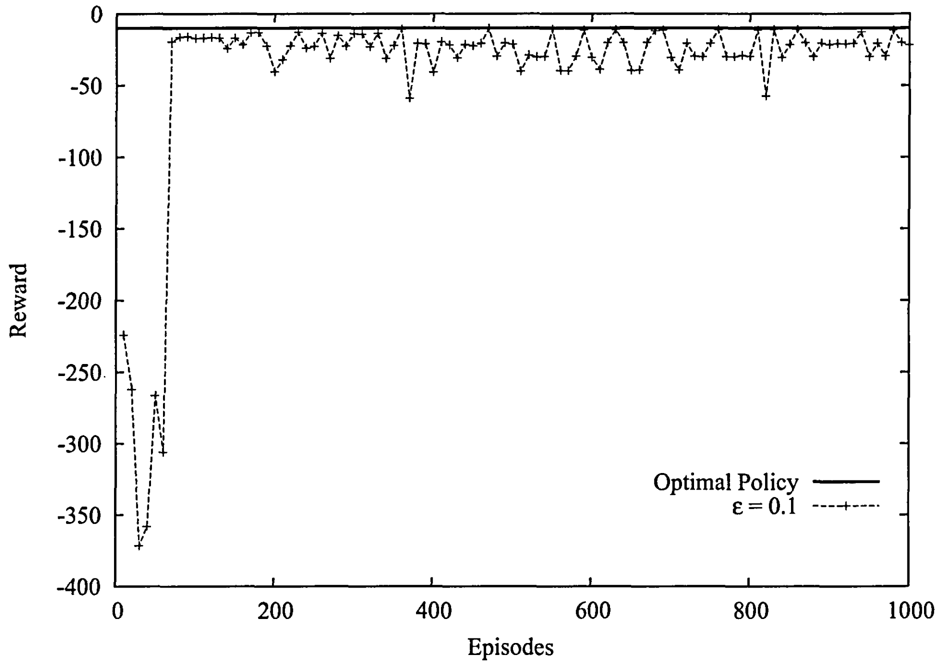


Figure 4.19: The on-line performance of q-learning method.

The transition time was 60 episodes long and was much longer than in the case of the SARSA method where the controller was able to successfully accomplish the task after only executing 10 trials (see Figure 4.18 for comparison). The values of rewards are also much lower reaching the average of -371 for the episodes 30 to 40 (the lowest value reached by the SARSA method was registered as 131). This indicates that the controller was struggling at the early stage of the learning process.

Later the rewards stabilise just below the line marking the optimal reward (-10 in this example). The stable cumulative reward curve means that the knowledge about the environment has been gained and finally the state-action relationship was derived. The "noisy" shape is caused by the agent performing exploratory actions (like in previous cases, the value of ϵ was constant throughout the simulation).

The number of failures the agent experienced during the simulation is presented on Figure 4.20 below. Like in the previous case, the grey blocks represent the abnormal termination of the task caused by the agent falling off the cliff. The height represents the sum of failures per 10 episodes.

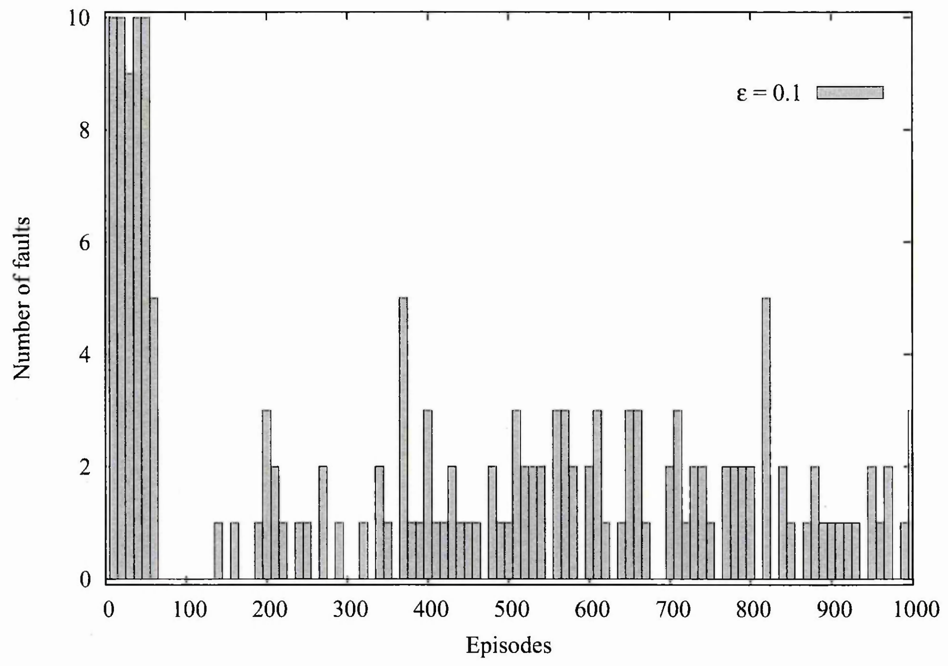


Figure 4.20: Number of the agent's failures using q-learning algorithm.

It can be seen from the Figure 4.20 that the number of failures is much higher than in the case of SARSA (see Figure 4.18 for comparison). It is caused by the fact that the q-learning method, due to its design, learns an optimal policy. In the case of the cliff-walk environment it inevitably means falling off the cliff during the exploratory movements. This significantly affects the agent's on-line performance.

The q-learning agent did not accomplish the task 167 times during a 1000 episodes simulation. This is a far greater number compared to the SARSA controller which failed only 11 times.

The learnt state action relationship is presented in Table 4.2. The safe and optimal policies were derived by the author from the relevant trajectories presented on Figure 4.16.

State	Learnt Policy			
	SARSA		Q-learning	
	Simulated	**Safe**	Simulated	**Optimal**
0	N	–	E	–
1	E	**E**	E	–
2	N	**N**	E	–
3	E	**E**	E	–
4	S	**S**	S	**S**
5	S	**S**	S	**S**
6	N	–	N	–
7	N	–	N	–
8	N	–	E	**E**
9	N	–	N	–
10	N	**N**	E	**E**
11	N	–	N	–
12	W	–	N	**N**
13	N	–	N	–
14	**G**	**G**	**G**	**G**
15	N	–	N	–

Table 4.2: Learnt policies for SARSA and q-learning methods.

Only the states which can be experienced during those walks were analysed. Although the agent, during the exploratory movements, could experience and learn the actions for all possible states. The redundant states for each policy are marked as "–".

It is clear that both, SARSA and q-learning agents learnt the expected policies. This means that assuming greedy action-selection method the SARSA controller will follow the safe path to GOAL and the q-learning agent will walk alongside the cliff.

The fact that q-learning learns the optimal policy independently of applied action-selection method is an important feature and the algorithm is classified as an off-policy learning method.

The comparison of the on-line performance of SARSA and q-learning methods is presented on Figure 4.21. For the clarity of presentation the Bezier curve algorithm was applied on the simulated data.

The SARSA agent quickly learnt the policy and converged to the reward value corresponding to safe trajectory. The transition time in the case of the q-learning controller is much longer and the rewards are much lower compared to the SARSA graph. This indicates a huge number of failures during the initial stage of the simulation (see Figure 4.20 for comparison).

After the transition time both methods converge to the similar value of reward. Although it cannot be assumed that the algorithms performed in similar ways. The q-learning agent with the greedy action-selection method would perform much better than SARSA. Due to random actions executed in the dangerous zone the controller receives poor rewards from the environment.

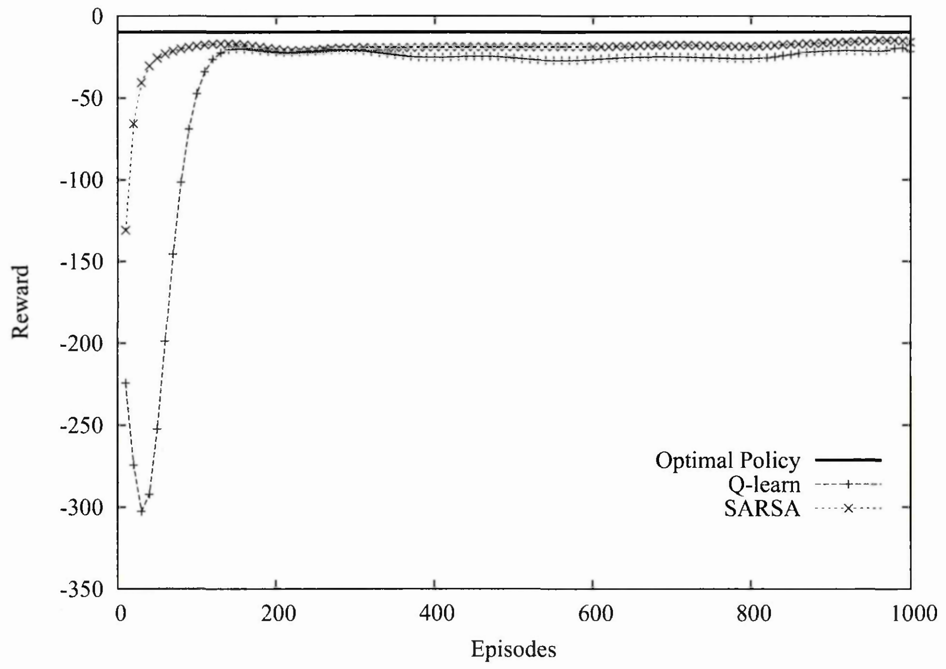


Figure 4.21: Comparison of q-learning and SARSA algorithms on-line performances.

4.5.2 State classification method evaluation

The ART2 algorithm was chosen for purpose of this project. It differs from ART1 structure only in the type of the input patterns. The price for that additional capability is an increase in complexity on the F_1 processing level which contains a number of sub-layers that serve to remove noise, enhance contrast and to normalise analogue input pattern. The overall structure of the ART2 network is shown in Figure 4.9. This complicated approach allows inputting to the controller raw, unprocessed data from the F/T sensor.

Choosing to partition the state space, it was divided into a set of classes using information extracted on-line from the part geometry. The author's approach significantly differs from the other researchers [5–7] where the geometrical model was supplied and extracted from the CAD drawing. In this research the geometry is learnt on-line in an unsupervised manner using only force and torque signals. For this purpose, the ART network was used. As described before (see Section 4.3.2), ART is a fast classifier with the ability to respond quickly to previously learnt data and remains able to learn when novel data is presented (stability-plasticity dilemma).

The author's implementation is based on the algorithm described by Ferrman and Skapura [16]. After the initial tests the problems with data stabilisation on the F_1 layer occurred and a few modifications had to be made (see Appendix D).

The contact between peg and hole can be modelled with the Newton equations for the rigid body. Furthermore, representing the insertion as a quasi static transition of states simplifies the equations greatly, removing them from the time domain.

In Brignone's controller design the network was presented with a set of unary vectors normal to the surfaces forming the hole. This helped to distinguish and assign the class number to the geometry feature. The idea of using unary vectors to describe orientation of planes in 3-dimensional Cartesian space is well known and was applied early by researchers [5–7]. However, the author does not consider network pre-training to be necessary. Omitting the geometry information given by an expert during the first stage of the learning process will not affect the controller's ability to learn, dealing with particular plane.

The raw F/T signal was set as an input to the network. The signal preprocessing was not necessary since normalisation and contrast enhancement is performed by the ART2 network. After successful classification of the state description in the form of class number was sent as an input to the learning agent.

To test the implementation two sets of data were prepared by the author. First a number of supervised insertions was performed to collect the force and torque signal samples.

The 20224 vectors were collected during that task. In the second stage, five random signals were chosen out of the main set of data. These vectors were used to test the stability of classification by introducing them at the network input during the learning process. This should happen every 1000 epochs.

The ART2 is an unsupervised, self-organising type of neural network. Due to it design it is hard to distinguish between the learning and working stage of the network. The network updates its weights always when the data vector is presented on the input so the learning takes place all the time and should not be switched off during the simulation. This will inevitably lead to some reorganisation in the classes structure.

As stated in Section 4.3 the reinforcement learning methods require clear state description for proper operation. For this reason, tuning the learning parameters (especially ρ in the case of ART) is an important task during the controller design. The class division should be fine enough to be able to pick up all relevant geometry features. However, if the number of states rises too high it will significantly affect the controller's performance since the agent has to learn the best action for each generated class.

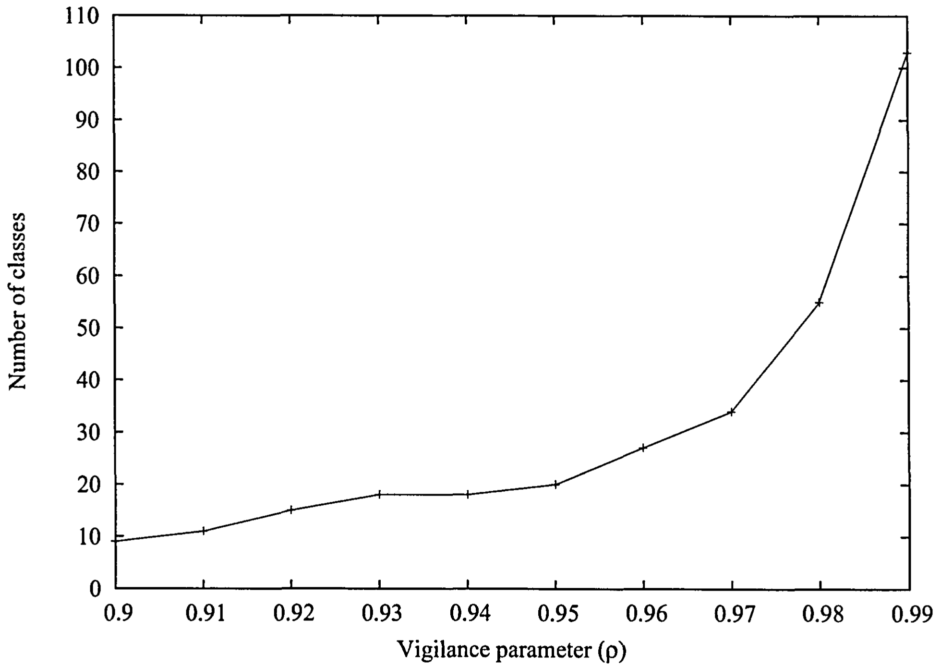


Figure 4.22: The vigilance parameter against the number of generated classes.

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	4	5	10	13	7
2000	4	5	6	3	7
3000	4	5	6	3	7
4000	4	5	6	1	7
5000	4	5	6	1	7
6000	4	5	6	1	7
7000	4	5	10	13	7
8000	4	5	10	13	7
9000	4	5	10	13	7
10000	4	5	10	13	7
11000	4	5	10	13	7
12000	4	5	10	13	7
13000	4	5	10	13	7
14000	4	5	10	13	7
15000	4	5	10	13	7
16000	4	5	10	13	7
17000	4	5	10	13	7
18000	4	5	10	13	7
19000	4	5	10	13	7
20000	4	5	10	13	7

Table 4.3: Classification stability of ART2 algorithm ($\rho = 0.95$).

When the number of clusters is too small the difference between them becomes significant. If the reorganisation of classes occurs (e.g. at early stage of learning) the reinforcement learning agent will have to follow it and retrain. This is a lengthy process and will eventually lead to low on-line performance.

The graph from the Figure 4.22 presents how the number of classes changes with the vigilance ratio (ρ).

A number different values of vigilance (ρ) were applied and tested on the described early data set. As expected (see Section 4.3.2 for details), the ART network became more sensitive with the increase of vigilance ratio. It means that the same data set will be classified into a greater number of states.

The stability of learning is the great advantage of the ART family algorithm. When an unknown pattern is presented to the network it will learn it without forgetting previously acquired information. Unlike the backpropagation neural network, the ART does not need to be retrained with the arrival of an unseen vector. The learning process takes place throughout the application lifetime. It gives an opportunity to build an independent self-adapting system although there is a danger of class reorganisation. This results in the same vector being assigned to different classes throughout the simulation.

The network class numbers returned by ART2 for five random vectors are presented in Table 4.3. It could be seen that vectors "A", "B" and "D" were consistently recognised and assigned to the same state. The exception is episode one when no pre-learnt knowledge is present and all the weights are set to zero.

During the classification of vectors "C" and "D" the state number was changing and the output stabilised after 7000 epochs. This reorganisation could be disadvantageous especially if accompanied by low values of vigilance. However, with the increase of ρ the contrast between classes is reduced so there is a good chance that the best responses of reinforcement agent for classes "6" and "10" (for the vector "C") are exactly the same. In this case the controller will not have to relearn its actions even if the reorganisation occurs.

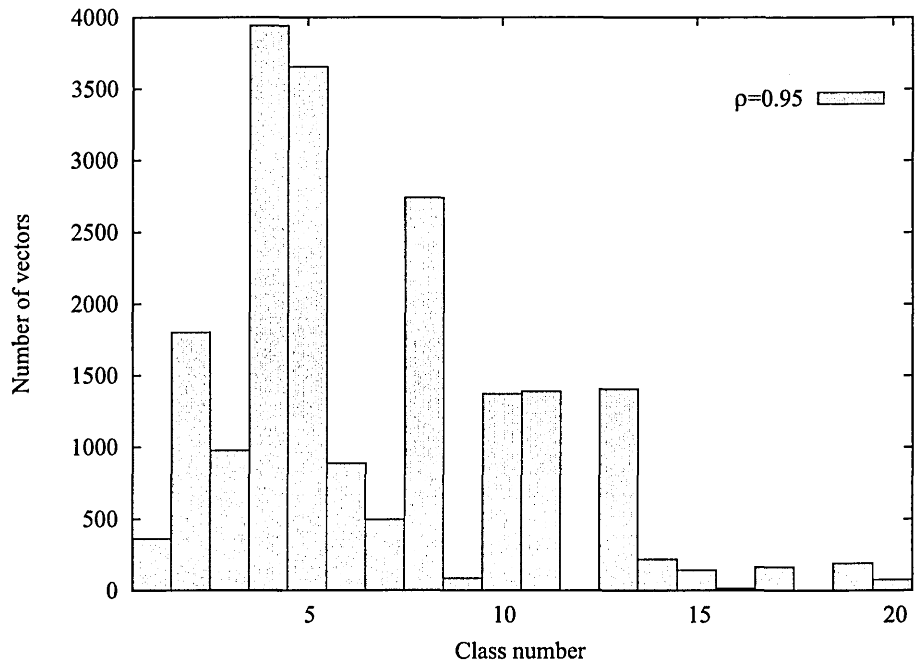


Figure 4.23: State classification using ART2 histogram.

The histogram from Figure 4.23 shows how many vectors were classified to the particular classes. Twenty different states were generated by the ART network for ρ=0.95 during the simulation. It could be seen that the classes "6", "10 and "13" were amongst the most popular. This indicates that those states are well known to the system and the reorganisation in case of vectors "C" and "D" should not affect the reinforcement learning process.

The tables featuring the number of generated classes for given ρ as well as more histograms can be found in Appendix A.2.

4.6 Discussion

In this Chapter the methodology of the artificial intelligence controller design was presented. The proposed sandwich structure divides the learning process into two main subtasks: the decision-making agent and state recognition subsystem. This allows the application of different artificial intelligence methods to learn the assembly tasks.

The reinforcement learning algorithms were chosen by the author as the decision-making part of the controller. They were used by the researchers [4,12,18,24,28] to learn peg-and-hole assembly. All approaches presented in Chapter 2 differ in methodology and choice of learning methods but share some common disadvantages. The poor on-line performance or high level of supervision are the main reasons for the learning process being long and often unstable, or quick but lacking the self-adaptivity.

The choice of the information sources is also an important issue. Many different types of data could be applied to help the learning process. The geometric representation of mating parts extracted from CAD drawings [4], or peg orientation information [18] are amongst the various information sources utilised.

Reinforcement learning methods, due to the iterative learning nature, are the most appropriate for solving automated peg-and-hole assembly problems. They learn by interaction with the environment and make decisions based on the current state and the reward from the environment. This means learning from experience by capturing the most significant aspects of the environment to solve the problem.

The significant difference to the previous designs is the fact that the controller addresses the whole problem instead dividing it into a series of subtasks. This is also an important feature of reinforcement learning approach.

The way of combining different signals to produce a good state description is very important for knowledge acquisition. The states that feature the Markovian property are ideal for learning. They include the past experiences and so all important information for the future decision-making process is retained. The fact that past sensations are part of a current state makes the decision-making process history independent. All that matters for the future is included so the agent can predict the next state (s_{t+1}) and reward (r_{t+1}) based on current state (s_t) and the action (a_t).

In the proposed design the hole's geometry is divided into set of classes. All that matters for the future is the current, sensed force and torque signal classified by the ART network. This allows the agent quick, history independent, learning. The controller optimises the reward function and so the number of steps to achieve the goal. This also should involve a significant reduction in contact forces as the number of episodes increases.

Adaptive Resonance Theory solves the learning instability problem suffered by standard feed-forward networks. The weights which have captured some knowledge in the past continue to change as new knowledge comes in. There is, therefore, a danger of losing the old knowledge with time. The ART neural network solves the problem by introducing the *resonance* which can be attained in two ways. If the network has learnt to recognise the input pattern before then the resonant state will be achieved quickly and the memory of the stored pattern will be reinforced. If the input vector is not recognised the network will look for stored patterns in the database. If no match is found the network will enter the resonant state and the new pattern will be stored for the first time.

This approach gives an opportunity to develop an independent, unsupervised, and what is important, self-adapting state recognition module. As stated before, for purpose of this project the state signal was built from force and torque signal. With a change of the geometry the new set of classes is introduced to the controller. The ART with stability-plasticity problem solved is the best choice to quickly classify the complex environment.

The applied ART2 algorithm accepts analogue data so the preprocessing was not necessary. It is also a fully scalable approach since more additional information can be supplied to the system and processed by the clustering module to produce a state signal for the reinforcement learning agent.

According to the author, the force signal is the only necessary source of information about the environment required to perform successful peg-and-hole assembly. Moreover, just the direction of the acting forces (and resulting torques) carries enough information to perform a successful assembly. The actual value of force is not essential to accomplish the task e.g. the human is able to insert a door key to the hole without vision or the knowledge of the exact force values. The proposed controller design is expected to be fast and reliable using just the necessary feedback signals.

For the purpose of this project, the chosen algorithms were implemented by the author and tested on the simulated environment. This provided an opportunity of understanding the learning ratios and to test different programming approaches. It was noted, during tests of ART2, the algorithm sometimes was not able to stabilise data on the F_1 layer, modifications to Ferrman and Skapura's [16] implementation had to be made. The F_0 layer was added to the previous configuration and the equations from Section 4.3.2 had to be modified. The final relationship that governs activities on each of the sub-layers can be found in Appendix D.

The different action-selection policies were also analysed. It was clear after simulations that the exploration-exploitation balance is very important and can significantly affect the on-line performance of the algorithms. In classic (stationary) peg-and-hole experiments, where the geometry of parts does not change throughout, the best solution is to reduce either ϵ or β from episode to episode. This guarantees the agent will explore less and use pre-learnt knowledge instead. It is assumed that together with the number of past episodes the experience and knowledge about the environments will rise.

After close analysis of the results from the simulation tests it is clear that the on-line performance of both SARSA and q-learning methods are similar. The SARSA agent quickly learnt and stabilised onto the safe trajectory. However, the q-learning controller's performance was worse. It is caused by the fact that in this case the optimal trajectory was learnt. This inevitably causes the agent to fall off the cliff after executing the random (exploratory) actions. It is important to note that the q-learning agent would produce better results if the change of exploration-exploitation balance was applied (after the number of episodes no exploration is allowed).

The state clustering module was tested on force vectors acquired from previous experiments. The results show that the number of classes could be controlled with the vigilance ratio ρ. The ART2 network responded to the changes of ρ as expected: the number of classes was rising from 10 to 104 as the vigilance approached the value of 1.0. The data was acquired from the circular peg-and-hole experiments. Brignone in his research [4] proposed a 16 classes division for the circular peg problem. This includes the 8 classes for separate planes forming the hole and 8 classes called "virtual" for the edges.

It was clear that with the value of ρ=0.93 and ρ=0.94 applied the best classification for this type of geometry was secured. The network produced a stable clustering with 18 different classes in both cases. The higher value of vigilance would result in the greater number of classes which makes the state clustering module more sensitive. However, too many states will significantly extend the learning process. The reason for that lies in the second layer of the controller (the learning agent), which needs to develop a good strategy for every, sensed state. So, the greater number of classes produced by ART2 means more states to learn and poor on-line performance during the initial stage of the insertion.

It is also important that the number of classes generated by the ART2 network was not too small. In this case the controller, due to insufficient information about the environment, would not be able to learn the optimal strategy to achieve the goal.

This balance is essential for quick learning. The value of vigilance ρ was investigated and setup by the author at the value of 0.94. However, this supervisory influence does not affect the unsupervised nature of the controller. The data reorganisation and learning within the clustering and the decision-making modules is fully automatic. For practical reasons, due to the extensive time of the experiments, the number of classes had to be limited by setting the value of ρ manually.

All the algorithms performed well working in simulated environments. The results show that both modules of the proposed controller are implemented well and can be applied in real life experiments. It is important to adjust the learning ratios values since they significantly affect the performance of the controller.

Chapter 5

Experimental Methodology

In this Chapter the methodology of the experiments will be explained. To verify the performance and learning ability of the porpoised controller it was applied onto real life system (described in Chapter 3). The different experiments with different part geometries were carried out to evaluate the system.

5.1 Environment evaluation

The artificial intelligence controller was designed to solve motion problems working in an unknown environment. The state description with Markovian property is the best way of describing the environment for reinforcement learning methods. However, in real life applications it is usually difficult to build such a signal. The feedback data from the environment is often noisy and inaccurate. The control signal, due to machine dynamics, also carries certain amount of error. To efficiently deal with those problems the controller needs to be able to quickly adapt to new situations and despite the changes (e.g. non-stationary environments) to derive a successful strategy to accomplish the task.

During the peg-in-hole insertion two major signals are exchanged between the robot and its environment. The feedback data is derived from force and torque signals acquired by the JR3 sensor. The control signal is sent to the robot in the form of incremental movement commands. The final response depends on the robot's accuracy, resolution and repeatability.

In the next two sections the results from experiments to measure the sensor's and robot's accuracy are presented.

5.1.1 Sensor accuracy

The JR3 sensor was described in details in Chapter 3. It was calibrated and certified by the manufacturer for a maximum load of 15 lbs on X and Y axes and 30 lbs on Z axis.

The manufacturer does not provide a device driver so an appropriate program in 'C' language had to be written. To evaluate the communications, filtering ability, unit scaling factors and the coordinate transformation procedure, a simple test was carried out.

The sensor was mounted on the Puma 560 robot's wrist. To assure that the TOOL coordinate system axis is aligned with the robot's WORLD coordinate system the "DO ALIGN" command was released from robot terminal. The sensor's internal coordinate system was aligned with the robot's TOOL CS. This was done executing a set of rotational commands (see JR3 sensor manual for details [26]). The load (value of 5 Newtons) was suspended underneath the sensor. Since the direction of WORLD Z axis is parallel (within the robot's accuracy) to the gravitational force, the expected sensor's reading should be on Z axis only.

The sensor features different types of pre-processed output data. The readings of 31Hz filtered together with the raw data were presented on the graphs from Figures 5.1 and 5.2. The presented value of force and torque are representing the calculated lengths of the relevant vector. It was calculated using the equations below:

$$F = \sqrt{{F_x}^2 + {F_y}^2 + {F_z}^2} \tag{5.1}$$

$$M = \sqrt{{M_x}^2 + {M_y}^2 + {M_z}^2} \tag{5.2}$$

The force and torque acquisition was carried out for 60 seconds. First the filtered sensor output was setup and recorded followed by raw data collection. Note, "filtered" and "unfiltered" signals are not from the same time frame - the force and torque plots below are uncorrelated.

It can be seen from the graphs below that filtering the data affects the magnitude of measured signal. The amount of noise during the force reading is significantly reduced with a 31Hz filter applied. The average value is also closer to the expected value of 5N than in the case of raw data reading.

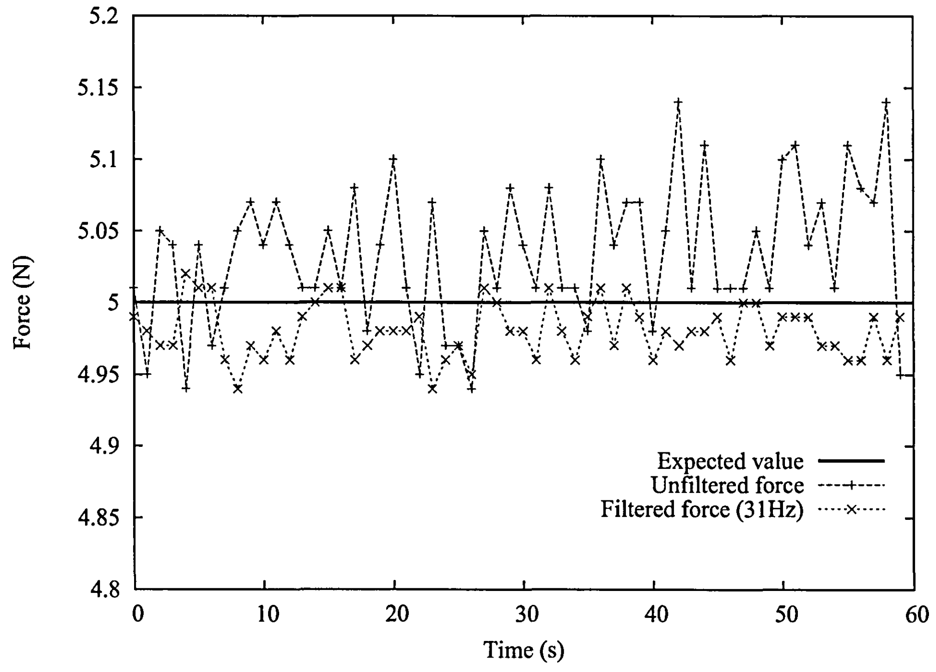


Figure 5.1: Force results for 5N load applied on the sensor (uncorrelated signals).

The torque results are presented on the graph from Figure 5.2. There is much greater difference (average of 0.1 Nm) between filtered and unfiltered signals. The load is suspended directly underneath the sensor so the torque readings should be close to the value of 0.0. However the design of the load suspension system can cause small values to appear on X and Y axes. The only true torque free axis in this configuration is Z (see Figure B.9 in Appendix B).

According to the manufacturer, the sensor's accuracy was tested as a part of a quality assurance system so the author decided not to undergo the complex experiments to establish the resolution and reading errors. It was assumed that the values from the manual and calibration certificate are correct, accurate and up to date.

All measured values prove that under certain circumstances (e.g. wrong filter settings) the feedback signal can be noisy but never misleading. The adaptive controller needs to be tuned so it can extract the relevant information from the force and torque array and classify the signal correctly. In the presented case analysing just the values of acting forces one can deduce the applied load conditions. Moreover, having the information about the values acting on each of the three axes (see Figures included in Section B) significantly increases the chances for successful classification.

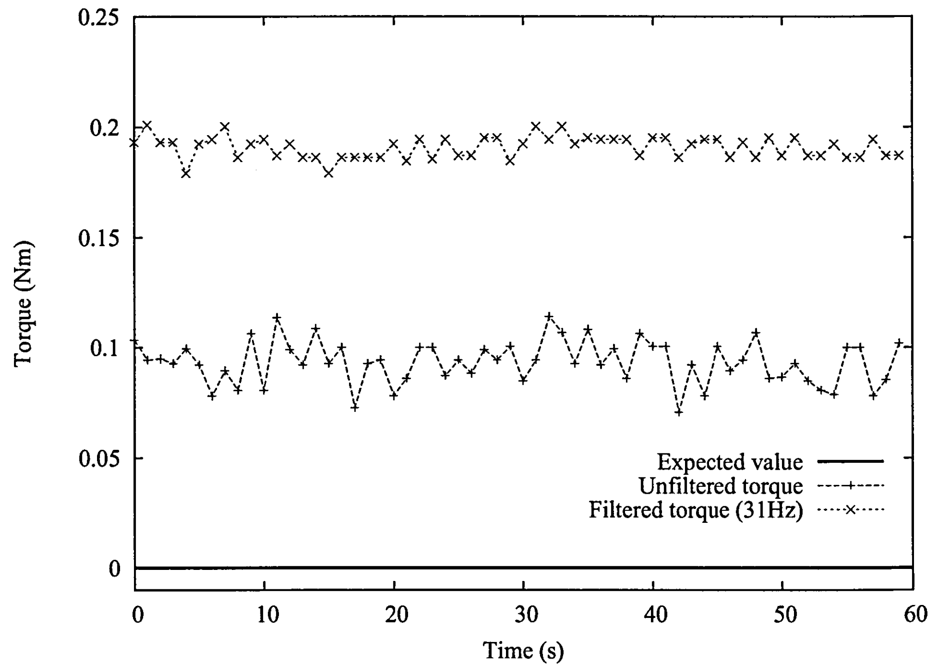


Figure 5.2: Torque results for 5N load applied on the sensor (uncorrelated signals).

The detailed graphs for each vector component (F_x, F_y, F_z, M_x, M_y, M_z) can be found in Appendix B.

5.1.2 Robot positional accuracy

The robot and its integral computer system are the main parts of the learning agent's environment. The incremental motion commands are sent to the Puma's control system. Here the inverse kinematics calculations are performed and set of positions and velocities instructions are sent to all six servo actuators.

The arm accuracy is limited by many different, and usually independent, factors. The

No.	DTI gauge reading [mm]					
	X-0.2			X+0.2		
	X	Y	Z	X	Y	Z
1	-0.21	-0.01	+0.012	+0.21	+0.02	-0.014
2	-0.16	+0.00	-0.038	+0.22	+0.02	+0.012
3	-0.19	+0.09	-0.018	+0.15	+0.01	+0.018
4	-0.19	+0.00	-0.018	+0.14	+0.01	+0.020
5	-0.18	-0.01	-0.016	+0.18	+0.02	+0.022

Table 5.1: Robot incremental motion accuracy measurements for X axis.

No.	DTI gauge reading [mm]					
	Y-0.2			Y+0.2		
	X	Y	Z	X	Y	Z
1	+0.01	-0.16	-0.002	+0.01	+0.22	-0.002
2	+0.02	-0.18	-0.012	+0.00	+0.19	+0.000
3	+0.00	-0.19	+0.002	+0.00	+0.17	-0.012
4	+0.01	-0.18	+0.014	+0.01	+0.20	-0.010
5	+0.01	-0.22	-0.010	+0.00	+0.21	-0.004

Table 5.2: Robot incremental motion accuracy measurements for Y axis.

No.	DTI gauge reading [mm]					
	Z-0.2			Z+0.2		
	X	Y	Z	X	Y	Z
1	+0.03	+0.01	-0.172	+0.06	+0.02	+0.212
2	+0.03	-0.01	-0.218	+0.01	+0.00	+0.224
3	+0.02	+0.00	-0.218	+0.03	+0.00	+0.212
4	+0.02	+0.01	-0.220	-0.01	+0.01	+0.228
5	+0.04	+0.00	-0.226	-0.01	+0.02	+0.226

Table 5.3: Robot incremental motion accuracy measurements for Z axis.

main ones include: the robot's mechanical rigidity and its dynamics, joint backlash or resolution and type of the control system.

For the purpose of this project only the incremental translations and rotations are executed. However, the positional repeatability can also play a significant role since all the insertions are started from the "APR" position. It is a fixed point exactly above the entrance to the hole (see the next Section for details of the experimental design).

A set of experiments and measurements were performed to evaluate the accuracy of robot motion and its positional repeatability. As in the case of the force and torque sensor the tests were designed as a part of an environmental uncertainty estimation. They were not meant to produce a detailed output about the robot's performance. This data was supplied by the manufacturer in the user manual.

Three plunger dial gauges were used to measure the displacement. The aluminium square peg was attached to the robot's gripper. The arm was positioned so the Z axis of TOOL coordinate system was parallel to Z of WORLD CS. This also means that the axis direction is normal (within robot's accuracy) to the mounting table.

The sensors were positioned in the way so the measured distance was along the X, Y and Z axes in TOOL coordinate system. However this was done manually and some inaccuracies are expected.

First the accuracy of incremental motion was measured. The robot was setup to move 0.2mm into positive and negative directions along the X,Y and Z axes. After each move the reading was taken and the plunger gauges were reset to the value of 0.00mm. Exactly the same procedure was repeated five times for each direction on each axis.

The results from the experiment can be seen in Tables 5.1, 5.2 and 5.3.

5.2 Plan of peg-in-hole experiments

The controller, proposed in Chapter 4 was applied on a set of real life peg end-hole experiments. This type of insertions are amongst the most common operations in automated assembly. The basic 2-dimensional and 3-dimensional peg-and-hole tasks were widely analysed by the researches in the past (see Chapter 2 for further reference). Due to the geometric simplicity it is perfect environment for learning basic manipulative skills with artificial intelligence methods.

5.2.1 Initial experiments

Due to the complexity of the task and implemented algorithms a number of initial tests need to be performed. The system should be tested in its simplest form so the circular peg-and-hole set was used. This type of geometry is the easiest to learn by the controller since it does not include the torque around the Z axis.

To speed up the learning process a limited number of actions were applied. The controller can only perform the incremental translations in both directions along the X, Y axes. The insertion direction will be also supplied to the controller.

5.2.2 SARSA and q-learning on-line performance evaluation

During this part of experimental work the SARSA and q-learning algorithm's ability to learn was tested and compared.

Both algorithms with ϵ-greedy and Gibbs distribution action-selection policies were applied on the same insertion task. The circular peg with chamfered hole was used for the purpose of this experiment. The set of incremental rotations was added to the action list.

The results from the tests allowed the author to compare the performance of both learning methods. The action-selection policy and its influence was also investigated. It is important that all experiments in this part of the project are performed under exactly the same conditions.

5.2.3 Different peg geometries analysis

To test the controller's flexibility the different geometries of the mating parts were applied and tested. As mentioned in the sections before, the initial experiments as well as reinforcement learning methods' performance evaluation were carried out using the circular peg-and hole tests.

The squared peg and chamfered hole was chosen for another test. The presence of sharp corners and the torque around Z axis significantly increased the complexity of the insertion task. The same controller architecture was applied onto the system. Due to the extensive time of the experiments only the q-learning method with Softmax action-selection policy was analysed. Following the simulation's analysis presented in Section 4.5.1 and 4.5.2 it is expected to learn the insertion strategy quickly. Using the same algorithms and experiment's settings will help to compare the on-line performance of the q-learning reinforcement methods applied onto two different peg geometries.

5.2.4 Friction analysis

In this part of the research the influence of different materials was investigated. A peg with a rubber end was manufactured for this purpose. It was applied onto the chamfered circular hole. The q-learning algorithm with Softmax action-selection method was used.

As stated in Section 4.5.2 the contact between peg and hole, can be modelled with the Newton equations for the rigid body. The direction of the resulting contact force depends on the friction between the mating parts. Due to the controllers design and clear distinction between the state recognition and decision-making modules the change of peg material should not influence the reinforcement learning agent performance.

The resolution of the ART2 algorithm is governed by the vigilance ration (ρ). Assuming it is setup to the right value and the number of output units is large enough to accommodate all encountered states, no significant difference in on-line performance was expected.

The aluminium peg is far more rigid than its rubber equivalent. This may lead to problems not directly related to the friction between mating parts. If two different force and torque signals are wrongly classified the reinforcement learning agent will try to adapt to the change. This will inevitably lead to a drop in the agent's performance.

The results from these experiments will be compared with the relevant output from the experiments where aluminium parts were applied. For easy data comparison the same experimental conditions was applied as in the case of tests described above.

5.2.5 Nonlinear case analysis

Asada in his research [1,2] defines the linear and non-linear compliance for peg-and-hole automated assembly. As shown in Section 2.1 the non-chamfered peg-in-hole insertion is a good example to test the intelligent controller's ability to deal with both types of problems.

Again, to speedup the tests and ease the analysis of the results, the simple circular geometry peg was used.

5.3 Discussion

In this Chapter the empirical part of this research was planned and analysed. After performing a set of experiments the evaluation of the environment accuracy was possible.

There are three distinguishing factors which contribute to the environmental uncertainty:

- The geometry of the parts. The agent needs to learn to classify the 3-dimensional domain without any supervisory influence. At the same time the best action for each state is being learnt. This is the main reason why state clustering module stability is an important issue. It means that the same vector should always be assigned to the same class throughout the whole experiment.
- The environmental feedback quality. The signal features the errors which need to be taken into account during the analysis. The controller must be able to learn to cope with the noise and classify the the force and torque vectors accurately.
- The control signal quality. This includes all the errors associated with the robot motion. The intelligent agent must be able to correct all the inaccuracies caused by the arm movement

The intelligent algorithm proposed in this thesis was designed to be able to work with complex, 3-dimensional geometry under high uncertainty. It is clear from results presented in Figures 5.1 and 5.2 that the sensor signal is noisy but not random. Also the accuracy of torque readings can be particularly affected by filtering the data. The maximum force error reading was less than ± 0.15 N. The maximum torque error never exceeded the value of ± 0.12 N for unfiltered data and ± 0.21 N with 31Hz filter applied on the output data.

The sensory reading, due to the noisy nature, presented a challenge for the state classification module. It is essential for this part of the controller to be able to cope with the supplied feedback signal without any preprocessing applied. As stated in Section 4.3.2 neural networks build up their internal computational structures through experience. This includes the ability to classify the patterns despite the presence of noise.

The reading errors caused by reasons other than noise have to be acknowledged and compensated by both AI controller's subsystems interacting together. The ART2 and reinforcement learning are self organising and adaptive methods. Although both should be able reorganise in such a way that all systematic disturbances are easily compensated for. This excludes the random errors which cannot be easily dealt with. The reason lies in the fact that the reinforcement learning takes place after the completion of an episode (failed or successful). The issue of random errors and system recovery will be explained later in this thesis during the analysis of the results.

After analysing the data it was assumed that the quality of force and torque feedback signal is good enough to learn the strategy for the automated assembly. For safety reasons the insertion is always interrupted when the force value (calculated using equation 5.1.1) is greater than 20N. The maximum measured error was smaller than 0.75 percent of that range. However, the direction of the force and torque vectors is far more important than the actual value of it. The insertions performed with the human hand are performed in a similar manner since the brain process the force signals based on the direction of its occurrence and a simple judgement of the increase or decrease of the sensation.

The learning agent affects the environment by sending incremental motion commands to the robot. The signal is later translated and executed by the controller. The manipulator's positional accuracy plays a significant role in the learning process. The large distance and direction error would cause "confusion" to the agent forcing it to retrain.

It is important to remember that the decision making module learns the state-action relationship. The same action number corresponding to two or three different arm movements would cause the controller to constantly reorganise its knowledge to adapt to constantly changing conditions. This is similar to the chess game when one or more figures can move around the board randomly, without any rules applied. This situation would inevitably cause a chaos which could be solved only by avoiding those moves.

Having the state clustering module tuned and producing the satisfactory results (stable classification) it is assumed that the robot would execute the incremental movements at least in the appropriate direction. Like in the case of force and torque signal the system, due to its design, should easily learn to correct any systematic inaccuracies. The random positioning errors can have a particularly bad effect on the controller's on-line performance.

Let us assume that the controller has learnt the good strategy for the peg-in-hole insertion. This means that for each state the agent has learnt the best estimated action to take. If the robot executes the requested move with a large positioning error there is a good chance that it will cause a collision. The interrupt condition is followed by the low reward from the environment which can downgrade the action that caused it. From the agent's perspective, the best action for the given state is no longer the favourite one. This can cause the disruption of the knowledge (only for a given state). The controller due to its ability to adapt to changing conditions can recover from such a condition but at the cost of the on-line performance which will inevitably drop.

The results presented in Tables:s 5.1, 5.2, 5.3 show that the robot's positional accuracy on all three axis is fully acceptable for peg-in-hole assembly. The direction of the commanded movement was maintained. The robot showed a level of inaccuracy in the distance which can be compensated for by the learning agent.

As stated many times in this thesis the controller was designed and built to be able to work with environments featuring a high level of uncertainty. The set of experiments was designed to evaluate this ability.

At the beginning of each experiment, the learning agent does not know anything about its environment. It is essential to acknowledge that no advance knowledge is supplied to the system. The supervisor sets up the learning ratios and decides about the direction of the insertion. However, this is only done to speedup the learning process.

Considering the geometrical complexity, signals inaccuracies and noise it is clear that the intelligent controller faces a difficult task to learn the assembly strategy. All the mentioned factors are part of the real world and form the difficult environment featuring high levels of uncertainty.

A set of experiments were planned to evaluate the controllers performance. Different aspects of learning were investigated. The empirical part of the thesis includes the investigation of the effects of different learning methods applied on the same geometry. The influence of action-selection methods onto controller performance was also analysed. The materials with different friction were applied as well as non-linear insertion case.

The on-line performance of different learning methods combined with the ART2 state clustering module was also investigated. To save the experimental time it was decided that during the tests the insertion task would always start from the same position and orientation. It can be proved that if the reinforcement learning agent can develop successful strategy from the given START point it can be done from any (random) position [36]. This approach allows the author to compare the results from initial stages of the insertion.

The force and torque sensor was mounted on the robot's wrist. All the incremental movements were related to the TOOL coordinate system. Because of the directions of the main axis (TOOL's and sensor's coordinates) the intelligent agent can learn the insertion even if the hole's central axis is not vertically aligned. The different orientations of the mating part should not affect the performance and learning abilities of the controller. Also the orientation of the force and torque sensor axes is irrelevant from the agent's point of view. As stated before, to be able to learn the assembly strategy the only requirement is for the sensor's axes to be parallel to the robot's TOOL coordinate system. The state clustering system treats each force vector as a pattern so the direction of X, Y and Z can be random. The force signal can be post-processed in such a way so the resulting vectors comply with the Newtonian model of contact forces.

The main purpose of experimental investigation is to evaluate the usefulness and applicability of the sandwich structure of the controller. The author claims that the combination of different, specialised artificial intelligence methods can produce a fully independent, self-adaptive and unsupervised controller.

Chapter 6

Results analysis

The techniques described in the previous chapter have been combined into an intelligent assembly architecture. The aim of this research is to provide a stable system which has a self-adapting capability, so that assembly operations can be performed in an environment under extreme uncertainty. The flexibility of software design was also an important issue.

The chapter is organised into sections where the results from specific experiments are presented.

6.1 Initial experiments

The circular peg-and-hole set was used for the purpose of initial tests. The number of actions the reinforcement learning controller can execute was restricted to 5. Only the positive and negative translations along the X, Y axes were allowed. The other actions, including the rotations about the X, Y and Z axes, had been disabled. The insertion direction was enforced by disabling all the movements on the negative Z direction.

This setup, however simple and limited, allows the author initial analysis and algorithm tests. It should be emphasised at this point that so far both subsystems were tested and simulated separately. The purpose of these tests was to test the stability of the algorithms working together and to adjust the learning ratios. Unlike the main experiments the initial tests were run for only 500 episodes. The one step q-learning algorithm was applied with ϵ-greedy action-selection policy. The value of ϵ was constant throughout the experiment and set to the value of 0.1 (greedy).

After a number of runs the algorithms proved to work efficiently and significant improvement in on-line performance was noted.

It can be seen from Figure 6.1 that during first 50 episodes the controller showed constant improvement of its on-line performance. The experiment was started without any pre-training so the system had no experience and knowledge about the environment. The first trials failed due to an excess of contact forces. During that time the controller was exploring the environment and learning the necessary manipulative skills to successfully accomplish the insertion

It can be assumed that after 50 episodes the controller managed to gain the necessary knowledge to insert the peg into the hole. The constant improvement of the performance was registered. It stabilised at the reward value between -100 and -80.

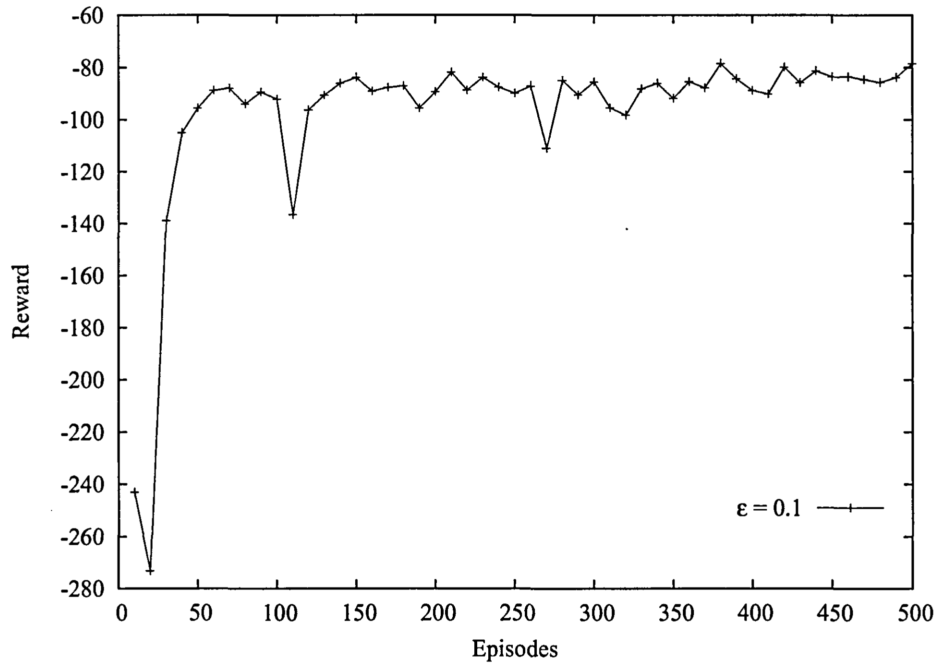


Figure 6.1: On-line performance of the controller during the initial tests.

The sudden drop in knowledge (episodes: 110 and 270) was caused by exploratory moves due to ϵ-greedy action-selection policy. This behaviour is fully acceptable especially with regards to the fact that the controller quickly recovered from those failures. The knowledge about the environment was maintained and stable operation was resumed.

The initial period, when the controller acquires the basic skills, plays a significant part in the learning process. The apparent drop of the on-line performance curve from the value of -243 to -273 after the first 20 episodes does not always correspond to the loss of knowledge. As explained in Chapter 4 the learning agent receives a reward of -1 for every action resulting with the occurrence of the contact between peg and the hole. However, in the case of a collision an extra penalty of -100 is applied. During the very first insertion attempts the controller usually fails after just a few moves. This is the major reason why the values of cumulative rewards are relatively low. The graph representing the controller's on-line performance (see Figure 6.1) should always be analysed against the histogram from the Figure 6.2 below.

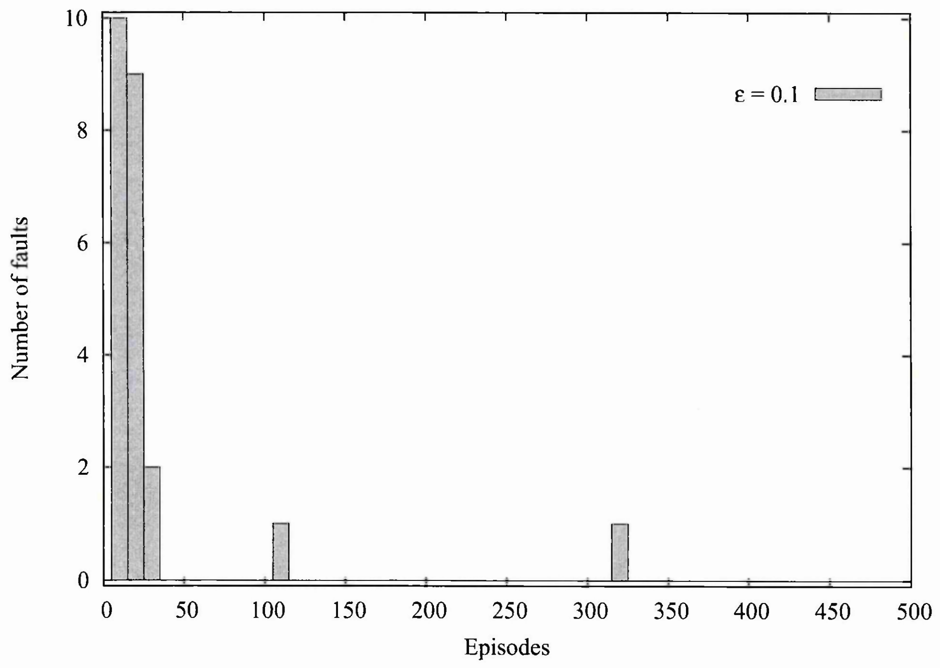


Figure 6.2: Number of failed insertions throughout the experiment.

As expected, during first 40 episodes the controller was unable to successfully accomplish the insertion. Due to lack of knowledge abut the geometry of the parts, position and orientation of the peg it was impossible to take a logical decision about the next action. This inevitably caused the execution of movements towards the wall and an increase of contact forces value.

The moving average window size of 10 was applied on the data from Figure 6.1. It could be seen that during first 10 episodes all insertion failed and were interrupted due to excessive contact force value. However, the intelligent agent was constantly improving its on-line performance as the number of insertion increased. This suggests that the learning process was taking place even during the interrupted (failed) trials.

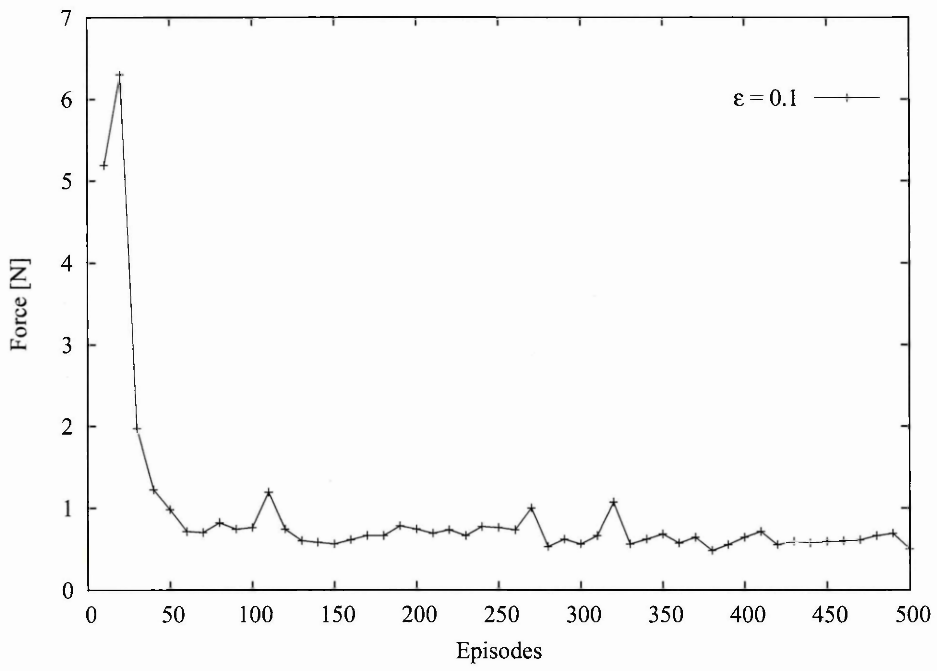


Figure 6.3: Force value throughout the experiment.

The on-line performance graphs and histograms are important tools in helping to assess and analyse the artificial intelligence algorithms. However, considering the peg-in-hole application the value of contact forces plays as important role in speed of learning during the knowledge acquisition process. It is obvious that ideal controller should be fast and able to develop the strategy of insertion minimising the acting forces between mating parts.

The graph from the Figure 6.3 presents how the values of contact forces changed with the episodes. The values were calculated using all three F_x, F_y and F_z components of force vector using the Equation 5.1.1.

The acting forces during first few episodes were very high. The experiment was set up in such a way that when the value of contact force exceeded 5 newtons the insertion was immediately stopped. Analysing the graph from Figure 6.3 it is clear that the force data directly corresponds to the number of insertion failures presented earlier (see Figure 6.2).

The value of contact force was gradually reduced as the experiment progressed. This, very significant fact, together with on-line performance improvement demonstrates the controller's ability to learn and gather the relevant knowledge to develop an insertion strategy.

After the successful accomplishment of the initial tests it was decided to apply the proposed intelligent controller onto more complicated tasks involving the investigation of different geometries of the parts as well as different mating parts' material and, finally, different reinforcement algorithms.

6.2 Chamfered, circular peg-in-hole insertion

In this section the results from experiments using a 45 degrees chamfered hole with a circular cross-section peg will be presented (see Figure 6.4). To compare the data from experiments every insertion was run under exactly the same conditions. The summary of the applied settings are shown in the Table B.1 (see Appendix B.2).

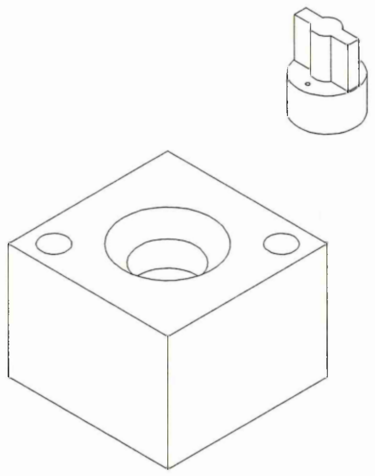


Figure 6.4: Peg and hole model used during the experiments.

Different peg materials and geometries were tested and analysed for purpose of this thesis. In the sections below the results from experiments using the aluminium components are presented.

6.2.1 Aluminium components

All the data presented in this section's graphs were post-processed in exactly the same way. This approach allows direct comparison between different methods and algorithms. The starting conditions for all peg-in-hole insertions were also unified throughout the experiments and were set as follows:

- The peg was positioned so its main axis was shifted 1.6 mm on X and -2.2 mm on Y direction from the hole centre.
- The angular misalignment was set up to the value of +1.8 degrees about X axis and +0.4 degrees about Y axis.
- The 0.2 mm and 0.2 degrees was applied as motion incremental step for the translations and rotations respectively.

Q-learning method performance

The detailed description of one-step q-learning algorithm used for the purpose of this experiment can be found in the Chapter 4. It differs from the other reinforcement learning methods. The learnt action-value function Q, directly approximates the optimal action-value function Q^*, independent of the policy being followed. This off-policy approach can be useful in the peg-in-hole insertion application since it allows learning the insertion strategy despite of the applied action-selection policy.

Two different action-selection methods were applied on the same q-learning algorithm implementation. Both experiments were performed under the same conditions. Comparing the graphs from Figure 6.5 and Figure 6.6 it can be seen that despite of the applied action-selection method the controller was able to to learn the insertion strategy.

The on-line performance results from the experiments involving the application of q-learning algorithm with ϵ-greedy action-selection method was presented in the Figure 6.5. During first 70 episodes the controller was learning the geometry features and force-torque relationship necessary to develop the insertion strategy. As in the case of initial experiments during that time the agent experienced a great deal of contact forces often exceeding the safe value and eventually interrupting the insertion task. The number of failed trails can be read from the histogram from Figure B.15 in Appendix B.2.

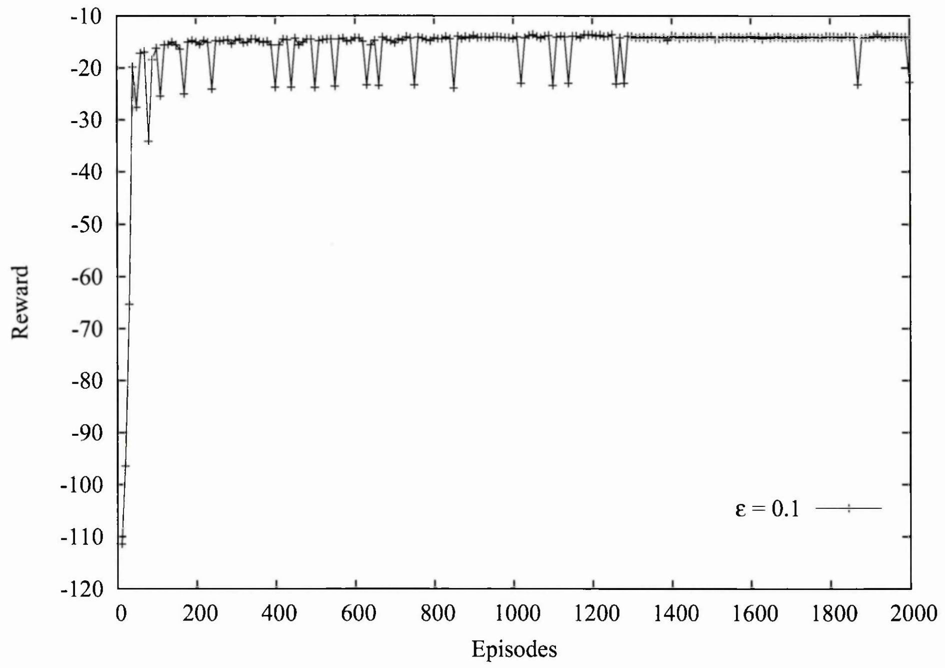


Figure 6.5: Performance of q-learning algorithm - ϵ-greedy action-selection policy.

In this case the probability of random movement is constant during the experiment. The occasional falling spikes are caused by the controller failing to accomplish the insertion. It is due to the random actions selected by ϵ-greedy policy which distract the controller from the known path and eventually cause the failure of the task. This behaviour is fully acceptable and can be minimised by reducing the amount of environment's exploration with the increase of the number of trials.

In the next graph (see Figure 6.6) data from the same experiment but with Softmax action-selection policy applied was presented. As stated in Section 4, when this method is used, actions with higher q-values are preferred. This basically means that the "good" actions have more chance to be chosen than the bad ones.

It is significant that in the case of both policies the learning agent managed to develop the strategy for the insertion. In either case the controller stabilised its on-line performance on the level of -15. Based on theoretical analysis and simulations presented in Chapter 4 it can be assumed that the insertion strategy was learnt in both cases despite of the action-selection method applied.

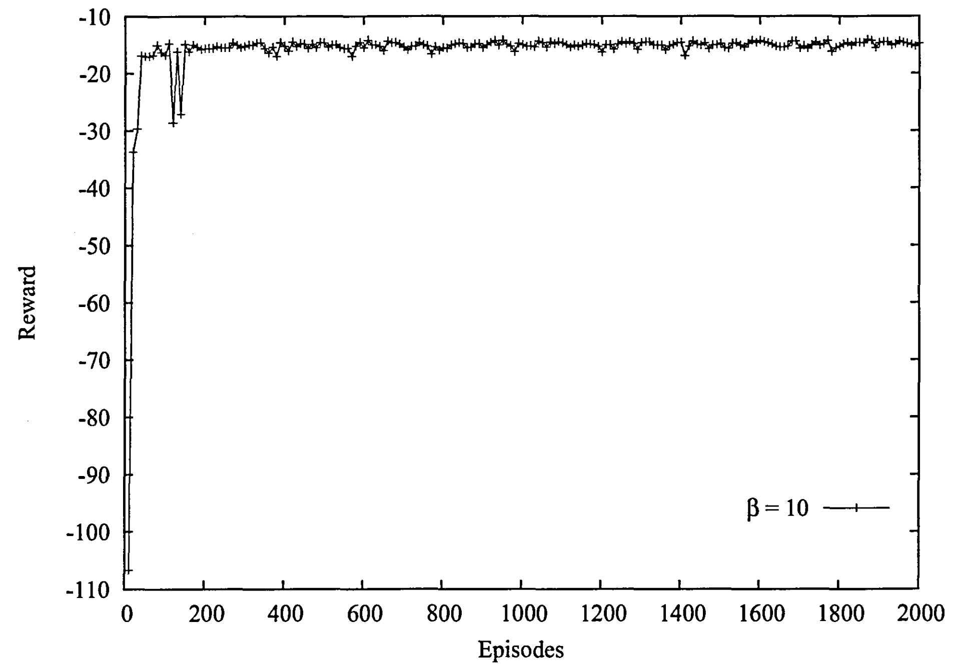


Figure 6.6: Performance of q-learning algorithm - Softmax distribution policy.

It is clear that the learning curve presented on Figure 6.6 is much smoother than the curve from Figure 6.5. The reason for this lies in the nature of Gibbs distribution method which significantly reduces the number of random actions.

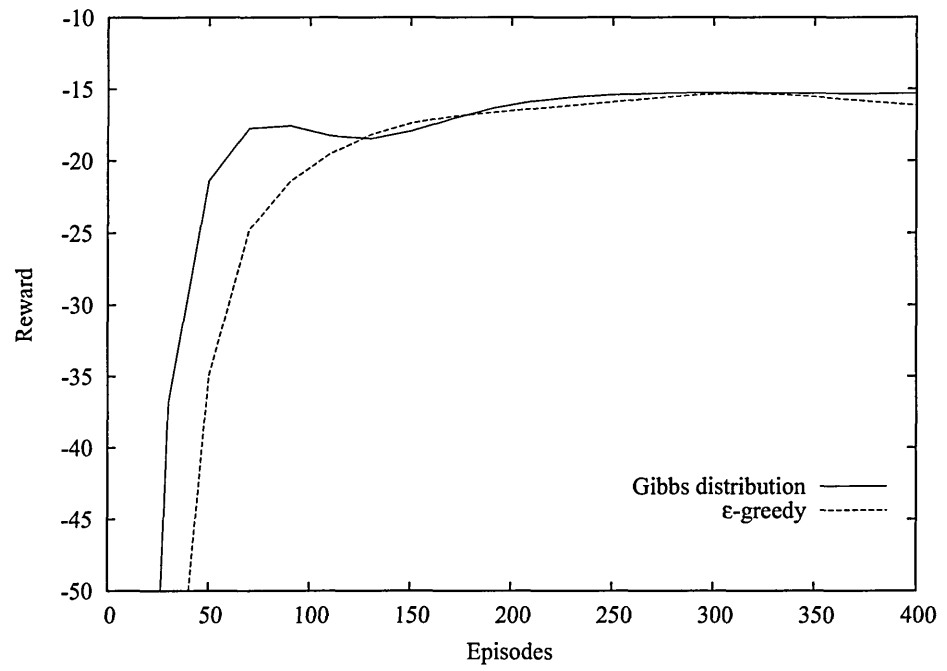


Figure 6.7: Softmax and ϵ-greedy policy during first 400 episodes.

It is also important to mention that, in the case of chamfered circular hole, the q-learning algorithm with Gibbs distribution applied showed the ability of rapid learning. The Figure 6.7 presents the comparison of both action-selection methods. Only the first 400 episodes were plotted and for clear presentation purpose the data points were interpolated using Bezier approximation function. The raw data can be found in Appendix B.2 (see Figure B.16).

The learning curve for the Softmax method quickly rises and stabilises on the value of -15. In the case of ϵ-greedy policy, due to a greater number of random actions, the on-line performance graph more steadily approaches the optimal value.

Despite those differences both methods proved to be highly efficient learning the basic skills for successful insertion task.

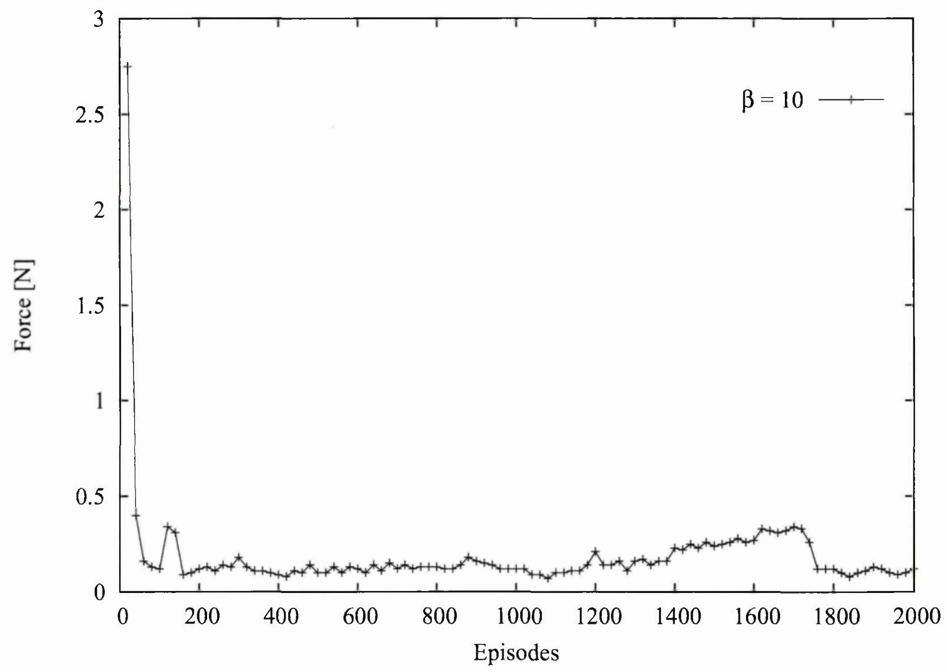


Figure 6.8: Force data from the peg-in-hole experiment - Softmax policy.

The force value and its change throughout the experiment can be seen in Figure 6.8. The data was acquired during the the insertion with q-learning algorithm with Softmax action-selection policy applied. The results from tests when the ϵ-greedy distribution was used can be found in the Section B.2.1.

The significant reduction of contact forces can be observed while the experiment progresses. The high values, caused by the collisions between mating parts were lowered (within first 40 episodes) and never exceeded the value 0.33 N.

The occasional sharp increase of the force is caused by collisions due to random actions taken. This can also be observed on the graphs from Figure 6.6.

The graph from Figure 6.9 presents the change of torque values. Similarly to the force signal torque was very quickly reduced and stabilised at the value of around 0.4 Nm. The very high torque values combined with high contact forces recorded during first 40 episodes suggests the occurrence of jamming conditions during the initial stages of the learning process. The controller learnt to recognise those error conditions and to avoid them in the future.

The significant reduction of both contact forces and the resulting torques is also a clear indication that the intelligent controller was able to learn the necessary skills to perform the peg-in-hole assembly. No additional knowledge about the geometry, part's orientation or position was supplied to the system.

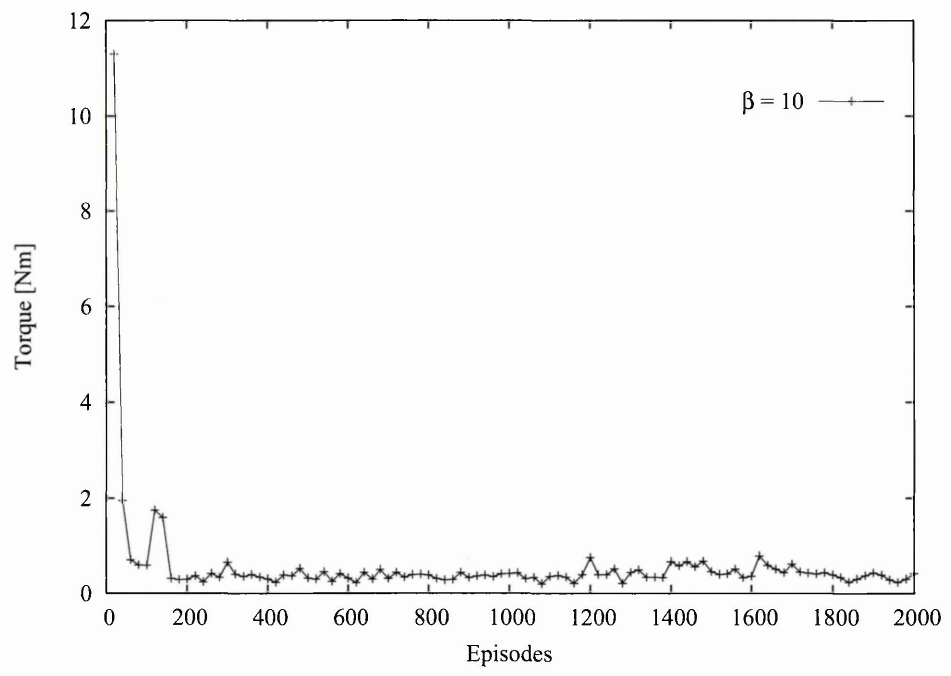


Figure 6.9: Torque data from peg-in-hole experiment.

The data presented on the histogram below (see Figure 6.10) represents the number of unsuccessful insertions per 10 episodes. It clearly indicates the sudden drop in error conditions caused by high contact forces. The results from Figure 6.10 conclude the initial evaluation.

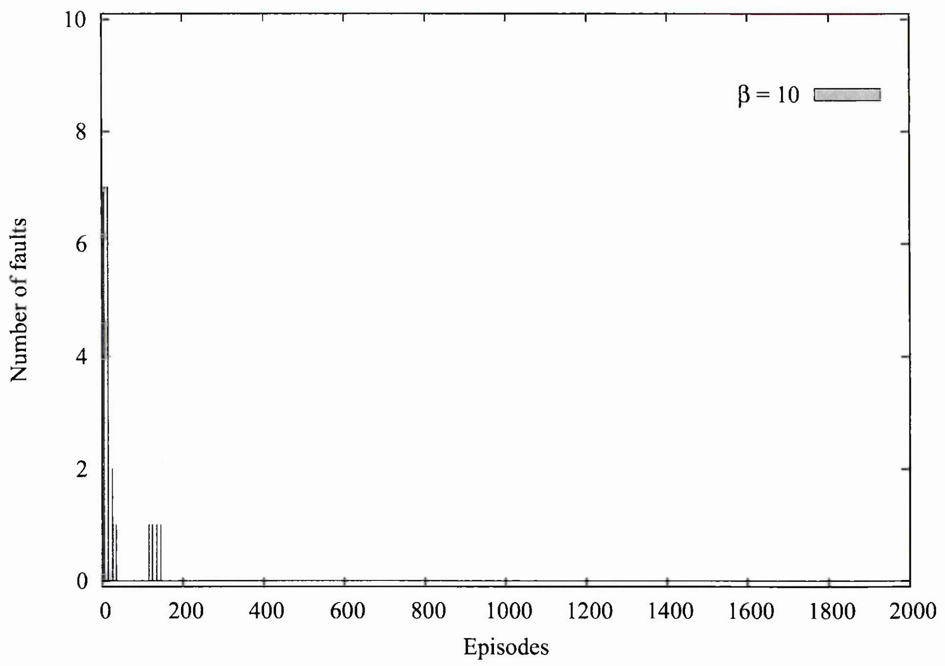


Figure 6.10: Number of failed insertions during peg-in-hole experiment.

The artificial intelligence controller with q-learning method used for knowledge acquisition applied on the chamfered circular peg-in-hole insertion proved to work very well. The ability to deal with a complex 3-dimensional environment was achieved. After 40 episodes transition time the agent very quickly approached and stabilised its performance on the value of cumulative reward -15. After reaching this level it can be assumed that the controller learnt the necessary skills to successfully accomplish the task.

SARSA performance

In this part of the thesis the results from the experiments using the SARSA algorithm will be presented. The same parts were used and the same starting conditions were applied.

Similarly to q-learning SARSA gathers the knowledge from the state-action pairs transitions but it learns the strategy directly. The learning of an optimal strategy relies on applied action-selection method. Each state-action pairs must be visited an infinitive number of times to gain the best possible solution. The Softmax action-selection method was applied (see Section 4.2.1 for details).

The on-line performance of SARSA reinforcement learning agent applied on the circular chamfered peg-in-hole experiment is presented in Figure 6.11. It took 230 episodes of transition until the controller managed to stabilise its cumulative reward close to the value of -18. The occasional fall of performance can be observed during episodes 630, 1180, and 1550 and was caused by insertion failure. It is clear from the graph that the controller's general performance was not affected and the agent was able to maintain its on-line performance after the collision occurrence.

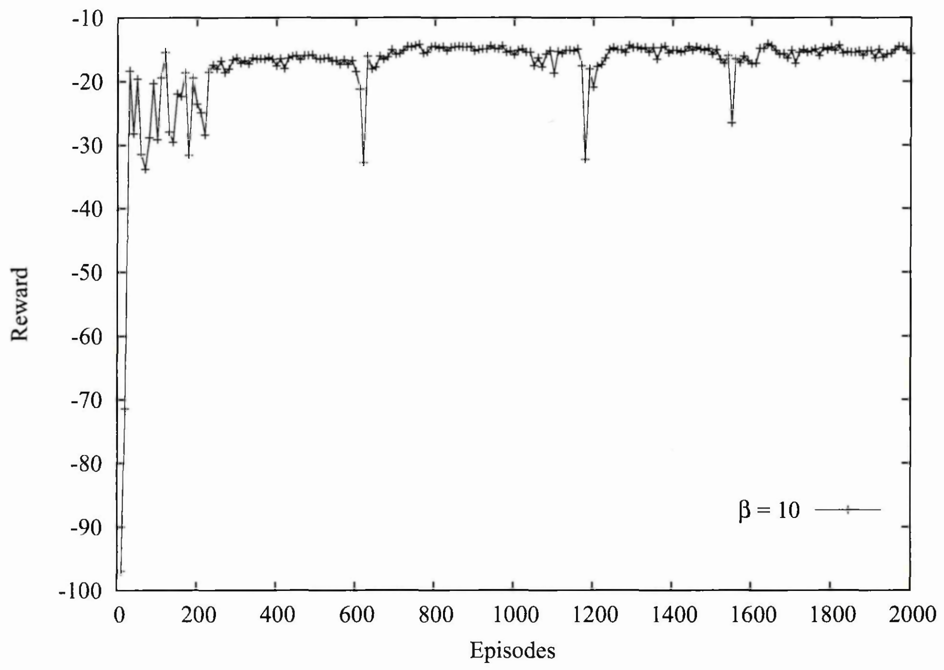


Figure 6.11: Cumulative rewards data acquired using SARSA algorithm.

The force data from the experiment is presented in Figure 6.12. The force values around episodes 1180 and 1550 suggest that the first event (episode 1172) caused the agent to reorganise the strategy. The on-line performance level was maintained but the contact forces between mating parts were constantly increasing eventually exceeding the safety trigger at episode 1550. After that the agent managed reorganise its knowledge again and reduce the forces eventually stabilised at the value of 0.07 N.

This behaviour is caused by continuous learning process combined with action-selection method which takes place throughout the experiment. The random actions are still possible which is necessary to converge to the optimal action-value function (Q^*). The strategy leading to an increase of contact forces was replaced after episode 1172 by the actions leading to low, stable values.

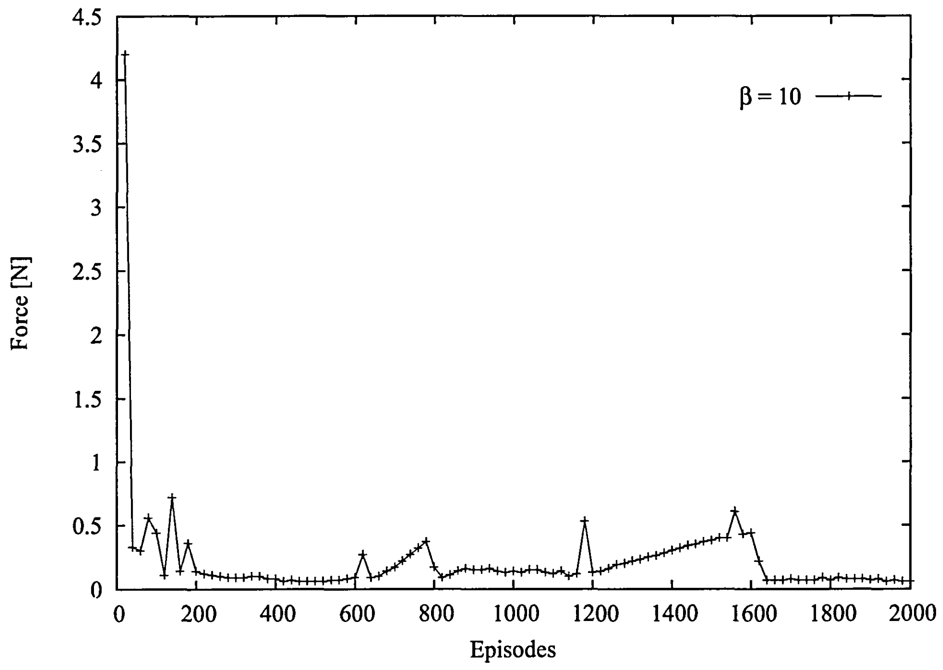


Figure 6.12: Force data acquired using SARSA algorithm.

A similar pattern can be observed in Figure 6.13 which represents the torque values. The occurrence of three spikes after episode 600 combined with high force values indicates jamming occurring. The controller was not designed to cope with those situations so the insertion was interrupted.

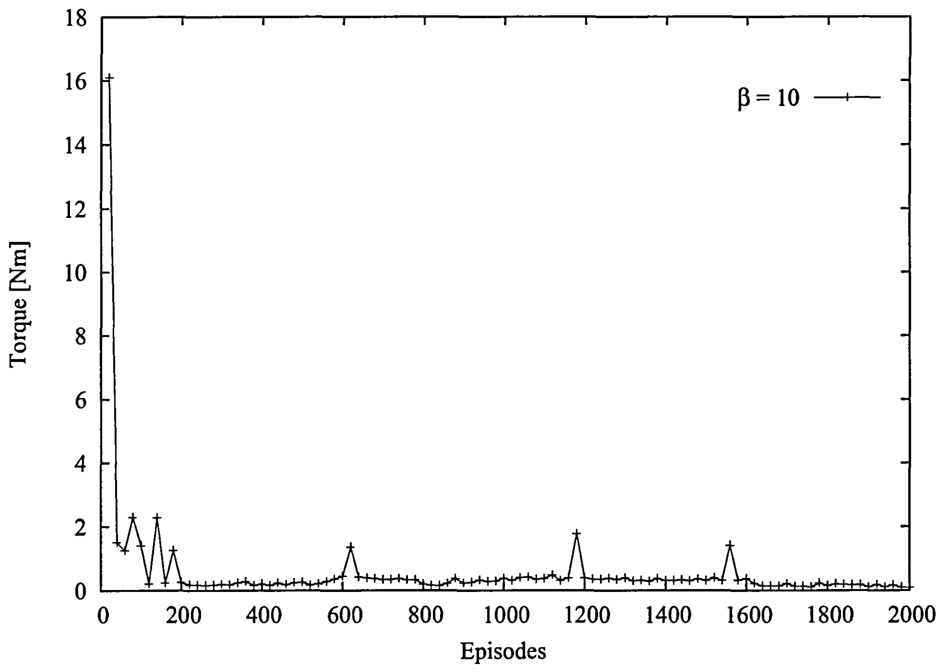


Figure 6.13: Torque data acquired using SARSA algorithm.

This error condition, however undesirable, is the inevitable consequence of the learning method applied. It is important to mention at this stage that the selection of random actions can be reduced to a minimum (no random actions allowed) by controlling the value of β in Gibbs distribution (see Section 4.2.1 for details).

In the histogram from Figure 6.14 the number of failed (jammed) insertions is presented. Like in the case of previous experiments a significant improvement can be observed. The intelligent agent managed to reduce the number of collisions from 8 per 10 episodes during the initial stages of insertion to 0 after 200 episodes. The occasional spikes at the values 630, 1180 and 1550 corresponds to jammed conditions explained above, caused by the occurrence of random movements.

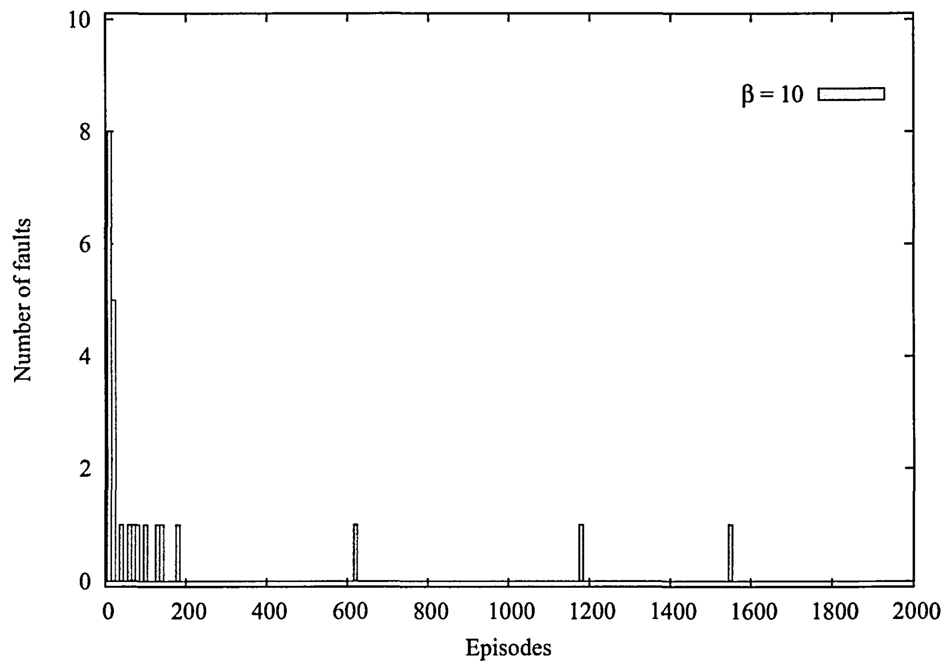


Figure 6.14: Number of the agent's failures using SARSA algorithm.

6.2.2 Rubber peg test

To evaluate the controller's ability to learn the assembly with different materials the rubber peg was designed and manufactured. A cylindrical rubber end was glued to the aluminium frame. This guaranteed the firm grip between the robot's end-effector and the mating part. The cylindrical, chamfered hole was used for purpose of this test. It was entirely built from aluminium. The main purpose of this experiment was to investigate if the stiffness or friction between the assembly components can affect the learning ability.

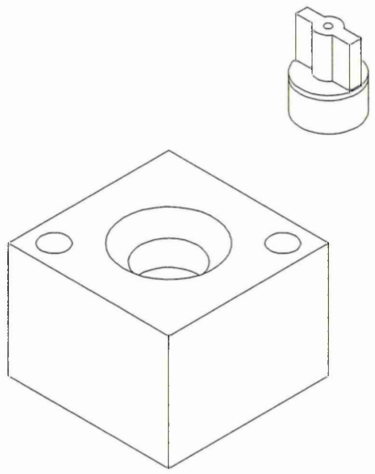


Figure 6.15: Peg and hole model used during the experiments.

On the graph from Figure 6.16 the cumulative reward against the number of episodes was presented. The starting conditions were set up in exactly the same way as in the case of previous experiments. The q-learning algorithm with Softmax action-selection method was applied.

The transition time when the controller learns the geometry features and force-torque relationship was 40 episodes long. After this time the agent stabilised its on-line performance on the level of 16. This corresponds to the number of times when the contact forces were detected and a decision making process took place. A single collision occurred during the episode 129. However, this isolated event did not affect the future rewards. The on-line-performance stayed at the stable level until the end of the experiment.

Comparing the graph from Figure 6.16 with the one from Figure 6.6 where the same algorithms and action-selection methods were used it can be seen that both agents performed similarly throughout the experiment. The major, noticeable difference is the quality of performance curve in its stable area. The results from experiments where only aluminium components were applied are more stable (close to the straight line) compared to the ones when the rubber peg was used.

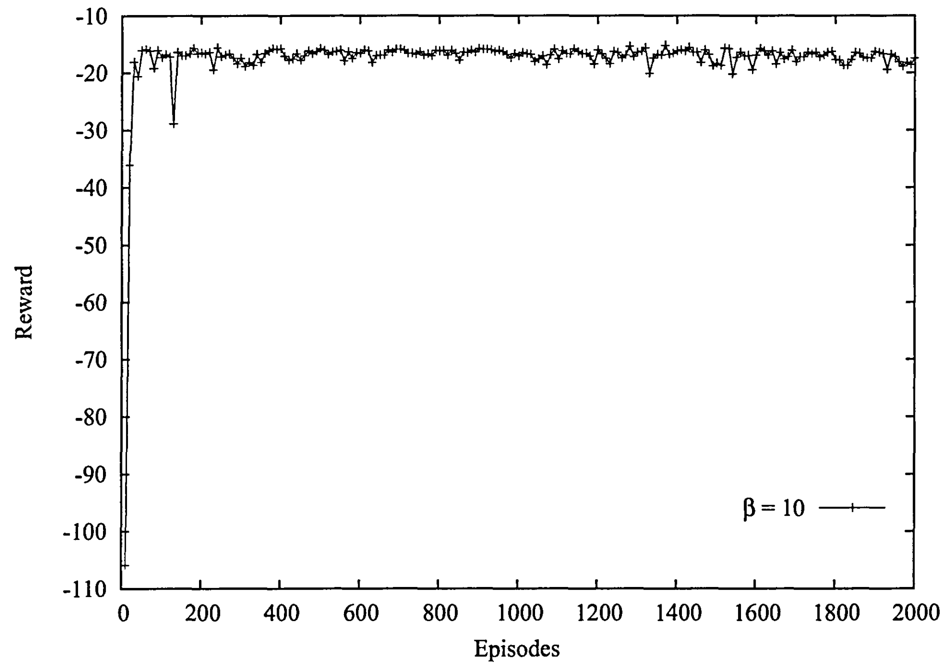


Figure 6.16: On-line performance data from peg-in-hole experiment.

The force readings (presented on the graph from Figure 6.17) were also significantly reduced within first 40 episodes. The initial value of 1.66 N dropped below 0.2 N. The spike at episode 129 was caused by the sudden increase of contact forces leading to the interruption of the insertion.

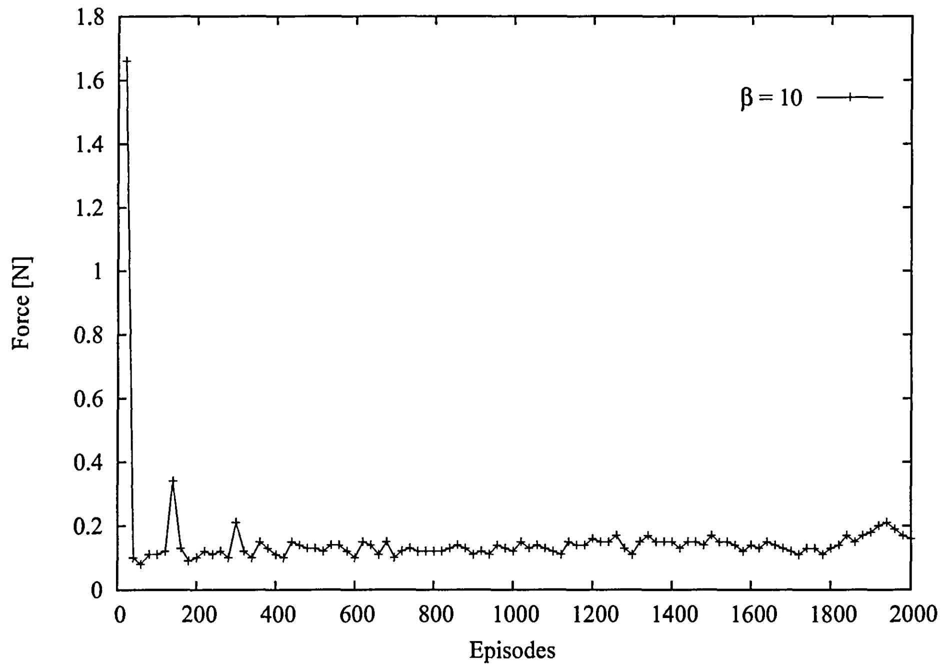


Figure 6.17: Force data from peg-in-hole experiment.

The torque values are presented in Figure 6.18. The transition time was 40 episodes long. This value is the same for on-line performance and force data. After initial readings of 6.97 Nm the contact torques were reduced to the value below 0.5 Nm. This was the result of the agent's ability to learn the insertion strategy.

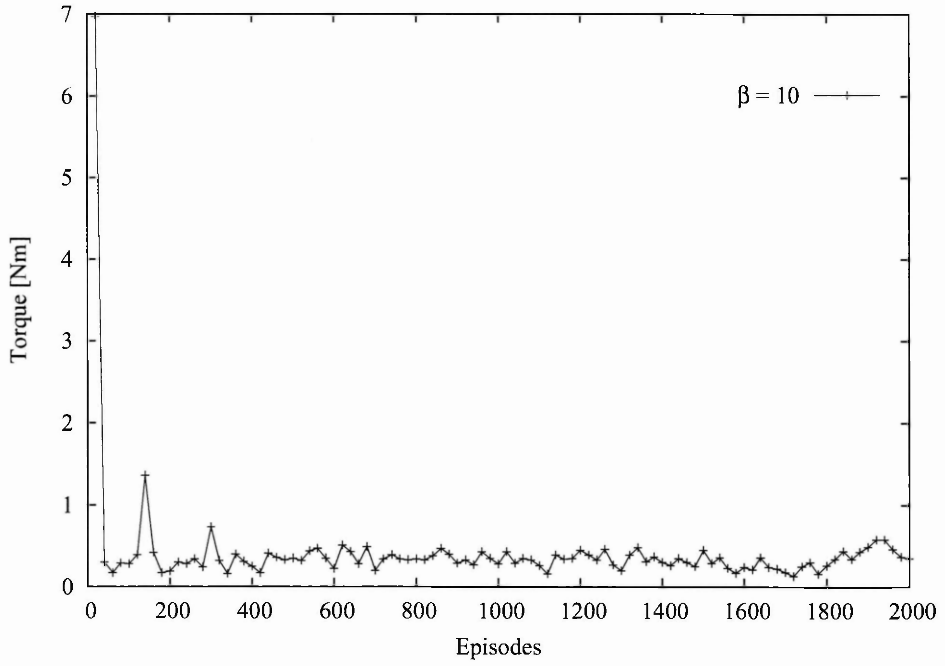


Figure 6.18: Torque data from peg-in-hole experiment.

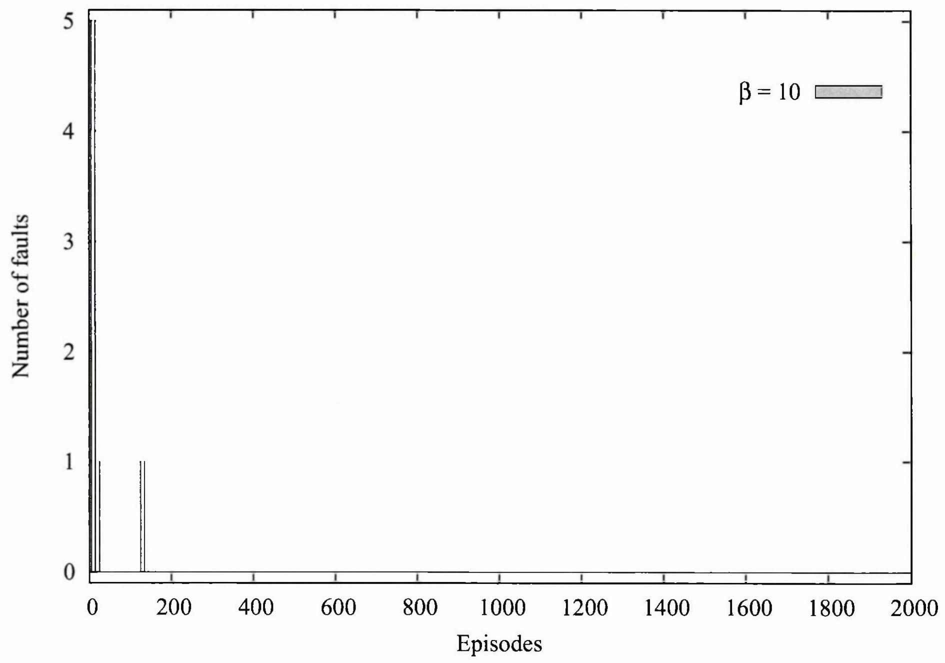


Figure 6.19: Number of failed insertions per 10 episodes - rubber peg applied.

Both force and torque graphs feature a clear initial, transition and steady periods. The number of failed insertions per 10 episodes is presented on the histogram above (Figure 6.19). Five collisions were detected during the initial stage of the insertion. It was followed by only two more interruptions later during the experiment.

6.3 Chamfer-less circular peg-in-hole insertion problem

In this section the results from the experiment using a cylindrical chamfer-less hole are presented. The results using q-learning algorithm with Softmax action-selection method are presented below. The data from the experiment with ϵ-greedy action-selection method applied can be found in Appendix B.2.

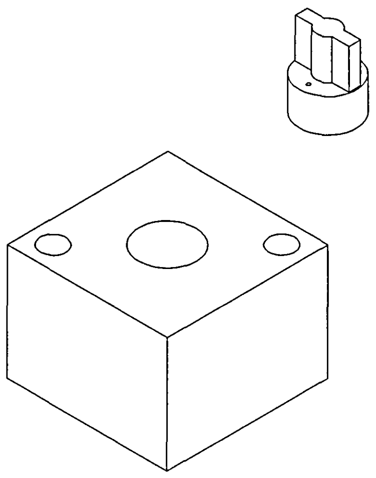


Figure 6.20: Peg and hole model used during the experiments.

As the assembly task becomes more complicated the force-velocity data is no longer linearly mappable. In that case applying a linear compliance unavoidably causes the error. Asada [1,2] used a chamfer-less peg-in-hole insertion task to illustrate this problem (see Chapter 2 for further details).

In case of the chamfer-less peg-in-hole insertion the force to velocity mapping is against the principle of superposition which is the one of most fundamental properties of linear mapping. Therefore that this task should be considered nonlinear.

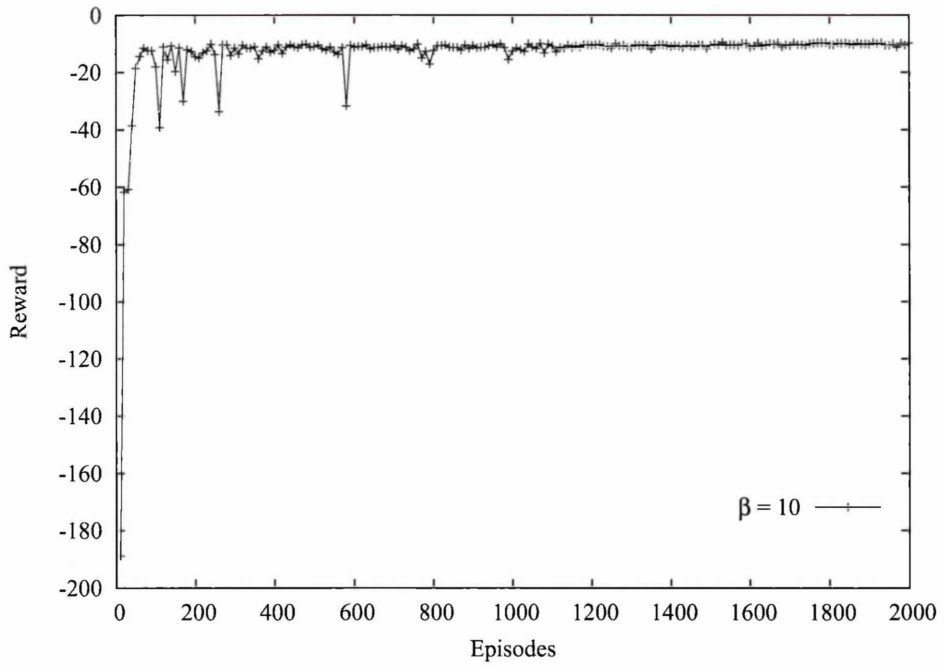


Figure 6.21: On-line performance data from peg-in-hole experiment.

Analysing the on-line performance graph from Figure 6.21 it could be seen that the transition time was 270 episodes long. During first 40 episodes the controller was struggling to perform the successful assembly. Later on the number of insertions were accomplished but the steady state was reached after episode 270. The controller stabilised the cumulative reward at the value around -11.

The occasional drop in knowledge was caused by the collision condition during episode 572. Since the knowledge about the environment was well established at this stage (after exactly 312 successful insertions), this failure did not affect the later performance.

The contact forces were registered on a higher level than in the case of previous experiments (see Figure 6.22). As expected the high value (2.72 N) within 60 episodes was reduced to the average value of 0.1 N. The occasional spikes of 0.54 N, 0.50 N, 0.39 N and 0.34 N were caused by the jamming condition and correspond to the drop in knowledge experienced during episodes 102, 163, 260 and 572. Since episode 163 the force signal has never raised above value of 0.5 N. It can be seen from the graph above that the curve did not stabilise at certain value like in the case of previous experiments. The contact force value was rising slowly reaching the peak value of 0.24 N at episode 760. The process was reversed and a minimum value of 0.08 was registered at episode 1120.

The later rise to 0.34 N was also reduced to the final value of 0.2 N at episode 2000.

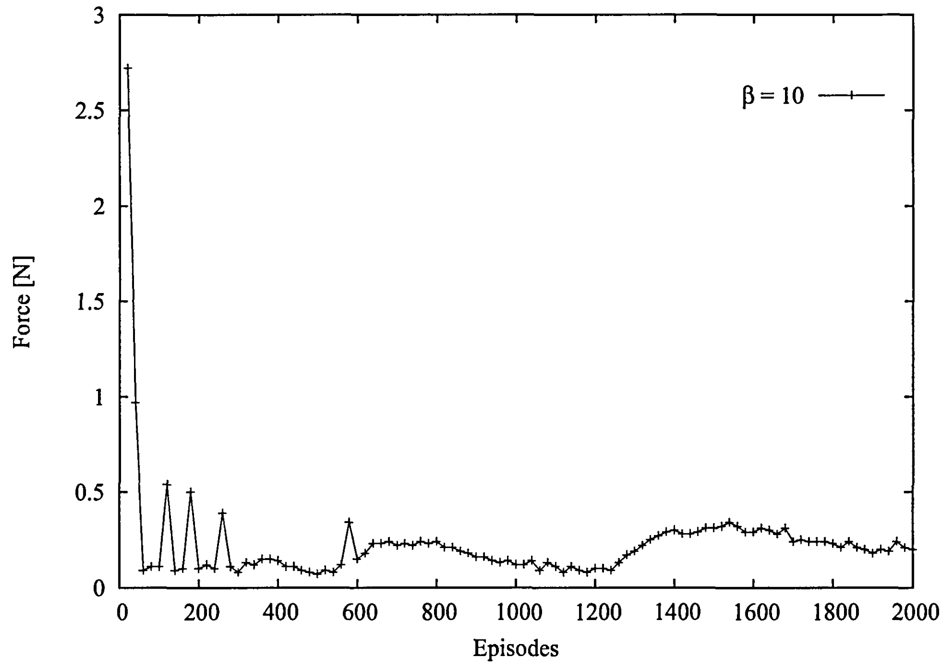


Figure 6.22: Force data from peg-in-hole experiment.

The stable torque curve was registered during the experiment (see Figure 6.23). Similarly to contact force data, the four spikes in the stable area indicate the jamming condition.

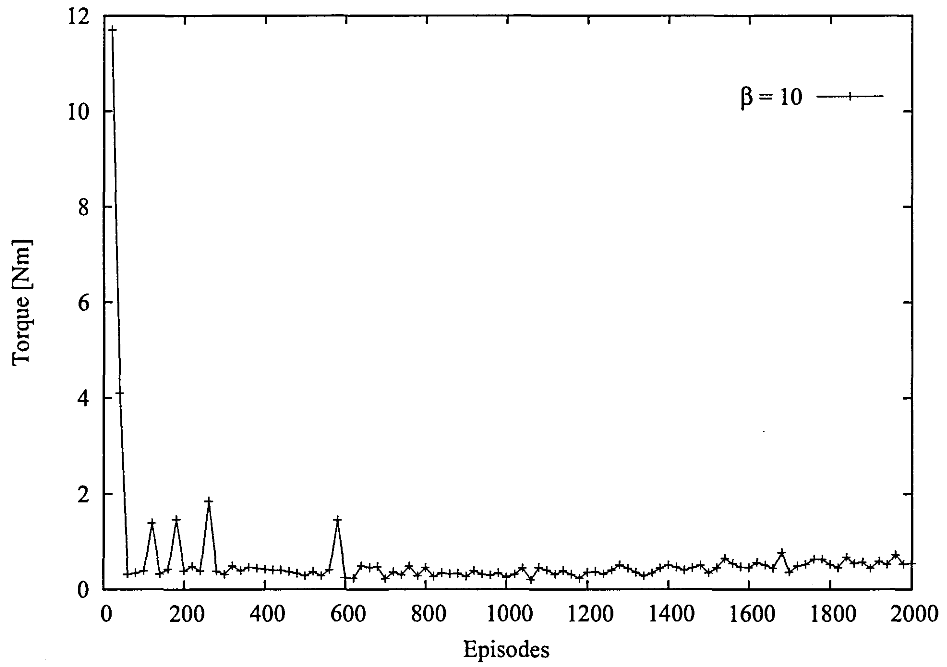


Figure 6.23: Torque data from peg-in-hole experiment.

The top value of torque (11.7 Nm) was steadily reduced and after 60 episodes stabilised at the value of 0.3 Nm. After 260 episodes, when the last peak occurred (1.85 Nm) the controller was able to maintain the knowledge about its environment and carry on complete insertions.

The last jammed condition was registered during episode 572. The number of unsuccessful insertions is presented in the form of a histogram in Figure 6.24. During the first 10 episodes 8 were terminated due to excess of contact force and torque values. The controller was introduced to the environment without any knowledge. Most of basic information about the geometry and contact forces relationship was acquired during the initial stage of the insertion.

The four jammed conditions, mentioned earlier in this Section, are also present in the histogram. Similarly to previous experiments the sporadic occurrence of failure did not affect the controller's performance or its learning ability. The intelligent agent was able to ignore them and maintain all necessary skills.

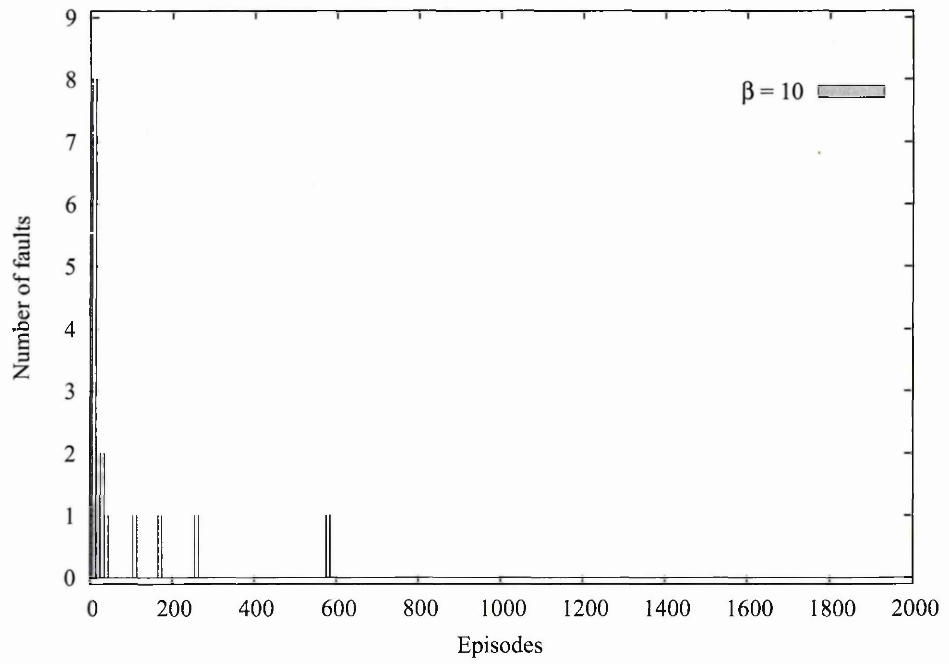


Figure 6.24: Number of failed insertions (per 10 episodes).

As mentioned at the beginning of this Section, the data from the experiment using ϵ-greedy action-selection method accompanied by a short analysis is presented in the

Appendix B.2. Due to the similarity of the results described above, the author decided not to include them in the main body of the thesis.

6.4 Square peg analysis

The results from the experiment with squared chamfered peg-in-hole insertion was presented. This is the most complex geometry analysed so far. It involves the highest number of planes and the additional two actions were introduced to the agent. Since all the previous insertions involved the insertion of the peg with circular cross-section into a chamfered or chamfer-less, circular hole, the rotations about TOOL Z axis were not implemented. This solution saved computational time as well as the time designated for empirical evaluation of the controller.

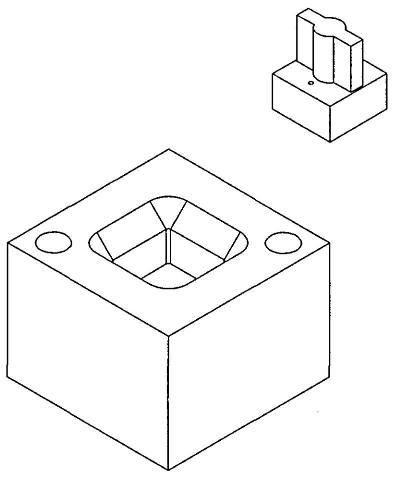


Figure 6.25: Peg and hole model used during the experiments.

In the case of the square peg the rotations about Z axis (in both directions Z+ and Z-) had to be included. This means that for every detected and classified state the agent has two more actions to chose from.

Additionally this type of geometry increases the chance of class overlap. This certain state can be detected during the initial stage of the insertion when the peg makes the contact with chamfered part of the hole. It is possible that due to limited sensor's resolution and ART2 classification setup (parameter ρ) that another state, detected during the second stage of insertion will be assigned to the same class as the previous one. This will inevitably cause confusion since the same state would require two different actions in regard to the stage of insertion.

The solution for the described problem together with controllers ability to recover from the jamming condition was not a part of this research. These problems and suggested solutions will be described in more detail in the next Chapter (Section 7.4).

The graph from Figure 6.26 shows the on-line performance of the q-learning algorithm with Softmax action-selection method applied on square peg-in-hole insertion.

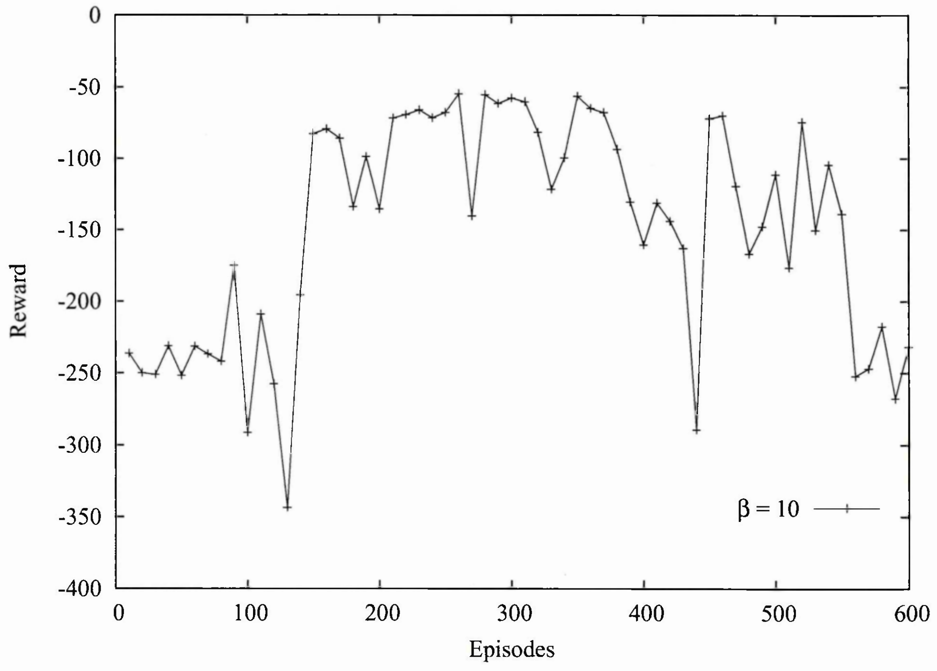


Figure 6.26: On-line performance data from peg-in-hole experiment.

Only the first 600 episodes are presented. The experiment was stopped after 1800 insertions due to the learning difficulties experienced. Several experiments were undergone with similar results. Only the recent results are presented for the purpose of this thesis. The full 1800 episodes graphs are included in Appendix B.2.

Due to the complexity of the task after a series of initial experiments it was decided to change the standard learning parameters and starting conditions. The following settings were used:

- The peg was positioned so its main axis was shifted -0.6 mm on X and +0.2 mm on Y direction from the hole centre.
- The angular misalignment was set up to the value of -1.8 about X, +1.4 about Y and -0.9 axis.
- The 0.1 mm and 0.1 deg. was applied as a motion incremental step for the translations and rotations respectively.
- The 0.2 mm was applied for every movement on Z+ direction. This value was kept the same to speed up the progress towards the bottom of the hole.
- -200 reward was given for every insertion interrupted due to contact forces excess. The higher reward was necessary since the step was reduced to 0.1 mm which required the controller to perform more actions to reach the goal.
- -50 reward was given if more that 100 movements were executed without any progress towards the bottom of the hole. The insertion was also stopped at this stage. This was applied to prevent "sinking" of the peg due to the large number of rotation actions applied.

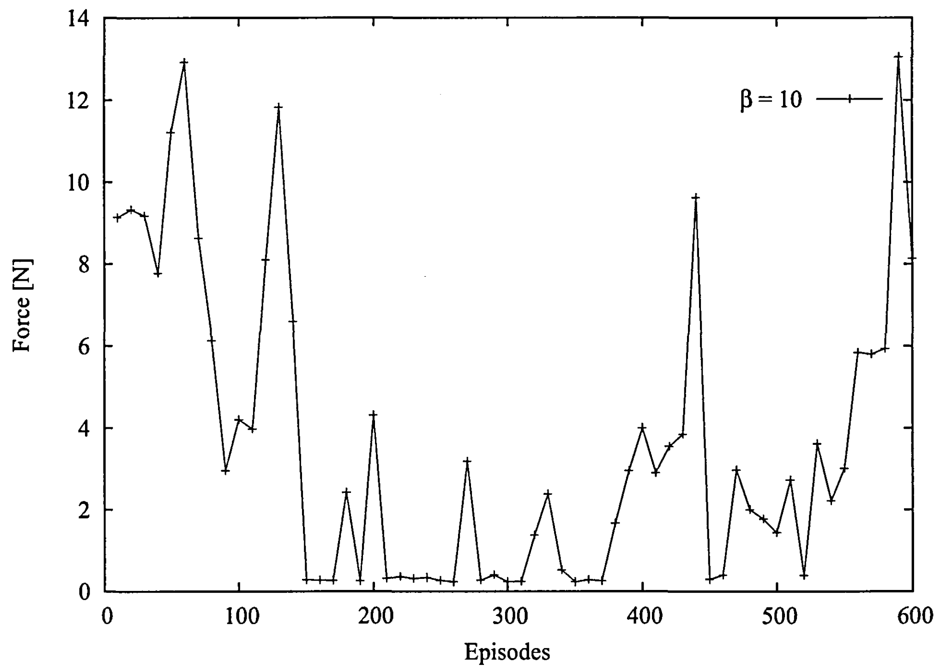


Figure 6.27: Force data from peg-in-hole experiment.

It can be seen from Figure 6.26 that the graphs are more erratic and non-stable. However clear signs of learning can be seen. The first successful insertion was detected after episode 86. The series of fully completed tasks was registered later, after episode 138. The insertion path was learnt, improved and taken until episode 547 when the series of collisions happened. Due to the applied settings (the high reward of -200 for every collision) the knowledge was disrupted and eventually lost. During later parts of the experiment (not shown in the included Figures) the agent tried to recover from that error condition.

The contact forces are significantly larger than in the case of experiments with the circular peg (see Figure 6.27). Top value of 12.92 N was registered after 60 episodes. After episode 150 the agent was able to reduce the contact forces to the value 0.3 N. This was maintained for several insertions often interrupted by the peak values (as high as 4 N) caused by jammed condition. Eventually, as the controller lost the knowledge the forces increased reaching the top value again. At this stage it can be assumed that most of the information about the environment was lost.

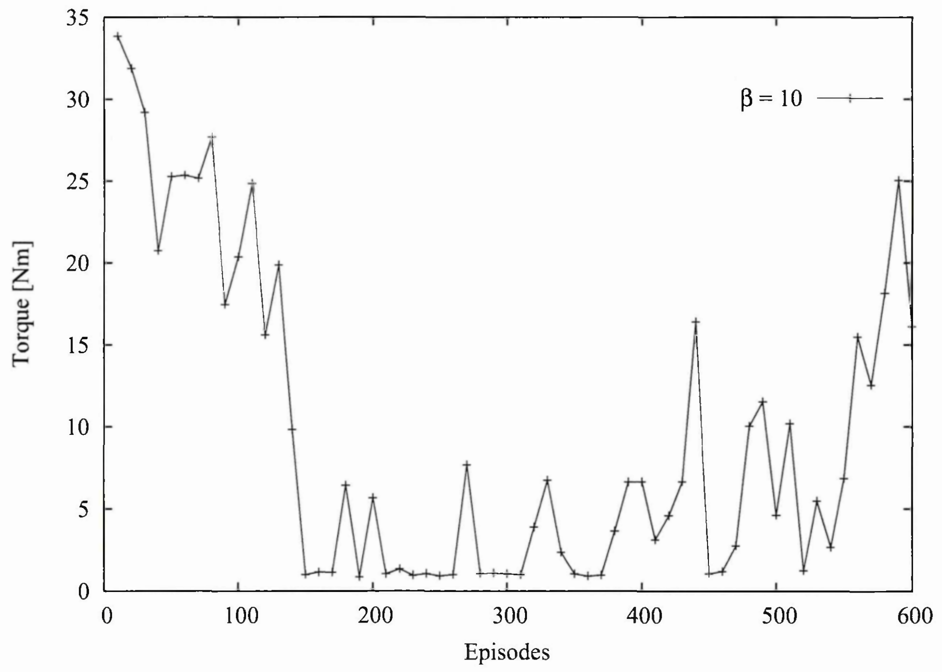


Figure 6.28: Torque data from peg-in-hole experiment.

The torque data shows a very similar pattern (see Figure 6.28). The initial value of 33.85 Nm was gradually reduced to the average value of 1 Nm. Again the steady state was often interrupted by the sudden increases in torque caused by the jammed peg.

The loss of knowledge caused by a series of severe collisions caused the significant increase of sensed torques. The lack of algorithms for jammed peg recovery caused the insertion to be interrupted and reward -200 was released. This can significantly affect the learning process disrupting pre-learnt state-action relationship. As stated before, the controller never managed to fully recover from these conditions. Analysing further the data from the experiment several reasons for this behaviour can be derived. More detailed discussion is included in the Section 6.5 at the end of this Chapter.

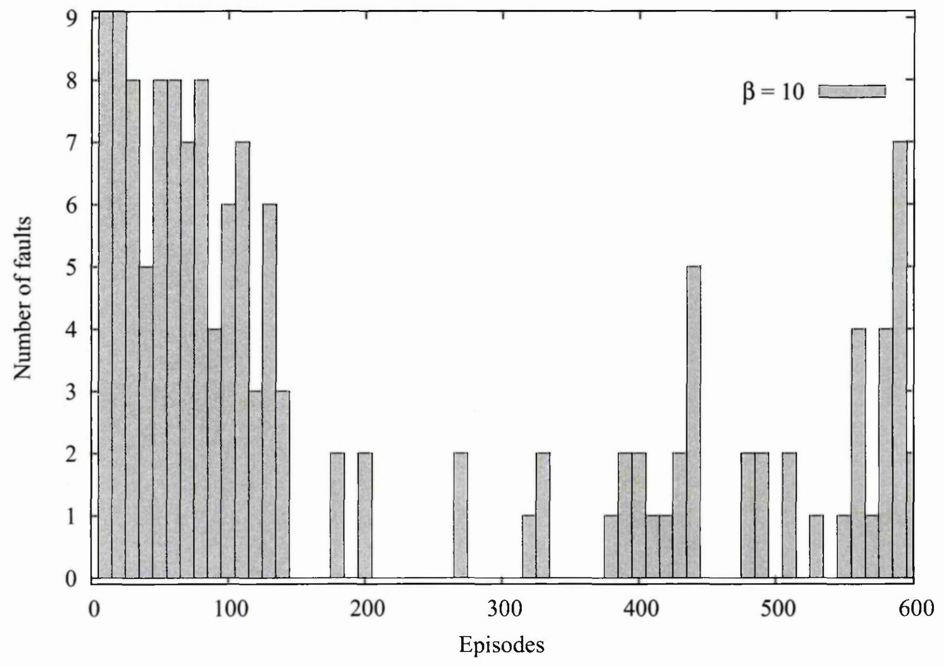


Figure 6.29: Number of failed insertions (per 10 episodes).

The short but present steady state proves the controllers ability to learn to deal with the different geometries of mating parts. It can be seen from Figure 6.29 that after 140 episodes transition time the controller managed to perform more than 250 successful assemblies. In some cases 110 consecutive insertions were accomplished.

6.5 Discussion

In this Chapter the results from the series of experiments were presented. The main purpose of empirical evaluation was to validate the proposed controller's ability to learn the state-action relationship working with the environments under severe uncertainty.

Every experiment was started with no external knowledge supplied to the system. Only the guidance towards the hole's bottom was implemented (hard-coded). The system was also designed in the way so the contacts with the referencing surfaces were maintained. These two, together with the conformity to the geometric constrains due to the contacts are Asada's [1,2,28] three basic requirements for valid peg insertions (see Chapter 2 for further reference).

All the weights in the ART2 artificial neural network were reset to the value of zero at the beginning of each experiment. Similarly, all the elements of the Q-function matrix were also set to zero. This was done to assure there is no pre-learnt knowledge left from the previous tests.

The starting conditions were the same for every experiment except the square, chamfered peg-in-hole task. The very simple reward function was applied namely: -100 for the collision, 0 for every movement in positive Z direction (insertion direction) and -1 for every other movement. Some additional modifications were necessary when chamfer-less circular, and chamfered square holes were considered. In those cases, the -50 was applied if more that 100 movements were executed without any progress towards the bottom of the hole. The insertion was also stopped at this stage. Due to the change of the step size the reward -200 had to be applied to the experiments involving the square peg geometry.

Two sensory signals were used namely: force and torque readings. No additional information was acquired during the experiments. The author would like to emphasise that the explicit position and orientation of the peg was unknown to the intelligent agent at every stage of the insertion.

The historical information was also used and is encoded within the reinforcement learning algorithms used for purpose of this project (see Chapter 4 for further details). As stated before the insertion direction can also be qualified as the information source.

Those two sensory and two descriptive information sources, according to the author, form the minimal requirements to perform the stable successful assembly with automatic, intelligent knowledge acquisition.

The q-learning and SARSA algorithms were tested together with ϵ-greedy and Softmax action-selection methods. A set of experiments were performed to evaluate the usefulness and on-line performance of above methods.

Three main stages can be distinguished in the knowledge acquisition process during the peg-in-hole insertion tasks:

- No knowledge period. At this stage, which usually takes 10-20 episodes, the controller does not know anything about its environment. In most of the cases 80% of the insertions failed due to the excess of contact forces. This feature was set up purely for safety reasons preventing damage of the force and torque sensor.

- Transition stage. During this time, which can take up to 70 episodes, the controller applies random actions to learn the geometry features of the mating parts as well as state-action relationship. This is the time when the main reorganisation of ART2 weight system and q-function matrix takes place.

- Steady state. This is the last period of the learning process. It is assumed that all the necessary information is acquired and controller is able to perform the insertion task without interruption. The state-action relationship reorganisation still takes place to guarantee the convergence to the the optimal action-value function (Q^*). The random actions are sporadic but still possible. The on-line performance curve slowly approaches the steady reward value. Occasional drops in knowledge at this stage are acceptable assuming that they do not affect the agent's future performance.

During the initial tests all three stages could be clearly distinguished. The experiment was simple and some of the controllers abilities were limited. Some adjustment of learning ratios were necessary and the real tests with cylindrical pegs were executed.

First, The aluminium components were taken into account. The q-learning algorithm was applied with both ϵ-greedy and Softmax action-selection methods. Both agents performed very well eventually reaching the same level of performance. It is important to mention that due to the system design and data acquisition method it was impossible to establish if the learnt policy is optimal. The ART2 belongs to a self-organising, unsupervised family of neural networks. The class rearrangement happens automatically and is governed by the algorithm described in detail in Section 4.3.2. This means that in the case of contact force patterns, description of each class is encoded within the network. The reinforcement learning module was designed to estimate the best action for each state. The class to force direction (and friction cone angle) decoding is necessary to confirm if the optimal policy was reached. This task was out of the scope of this project.

Comparing the graphs from Figure 6.5 and 6.6 it could be seen how the random actions can affect the performance. The frequent spikes present when ϵ-greedy method was applied are caused by the exploratory movements. These actions are responsible for the collisions or sending the controller to the area never experienced before. In the second case it usually takes a number of additional moves until high contact forces interrupt the task. This problem was eliminated when the Softmax method was used. In this case from the initial stages of the insertion the actions with greater probability to be better are preferred. In the other words, the controller prefers the "good" actions over the "bad" ones.

As expected, both methods in steady state converged to the same average value (-15). It is because the q-learning is an off-policy method and learnt action-value function (Q), directly approximates, the optimal action-value function (Q^*). Based on that it could be assumed that the optimal policy was learnt indeed.

Another experiment was designed to allow the comparison between q-learning method and SARSA. Exactly the same starting conditions with peg and hole geometries were applied. the experiment was run with the Softmax action-selection method.

The SARSA agent slowly approaches the value of cumulative reward reached early by the q-learning algorithm. The transition time of the SARSA controller is much longer and more erratic than in the case of q-learning. However, the major difference is in the steady state (compare Figure 6.6 and Figure 6.11). According to the theory and simulations presented early in this thesis, the SARSA agent should be able to learn the safe policy and to avoid the cliff (see Section 4.5 for details). It is clearly not the case when the experimental data is considered. The SARSA agent with the same action-selection policy applied caused the collision seven times during the stable state. Only one interrupted task was recorded when q-learning agent was used.

This behaviour was expected and is caused by the fact that the contacts with the referencing surface must be maintained. For that reason to learn the safe path (away from the cliff) is very difficult in peg-in-hole application. The q-learning agent was working without causing the collisions during the steady state because of its off-policy nature and ability to learn the optimal path.

Analysing the force graphs (Figure 6.8 and Figure 6.12) it can be seen that the SARSA agent managed to maintain the contact forces at a slightly higher level than q-learning controller. In some cases, due to random collisions the value exceeded even 0.6 N. The graph from Figure 6.12 is also less stable. Also 3 more collisions were detected during the steady state when the SARSA agent was applied (compare the graphs from Figures 6.10 and 6.14). In both cases the force value was steadily increasing after 1100 episodes reaching the top value within 450 episodes. After that the controller managed to reduce the contact forces back to the low average value of 0.2 N. The experimental data from the tests using the SARSA agent suggests that this behaviour was triggered by the collision. However, because a very similar pattern was recorded with the q-learning controller but with no interruption present it can be assumed that knowledge reorganisation took place. Both controllers are fully adaptive throughout the whole experiment (the learning takes place all the time) and they managed to eventually reduce the contact force values.

Generally both the SARSA and q-learning agents performed similarly on the cylindrical peg-in-hole experiment. However, due to more stable contact forces graph, less erratic on-line performance curve, and lower contact force level it was decided to choose the q-learning algorithm with Softmax action selection policy for further experiments.

To investigate if the application of different materials can affect the controller performance the test with the rubber peg was designed. To be able to compare the results from the previous experiments exactly the same starting conditions were applied. Analysing the on-line performance of the q-learning controller using the aluminium components (see Figure 6.6) with the one with rubber applied (see Figure 6.16) it can be seen that the different peg's material does not affect the agent's on-line performance. Both controllers converged to the same cumulative reward value and both graphs feature long periods (about 800 episodes) of stable performance, successfully executing the insertion tasks.

The difference can be seen when the contact forces and torques are taken into account. During the experiments with the rubber peg the forces during the steady state were measured at the very low level. Only once during episode 1940 they raised above the value of 0,2 N. The graph does not feature the 400 episode long increase of contact forces clearly present when the aluminium components were investigated.

Based on presented results, it can be stated that the application of the rubber type peg did not affect the controller's ability to learn. The usage of a less rigid material resulted in lower, more stable force reading during the steady state. It also significantly reduced the number of collisions. Only seven unfinished tasks were recorded throughout 2000 episodes. According to the author the reason for that lies in the damping properties of rubber as well as the rigidity of the peg. The friction between mating parts should not affect the performance unless it causes the class overlap problem (the same state requires two entirely different actions depending on peg's orientation). Because the position and orientation are not included as an input to the system the controller will not be able to deal with such a condition.

To evaluate the controllers ability to deal with the complex geometry when force-velocity relationship is not linearly mappable the chamfer-less hole was used during the experiment. The cylindrical aluminium peg was used. As stated before to avoid the peg sinking condition when the controller performs a high number of consecutive rotations a different reward function was applied. The value of -50 was released after 100 actions executed without any progress towards the bottom of the hole. This also prevented the controller from being able to "walk" along the top surface of the hole eventually losing contact with the geometry. The hole search algorithms were not implemented since it was not part of the research investigation.

Before the change to the reward function a number of unsuccessful experiments was executed. The results are not included in this thesis since every of the 2000 episodes was interrupted due to contact forces excess or loss of the contact with the referencing surface. Basically the controller learnt to perform a set of horizontal movements away from the hole well beyond the physical limits.

After the change the on-line performance was improved significantly (see Figure 6.21). The transition time is much longer than in the case of previous insertions (about 270 episodes). This indicates that the controller was particularly struggling with the first part of the hole's geometry. After the strategy for the entrance section was developed the the rest of insertion task was quickly learnt. The sporadic collision registered during the episode 572 did not affect the controller's performance. The force and torque graphs are similar to those presented before. The curve from Figure 6.22 is less stable than in the case of chamfer-less hole experiments but the agent managed to maintain the force values below 0.5 N throughout the experiment.

The last experiment performed for purpose of this project involved the insertion task with square peg and chamfered hole applied. This type of geometry presents new difficulties to the intelligent controller. The rotation about Z-axis was introduced to the system. The presence of 2 new actions increases the learning time and the chance of class overlap occurrence. Moreover, defined by Brignone [4] ”virtual classes” (four corners of square hole) present additional problems to the controller. Constant incremental steps combined with the possibility of rotating the square peg about its main axis can cause confusion during the learning process. The orientation of the peg is not a part of information sources supplied to the system. The decisions are mainly taken based on force and torque information. The learnt correct rotation about Z-axis may cause collision when the peg is slightly tilted inside the hole.

Despite of the described problems, the results from the square peg-in-hole experiment were very promising. All the graphs are erratic and lack stability when compared to the data from the experiments with cylindrical peg applied. However, clear signs of the controller's ability to learn how to cope with this complex geometry can be noticed. The significant reduction of forces and torques can be observed between 140 episodes transition time. The agent managed to work with no interruption in some cases for more than 110 episodes. Later during the experiment, due to a series of collisions the controller had lost the knowledge and never recovered from it. An interesting pattern was developed towards the end of the experiment. The controller was systematically given the reward -50 for the amount of movements without any progress towards the bottom of the hole. After close investigation it turned out that the agent fell into the never ending loop rotating the peg +0.1 of the degree about Z axis (class 3 was detected) followed the rotation of -0.1 degree about the same axis (class 0 was detected). Due to Softmax action-selection policy applied where the random actions are reduced while the experiment progresses the controller was not able to learn the alternative route.

Different starting conditions, step size and the reward function were applied so the results from square peg experiments cannot be directly compared with the previous tests.

The controller performed exceptionally well with the cylindrical peg and hole geometry. After 40 episodes the agent managed to established the strategy to successfully accomplish the peg-in-hole insertion task. Some difficulties were experienced and adjustments to the starting conditions and reward function had to be made when the chamfer-less circular and chamfered square holes were applied. However, learning skills and abilities to cope with complex 3-dimensional geometry are clearly present.

According to the author's knowledge the proposed controller with exceptionally low degree of supervision was never used before on such a complex geometry. Considering the results from the experiments it can be concluded that the agent was able to learn the strategy for successful peg-in-hole assembly working with complex 3-dimensional geometry under high degree of uncertainty.

Chapter 7

Conclusions and future work

In this part of the thesis the final summary and conclusions will be presented. The chapter is divided into a set of subsections where the work done, results summary, the final conclusions and the proposal for future work are included.

7.1 Work done

1. **System design.** A stable connection using the terminal emulation system was established. After solving initial problems the robot is able to perform up to 3 movements per second. However, due to the computational load the movement speed had to be restricted to 1 per second. The floppy drive emulator was implemented to load the VAL II operating system from an external disk. The force and torque driver designed as a GNU/Linux kernel module was tested and the scaled data from F/T sensor was also successfully read.

2. **Controller design and implementation.** The sandwich structure of the intelligent agent was proposed. It featured two major layers: State Recognition module where the detection and localisation of the contact points were performed and the Decision Making subsystem where the decision about the next action based on the calculated reward and the current state took place. The controller was designed to receive two signals from the environment. The force and torque signals were fed alongside the

reward feedback. The number corresponding to the location of the contact point was produced by the State Recognition module. This information together with the reward from the environment was sent as an input to the Decision Making subsystem. The decision about the next action was taken and the signal was decoded to the incremental motion command understandable to the robot. This close loop control system was implemented using ART2 Artificial Neural Network, q-learning and SARSA reinforcement learning algorithms. The system was designed in modular form and for flexibility the data structures were also implanted.

3. **Algorithm simulations.** All the algorithms were implemented and tested on simulated data before being applied onto the real-life peg-in-hole insertion. Every part of the controller was evaluated separately. This approach gave the author an opportunity to understand the information flow within the Artificial Intelligence algorithms and to learn how the different ratios affect the learning speed and overall performance. Both subsystems were tested separately to evaluate the implementation and usefulness of proposed algorithms. During tests of the ART2 implementation, the algorithm sometimes was not able to stabilise data on the F_1 layer. A different method had to be implemented (see Appendix D). Here the benefits of the modular design could be seen allowing the testing of a problem to be conducted quickly and accurately. Using this approach the author was able to switch to a different implementation and compare it without any interference to the main application code.

4. **Environment's evaluation.** A set of measurements was taken to establish the robot's accuracy and repeatability. The signal from the force and torque sensor was acquired under controlled conditions (preset load). All the data was collected to establish the area and level of uncertainty (e.g. signal errors) the artificial controller would need to learn to cope with and compensate for.

5. **Plan of experiments.** The proposed controller was applied on a set of real life peg-in-hole experiments. This type of insertions are amongst the most common operations in automated assembly. The set of experiments was designed to evaluate

the intelligent agent's ability to learn state-action mapping for different geometrical features. This includes working with cylindrical and square peg geometry. Both chamfered and chamfer-less holes were taken into account as well as the ability of the controller to deal with the different component's materials (rubber and aluminium pegs were used).

6. **Empirical evaluation.** The main purpose of the experimental investigation was to evaluate the usefulness and applicability of the sandwich structure of the controller. The author claims that the combination of different, specialised artificial intelligence methods can produce a fully independent, self-adaptive and unsupervised controller. Different aspects of learning were investigated. The empirical part of the thesis includes the investigation of the effects of different learning methods applied on the same geometry. The influence of action-selection methods onto controller performance was also analysed. Materials with different friction factors were applied as well as a non-linear insertion case.

7.2 Results summary and final conclusions.

The proposed controller was applied on a set of tests to evaluate its on-line performance working with complex, 3-dimensional environments.

Every experiment was started with no external knowledge supplied to the system. Only the guidance towards the hole's bottom was implemented.

All three stages of knowledge acquisition defined by the author can be clearly distinguished on the presented performance graphs. The transition time was 40 episodes long when cylindrical peg was used. It took much longer (up to 270 episodes) to learn the insertion task when the chamfer-less or square problem were considered.

The q-learning and SARSA algorithms performed very similarly. They converged to the same value of cumulative rewards (-15). The transition time when SARSA algorithm was used was extended to 200 episodes while it took 40 episodes for the q-learning agent to stabilise the learning curve.

Occasional collisions (jammed condition) were registered during the steady state of the knowledge acquisition. They were caused by the agent executing the random actions following the action-selection policy. However, the interrupted insertions did not affect the pre-learnt knowledge and did not influence the controller's on-line performance.

A significant reduction in force value during the initial stage of the learning process was recorded. The force was reduced to $1/10^{th}$ of the initial value. Some fluctuations were recorded but when the cylindrical peg was considered the value of contact forces never exceeded 0.5 N during the steady state.

Similarly the torque was quickly reduced (usually by the same factor). After that it stabilised and only the occasional jammed peg condition caused the torques to rise.

The application of the rubber type peg did not affect the controller's ability to learn. The usage of less rigid material resulted in lower, more stable force reading during the steady state. It also significantly reduced the number of collisions.

To investigate the controller's ability to deal with the tasks involving the geometry where the force-velocity domains are not linearly mappable (non-linear problems) the chamfer-less cylindrical hole was used. The transition time was much longer than in the case of insertions into chamfered hole (about 270 episodes). This indicates that the controller was particularly struggling with the first part of the hole's geometry. After the strategy for the entrance section was developed the the rest of insertion task was quickly learnt.

Some problems were encountered when the square peg was analysed. The presented graphs lack a long, uninterrupted steady state. However the the set of successful insertions can be observed. That clearly indicates that the intelligent agent was able to develop the strategy for square peg-in-hole insertions. The suggestions for future work to improve the on-line performance are included in the next section of this chapter.

Based on the results of conducted experiments it can be concluded that the main aim of this thesis was fully achieved. The fully unsupervised controller with the ability to deal with complex, 3-dimensional geometry was proposed, implemented and tested. The knowledge acquisition was clearly present in all investigated cases. Also, the contact forces and torque values were significantly reduced while the insertion progressed.

According to the author's best knowledge all aims and project objectives included in Section 1.1 were fully fulfilled, namely:

- Different information sources were analysed and applied to the system. The sensory signals in form of force and torque readings and descriptive in the form of assembly direction were employed to peg-in-hole insertion. It was decided and empirically proved that any form of pre-training or task description is not necessary to successfully learn the assembly. The history was also included and was embodied as a part of the reinforcement learning algorithms.

- The automatic state recognition and clustering was investigated. The contact states were analysed and classified using the implemented module for geometry classification.

- Unsupervised motion generation was achieved using the intelligent decision making agent, able to learn the state-action map.

- The stable and flexible system able to support several different AI architectures was implemented.

- The controller's performance and its usefulness was tested and validated using a range of mechanical components. The real-life, 3-dimensional peg-in-hole assembly tasks were performed. The influence of the different features of the mating parts (e.g. peg's material) was also investigated.

The system was initially designed to work with Puma 500 series robots. However, due to mechanical failure of the manipulator, the proposed system was later transferred, with no modification, onto a Staubli RX90 robotic arm. This demonstrates the portability and flexibility of the design and methodology. The repeatability of the results was also observed with no difference in recorded on-line performance or F/T values when both robots were applied.

The agent performed exceptionally well with the cylindrical peg-and-hole geometry. After short initial stage it managed to established the strategy to successfully accomplish the insertion task. Some difficulties were experienced when chamfered, square hole was investigated but the learning skills and abilities to cope with complex 3-dimensional geometry are clearly present.

7.3 Original contribution

As stated in the previous section, the main aim of this study was to develop, implement and validate an intelligent controller capable of learning basic manipulative skills during the peg-in-hole assembly. The complexity of geometry, nonlinearity of the insertion task and noisy feedback signals from the sensors are the main factors making the task very difficult to deal with.

Different information sources were applied and combined to develop a fully unsupervised, intelligent controller. In the authors design, no class labelling or geometry feature pre-training takes place. Only force and torque signal together with the direction of insertion was supplied to the controller.

The controller learns the geometry of the parts using self organising Artificial Neural Network. In the next step an unsupervised decision about the appropriate action is taken.

According to the author's best knowledge, the "sandwich" structure of the controller and the choice of algorithms are unique and have not been investigated before. The proposed design and selection of information sources proved to be able to cope in an environment with high level of uncertainty without the loss of on-line performance.

The improvement in system design was also an important part of this research. The modular form of the software and robust method of robot-computer connection were developed to fulfil that requirement.

The system was initially designed to work with Puma 500 series robots. However, due to mechanical failure of the manipulator the proposed system was later transferred with no modification onto Staubli RX90 robotic arm. This demonstrates the portability and flexibility of the design and methodology.

7.4 Future work

The proposed controller has shown some limitations described earlier in this thesis. Some improvements suggested by the author are presented in this section below. Most of the future work should be focused on improving the controllers performance in working with square peg geometry.

1. **Angular misalignment problem.** As described earlier (Section 4.3) the state clustering methodology is based on a successful match between F/T signal and vector normal to the surface. The F/T coordinate system is fixed in the sensor centre point. Because of this configuration the F/T reference frame rotates with the robot gripper. The unary vectors which describe the orientation of each surface forming the hole are calculated using constant coordinate system fixed at the bottom centre of the hole. Since the orientation of the peg is not known the transformation between these two systems can not be calculated. This problem could be solved in a number of ways:

    - Use the peg orientation description gained from the robot. This approach increases the number of information sources. Since the author's aim is to use the minimum accessible data to perform successful assembly this approach should not be considered in the future.
    - F/T sensor is fixed underneath the hole. This is an ideal solution since both coordinate systems (for peg and the hole) are constant and fixed in the same point.
    - Installation of two F/T sensors. This allows the peg orientation to be calculated by comparing the force signal from two sensors: one mounted on the robot gripper and the second fixed underneath the hole. This approach involves the addition of new source of data to the system. It is also an expensive solution because the second F/T sensor needs to be obtained.

- Application of fuzzy logic. Using fuzzy logic and history data should help to distinguish between two overlapping classes. This approach does not require any change to the system architecture. It involves alteration of clustering algorithms by adding a fuzzy logic module. This could be done by changing ART2 type of ANN to a FuzzyART or implementation of fuzzy function as a separate module.

2. **Jammed condition recovery.** During the experiments with square peg geometry it was noticed that the mating parts had a tendency to jam against each other. This situation results in a sharp rise of both force and torque values. To speedup the learning process with the more complex geometry than the cylindrical peg a module for detection and recovery should be designed and implemented. This will inevitably involve the the implementation of actions towards negative Z direction. In this case the insertion direction and assembly goal for reinforcement learning needs to be redefined.

3. **Action domain problem** Currently the controller chooses from discrete set of action which represent the constant step (usually 0.2 mm) towards given direction. Also implemented reinforcement learning tasks are generally treated in discrete time steps. At each time step, (t), the learning system receives some representation of the environment's state (s_t), it takes an action (a_{t+1}), and one step later it receives a scalar reward (r_{t+1}), and finds itself in a new state (s_{t+1}). The limited number of actions, five in total, can cause the problems described in the previous chapter (see Chapter 6 for reference). Widening the choice by adding the combined movements on two different axes, the never ending loop trap could be avoided. Also introducing the actions with different step size may increase the controller's on-line performance.

4. **Reward function investigation.** The proposed controller is given reward value of -1 for every performed movement, -100 when the collision occurs and 0 for successfully accomplished task. This relatively simple reward function was altered

when chamfer-less and square holes were investigated. Additional value of -50 was given when the controller performed more than 100 movements without any progress towards the goal. The results suggested that implementing even more sophisticated rewards can also improve the learning skills.

5. **State clustering module.** Currently the ART2 artificial neural networks algorithm is implemented. Because it accepts the analogue values on the input the scaled real force and torque data can be applied on the input. The drawback of this solution lies in extensive computational time. Additional layers had to be added to the network structure for vector normalisation and contrast enhancement. Since the signal from the sensor is read in the form of an unsigned integer number the faster ART1 artificial neural network could be used for purpose of force and torque vectors classification.

6. **Tests using a wider range of mechanical components.** The final version of the controller should be tested and validated using all range of peg and hole configurations, raging from circular, square and finally triangular.

References

[1] H. Asada. Teaching and learning of compliance using neural nets. In *Proceedings of 1990 IEEE International Conference on Robotics and Automation*, volume 2, pages 1237–1244, 1990.

[2] H. Asada. Representation and learning of nonlinear compliance using neural nets. *IEEE Transactions on Robotics and Automation*, 9(4):863–867, 1993.

[3] J.D. Barber. Mantran: A symbolic language for supervisory control of an intelligent remote manipulator. Master's thesis, MIT, 1967.

[4] L. Brignone. *A geometrically validated approach to intelligent robotic assembly.* PhD thesis, The Nottingham Trent University, April 2002.

[5] L. Brignone, M. Howarth, K. Sivayoganathan, and V. Balendran. Contact localisation: A novel approach to intelligent robotic assembly. In *INNS/IEEE Intl. Joint Conf. on Neural Networks 2001*, 2001.

[6] L. Brignone, M. Howarth, K. Sivayoganathan, and V. Balendran. Simulation of contact states using peg in hole insertions. In *Proceedings of Simulation '99, UCLse/UK Sim Conference on Simulation*, pages 77–81, 2002.

[7] L. Brignone, K. Sivayoganathan, V. Balendran, and M. Howarth. Mechanical assembly force/torque data analysis as a foundation for task level programming. *Advances in Manufacturing Technology*, XIV:163–169, 2000.

[8] F.M. Campos, J.M. Caladoa, L.F. Baptistab, and J.M. S da Costa. On the impedance parameter selection for robot force control. *11th International Conference on Advanced Robotics*, 2003.

[9] A.G. Carpenter and S. Grossberg. Atrmap: Supervised, real-time learning and classification of non-stationary data by self-organising neural network. *Neural Networks*, pages 565–588, 1991.

[10] E. Cervera and A. del Pobil. Geometric reasoning for fine motion planning. *IEEE 0-8186-6995-0/95*, 1995.

[11] E. Cervera and A. del Pobil. Learning strategies for sensor-based manipulation tasks. *IEEE 0-8186-8138-1/97*, pages 54–59, 1997.

[12] E. Cervera and A. del Pobil. Sensor-based learning for practical planning of the fine motion in robotics. *Information Sciences*, 145:147–168, 2002.

[13] E. Cervera, A. del Pobil, E. Marta, and M. Serna. A sensor-based approach to motion in contact in task planning. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 468–473, 1995.

[14] C. Cheah and D. Wang. Learning impedance control for robotic manipulators. *IEEE Transactions on Robotics and Automation*, 14(3), 1998.

[15] H.A. Ernst. Mh1-a computer operated mechanical hand. Master's thesis, MIT, 1961.

[16] J. Ferrman and D. Skapura. *Neural networks. Algorithms, applications and programming techniques.* Computation and neural systems series, 1991. ISBN: 0–201–51376–5.

[17] S. Grossberg. Self-organizing neural networks for stable control of autonomous behaviour in a changing world. *Elsevier Science Publishers*, 1993.

[18] V. Gullapalli. Skillful control under uncertainty via direct reinforcement learning. *Robotics and autonomous systems*, pages 237–246, 1995.

[19] V. Gullapalli, J.A. Franclin, and H. Benbrahim. Acquiring robot skills via reinforcement learning. *IEEE Control Systems*, pages 13–24, February 1994.

[20] V. Gullapalli, R.A. Grupen, and A.G. Barto. Learning reactive admittance control. In *Proceedings of IEEE International Conference on Robotics and Automation*, May 1992.

[21] H. Hanafusa and H. Asada. A robotic hand with elastic fingers and its application to assembly process. In *Proceedings of FAC Symposium on Information Control Problems in Production Engineering*, pages 127–138, October 1977.

[22] S. Haykin. *Neural Networks. A Comprehensive Foundation.* Macmillan College Publishing Co. Inc., 1994.

[23] D. Henriksson, R. Johansson, and A. Robertsson. Observer-based impedance control in robotics. *5th IFAC Symposium Nonlinear Control Systems DNOLCOS 01E*, 2001.

[24] M. Howarth. *An Investigation Of Task Level Programming For Robotic Assembly.* PhD thesis, The Nottingham Trent University, January 1998.

[25] S.H. Huang and H. Zhang. Artificial neural networks in manufacturing: Concepts, applications, and perspectives. *IEEE Transactions on components, packaging, and manufacturing technology*, 17(2), 1994.

[26] JR3 Inc. *JR3 DSP-based Force Sensor Receivers. Software and Installation manual*, 1994.

[27] S. Kang, K. Komoriya, K. Yokoi, T. Koutoku, and K. Tanie. Utilization of inertial effect in damping-based posture control of mobile manipulator. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, 2001.

[28] S. Lee and H. Asada. Assembly parts with irregular surfaces using active force sensing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2639–2644, 1994.

[29] I. Lopez and M. Howarth. Learning manipulative skills with art. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and systems*, pages 578–583, 2000.

[30] I. Lopez and M. Howarth. Knowledge acquisition and learning in unstructured robotic assembly environments. *International Journal of Information Sciences*, March 2002.

[31] I. Lopez-Juarez. *On-line Learning for Robotic Assembly Using Artificial Neural Networks and Contact Force Sensing*. PhD thesis, The Nottingham Trent University, April 2000.

[32] J. Roy and L. Whitcomb. Adaptive force control of position/velocity controlled robots: Theory and experiment. *IEEE Transactions on Robotics and Automation*, 18(2), 2002.

[33] W.S. Sarle. Neural network faq, part 1 of 7: Introduction, periodic posting to the usenet newsgroup comp.ai.neural-nets, 1997.
URL: ftp://ftp.sas.com/pub/neural/FAQ.html.

[34] J. Simons, H. Van Brussels, J. De Schutter, and J. Verhaert. A self-learning automation with variable resolution for high precision assembly by industrial robots. *IEEE Transactions on Automatic Control*, 5(1109–1113), 1982.

[35] S.N. Simunovic. Force information in assembly processes. In *Proceedings of 5th International Symposium Industrial robots*, pages 415–431, 1975.

[36] R. Sutton and A. Barto. *Reinforcement learning. An introduction.* The MIT Press, 1998. ISBN: 0-262-19398-1.

[37] J. Tidd. Divergent trends in robotic assembly in the uk and japan,assembly automation. *Assembly Automation*, 8(4):211–212, 1988.

[38] UNECE. World robotics 2002 - statistics, market analysis, forecasts, case studies and profitability of robot investment, 2002.

[39] UNECE. World robotics 2003 - statistics, market analysis, forecasts, case studies and profitability of robot investment, 2003.

[40] Unimate Industrial Robot. *Programming manual User's guide to VALII version 2.0*, 1986.

[41] D.E. Whitney. Force feedback control of manipulator fine motions. *ASME Journal of Dynamic Systems, Measurement, Control*, 98(2):91–97, 1977.

[42] D.E. Whitney. Quasi-static assembly of compliantly supported rigid parts. *Journal of Dynamic Systems, Measurement, Control*, 104:65–77, 1982.

[43] D.E. Whitney. Historical perspective and state of the art in robot force control. *The Journal of Robotic Research*, 6(1), 1987.

[44] D.E. Whitney and J.L. Nevins. What is the remote centre compliance (rcc) and what can it do? In *Proceedings of the 9th International Symposium on Industrial Robots*, pages 135–152, 1979.

[45] D. Xiao, B.K. Ghosh, N. Xi, and T.J. Tarn. Sensor-based hybrid position/force control of a robot manipulator in an uncalibrated environment. *IEEE Transactions on Control Systems Technology*, 8(4), 2000.

[46] B. Yang and H. Asada. Learning compliance control laws for robotic assembly. an application of adaptive reinforcement learning control. In *Proceedings of 1993 International Joint Conference on Neural Networks*, pages 1821–1823, Japan, 1993.

[47] A. Zaknich. Artificial neural networks an introductory course, 1998. URL: http://www.maths.uwa.edu.au/ rkealley/ann_all/ann_all.html.

Appendix A

Simulation results

In this chapter the additional results from simulations described in Section 4.5 are presented. All the tests were performed under exactly the same conditions as those described in the main body of the thesis. This includes the learning parameters and applied action-selection methods.

A.1 Action selection policy evaluation (q-learning)

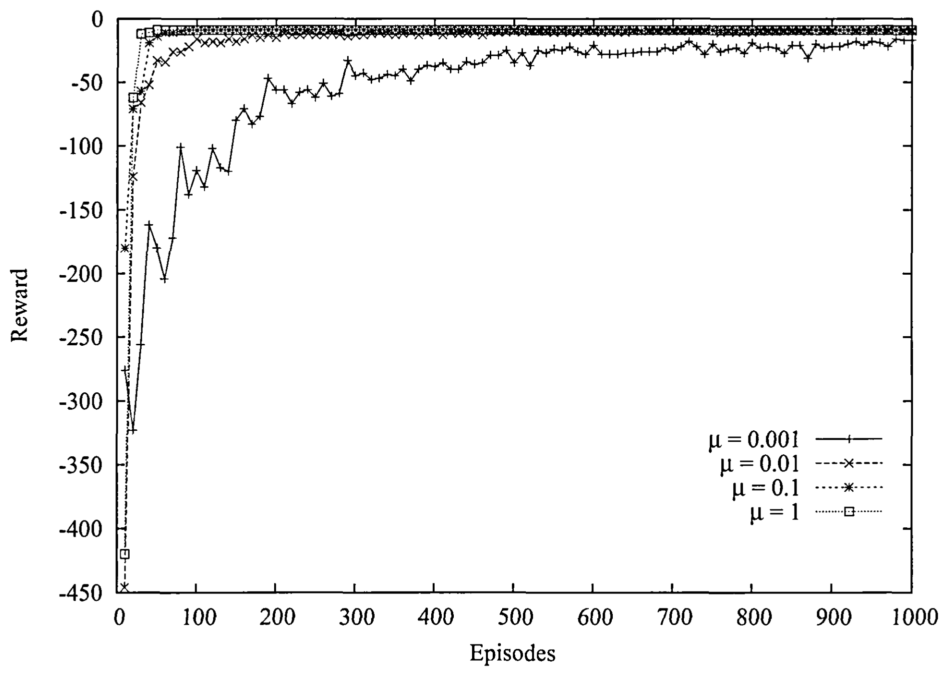


Figure A.1: ϵ-greedy policy evaluation using q-learning algorithm.

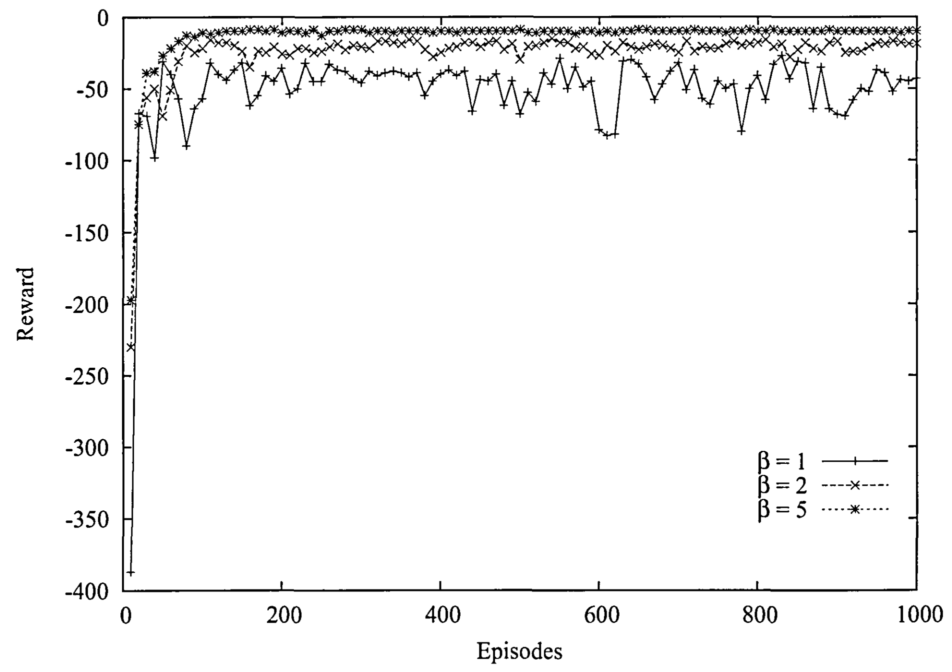


Figure A.2: Softmax policy evaluation using q-learning algorithm.

State	Learnt Policy							Optimal policy
	ϵ-greedy				Softmax			
	μ=0.001	μ=0.01	μ=0.1	μ=1	β=1	β=2	β=5	
0	E	E	E	E	E	E	E	–
1	E	E	E	E	E	E	E	**E**
2	E	E	E	E	E	E	E	**E**
3	E	E	E	E	E	E	E	**E**
4	N	N	N	N	N	N	N	**N**
5	S	S	S	S	S	S	S	**S**
6	S	S	S	S	S	S	S	**S**
7	S	S	S	S	S	S	S	**S**
8	N	E	N	E	E	N	E	–
9	E	E	E	E	E	E	E	**E**
10	N	W	S	W	N	N	E	–
11	N	N	N	N	N	N	N	–
12	N	N	W	E	N	N	N	–
13	N	N	N	N	N	N	N	–
14	**G**	**G**	**G**	**G**	**G**	**G**	**G**	**G**
15	N	N	N	N	N	N	N	–

Table A.1: Learnt policies for different action-selection methods.

A.2 State classification method evaluation

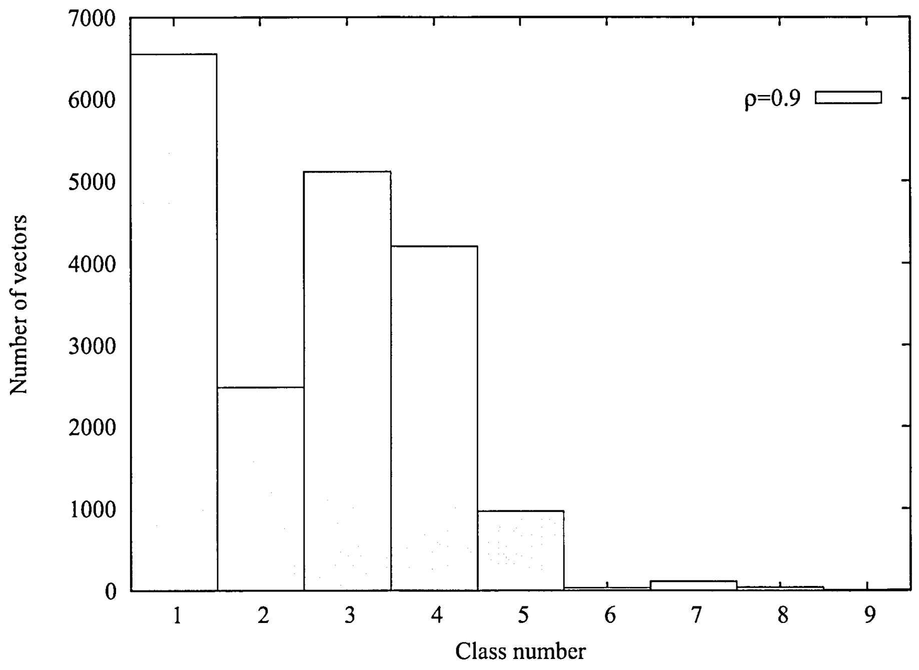


Figure A.3: State classification using ART2 (ρ=0.9).

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	3	4	1	2	5
2000	3	4	1	2	5
3000	3	4	1	2	5
4000	3	4	1	2	5
5000	3	4	1	2	5
6000	3	4	1	2	5
7000	3	4	1	2	5
8000	3	4	1	2	5
9000	3	4	1	2	5
10000	3	4	1	2	5
11000	3	4	1	2	5
12000	3	4	1	2	5
13000	3	4	1	2	5
14000	3	4	1	2	5
15000	3	4	1	2	5
16000	3	4	1	2	5
17000	3	4	1	2	5
18000	3	4	1	2	5
19000	3	4	1	2	5
20000	3	4	1	2	5

Table A.2: Classification stability of ART2 algorithm ($\rho = 0.9$).

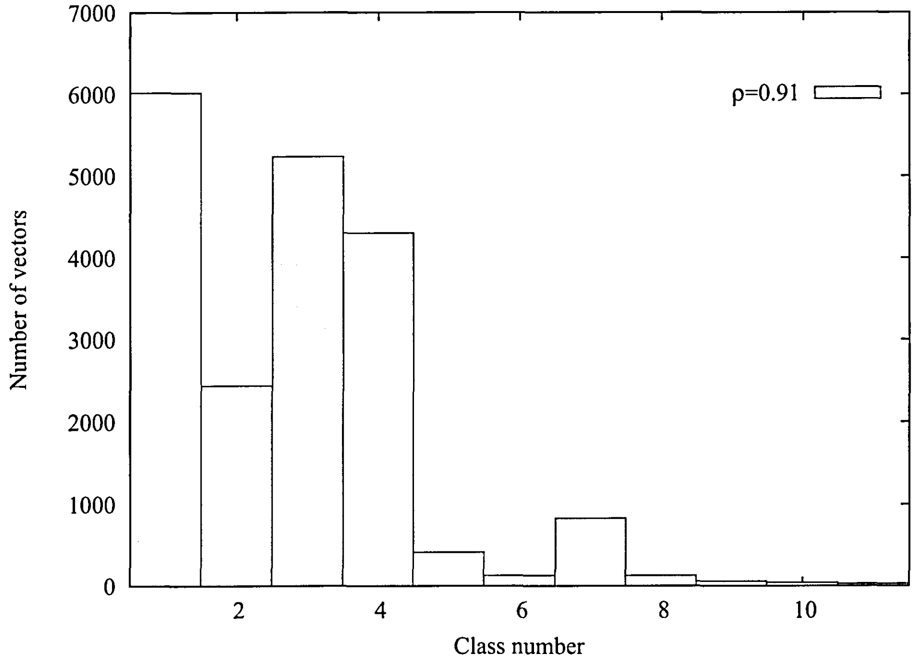


Figure A.4: State classification using ART2 (ρ=0.91).

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	3	4	1	2	7
2000	3	4	1	2	7
3000	3	4	1	2	5
4000	3	4	1	2	5
5000	3	4	1	2	5
6000	3	4	1	2	5
7000	3	4	1	2	5
8000	3	4	1	2	7
9000	3	4	1	2	7
10000	3	4	1	2	7
11000	3	4	1	2	7
12000	3	4	1	2	7
13000	3	4	1	2	7
14000	3	4	1	2	7
15000	3	4	1	2	7
16000	3	4	1	2	7
17000	3	4	1	2	7
18000	3	4	1	2	7
19000	3	4	1	2	7
20000	3	4	1	2	7

Table A.3: Classification stability of ART2 algorithm ($\rho = 0.91$).

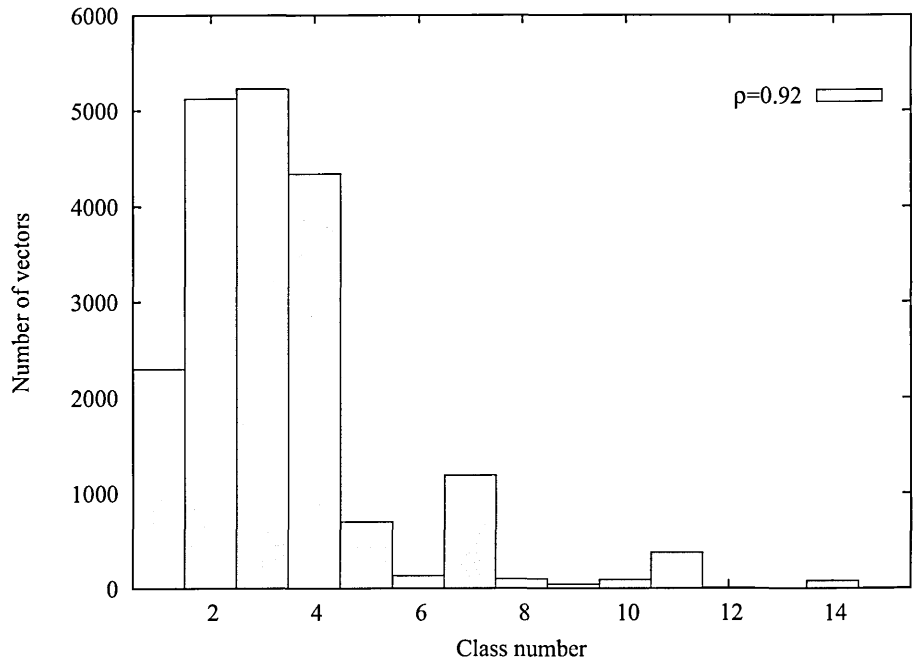


Figure A.5: State classification using ART2 (ρ=0.92).

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	3	4	5	1	8
2000	3	4	2	1	7
3000	3	4	2	1	7
4000	3	4	2	1	7
5000	3	4	2	1	7
6000	3	4	2	1	7
7000	3	4	2	1	7
8000	3	4	2	1	7
9000	3	4	2	1	7
10000	3	4	2	1	7
11000	3	4	2	1	7
12000	3	4	2	1	7
13000	3	4	2	1	7
14000	3	4	2	1	7
15000	3	4	2	1	7
16000	3	4	2	1	7
17000	3	4	2	1	7
18000	3	4	2	1	7
19000	3	4	2	1	7
20000	3	4	2	1	7

Table A.4: Classification stability of ART2 algorithm ($\rho = 0.92$).

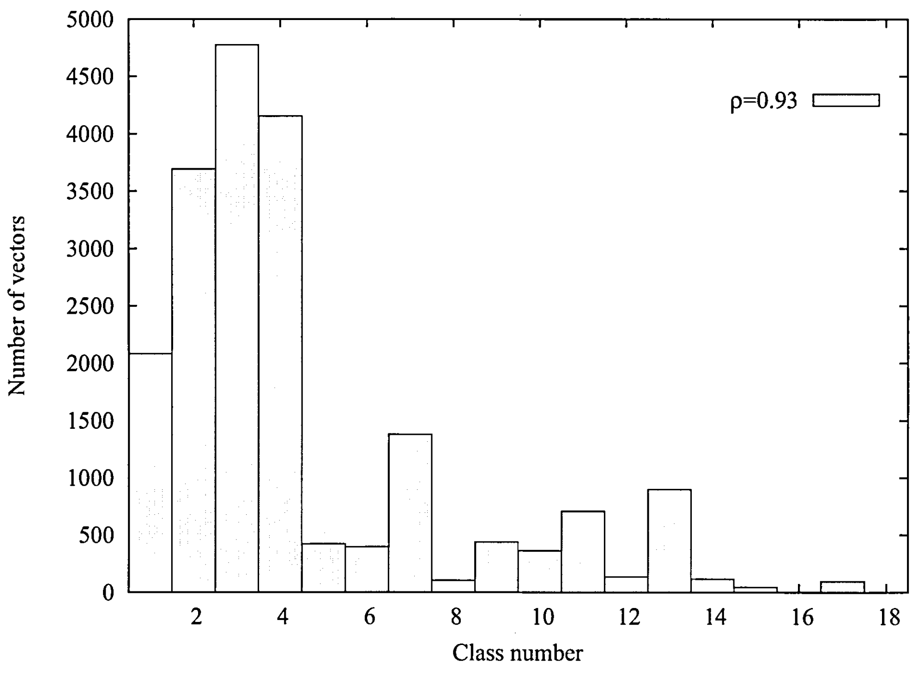


Figure A.6: State classification using ART2 (ρ=0.93).

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	3	4	7	2	5
2000	3	4	7	2	5
3000	3	4	7	2	5
4000	3	4	2	2	5
5000	3	4	2	2	5
6000	3	4	2	2	5
7000	3	4	2	2	5
8000	3	4	2	2	11
9000	3	4	2	2	11
10000	3	4	2	2	11
11000	3	4	2	2	11
12000	3	4	2	2	11
13000	3	4	2	2	11
14000	3	4	2	2	11
15000	3	4	2	2	11
16000	3	4	2	2	11
17000	3	4	2	2	11
18000	3	4	2	2	11
19000	3	4	2	2	11
20000	3	4	2	2	11

Table A.5: Classification stability of ART2 algorithm ($\rho = 0.93$).

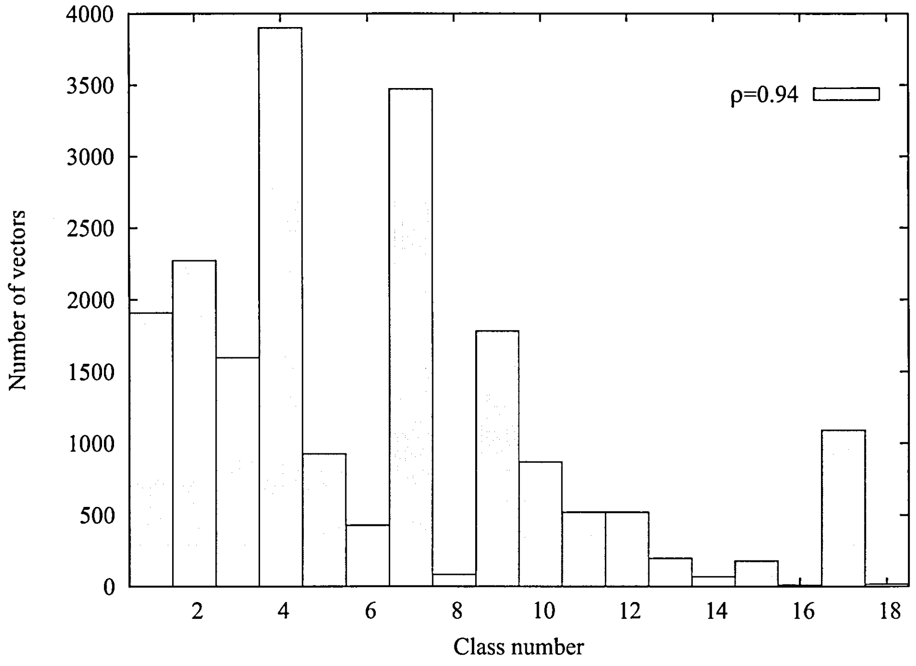


Figure A.7: State classification using ART2 (ρ=0.94).

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	3	4	5	1	6
2000	3	4	5	11	6
3000	3	4	5	1	6
4000	3	4	5	1	6
5000	3	4	5	1	6
6000	3	4	5	1	6
7000	3	4	5	1	6
8000	3	4	9	1	6
9000	3	4	5	1	6
10000	3	4	5	1	12
11000	3	4	5	1	12
12000	3	4	9	1	12
13000	3	4	9	1	12
14000	3	4	5	1	12
15000	3	4	5	1	12
16000	3	4	5	1	12
17000	3	4	5	1	12
18000	3	4	5	1	12
19000	3	4	5	1	12
20000	3	4	5	1	12

Table A.6: Classification stability of ART2 algorithm ($\rho = 0.94$).

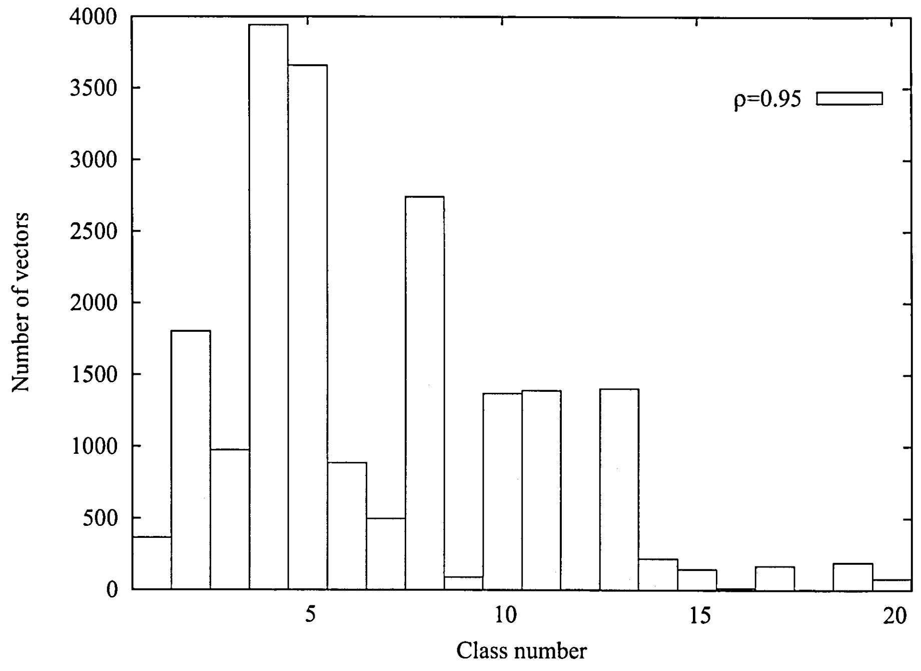


Figure A.8: State classification using ART2 (ρ=0.95).

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	4	5	10	13	7
2000	4	5	6	3	7
3000	4	5	6	3	7
4000	4	5	6	1	7
5000	4	5	6	1	7
6000	4	5	6	1	7
7000	4	5	10	13	7
8000	4	5	10	13	7
9000	4	5	10	13	7
10000	4	5	10	13	7
11000	4	5	10	13	7
12000	4	5	10	13	7
13000	4	5	10	13	7
14000	4	5	10	13	7
15000	4	5	10	13	7
16000	4	5	10	13	7
17000	4	5	10	13	7
18000	4	5	10	13	7
19000	4	5	10	13	7
20000	4	5	10	13	7

Table A.7: Classification stability of ART2 algorithm (ρ = 0.95).

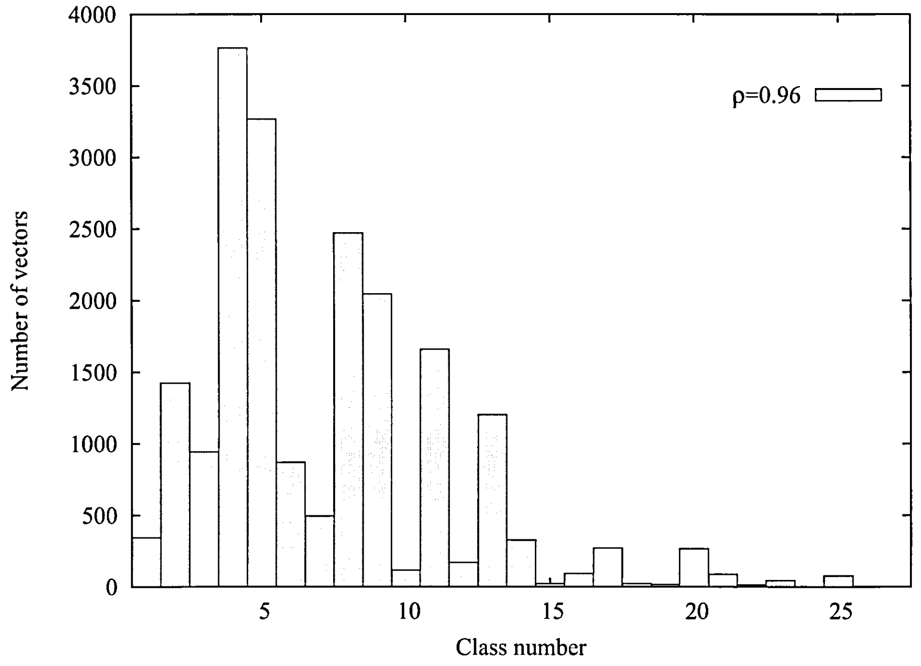


Figure A.9: State classification using ART2 (ρ=0.96).

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	4	5	11	13	7
2000	4	5	11	13	7
3000	4	5	11	13	7
4000	4	5	11	3	7
5000	4	5	11	13	7
6000	4	5	11	1	7
7000	4	5	11	1	7
8000	4	5	11	1	7
9000	4	5	11	13	7
10000	4	5	11	13	7
11000	4	5	11	13	7
12000	4	5	11	13	7
13000	4	5	11	13	7
14000	4	5	11	13	7
15000	4	5	11	13	7
16000	4	5	11	13	7
17000	4	5	11	13	7
18000	4	5	11	13	7
19000	4	5	11	13	7
20000	4	5	11	13	7

Table A.8: Classification stability of ART2 algorithm (ρ = 0.96).

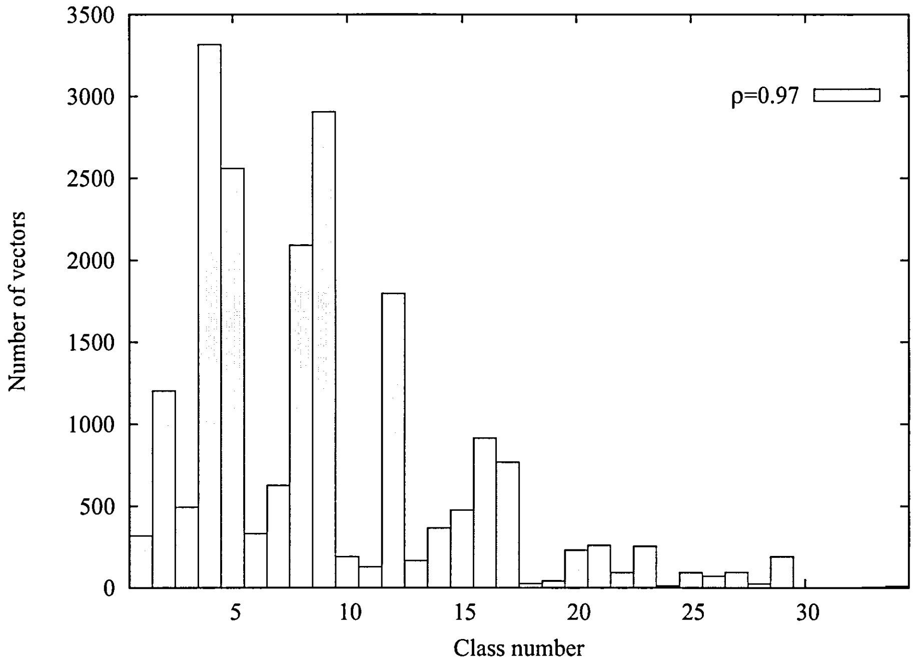


Figure A.10: State classification using ART2 (ρ=0.97).

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	4	5	12	16	6
2000	4	5	12	16	6
3000	4	5	12	16	6
4000	4	5	12	16	6
5000	4	5	12	16	6
6000	4	5	12	16	6
7000	4	5	12	16	6
8000	4	5	12	16	6
9000	4	5	12	16	6
10000	4	5	12	16	6
11000	4	5	12	16	6
12000	4	5	12	16	6
13000	4	5	12	16	6
14000	4	5	12	16	6
15000	4	5	12	16	6
16000	4	5	12	16	6
17000	4	5	12	16	6
18000	4	5	12	16	6
19000	4	5	12	16	6
20000	4	5	12	16	6

Table A.9: Classification stability of ART2 algorithm (ρ = 0.97).

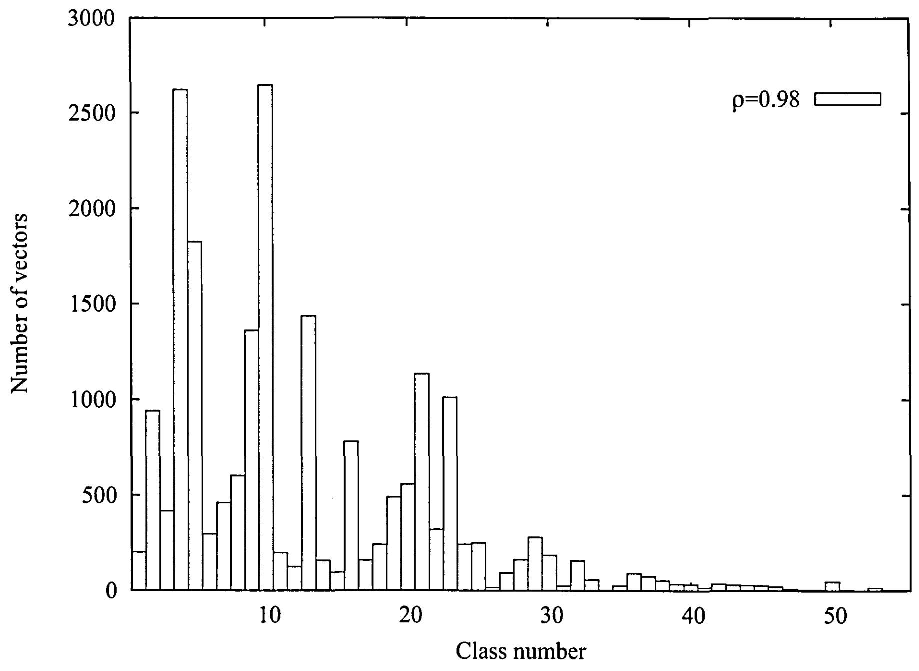


Figure A.11: State classification using ART2 (ρ=0.98).

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	4	5	13	19	24
2000	4	5	13	19	24
3000	4	5	13	19	24
4000	4	5	13	19	24
5000	4	5	13	19	24
6000	4	5	13	19	24
7000	4	5	13	19	24
8000	4	5	13	19	24
9000	4	5	13	19	24
10000	4	5	13	19	24
11000	4	5	13	19	24
12000	4	5	13	19	24
13000	4	5	13	19	24
14000	4	5	13	19	24
15000	4	5	13	19	24
16000	4	5	13	19	6
17000	4	5	13	19	6
18000	4	5	13	19	6
19000	4	5	13	19	6
20000	4	5	13	19	6

Table A.10: Classification stability of ART2 algorithm (ρ = 0.98).

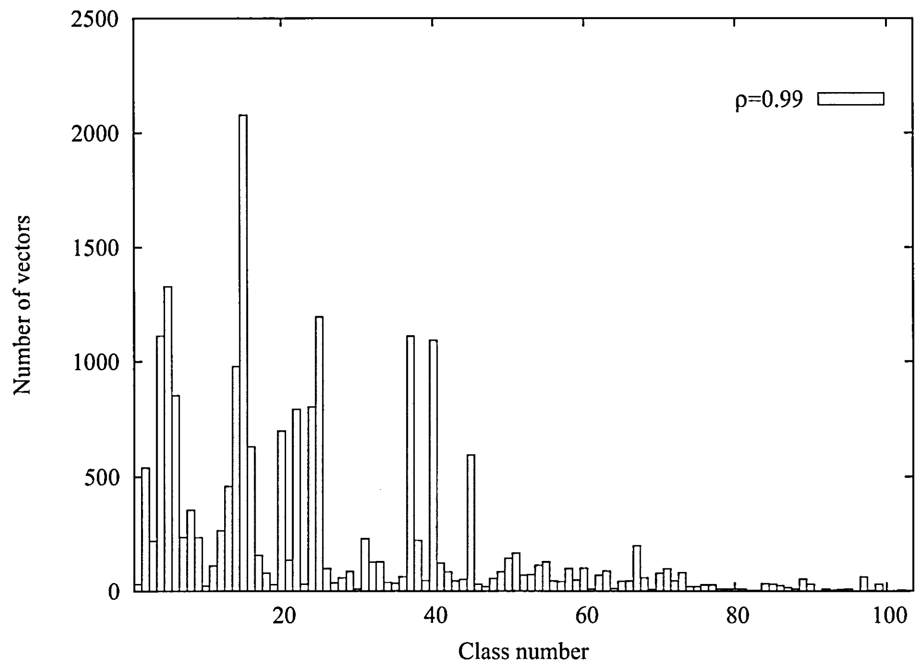


Figure A.12: State classification using ART2 (ρ=0.99).

Epoch	Class number				
	Vector A	Vector B	Vector C	Vector D	Vector E
1	1	1	1	1	1
1000	5	40	20	42	11
2000	5	40	20	37	11
3000	5	40	20	37	11
4000	5	40	20	37	11
5000	5	40	20	37	11
6000	5	40	20	37	11
7000	5	40	20	37	11
8000	5	40	20	37	11
9000	5	40	20	37	11
10000	5	40	20	37	11
11000	5	25	20	37	11
12000	5	25	20	37	11
13000	5	25	20	37	11
14000	5	25	20	37	11
15000	5	25	20	37	11
16000	5	25	20	37	11
17000	5	25	20	37	11
18000	5	25	20	37	11
19000	5	25	20	37	11
20000	5	25	20	37	11

Table A.11: Classification stability of ART2 algorithm (ρ = 0.99).

Appendix B

Experimental results

In this chapter the additional results from the experiments are presented. This includes evaluation of the environment's uncertainty, sensor accuracy and robot measurements. The data from real life peg-and-hole experiments is also presented. Different geometries and materials are investigated.

B.1 Environment evaluation

B.1.1 Sensor accuracy

The results included in the main body of the thesis represent the overall length of the vector calculated using the Equation 5.1.1 and Equation 5.1.1. In this section the force and torque readings for every axes are presented.

The sensor was mounted on the Puma 560 robot's wrist. To ensure that the TOOL coordinate system axis is aligned with the robot's WORLD coordinate system the "DO ALIGN" command was released from robot terminal. The sensor's internal coordinate system was aligned with the robot's TOOL CS. This was done executing a set of rotational commands (see JR3 sensor manual for details [26]). The load (value of 5 N) was suspended underneath the sensor. Since the direction of WORLD Z axis is parallel (within the robot's accuracy) to the gravitational force, the expected sensor's reading should be on Z axis only.

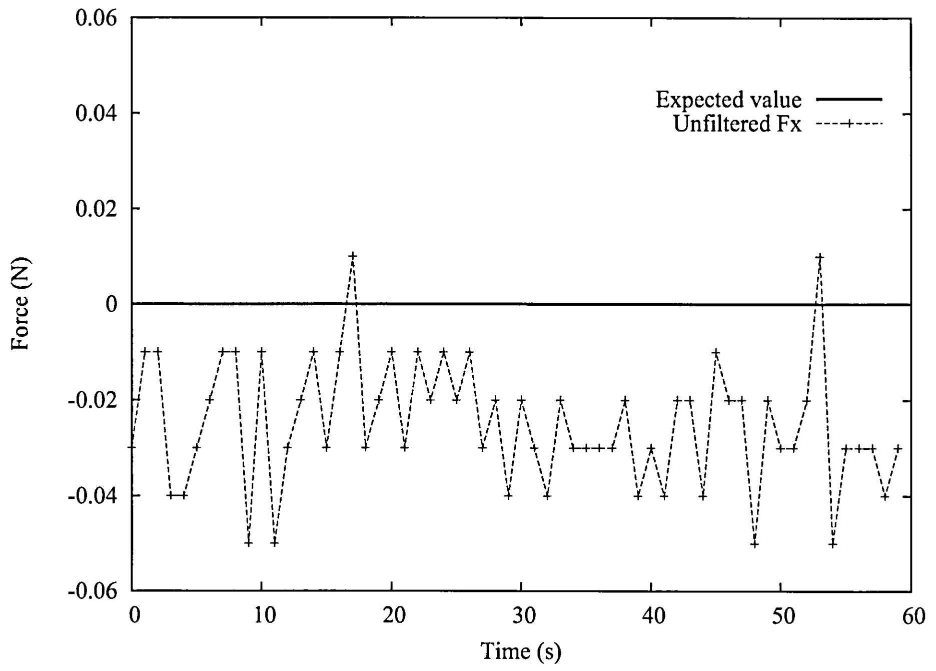


Figure B.1: Force reading on X-axis - unfiltered data.

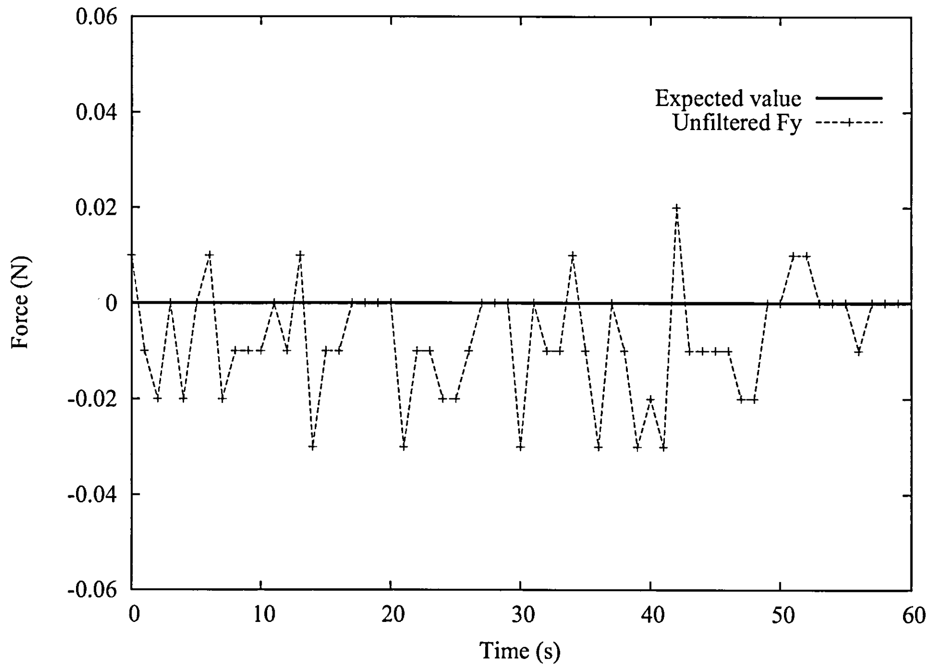


Figure B.2: Force reading on Y-axis - unfiltered data.

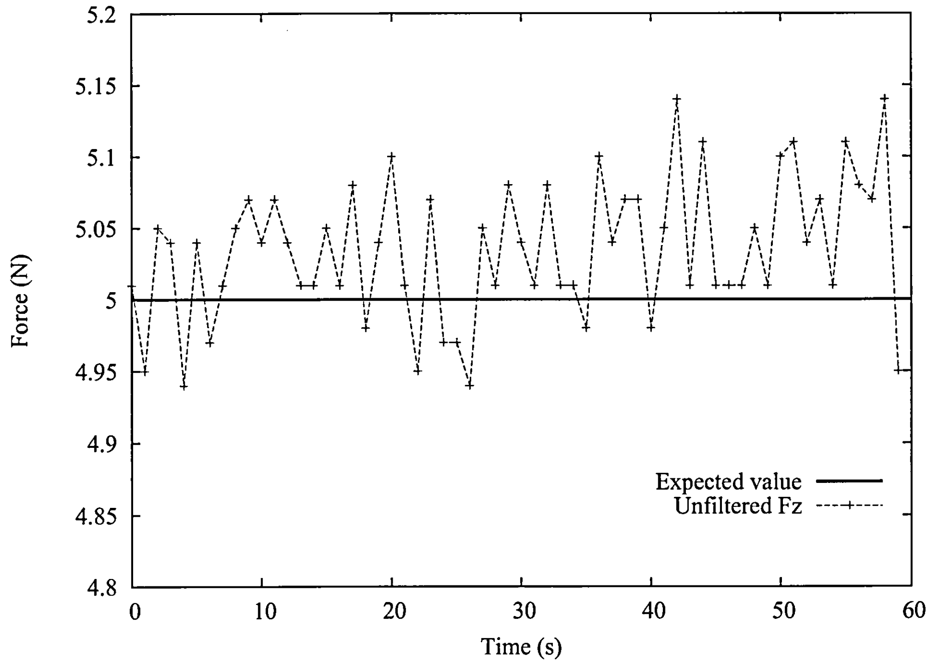


Figure B.3: Force reading on Z-axis - unfiltered data.

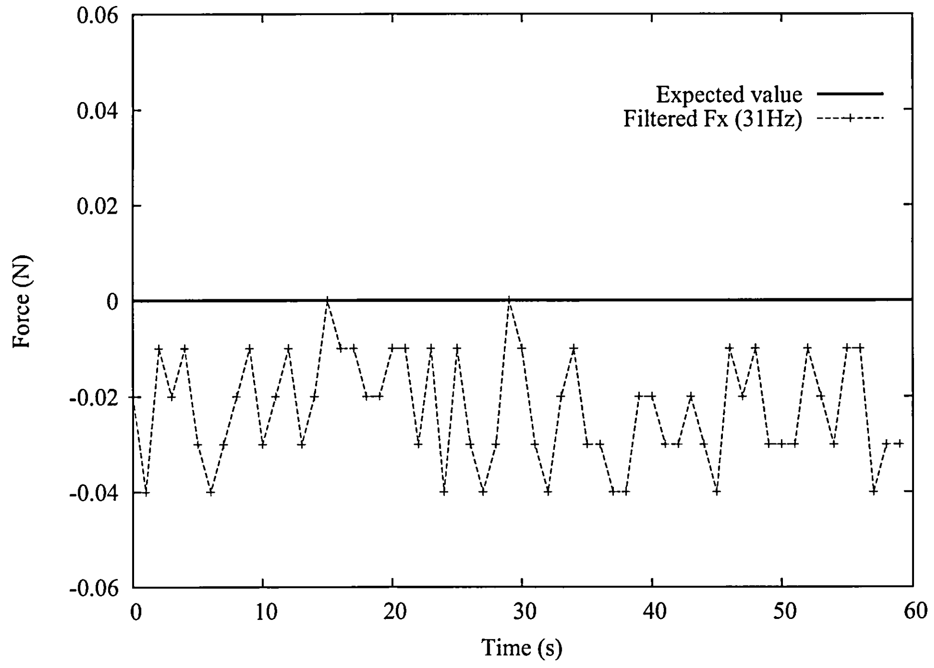


Figure B.4: Force reading on X-axis - 31Hz filter applied.

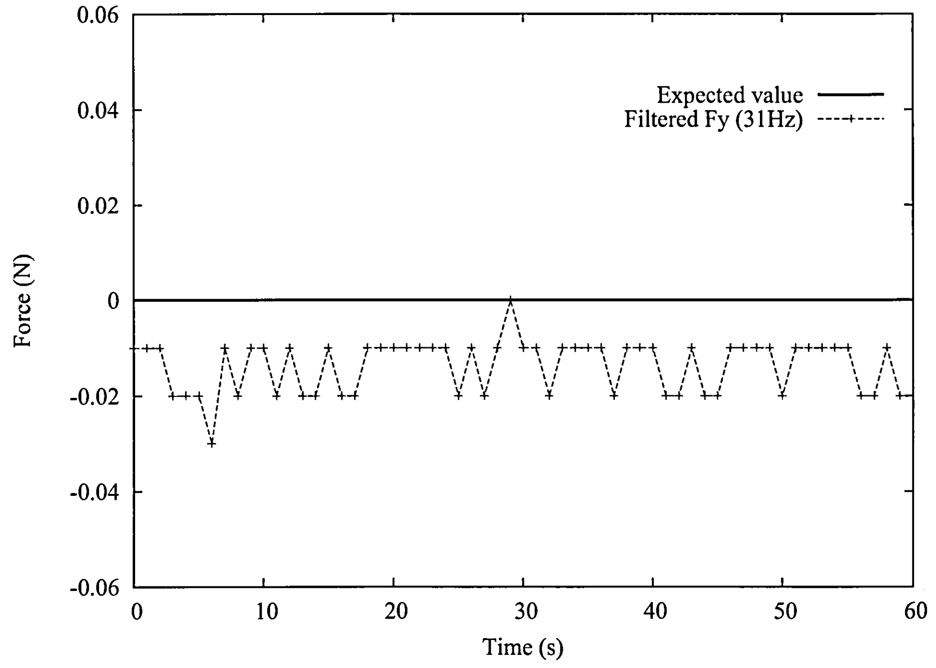


Figure B.5: Force reading on Y-axis - 31Hz filter applied.

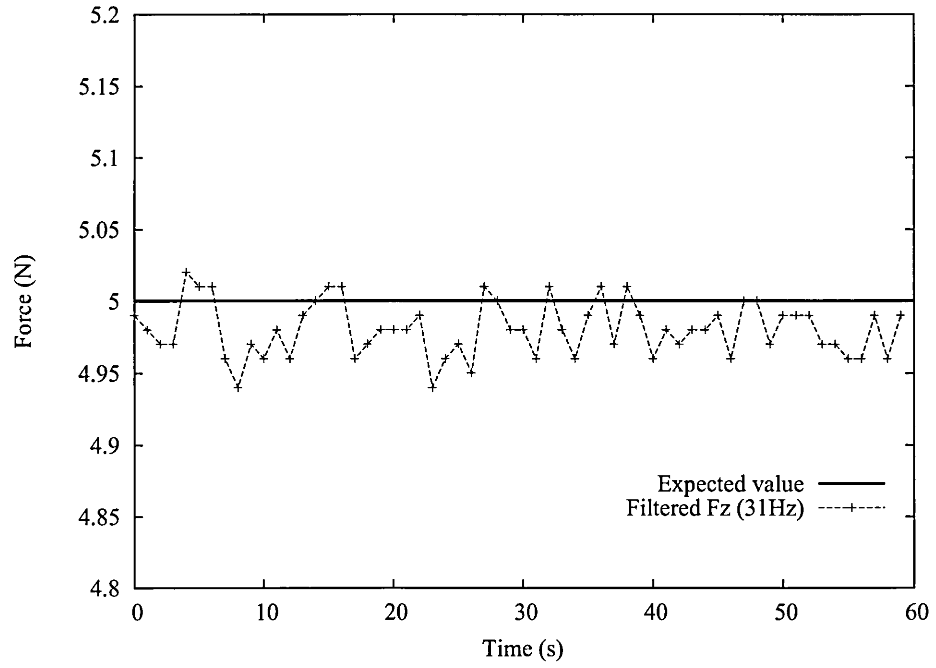


Figure B.6: Force reading on Z-axis - 31Hz filter applied.

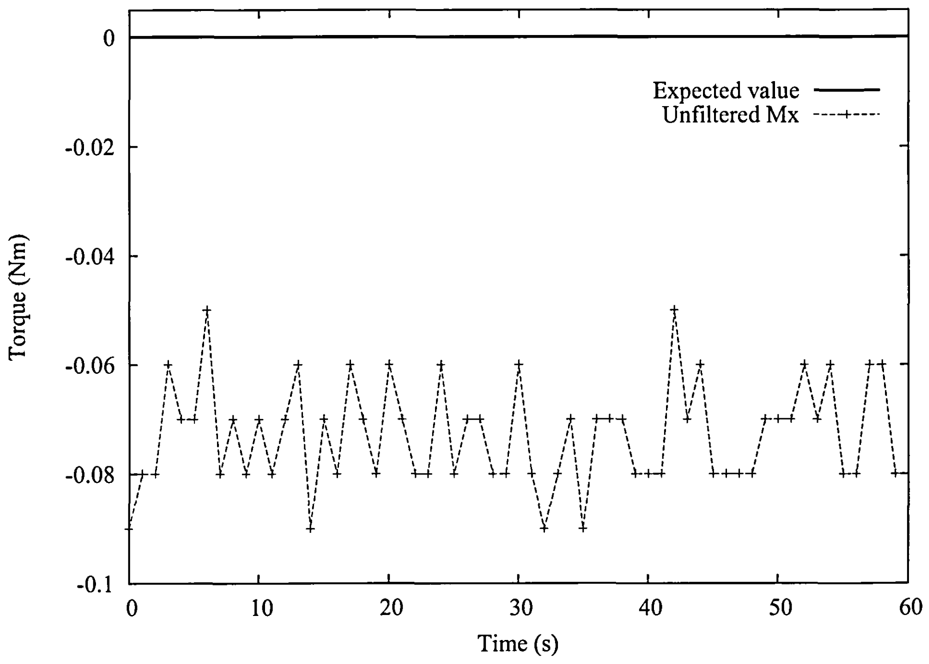


Figure B.7: Torque reading on X-axis - unfiltered data.

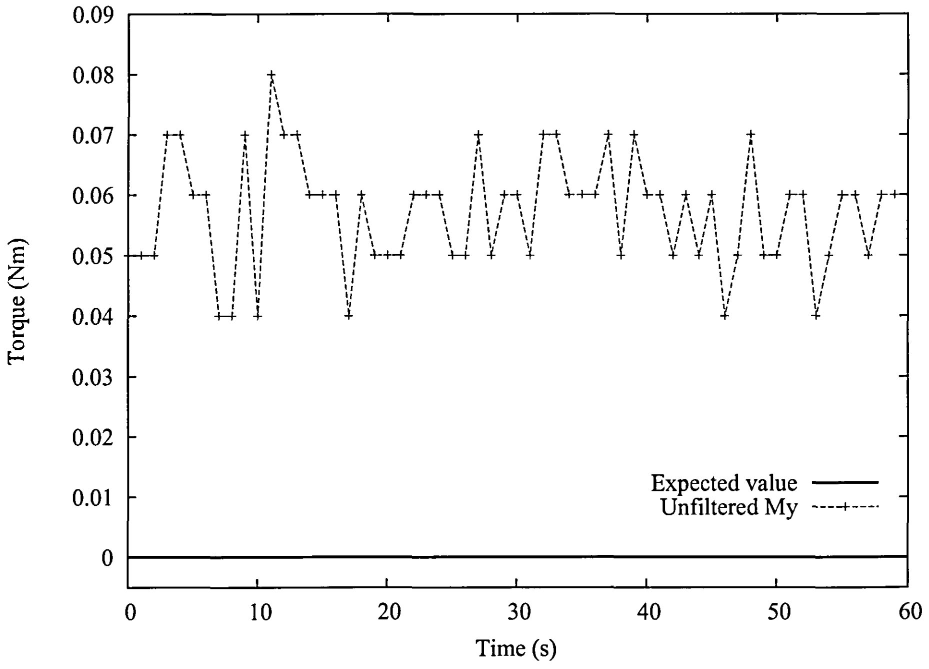


Figure B.8: Torque reading on Y-axis - unfiltered data.

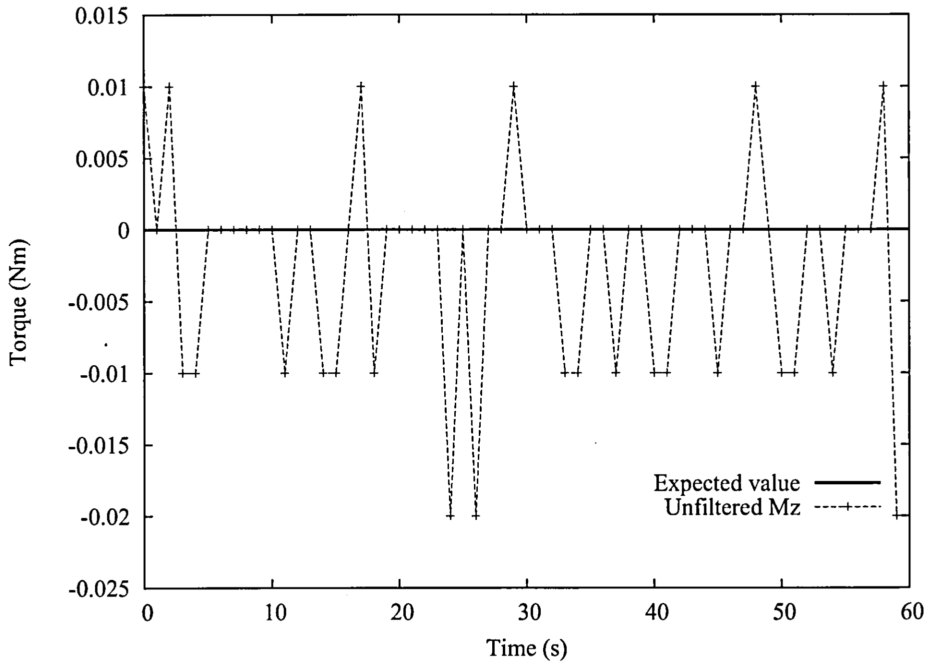


Figure B.9: Torque reading on Z-axis - unfiltered data.

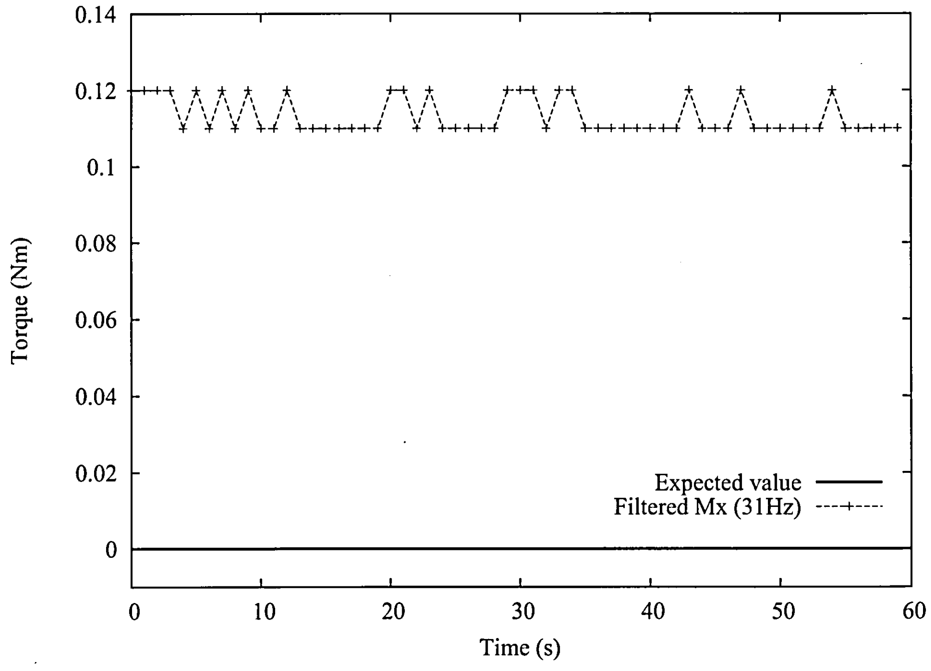


Figure B.10: Torque reading on X-axis - 31Hz filter applied.

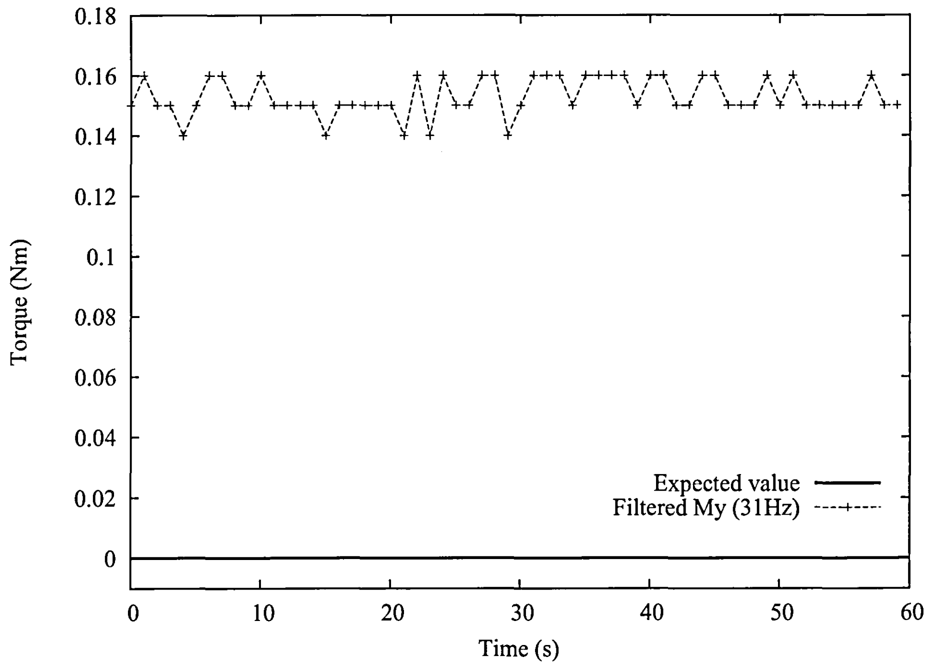


Figure B.11: Torque reading on Y-axis - 31Hz filter applied.

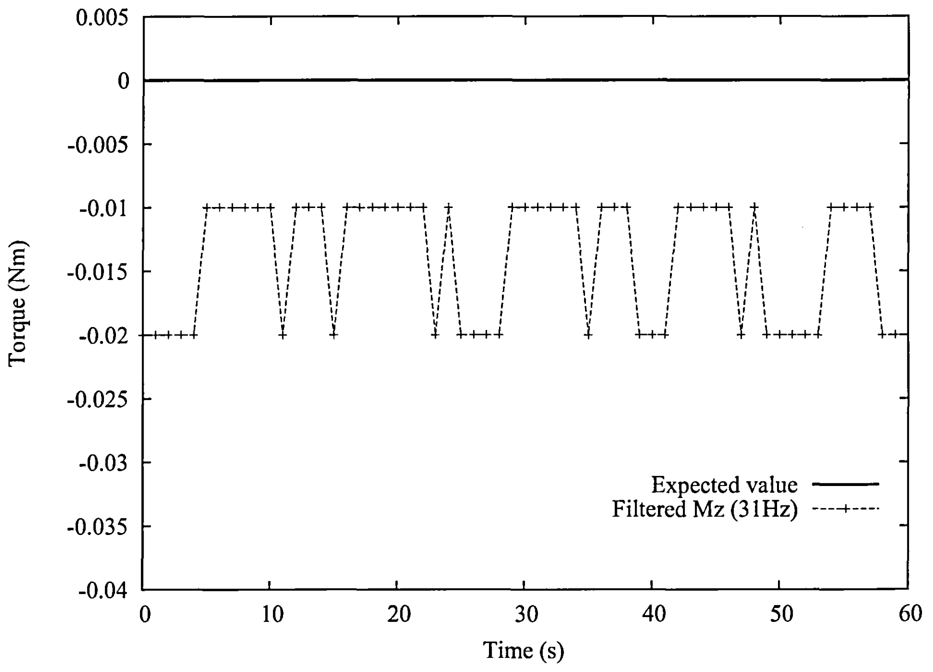


Figure B.12: Torque reading on Z-axis - 31Hz filter applied.

B.2 Experimental results

In this section the selected results from the experiments using real components are presented. The circular and square pegs were applied onto chamfered holes. The non-linear, chamfer-less peg-in-hole problem was also investigated.

The initial tests were performed using a Puma 560 robotic arm staring from a random position. The experiments (except the non-linear and square peg problem) were executed using the following settings:

Collision trigger	10 N
Translation Step X	0.2 mm
Translation Step Y	0.2 mm
Translation Step Z	0.2 mm
Rotation Step X	0.2 deg.
Rotation Step Y	0.2 deg.
Rotation Step Z	0.2 deg.

Table B.1: Standard settings for peg-in-hole insertion experiments.

B.2.1 Circular chamfer-less hole - additional results.

In this section the data acquired during the the insertion with the q-learning algorithm with ϵ-greedy action-selection policy applied is presented. It is a very similar approach to the greedy method. The agent still chooses the action with the highest estimated value but also allows exploration with a small probability ϵ. The method was implemented in a way so the value of ϵ depends on the number of the current episode. This reduces the amount of exploration as the number of plays increases. the probability of greedy action a_g was set according to Equation 4.1. This method together with the q-learning algorithm was applied onto the circular chamfer-less peg-in-hole insertion. To compare the results with the data acquired using the other type of hole geometries the Softmax action-selection methods were included in the main body of the thesis. The same starting conditions were applied to the test presented below.

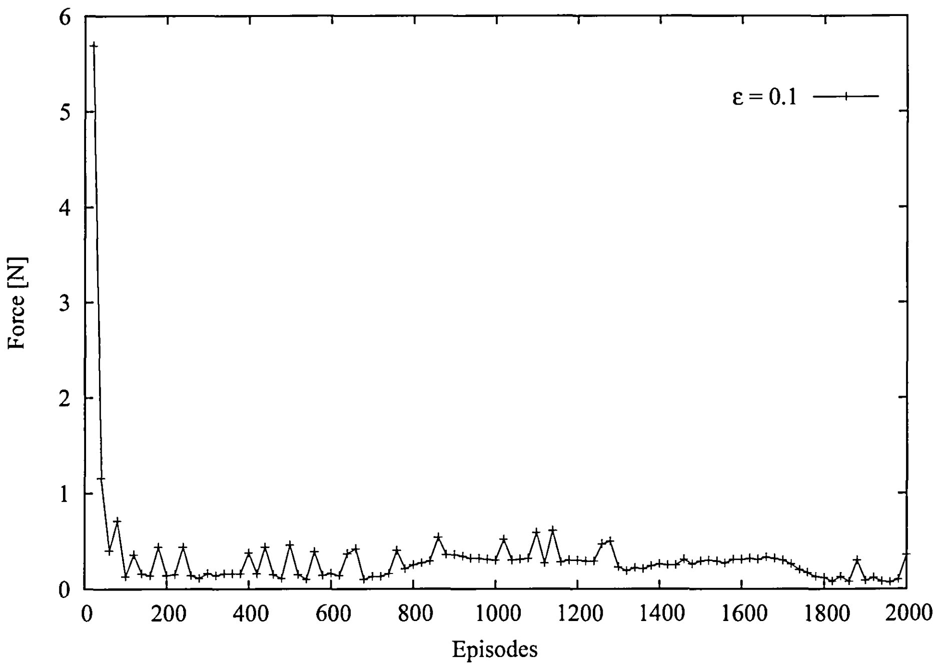


Figure B.13: Force data - chamfered hole, q-learning with ϵ-greedy applied.

The force value and its change throughout the experiment can be seen in the Figure B.13.

A significant reduction of contact forces can be observed while the experiment progresses. The high values, caused by the collisions between mating parts are quickly (within first 40 episodes) lowered to the value of below 1 Newton.

The occasional sharp increase of the force is caused by collision due to random actions taken (constant value of $\epsilon = 0.1$).

The graph from Figure B.14 presents the change of torque values. Similarly to the force signal torque was very quickly reduced and stabilised at the value of around 0.5 Nm. The very high torque values combined with high contact forces recorded during first 40 episodes suggests the occurrence of jamming conditions during the initial stages of the learning process. The controller learnt to recognise those error conditions and avoid them in the future.

The significant reduction of both contact forces and the resulting torques is also clear indication that the intelligent controller was able to learn the necessary skills to perform the peg-in-hole assembly. No additional knowledge about the geometry, part's orientation or position was supplied to the system.

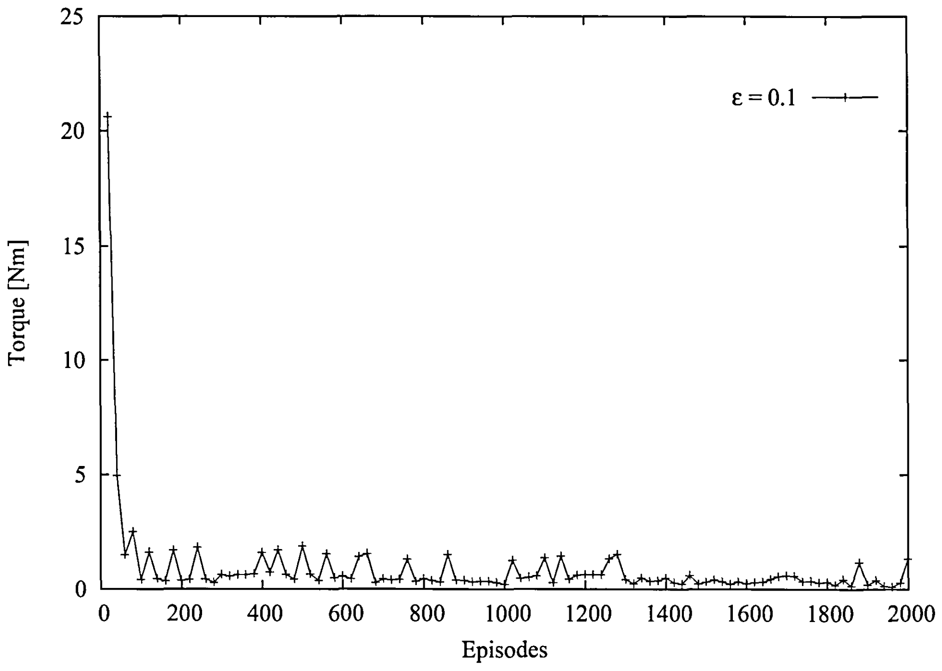


Figure B.14: Torque data - chamfered hole, q-learning with ϵ-greedy applied.

The data presented on the histogram below (see Figure B.15) represents the number of unsuccessful insertions per 10 episodes. It clearly indicates the sudden drop in error conditions caused by high contact forces. The results from Figure B.15 conclude the presented early data.

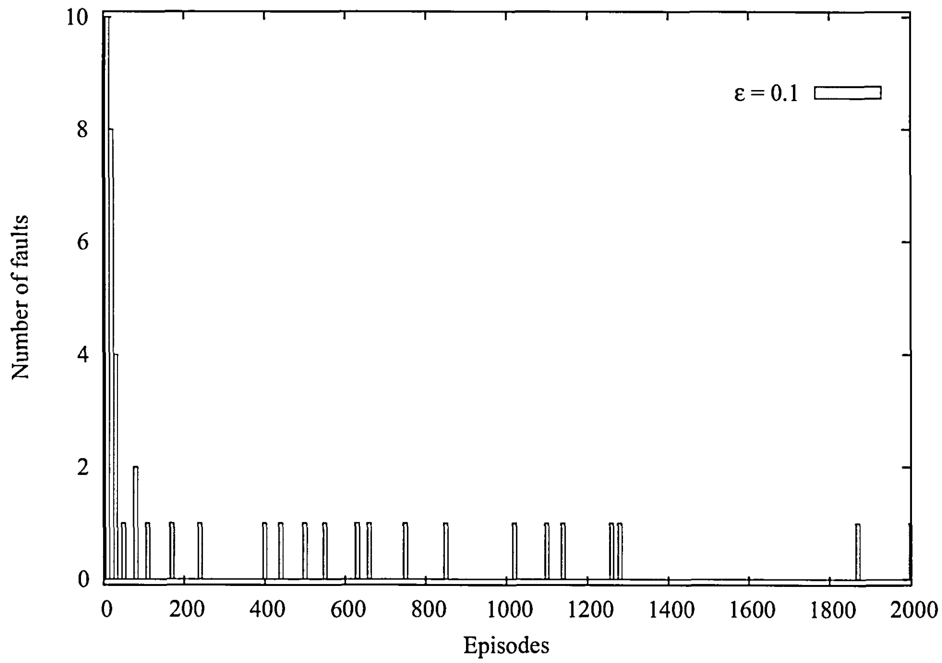


Figure B.15: Number of failed insertions (per 10 episodes).

B.2.2 Softmax and ϵ-greedy comparison data.

The raw data from the graph (see Figure 6.7) included in Section 6.2.1 is presented. For clarity the graph in the main body of the thesis only shows the first 400 episodes plotted and the data points were interpolated using Bezier approximation function.

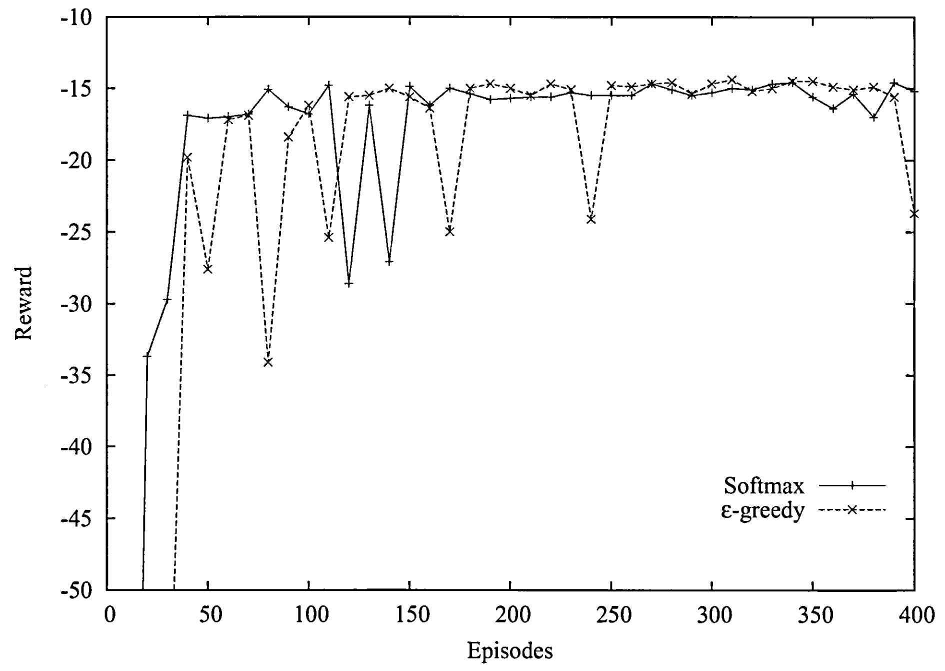


Figure B.16: Softmax and ϵ-greedy policy during first 400 episodes.

The on-line performance curve for the Softmax action-selection method quickly rises and stabilises on the value of -15. In the case of ϵ-greedy policy, due to a greater number of random actions, the on-line performance graph more steadily approaches the optimal value of cumulative reward.

B.2.3 Q-learning agent with ϵ-greedy policy applied on chamfer-less peg-in-hole problem

In this Section the additional results from chamfer-less circular peg-in-hole experiments are presented. In the main body of this thesis (see Section 6.3) data acquired during the tests with the q-learning algorithm and Softmax action-selection method applied were presented. Here the ϵ-greedy method was used instead. The starting conditions were exactly the same so the results can be directly compared to those included in Section 6.3

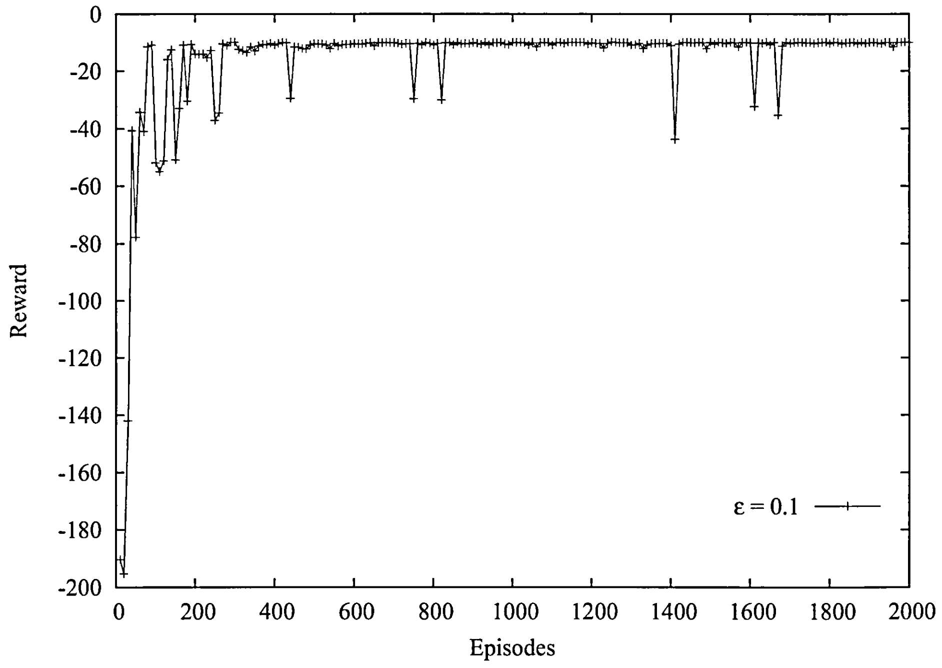


Figure B.17: On-line performance data from the experiment.

The on-line performance curve was presented on Figure B.17. The transition time was 180 episodes long. After that the controller stabilised the cumulative reward on the value -10. The stable period was occasionally interrupted by the occurrence of a collision between mating parts.

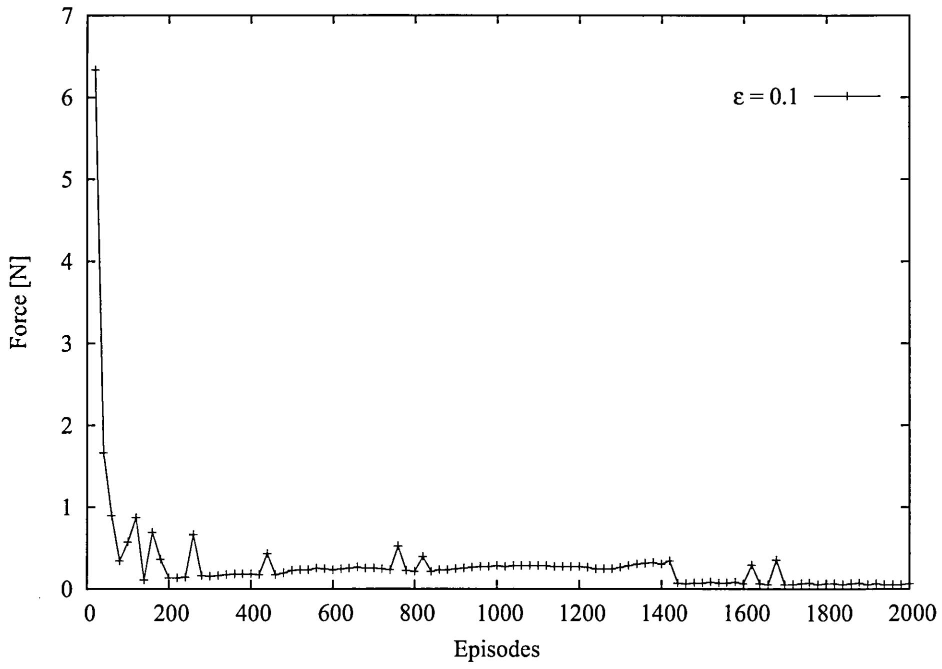


Figure B.18: Force data from peg-in-hole experiment.

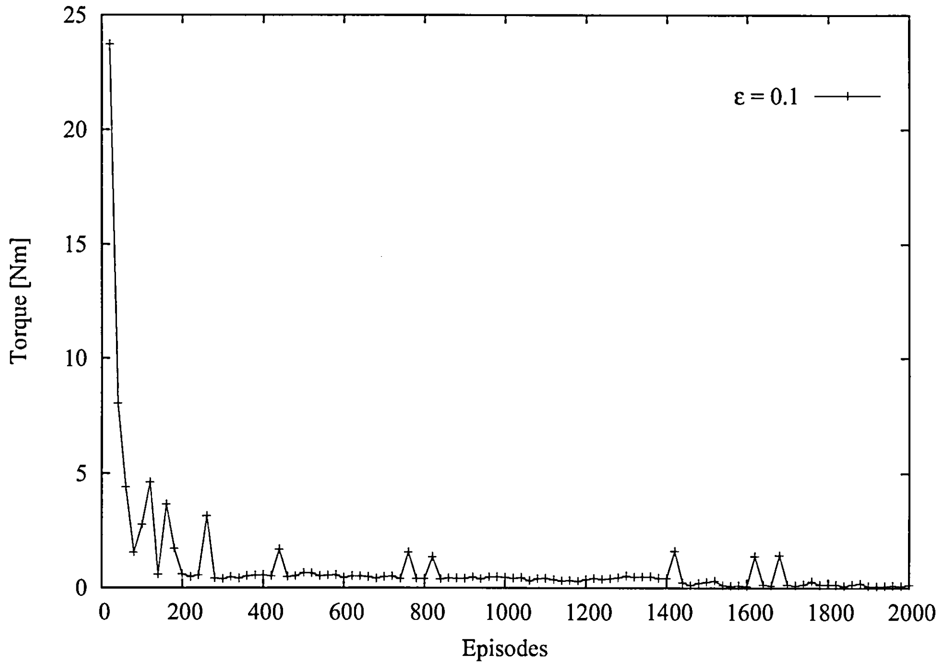


Figure B.19: Torque data from peg-in-hole experiment.

The sudden increases in force and torque values during episodes 249, 435, 745, 816, 1410, 1607 and 1661 (see Figures B.18 and B.19) are present on both graphs. this indicates the jamming condition. The episode numbers correspond to those from the histogram from Figure B.20 below.

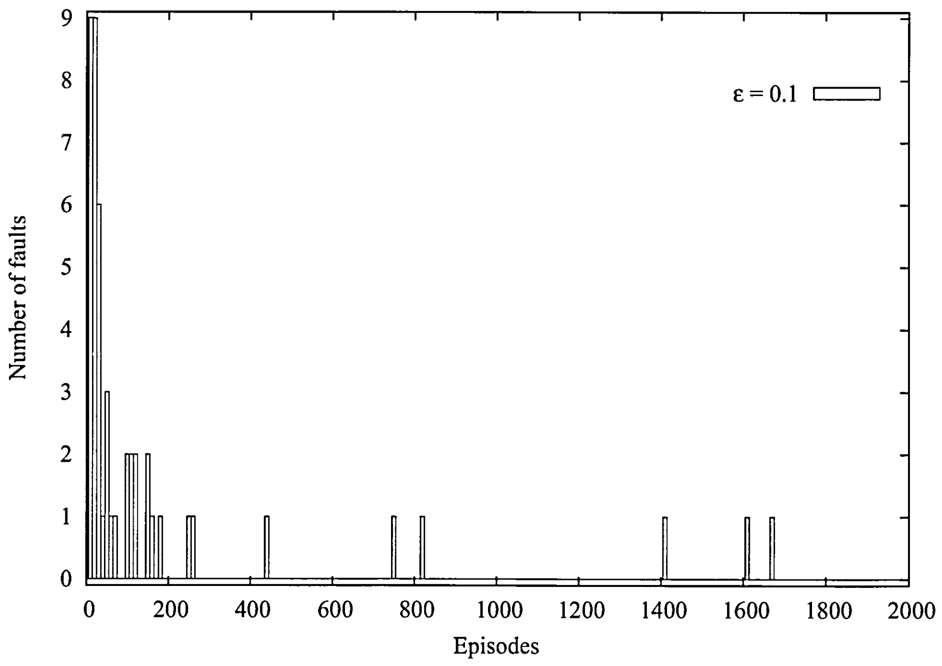


Figure B.20: Number of failed insertions (per 10 episodes).

B.2.4 Complete raw data from square peg-in-hole insertion.

In this Section the full data for the chamfered square hole is presented. The first 600 episodes were presented and discussed in the main part of the thesis. The experiment was stopped after 1800 episodes due to reasons explained before (see Section 6.4 for details).

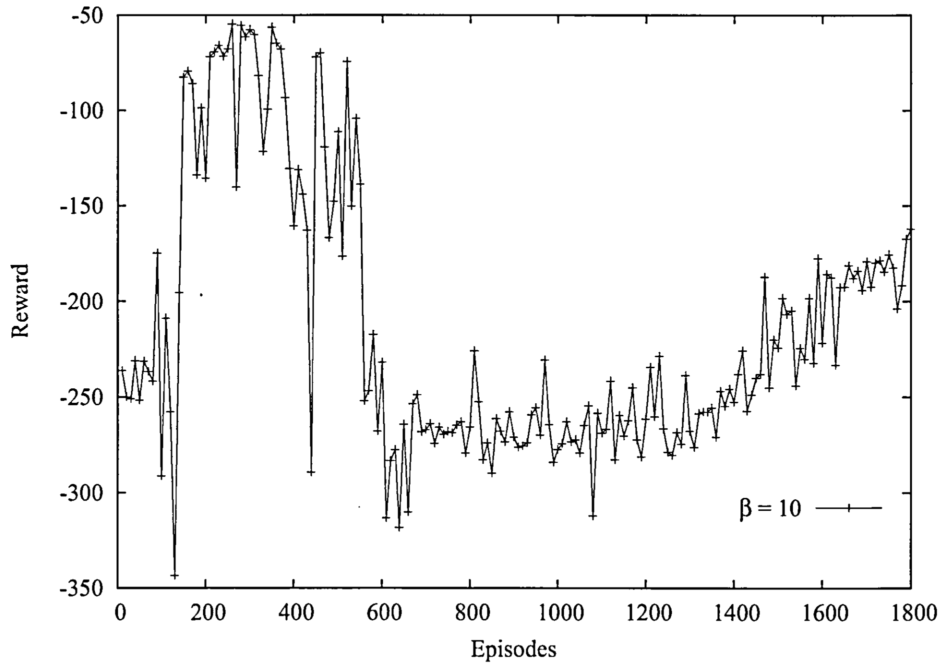


Figure B.21: On-line performance data from the experiment.

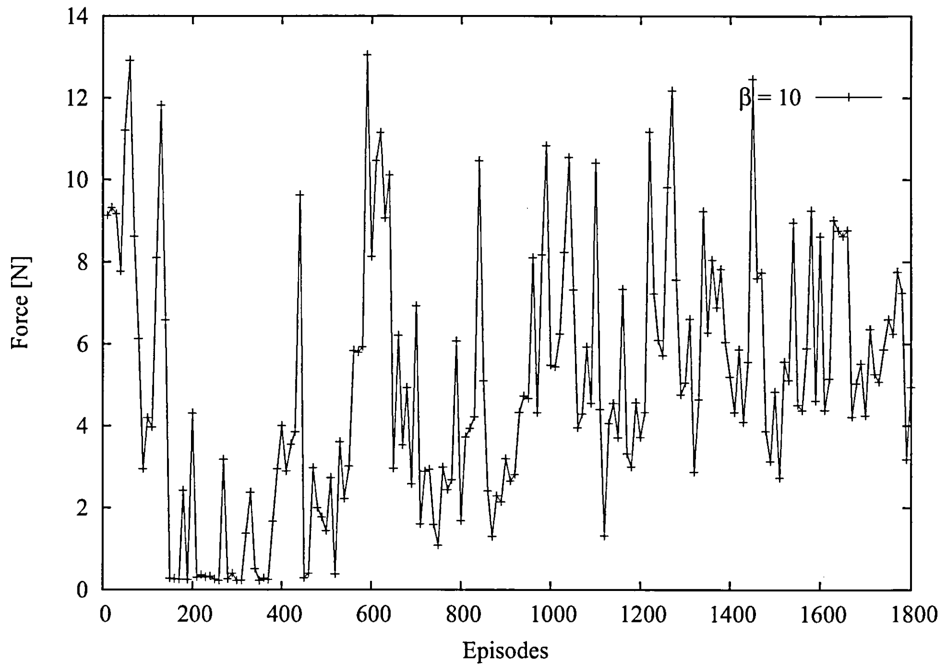


Figure B.22: Force data from peg-in-hole experiment.

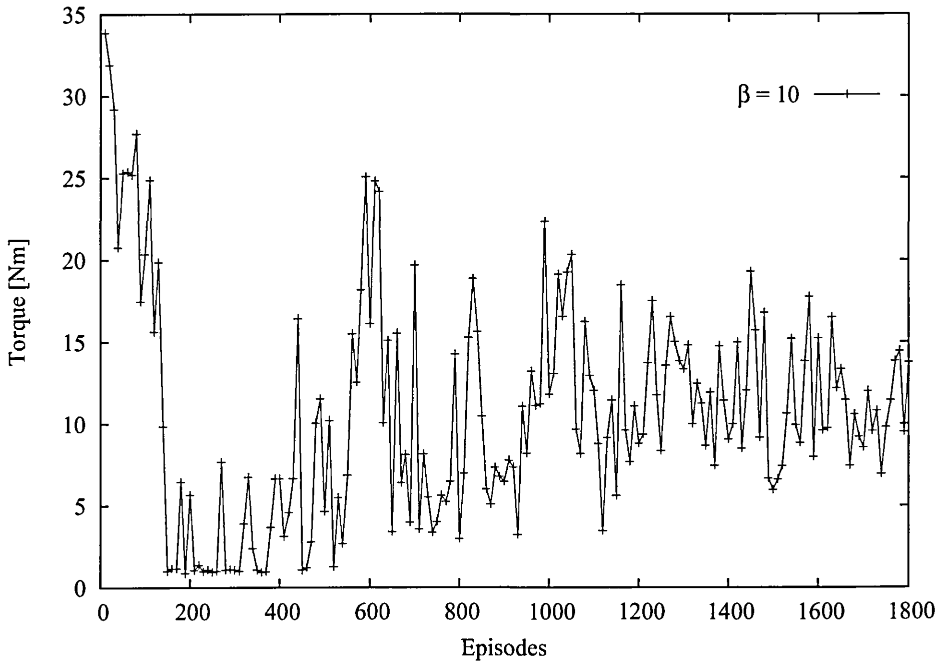


Figure B.23: Torque data from peg-in-hole experiment.

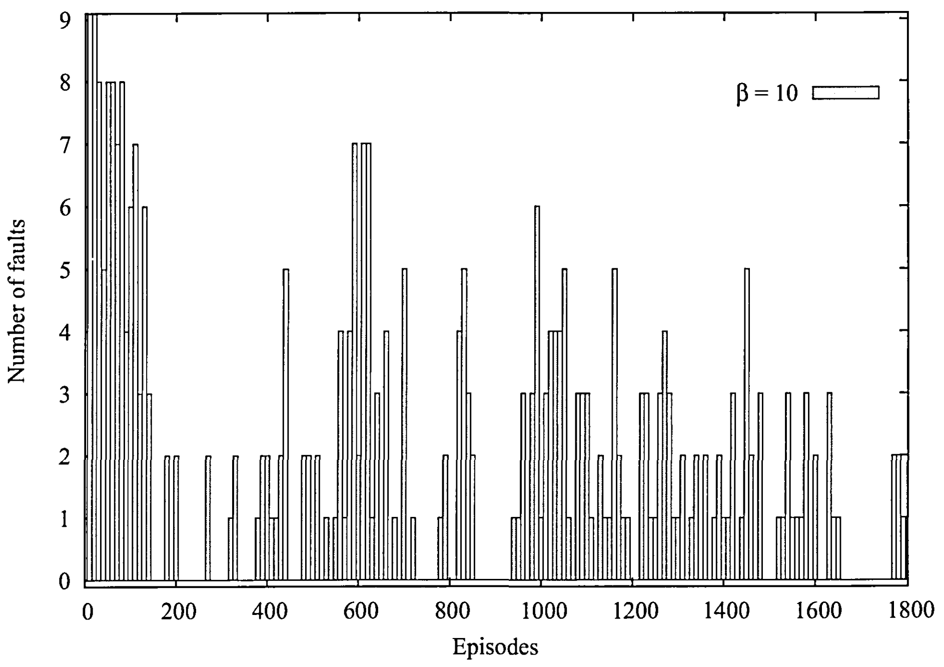


Figure B.24: Number of failed insertions (per 10 episodes).

It can be seen from the presented graphs that some improvement of controller design and the learning parameters is necessary. The suggestions for future work to extend the presented features namely: on-line performance, force and torque data readings can be found in the Section 7.4.

Appendix C

Rotations in TOOL Coordinate System

To request an incremental movement the `DO MOVE HERE:TRANS(X,Y,Z,O,A,T)` command is sent to VAL II. The `X`, `Y`, `Z` values represent the increment on each axis respectively. Although `O`, `A` and `T` does NOT represent rotations about those axes. These angles are formed by the TOOL CS when referenced to the `X`, `Y`, `Z` WORLD CS. The reference coordinates for WORLD mode are fixed to the robot arm base and the TOOL reference coordinates are fixed in the gripper with their origin at the centre of its mechanical interface.

When the TOOL mode is chosen the gripper is meant to move parallel to any of TOOL axis and rotate about those axes.

The angles `O`, `A` and `T` are defined as follows (after robot manual):

- `O` - a measurement of the angle formed between the WORLD `X` axis and a projection of the TOOL `Z` on the WORLD `XY` plane.
- `A` - a measurement of the angle formed between the TOOL `Z` and a plane parallel to the WORLD `XY` plane.
- `T` - a measurement of the angle formed between the TOOL `Y` and a plane parallel to the WORLD `XY` plane.

The output from the AI controller is in form of δ_x, δ_y, δ_z values, which represent movement correction and 3 values $\delta\theta x, \delta\theta y, \delta\theta z$ for rotation correction. Since the definition of those differ from the definition of O, A, T angles, the transformation between these values had to be found.

With help of Dr M. Rybaczuk from Wroclaw University of Technology the following algorithm was developed:

1. Calculate the rotation matrix based on the equation (summation convention over k):

$$\theta_{ij} = \delta_{ij}\cos\phi + (1-\cos\phi)\hat{n}_i\hat{n}_j + \sin\phi\epsilon_{ikj}\hat{n}_k \tag{C.1}$$

   where $\underline{\hat{\mathrm{n}}}$ is a unary vector describing orientation of the rotation axis, ϕ is a rotation angle, θ is a rotation matrix.

2. Having the rotation matrix we can calculate transformation from TOOL CS to WORLD CS:

$$\underline{\hat{\mathrm{b}}}^{(j)} = \theta_{i,j}\underline{\hat{\mathrm{a}}}^{(j)} \tag{C.2}$$

   where $\underline{\hat{\mathrm{b}}}^{(j)}$ and $\underline{\hat{\mathrm{a}}}^{(j)}$ are TOOL and WORLD CS unary vectors respectively.

3. Now using the description of O, A,T from robot manual we can describe relations between $\underline{\hat{\mathrm{a}}}^{(j)}$ and $\underline{\hat{\mathrm{b}}}^{(j)}$ as follows:

$$\cos(O) = \underline{\hat{\mathrm{a}}}^{(2)}\underline{\hat{\mathrm{t}}} \tag{C.3}$$

$$\cos(O) = \underline{\hat{\mathrm{a}}}^{(2)}\underline{\hat{\mathrm{b}}}^{(3)} \tag{C.4}$$

$$\cos(T) = \underline{\hat{\mathrm{a}}}^{(1)}\underline{\hat{\mathrm{b}}}^{(2)}. \tag{C.5}$$

   where $\underline{\hat{\mathrm{t}}}$ can be calculated as follows:

$$\underline{\hat{\mathrm{u}}} = \underline{\hat{\mathrm{u}}}_{\|} + \underline{\hat{\mathrm{u}}}_{\perp} = \underline{\hat{\mathrm{a}}}^{(3)}(\underline{\hat{\mathrm{b}}}^{(3)}\underline{\hat{\mathrm{a}}}^{(3)}) + \left(\underline{\hat{\mathrm{b}}}^{(3)} - \underline{\hat{\mathrm{a}}}^{(3)}(\underline{\hat{\mathrm{b}}}^{(3)}\underline{\hat{\mathrm{a}}}^{(3)})\right) \tag{C.6}$$

$$\underline{\hat{\mathrm{t}}} = \frac{\underline{\hat{\mathrm{u}}}_{\perp}}{|\underline{\hat{\mathrm{u}}}_{\perp}|} \tag{C.7}$$

After close investigation it became clear that `O`, `A`,`T` angles are components of Euler Z-Y-Z orientation description. The appropriate software was written to calculate the constant rotation increment value.

Appendix D

ART2 algorithm

During tests of the ART2 implementation the algorithm sometimes was not able to stabilise data on the F_1 layer. The F_0 layer was also added to the previous configuration. The equations from Section 4.3.2 had to be slightly modified. The final set of equations that governs activities on each of the sub-layers for layers F_0 and F_1 is as follows:

- for Layer F_0:

$$w'_i = I_i + au'_i \tag{D.1}$$

$$x'_i = \frac{w'_i}{e + \|\mathbf{w}'\|} \tag{D.2}$$

$$p'_i = u'_i \tag{D.3}$$

$$q'_i = \frac{p'_i}{e + \|\mathbf{p}'\|} \tag{D.4}$$

$$v'_i = f(x'_i) + bf(q'_i) \tag{D.5}$$

$$u'_i = \frac{v'_i}{e + \|\mathbf{v}'\|} \tag{D.6}$$

- for layer F_1:

$$w_i = q'_i + au_i \tag{D.7}$$

$$p_i = u_i + \sum_j g(y_j) z_{ij} \tag{D.8}$$

$$x_i = \frac{w_i}{e + \|\mathbf{w}\|} \tag{D.9}$$

$$q_i = \frac{p_i}{e + \|\mathbf{p}\|} \tag{D.10}$$

$$v_i = f(x_i) + bf(q_i) \tag{D.11}$$

$$u_i = \frac{v_i}{e + \|\mathbf{v}\|} \tag{D.12}$$

The function $f(x)$ is a threshold linear function. In the simulations the equations were computed in the sequence shown for each pass through the loop.

A *reset* vector is also calculated to record the match between the top-down expectation and the bottom-up priming, as follows:

$$r_i = \frac{q_i + cp_i}{e + \|\mathbf{q}\| + \|c\mathbf{p}\|} \tag{D.13}$$

Appendix E

Software description

E.1 Floppy drive emulator

First the data including the operating system boot sequence from the original VALII disk had to be extracted. To accomplish that dedicated software was written. The robot's floppy drive was directly connected to the RS232C port on PC computer. The connection cable had to be made. The software on PC was acting as robot's controller and initiated the transmission from the floppy according to the protocol regime. The all files were copied bit by bit over the designed link from the floppy drive directly to the hard drive on the host PC. When the data transmission was accomplished the floppy drive was unplugged. The connection cable was slightly modified and connected to the socket of the robot's controller. The program *pfloppy* was run in the background to allow floppy drive emulation. From the controller's perspective the whole system is seen as a CRT terminal and ordinary disk drive.

E.2 Data structures

The listings of ANSI C code for data structures design used for purpose of this project are presented below. This implementation gives the flexibility of building different network designs with different number of layers applied. The reinforcement learning agent's structure allows the easy implementation of both SARSA and q-learning methods.

The structures defining the topology (layers) of the ART2 neural network.

```
typedef struct {            // A LAYER OF A NET:
    int        units;       // number of units in this layer
    double*    w;           // W sublayer of F1
    double*    x;           // X sublayer of F1
    double*    v;           // V sublayer of F1
    double*    u;           // U sublayer of F1
    double*    output;      // output units
    double*    p;           // P sublayer of F1
    double*    p_old;       // old P units for stabilisation
    double*    w_old;       // old W units for stabilisation
    double*    q;           // Q sublayer of F1
    double**   weight;      // connection weights to unit
    BOOL*      inhibited;   // inhibition status of F2 unit
} LAYER;


typedef struct {            // A NET:
    LAYER*     F0;          // F0 layer
    LAYER*     F1;          // F1 layer
    LAYER*     F2;          // F2 layer
    int        winner;      // last winner in F2 layer
    double     a;           // parameter A of the ART2
    double     b;           // parameter B of the ART2
    double     c;           // parameter C of the ART2
    double     d;           // parameter D of the ART2
    double     e;           // parameter E of the ART2
    double     theta;       // parameter Theta
    double     rho;         // vigilance parameter
} NET;
```

The data structure for the reinforcement learning agent:

```
typedef struct {          // A REINFORCEMENT AGENT:
    int        states;    // number of states
    int        actions;   // number of actions
    double     gamma;     // Gamma learning ratio
    double     alpha;     // Alpha learning ratio
    double     epsilon;   // epsilon-greedy exploration
    double**   q;         // Q state-actions values
} AGENT;
```

E.3 DDCMP protocol

As mentioned in Section 3.3.2, the difference between the description in the robot's manual and real life controller's response was spotted. According to the manual, *fill* bytes are suppose to be added to each frame to be sent to the controller. In reality the fill bytes are added by the controller so the supervisory computer does not have to include them. The correct data flow for establishing connection looks as follows:

```
FROM VAL: (10) FF FF 05 05 C0 00 00 01 75 95 FF FF   <START>
FROM VAL: (10) FF FF 05 05 C0 00 00 01 75 95 FF FF   <START>
FROM VAL: (10) FF FF 05 05 C0 00 00 01 75 95 FF FF   <START>
   [...]
TO VAL:     (8) 05 07 C0 00 00 01 46 55              <STACK>
FROM VAL: (12) FF FF 05 01 C0 00 00 01 C0 55 FF FF    <ACK>
```

If the supervisor switch is enabled (`EN SUPERVISOR` from the robot terminal) the system will immediately follow the `ACK` with `LUN2 READ DATA` message:

```
FROM VAL II: (18) FF FF 81 04 C0 00 01 01 13 81
                  02 02 00 00 A0 78 FF FF            <DATA>
TO VAL II:    (8) 05 01 C0 01 00 01 91 95            <ACK>
```

Appendix F

List of publications

The printouts of conference publications are included in this Chapter.

Learning manipulative skills using an ANN approach based on sensorial and descriptive information sources

P. Chmielarczyk[1], **M. Howarth**[2], **L. Brignone**[3]
[1] School of Engineering, Sheffield Hallam University, Sheffield S1 1WB, UK,
email: P.Chmielarczyk@shu.ac.uk.
[2] School of Engineering, Sheffield Hallam University, Sheffield S1 1WB, UK,
email: M.Howarth@shu.ac.uk.
[3] Department of Mechanical and Manufacturing Engineering, The Nottingham Trent University, Nottingham NG1 4BU, UK,
email. lorenzo.brignone@ntu.ac.uk.

ABSTRACT

The aim of this research is to design a non-linear controller based on an Artificial Neural Network (ANN) implementation, which is able to perform an intelligent robotic assembly of mechanical components. Differently from other researchers' work, the main aim of this study is the investigation of the different information sources and their application to the "peg in hole" insertion problem. Amongst these, one source is sensorial (signal from force and torque sensor) and three are descriptive (previous actions database, geometry of parts and coded task description). To receive F/T data a six-axis sensor is used, the set of unary vectors normal to the surfaces forming the hole represents the geometry of components and a previous actions database contains the sets of successful insertions for different types of parts' geometry.

1 INTRODUCTION

There is widespread acceptance that the adoption of automated assembly operations could provide a solution to the high costs related to the employment of human labour and supervision in the execution of such processes. Early research work by Simunovic [9] concluded that most common assembly operations could be modelled as simple peg in hole insertions.

Traditional force control strategies have given way to applications of ANN and Artificial Intelligence (AI). The applications of AI to the peg in hole problem range from the employment of various ANN algorithms to the application of Fuzzy Inference Systems. Asada [1] pioneered the application of Backpropagation to assembly tasks, using the network's learning capabilities to approximate compliant control laws. Backpropagation based controllers were also developed by Gullapalli et al [3] and Howarth [4], these two approaches feature substantial differences in the architecture of the reinforcement learning rule. Cervera et al [2] defined a finite set of contact configurations and used Self Organising Maps to associate the current contact state to a predefined one. Lopez et al [5] applied a FuzzyARTMAP network to learn the 'pattern motion' association from a table of pre-taught templates. Brignone in his studies [6] used a similar algorithm but extended the problem by merging F/T data and parts geometry. In his work, the network was trained with a set of normalised vectors representing the normal to the surfaces, which form the hole.

A new approach to the design of the system is proposed. It includes the main process (ANN controller) and two child processes (F/T sensor driver and computer-robot connection module) running in parallel. The ANN controller was built in a modular form (as a plug-in). This simplified the whole system and increased its flexibility enabling different ANN controller architectures to be developed and tested.

2 METHODOLOGY

During this research, authors will extend the earlier work by investigating how the ANN controller should combine different information sources in order to generate a successful assembly strategy. The information sources mainly concern: task description, geometry and dimensions of parts, online force and torque contact data analysis and information derived from previous moves (experience). The originality of this approach lies in the definition of a methodology to merge information deriving from different sources to attempt an interpretation of the contact state, in terms of contact location and peg's attitude. The work carried out on data pre-processing and interpretation is used to provide the tools for the definition of the controller's decision-making strategy, which also includes the use of information on previous stages of the insertion.

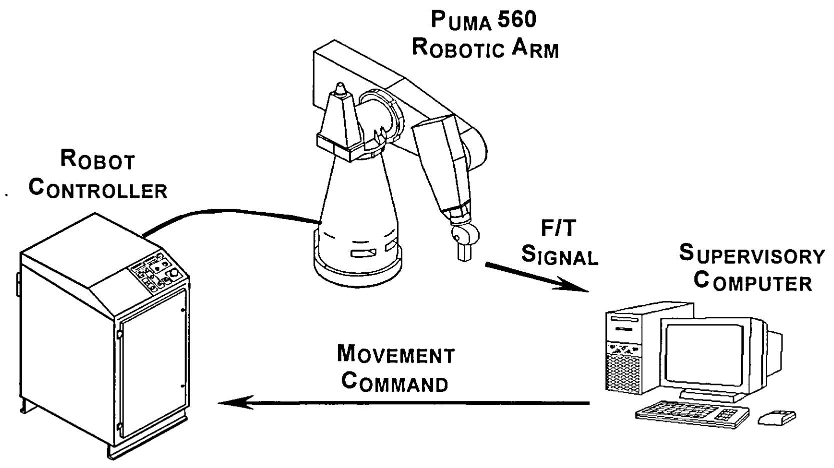


Figure 1 System configuration.

The robot arm and controller are the main part of the system. The Supervisory computer is connected to the controller via serial port. F/T sensor was mounted on the robot wrist. The signal is transmitted to the computer and then used as an input to the ANN controller. The new arm position and orientation values are calculated. The incremental motion request is sent to the controller using "supervisory" mode (see Figure 1).

2.1 ANN controller architecture

The ANN controller structure is shown in Figure 2. The controller architecture consists two main parts: ART network and reinforcement algorithm. In order to generalise large and continuous state space (s), the controller has to use function approximators. One of the simplest way to do this is to partition the continuous state space, and treat each class as an autonomous object.

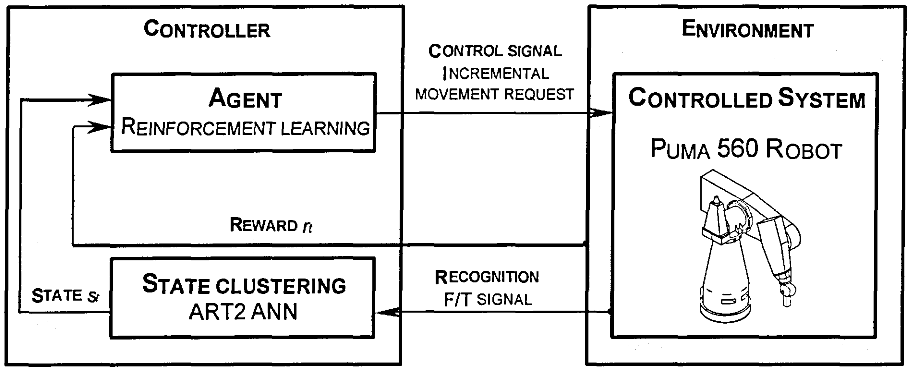


Figure 2 Controller architecture.

The state domain was divided into a set of classes using geometry information extracted from a CAD drawing. For this purpose, the ART network was used. The class number was set as input for reinforcement learning describing the actual contact forces as well as geometric position of its occurrence. Using this configuration, the reinforcement algorithm needs to learn how to deal only with a particular part of the component's geometry. This task decomposition is expected to speed up the learning process significantly.

2.1.1 Reinforcement learning

Following Sutton and Barto's [7] description, reinforcement learning problem is meant to be a straightforward framing of the problem of learning from iteration to achieving the goal. The basic one step *Q-learning* control algorithm was used. The method of *Q-learning* is based on the function Q, defined as followed:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a_{t+t}) - Q(s_t,a_t)\right] \tag{1}$$

where: t is a discrete time step, an agent receives representation of the environment; s is a finite set of states, an agent can be in; a is a finite set of actions, an agent may perform; Q is a transition function, mapping each state-action pair to a successor state; r is a reward function, mapping states-action pairs to payoffs, α is a learning rate and γ is a discount factor. These last two ratios are learning parameters.

As described above the class number (indicating a location) was used as the state description.

2.1.2 Adaptive Resonance Theory (ART)

To cluster the continuous state space using F/T signal and geometry information the *adaptive resonance theory* (ART) network was used. In this kind of ANN, information is repeatedly transferred back and forth between the layers. If the right pattern is developed during this process, a stable oscillation occurs. This is the neural network equivalent of resonance. Only at this stage does learning occur. This type of ANN is able to respond quickly to previously learned data and remains able to learn when novel data is presented.

Because of the analogue nature of F/T signal, the ART2 algorithm was chosen. It differs from ART1 structure only in the type of the input patterns. The price for that additional capability is an increase in complexity on the "F_1" processing level which contains a number of sub-layers that serve to remove noise, enhance contrast and to normalise analogue input pattern [8]. The overall structure of the ART2 network is shown in Figure 3 bellow.

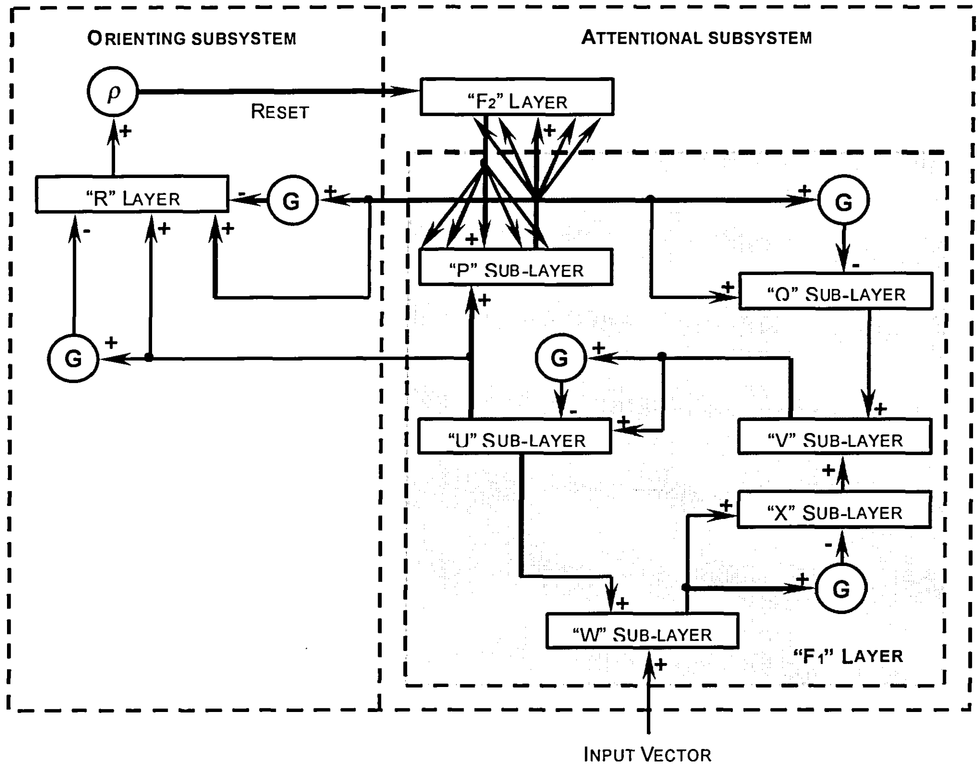


Figure 3 ART2 architecture [8].

The contact point localisation was proposed by Brignone [6]. The algorithm consists of two stages: network off-line training and signal classification. During off-line training, the network is presented with the set of unary vectors normal to the surfaces forming the hole. In the next step, the on-line F/T signal is presented to the pre-trained network, which returns the class number corresponding to the location of the contact.

2.2 Software design

The ANN and reinforcement algorithms were created using ANSI C language and compiled with GNU C compiler. Whole system was implemented in 200MHz Pentium PC under Debian GNU/Linux 3.0 operating system.

A new approach to the design of the system is proposed. The ANN controller was built in a modular form (as a plug-in). This simplified the whole design and increased its flexibility enabling different ANN controller architectures to be developed and tested. The overall structure is shown on the Figure 4.

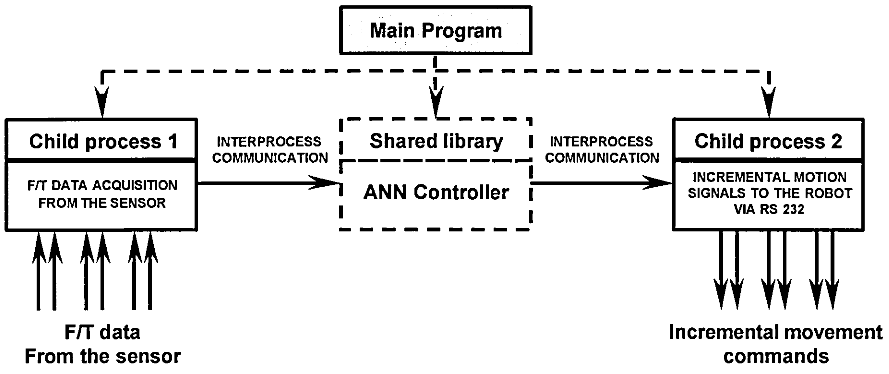

Main Program
Child process 1
F/T DATA ACQUISITION FROM THE SENSOR
INTERPROCESS COMMUNICATION
Shared library
ANN Controller
INTERPROCESS COMMUNICATION
Child process 2
INCREMENTAL MOTION SIGNALS TO THE ROBOT VIA RS 232
F/T data From the sensor
Incremental movement commands

Figure 4 Software architecture.

3 CONCLUSIONS

During initial tests, the controller was able to accurately cluster and classify the state domain. The ART network was responding quickly and accurately selecting the closest match using F/T signals and pre-taught geometry information. The new approach to the software design has proved its flexibility. A number of simple controller designs were developed and tested. Using this method switching between different controller designs during the insertion process was also possible.

Work continues to improve the capability of this system along with extensive testing using a range of mechanical components.

REFERENCES

1 **H. Asada**, "Teaching and Learning of Compliance using Neural Nets", Proceedings of the 1990 IEEE Int. Conf on Robotics and Automation,1990, pp 1237,1243.

2 **E. Cervera, A. del Pobil and M. Serna,** "A sensor-Based Approach to Motion in Contact Task Planning", Proc 1995 IEEE/RSJ Int. Conf on Intelligent Robots and Systems, 1995, 2: pp 468,473.

3 **V. Gullapalli**, "Skilful control under uncertainty via direct reinforcement learning", Robotics and autonomous systems, 1995, pp 237,246.

4 **M. Howarth,** "An Investigation Of Task Level Programming For Robotic Assembly". PhD thesis. The Nottingham Trent University, January 1998.

5 **I. Lopez, M. Howarth**, "Learning manipulative skills with ART", Proceedings of the IEEE/RSJ Int Conf on Intelligent Robots and systems,2000, pp 578,583

6 **L. Brignone, M. Howarth, K. Sivayoganathan, V. Balendran**, "Using contact information to enable decision making and to control peg in hole insertions", accepted for publication.

7 **R. Sutton, A. Barto**, "Reinforcement learning. An introduction", The MIT Press, ISBN 0-262-19398-1, 1998.

8 **J. Ferrman, D. Skapura**, "Neural networks. Algorithms, applications and programming techniques", Computation and neural systems series, ISBN 0-201-51376-5, 1991.

9 **S. N. Simunovic,** "Force Information In Assembly Processes", Proceedings of 5th Int. Symp. Industrial robots 1975; pp 415,431.

Learning contact states and basic manipulative skills - an AI approach based on sensorial and descriptive information sources

P. Chmielarczyk[1], M. Howarth[2]

[1] School of Engineering, Sheffield Hallam University, Sheffield S1 1WB, UK,
email: P.Chmielarczyk@shu.ac.uk.

[2] School of Engineering, Sheffield Hallam University, Sheffield S1 1WB, UK,
email: M.Howarth@shu.ac.uk.

ABSTRACT

The aim of this research is to develop a non-linear controller, which is able to perform an intelligent robotic assembly of mechanical components. The design is based on the implementation of Artificial Neural Network (ANN) and Reinforcement Learning (RL) algorithms. The main aim of this study is the investigation of the different information sources and their application to the "peg-in-hole" insertion problem. Amongst these, one source is sensorial (signal from force and torque sensor) and the others are descriptive (geometry of parts and coded task description). Force and torque data is obtained from a six-axis sensor. The presented results show that controller is able to learn the state-action relationship quickly, without supervision.

1 INTRODUCTION

Manufacturing could be described as a set of operations and activities performed in order to make a final product. It involves product design, planning, production, assembly, inspection and marketing. The assembly is one of the most complicated processes in manufacturing [12]. It often requires accurate motion control often guided by a vision system.

There is widespread acceptance that the adoption of automated assembly operations could provide a solution to the high costs related to the employment of human labour and supervision in the execution of such processes. The very first applications of robots in automated assembly techniques have relied on passive accommodation methods [10], simple sensing systems and the manipulator's programming language. These approaches show many restrictions that make them unable to deal with complex geometry, and when working with components in an unknown environment. Early research work by Simunovic [9] concluded that most common assembly operations could be modelled as simple peg-in-hole insertions.
The applications of AI to the peg-in-hole problem range from the employment of various ANN algorithms to the application of Fuzzy Inference Systems. Asada [1] pioneered the application

of backpropagation to assembly tasks, using the network's learning capabilities to approximate compliant control laws. Backpropagation based controllers were also developed later by Gullapalli [3] and Howarth [4], these two approaches feature substantial differences in the architecture of the reinforcement learning rule. Cervera [2] defined a finite set of contact configurations and used Self Organising Maps to associate the current contact state to a predefined one. Lopez [5] applied a FuzzyARTMAP network to learn the 'pattern motion' association from a table of pre-taught templates. Brignone in his studies [6] used a similar algorithm but extended the problem by merging F/T data and parts geometry. In his work, the network was trained with a set of normalised vectors representing the normal to the surfaces, which form the geometry of the hole.

Most of these early algorithm designs share a number of common disadvantages. The slowness and poor online performance of Reinforcement Learning algorithms and supervised learning methods of contact states are the major limitations. In this paper the novel AI controller architecture will be presented. Early results show its ability to learn the state-action mapping quickly and without the supervision. The F/T signal and coded task description are sufficient enough to perform a successful peg-in-hole assembly.

2 METHODOLOGY

During this research, authors will extend the earlier work by investigating how the AI controller should combine different information sources in order to generate a successful assembly strategy. The information sources mainly concern: task description; geometry; and online force and torque contact data analysis. The originality of this approach lies in the definition of a methodology to merge information derived from different sources to attempt an interpretation of the contact state, in terms of contact location and peg's attitude. The work carried out on data processing and interpretation is used to provide the tools for the definition of the controller's decision-making strategy.

The robot arm and controller are the main part of the system. The computer is connected to the controller via serial port. The JR3 six axis force and torque sensor was mounted on the robot wrist. The closed loop control system is formed by the signal from the sensor transmitted to the computer and then used as an input to the controller. The new arm position and orientation values are then calculated and the incremental motion request is sent to the controller using the robot controller's terminal emulation mode.

The controller's structure is shown in Figure 1. It consists two main parts: an ART2 neural network and a reinforcement learning algorithm. In order to speed up the learning process a large and continuous state space (*s*) needs to be clustered and generalised. One of the simplest methods is to partition the continuous state domain using ANN, and then treat each output class as an autonomous object.

2.1 State classification

To cluster the continuous state space using the force and torque signal, and geometry information the *adaptive resonance theory* (ART) network was used [11]. In this kind of ANN, information is repeatedly transferred back and forth between the layers. If the correct pattern is developed during this process, a stable oscillation occurs. This is the neural network equivalent of resonance. Only at this stage does learning occur. This type of ANN is able to respond quickly to previously learned data and remains able to learn when new data is presented.

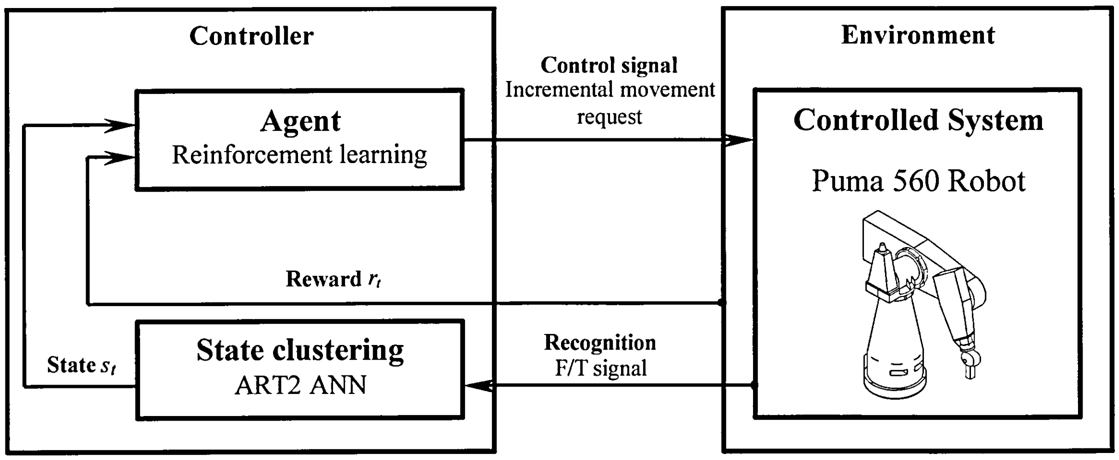


Figure 1 The controller's architecture.

Because of the analogue nature of the F/T signal, the ART2 algorithm was chosen. It differs from ART1 structure in the type of the input patterns. The price for that additional capability is an increase in complexity at the processing level, which contains a number of sub-layers that serve to remove noise, enhance contrast and to normalise the analogue input pattern [8].

In this configuration, the reinforcement algorithm needs to learn the best actions in relation to a finite number of states. This geometry and task decomposition is expected to speed up the learning process significantly.

2.2 State-action mapping learning

The basic one step *Q-learning* reinforcement learning algorithm was used to learn the state-action relationship. The method is based on the *Q*-function, defined as followed [7]:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a_{t+t}) - Q(s_t,a_t)\right] \qquad (1)$$

where: t is a discrete time step, s is a finite set of states, an agent can be in; a is a finite set of actions, an agent may perform; Q is a transition function, mapping each state-action pair to a successor state; r is a reward function, mapping states-action pairs to payoffs, α is a learning rate and γ is a discount factor (these last two ratios are learning parameters).

Due to its nature, the *Q*-function approximates the optimal action-value function despite of applied exploration policy. Theoretically, the optimal function can be learnt under the assumption that the controller will experience every state an infinite number of times.

The information flow within the controller looks as follows. First, the F/T signal was acquired and classified by ANN. The number indicating a location was used as the state description and set as an input to the reinforcement learning algorithm. The appropriate action was calculated and decoded. The incremental movement was sent to the robot, which caused the change in state, and then a new F/T vector was read which formed another input to the ANN.

3 RESULTS

The ANN and reinforcement algorithms were implemented using ANSI C language running on a Pentium PC under the GNU/Linux operating system. The controller's performance was validated during a number of experiments using a Unimation Puma 560 robotic arm. A circular peg and 45° chamfered hole was used during the experiments. The clearance between parts was 0.2mm. The robot was programmed to perform 2,000 successive iterations, each from a random start position. The insertion was considered successful and complete when 2/3 of the peg's height was inside the hole. The reward function was implemented as follows. If after the movement reacting forces were reduced by 50%, the controller was granted a reward of +5. When the contact force increased (by 20%) the controller was rewarded with a value of –5. To minimise the overall number of steps during the insertion, for every other movement, the value of –1 was applied. The ART2 vigilance parameter (ρ) was set to 0.95 and two different ε-greedy action-exploration policies were applied (see Figure 2). Nine different actions namely: 5 translations and 4 rotations were implemented.

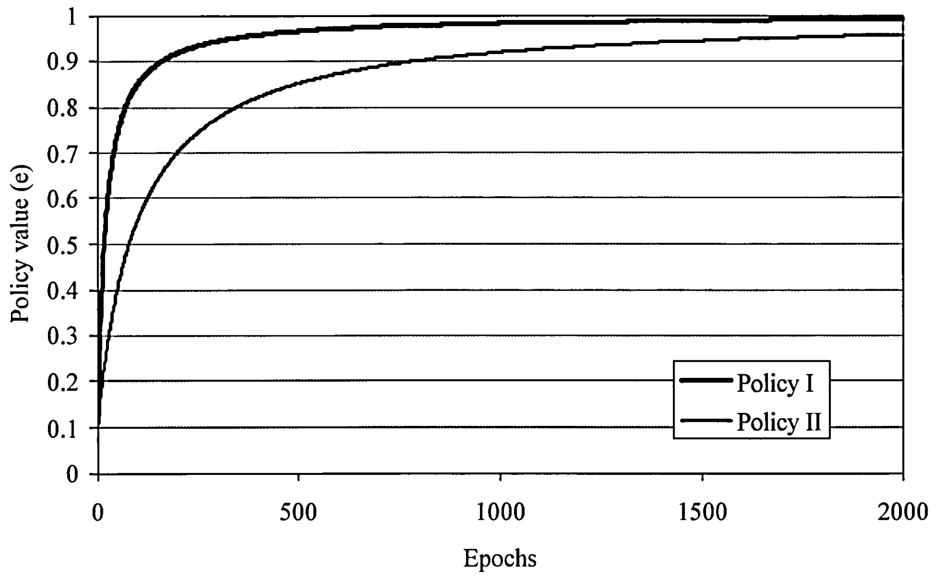


Figure 2 Applied action-exploration policies

The results from the experiments are presented in Figure 3, and Figure 4. In the case of a collision the value of –200 was added to the cumulative rewards. The presented data was filtered using a moving average window of ten values. Very low rewards during the first few iterations indicate that the controller was not able to successfully accomplish the peg-in-hole assembly. The transition time could also be observed. During this time, the controller was learning (with no supervision), the geometry features and state-action relationship. The controller with "Policy I" applied showed a better online performance. The increase in noise (see Figure 4) is caused mainly, by a higher number of random movements (more exploration) imposed by "Policy II".

The significant drop in knowledge after 1,600 iterations can also be seen on the graph in Figure 3. There are several reasons for this phenomenon, but at the time of writing, a conclusive reason is still being investigated. The knowledge drop may be caused by the ANN reorganisation its internal structure during the experiment. Work is being carried out to closely investigate the problem and to find an appropriate solution.

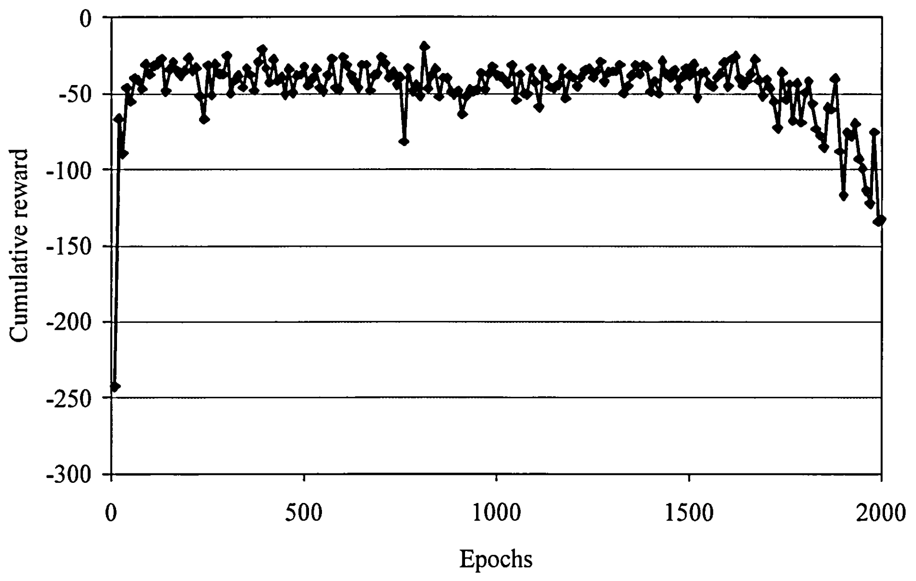


Figure 3 Learning results from the experiment with "Policy I" applied.

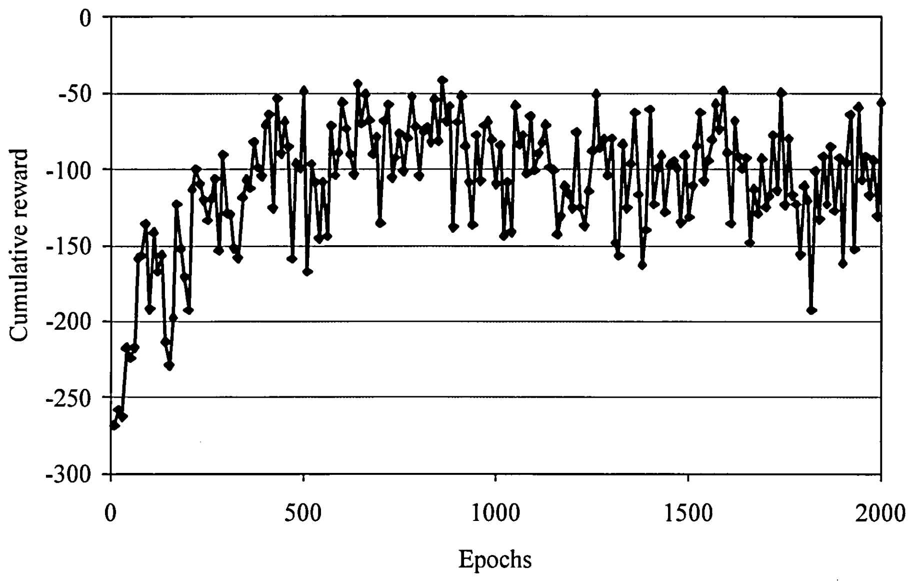


Figure 4 Learning results from the experiment with "Policy II" applied.

4 CONCLUSIONS

During initial experiments, the controller was able to accurately cluster and classify the state domain. The ART network was responding quickly and accurately to the environment and successfully selecting the closest match using F/T signals. The controller has showed the ability to quickly learn the relationship between sensor data and robot action, and hence to interact with the environment. The two information sources, F/T signal and coded task description (in form of the assembly direction), are sufficient to perform the peg-in-hole assembly. The controller was able to develop the insertion policy without any supervision. The geometry classification and state-action learning were done automatically. In comparison with previous researchers techniques, this approach does not require class labels; peg position and orientation; nor any form of off-line learning.

Future work involves experiments using different exploration policies (current work is assessing the performance of polices based on Boltzmann's distribution) and to validate the controller's performance by applying it to different geometry including the non-linear chamfer-less problem. The work on stability of the system is currently being carried out.

REFERENCES

1 **H. Asada**, "Teaching and Learning of Compliance using Neural Nets", Proceedings of the 1990 IEEE Int. Conference on Robotics and Automation, 1990, pp 1237-1243.

2 **E. Cervera, A. del Pobil and M. Serna,** "A sensor-Based Approach to Motion in Contact Task Planning", Proc 1995 IEEE/RSJ Int. Conference on Intelligent Robots and Systems, 1995, 2: pp 468,473.

3 **V. Gullapalli**, "Skilful control under uncertainty via direct reinforcement learning", Robotics and autonomous systems, 1995, pp 237-246.

4 **M. Howarth,** "An Investigation of Task Level Programming For Robotic Assembly". PhD thesis. The Nottingham Trent University, January 1998.

5 **I. Lopez, M. Howarth**, "Learning manipulative skills with ART", Proceedings of the IEEE/RSJ Int. Conf on Intelligent Robots and systems, 2000, pp 578-583

6 **L. Brignone, M. Howarth, K. Sivayoganathan, V. Balendran**, "Using contact information to enable decision making and to control peg in hole insertions", accepted for publication.

7 **R. Sutton, A. Barto**, "Reinforcement learning. An introduction", The MIT Press, ISBN 0-262-19398-1, 1998.

8 **J. Ferrman, D. Skapura**, "Neural networks. Algorithms, applications and programming techniques", Computation and neural systems series, ISBN 0-201-51376-5, 1991.

9 **S. N. Simunovic,** "Force Information In Assembly Processes", Proceedings of 5th Int. Symp. Industrial robots 1975; pp 415-431.

10 **D. E. Whitney,** Quasi-static assembly of compliantly supported rigid parts. Trans. of ASME, Jnl. of Dynamic Systems, Measurement and Control, 104, March 1982, pp. 65-77.

11 **G.A. Carpenter, S. Grossberg,** A massively parallel architecture for self-organizing neural pattern recognition machine. Computer vision, Graphics and Image processing. Vol. 37 1987, pp.54-115.

12 **J Tidd,** Divergent trends in robotic assembly in the UK and Japan, Assembly Automation 8(4), pp 211-212, November 1988.

Application of reinforcement learning methods onto peg-in-hole insertion task.

Pawel Chmielarczyk
Sheffield Hallam University
Sheffield, UK
p.chmielarczyk@shu.ac.uk

Martin Howarth
Sheffield Hallam University
Sheffield, UK
m.howarth@shu.ac.uk

1 Introduction

Assembly operations are amongst the most common and complicated in manufacturing process. Working in environments under high level of uncertainty they usually require an accurate motion control guided by a vision system.

The early applications of robots in automated assembly techniques have relied on passive accommodation methods, simple sensing systems and the manipulator's programming language. These approaches show many restrictions that make them unable to deal with a complex geometry of the components. In a typical force guided assembly task, the robot's trajectory corrections are calculated and applied, according to predefined control laws. The more sophisticated force control methods involve modification of trajectory based on continuous force feedback from the system and a task description.

Recently a new way of programming has been proposed. The intelligent control embraces substantially different techniques that include application of knowledge based Expert Systems, Artificial Neural Networks, Fuzzy Logic or Reinforcement Learning algorithms. All these algorithms are different in principles and based on different theories but their application share similar methodology: an attempt to create an effective mapping from sensory state space to the action domain.

Asada [1] laid the foundation for the research on the intelligent assembly process. His 2-dimensional controller, although simple, was a major break through in research on intelligent peg-in-hole insertion tasks. The main disadvantages of proposed design were its slowness and supervised method of learning. This involves the need for *a priori* knowledge about the environment.

Cervera [5] and later Brignone [3] used a geometrical approach to locate the contact states. Both methods proved to be fast and reliable but a level of supervision was needed to accomplish the task.

Gullapalli [9] applied his stochastic reinforcement method on a real life robot. For successful learning the peg's position and orientation description had to be supplied to the system.

Most current intelligent controller designs share a number of common disadvantages. The poor on-line performance or need for an explicit description of the environment are amongst the major drawbacks in automated assembly. Performing complex tasks (i.e. peg-in-hole insertion) the intelligent controller often deals with infinitive, 3-dimensional Cartesian space. Therefore, for efficient on-line state-action relationship, learning the state domain should be simplified.

The aim of this research was to design a non-linear controller based on an Artificial Neural Network and Reinforcement Learning algorithms implementation, which is able to perform an intelligent assembly of mechanical components.

During this research, the authors will extend the earlier work by investigating how the AI controller should combine different information sources in order to generate a successful assembly strategy. The information sources mainly concern: task description, geometry, on-line force and torque contact data analysis and information derived from previous moves. The originality of this approach lies in the definition of a methodology to merge information deriving from different sources to attempt an interpretation of the contact state, in terms of contact location and the peg's attitude.

2 Methodology

Early research work by Simunovic [12] concluded that most common assembly operations could be modeled as simple peg-in-hole insertions. This represents a large number of part mating operations carried out in industry. The peg-in-hole model, its dynamics, and acting forces have been widely studied by researchers [1, 6, 9, 10, 11, 2].

The unsupervised, self-organising algorithms have revolutionised the modern intelligent assembly methods and are state-of-the-art in robot force control. There is widespread acceptance that the adoption of automated assembly operations could provide a solution to the high costs related to the employment of human labour and supervision in the execution of such processes.

The use of adaptive and learning capabilities in the assembly process simplifies the implementation and improves the reliability of these systems. The main advantage of AI approach is the ability to solve problems without a detailed or explicit algorithm available for the solution. This has a tremendous impact on dealing with complex parts geometry or noisy sensory signals during the assembly process as well as the development of automatic error recovery methods.

2.1 The controller

The sandwich structure of the intelligent agent was proposed (see Figure 1). It features two major layers: *State Clustering* module, where the detection and localisation of the contact points were performed, and the *Learning Agent* subsystem, where the decision about the next action took place. The controller was designed to receive two signals from the environment. The F/T signal was acquired from the sensor and fed straight to the *State Clustering* module. This produces a number corresponding to the location of the contact point, called now a *state description*. This information together with the reward from the environment was sent as an input to the *Learning Agent* subsystem. In this module, the next action-selection takes place. It results in an action number which is later decoded to the incremental motion command understandable to the robot. The robot executes the move which causes the change in state. Then the new F/T vector is read and fed again to *State Clustering* module. This closed loop process is repeated until the stop condition occurs.

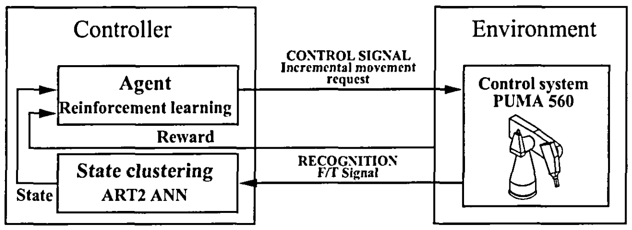


Figure 1: Controller architecture

The system was implemented in modular form. This approach allows the user to create different AI controllers and test them without any interference to the main application code. The specification of the application interface allows researchers to focus on controller development and implementation without coding the I/O operations needed to establish connection with robot.

2.2 The *Learning Agent*

This is the main, decision-making part of the controller. Here the next action-selection takes place. Many different algorithms were applied in the past to learn manipulative skills during peg-in-hole insertion. The authors decided to take a reinforcement learning approach to the problem. The complete insertion can be considered as an *episode* (or play) and every single movement as a *step*. The controller will learn the state-action relationship by continuously repeating the insertions. This will result in developing the optimal insertion strategy.

Two different reinforcement learning algorithms were implemented and individually tested namely; SARSA and q-learning. They both belong to the Temporal-Difference family but differ in the learning policy. The three different action-selection methods were also implemented and tested. All algorithms were evaluated with the simulated grid-world environment before being applied on a real life system.

Reinforcement learning tasks are generally treated in discrete time steps. At each time step (t), the learning system receives some representation of the environment's state (s_t), it takes an action (a_{t+1}), and one step later it receives a scalar reward (r_{t+1}), and finds itself in a new state (s_{t+1}).

The significant feature that distinguishes the reinforcement learning approach from the other artificial intelligence algorithms is its ability to learn from experience. The idea is to capture the most significant aspects of the environment to solve the problem as opposed to supervised learning algorithms, which are mainly focused on finding the methods of acting [13].

2.2.1 Q-learning

In Q-learning method, the learnt action-value function Q directly approximates the optimal action-value function Q^*, independently of the policy being followed (off-policy). The policy still has an effect in that it determines which state-action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated.

The Q-learning algorithm was implemented after Sutton and Barto [13]. The class number was set as an input describing the actual contact forces. Using this configuration, the reinforcement algorithm needs to learn how to deal with a particular part of the component's geometry. Moreover the self-learnt geometry information includes only orientation of the given plane. Based on that the agent can be trained with a set of surfaces separately and then be able to work efficiently with an object built from these planes. So, unless the part consists of a feature which has not been learnt before, the change of geometry (i.e. from circular to square peg) should not affect the controller's performance in the early stages of insertion.

2.2.2 SARSA

The Q-learning method learns the optimal policy. This could be a disadvantage in some applications especially when particularly low rewards are assigned to some movements. In those cases, despite of a pre-learnt optimal policy, due to exploratory random action choice the agent can experience an error condition (sudden increase of forces in the case of peg-and-hole insertion). The SARSA method, due to the fact that it learns a safe path, is not affected by this problem. It is still possible for SARSA (by applying an appropriate action-selection method) to converge to optimal policy.

The SARSA algorithm also belongs to the Temporal-Difference methods class. Similarly to Q-learning, SARSA gathers the knowledge from the state-action pairs transitions but it learns the policy directly. The on-line performance of the algorithm depends on the application but generally is better than in the case of Q-learning. The learning of optimal policy relies on applied action-selection method. Each state-action pairs must be visited an infinitive number of times to gain the best possible solution. This could be achieved by applying an ϵ-greedy action-selection method which converges in limit to greedy policy.

2.3 *State Clustering* module

To learn an appropriate action the reinforcement learning agent requires a clear definition of the state. This information can be constructed from almost any data acquired from the environment. In most cases the raw signals have to be preprocessed or classified and in some applications the state can be built from a combination of two or more features. According to reinforcement learning theory, the preprocessing system is nominally a part of the environment [13]. It is clear from Figure 1 that in the proposed design the preprocessing module is placed as a part of the controller. The reason for it is the fact that applied methods of classification are fully self-adapting and together with the reinforcement learning agent they form an intelligent controller. Although, it should be emphasised at this point that the preprocessing system is not a part of a reinforcement learning agent. Both subsystems work independently of each other and serve different purposes. The decision about the next action is taken by the agent without knowing how the state signal was constructed.

The choice of the best action for a particular state is not an easy task. The decision has to be made based on immediate rewards but the overall performance also needs to be taken into account. Making the choice by monitoring just the immediate sensations will significantly affect the agent's learning speed since, very often, the actions with poor immediate rewards can produce good results on the global (task) scale.

The states that feature the Markovian property play a significant role in the reinforcement learning process. The fact that past sensations are retained in the current state makes the decision-making process history independent. In the other words all that matters for the future is included so the agent can predict the next state (s_{t+1}) and reward (r_{t+1}) based on current state (s_t) and the action (a_t). This very important feature is essential for a fast, reliable decision-making process. The next action choice which is based on the state signal with Markovian property is as good as the decision based on the entire history of the task [13].

The *State Classification* module was designed to generalise the large and continuous state space. In the authors' application the only sensory signal from the environment is in the form of a force and torque signal. Because of that, the unsupervised pattern recognition or, in the other words, function approximation system needs to be implemented.

2.3.1 Action-selection methods

Reinforcement learning estimates the action taken rather than instructing the system. This feature has been widely used for function optimisation, and is a basis of evolutionary algorithms [13]. The agent evaluates how good an action taken is, but not if it is the correct choice. The instructive approach, on the other hand, is an action independent approach and indicates the best known solution.

The action dependant, evaluative feedback approach requires the agent to decide how far to explore the environment or to exploit the current knowledge. This decision making process is called *exploration-exploitation dilemma* and can significantly influence the on-line performance of the reinforcement learning algorithm [13]. The supervised methods implement the exploration-exploitation trade-off explicitly, relying heavily on the expert knowledge. Three different action-selection methods were implemented and tested by the authors. They differ in principles and applications:

- Greedy action-selection. This method relies entirely on knowledge exploitation. The agent always chooses the action with the highest estimated value. This means that no exploration of the environment takes place. It performs poorly with the complex environment and rarely produces an optimal result. Its limitation lies in the fact that it ignores other actions which may produce better results in the long run. However, this method performs very well in non-complex environments where the system knows each action after trying it once (reward variance equal to zero). In the case of greedy action-selection the probability of greedy action (a_g) is equal to one for every episode ($P(a_g) = 1$).

- ϵ-greedy action-selection. It is a very similar approach to the greedy method. The agent still chooses the action with the highest estimated value but also allows the exploration with a small probability ϵ. The method was implemented in the way so the value of ϵ depends on the number of the current episode. This reduces the amount of exploration as the number of plays increases. The probability of greedy action a_g was set as follows:

$$P(a_g) = 1 - \epsilon + \frac{\epsilon}{|a|} \tag{1}$$

where ϵ depends on the actual episode number.

The main advantage of this policy over the greedy action-selection method is the fact that with the increase of episodes, every possible action can be tried and evaluated an infinite number of times [13]. The ϵ-greedy action-selection method guarantees that during exploration time every action has an uniform chance to be chosen. This is independent of its estimated value so even the actions with the lowest values can be applied. This feature is highly undesirable in the environments where the "bad" actions could have serious implications on system safety and performance (i.e work-piece damage, robot collision).

- Softmax action-selection. In this method the probability of each action is a function of its estimated value. The uniform choice in the form of random function present in ϵ-greedy method has been replaced with a system of weighted actions. In Softmax selection policy the probability of choosing the action is usually governed by Gibbs distribution, described as follows:

$$P(a_i) = \frac{e^{\beta * a_i}}{\sum_j e^{\beta * a_j}} \qquad (2)$$

where β is a ratio which determines how much exploration the agent is allowed to take. When $\beta = 0$ the actions are selected randomly and for $\beta \to \infty$ the greedy policy is applied. In between the above limits the actions with the highest estimated value still have the highest probability of being chosen. Although the actions showing the lowest estimated values have a relatively small chance of being applied. This approach favour the "good" actions over the "bad" ones.

2.3.2 The ART2 ANN

Adaptive Resonance Theory (ART) was developed by Stephen Grossberg and Gail Carpenter over the period of 1976-86, during their studies of the behaviour of models of systems of neurons [8, 4]. ART was developed to solve the learning instability problem suffered by standard feed-forward networks. The weights which have captured some knowledge in the past continue to change as new knowledge comes in. There is, therefore, a danger of losing the old knowledge with time.

Adaptive Resonance Theory gets its name from the particular way in which learning and recall interplay in the network. In this type of network, information in the form of processing element outputs reverberate back and forth between layers. If the proper patterns develop, a stable oscillation occurs. This is the neural-network equivalent of resonance. Only during this stage can learning take place [7].

A *resonance* can be attained in one of two ways. If the network has learnt to recognise the input pattern before then the resonant state will be achieved quickly when the input vector is presented. During this state the memory of the stored pattern will be reinforced. If the input vector is not recognised the network will look in the stored patterns database for a match. If no match is found the network will enter the resonant state and the new pattern will be stored for the first time. Thanks to this design the network is able to respond quickly to previously learnt data, yet remains able to learn new patterns.

The author's implementation is based on the algorithm described by Ferrman and Skapura [7]. After the initial tests the problems with data stabilisation on the F_1 layer occurred and some modifications had to be made.

The contact between peg and hole can be modelled with the Newton equations for the rigid body. Furthermore, representing the insertion as a quasi static transition of states simplifies the equations greatly, removing them from the time domain. The raw F/T signal was set as an input to the network. The signal preprocessing was not necessary since normalisation and contrast enhancement is performed by the ART2 network. After successful classification of the state description in the form of class number was sent as an input to the *Learning Agent*.

Choosing to partition the state space, it was divided into a set of classes using information extracted on-line from the part geometry. The authors' approach significantly differs from the other researchers where the geometrical model was supplied and extracted from the CAD drawing. In this research the geometry is learnt on-line in an unsupervised manner using only force and torque signals.

3 Experimental setup

The system was implemented using Puma 560 and later Staubli RX90 robots. The robotic arm and controller are the main parts of the system. The *supervisory* computer is connected to the controller via a serial port. The sensor was mounted on the robot's wrist (see Figure 2). The F/T signal is transmitted to the computer and then used as an input to the AI controller. The new arm position and orientation values are calculated. The incremental motion request is sent to the controller using *terminal* communication mode.

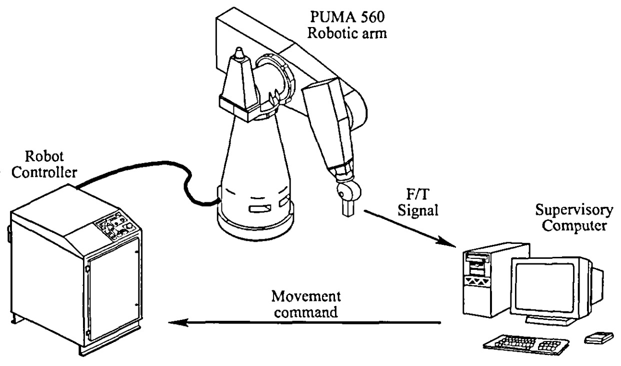


Figure 2: Experimental setup

The proposed design guarantees a stable communication and fast, bi-directional information flow between the devices. These characteristics are very important since the experimental setup is expected to work continuously for several days undertaking numerous peg-in-hole insertions.

The feedback signal to the *Learning Agent* was provided by Force and Torque (F/T) sensor mounted on the robot's wrist. The JR3 sensor with internal electronic and receiver card for ISA (IBM-AT) bus was chosen since it provides force and torque data with very low noise. The sensor was pre-calibrated with maximum loads 15 lbs on X and Y axes and 30 lbs on Z axis.

The influence of action-selection methods onto AI agent performance was analysed. The proposed controller was applied to a set of real life peg-and-hole experiments. The circular peg geometries were used, and insertions into chamfered and non-chamfered holes were performed. Materials with different friction factors (rubber and aluminium pegs) were used for mating parts.

4 Results

The proposed controller was applied on a set of tests to evaluate its on-line performance working with complex, 3-dimensional environments. Every experiment was started with no external knowledge supplied to the system. Only the guidance towards the hole's bottom was implemented.

Only the results from the experiments using Q-learning algorithm with Softmax action-selection policy are presented. The Q-learning and SARSA algorithms performed very similarly. They both converged to the same value of cumulative rewards (-15). The transition time when SARSA algorithm was used was extended to 200 episodes while it took 40 episodes for the Q-learning agent to stabilise the learning curve.

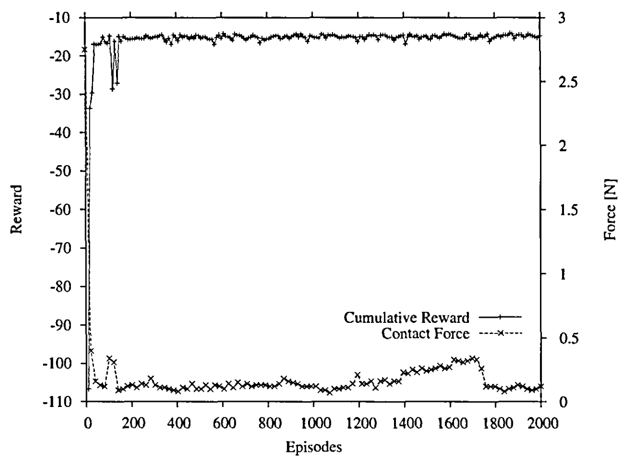


Figure 3: Q-learning agent applied on aluminium peg.

Occasional collisions or jammed conditions were registered during the steady state of the knowledge acquisition (see Figure 3). They were caused by the agent executing the random actions following the action-selection policy. However, the interrupted insertions did not affect the prelearnt knowledge and did not influence the controller's online performance.

The application of the rubber type peg did not affect the controller's ability to learn (see Figure 4). The usage of less rigid material resulted in lower, more stable force reading during the steady state. It also significantly reduced the number of collisions.

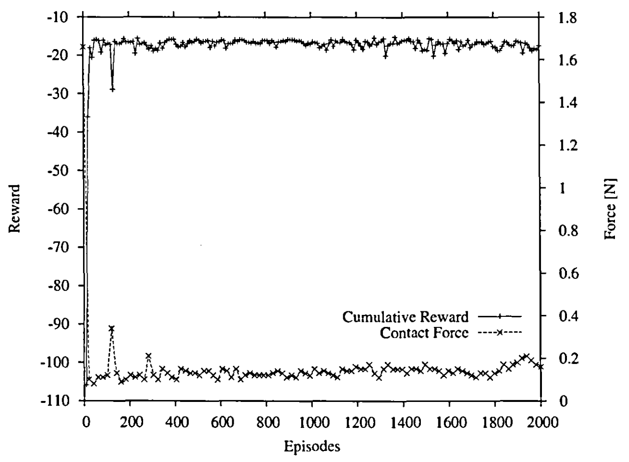


Figure 4: Q-learning agent applied on rubber peg.

To investigate the controller's ability to deal with the tasks involving the geometry where the force-velocity domains are not linearly mappable (non-linear problems) the chamfer-less cylindrical hole was used.

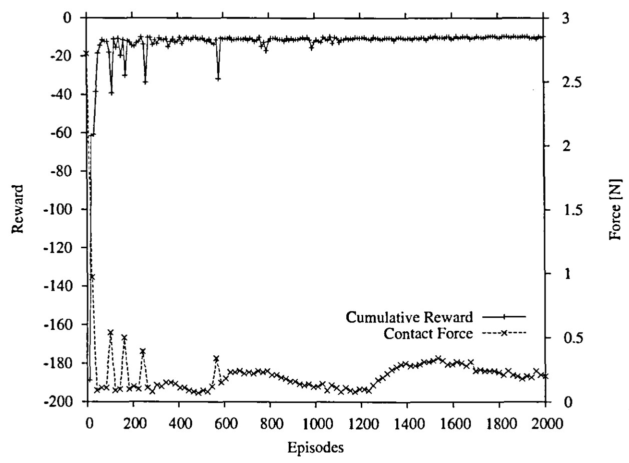


Figure 5: Q-learning agent applied on non-chamfered hole.

The transition time was much longer than in the case of insertions into chamfered hole (see Figure 5). This indicates that the controller was particularly struggling with the first part of the hole's geometry. After the strategy for the entrance section was developed the the rest of insertion task was quickly learnt.

5 Conclusions

Fast and stable knowledge acquisition was clearly present in all the cases investigated. A significant reduction in contact forces value during the initial stage of the learning process was recorded. The force was usually reduced to one tenth of the initial value. Some fluctuations were recorded but when the cylindrical peg was considered the value of contact forces never exceeded 0.5 N during the steady state.

The fully unsupervised controller with the ability to deal with complex, 3-dimensional geometry was proposed, implemented and tested. The knowledge acquisition was clearly present in all investigated cases. Also, the contact forces and torque values were significantly reduced while the insertion progressed.

Different information sources were analysed and applied to the system. The sensory signals in form of force and torque readings and descriptive in the form of assembly direction were employed to peg-in-hole insertion. It was decided and empirically proved that any form of pre-training or task description is not necessary to successfully learn the assembly. The history was also included and was embodied as a part of the reinforcement learning algorithms.

The automatic state recognition and clustering was investigated. The contact states were analysed and classified using the implemented module for geometry classification.

The proposed controller worked with no supervision and gained all the knowledge automatically from experience. The system was implemented and tested using a Puma 560 robotic arm and Staubli RX90.

The agent performed exceptionally well with the cylindrical peg-and-hole geometry. After short initial stage it managed to established the strategy to successfully accomplish the insertion task. Some difficulties were experienced when chamfered hole was investigated but the learning skills and abilities to cope with complex 3-dimensional geometry are clearly present.

References

[1] H. Asada. Teaching and learning of compliance using neural nets. In *Proceedings of 1990 IEEE International Conference on Robotics and Automation*, volume 2, pages 1237–1244, 1990.

[2] L. Brignone, M. Howarth, K. Sivayoganathan, and V. Balendran. Simulation of contact states during peg in hole insertions, 2002.

[3] L. Brigone. *A geometrically validated approach to intelligent robotic assembly*. PhD thesis, The Nottingham Trent University, April 2002.

[4] G. Carpenter, A. and S. Grossberg. Atrmap: Supervised, real-time learning and classification of nonstationary data by self-organising neural network. *Neural Networks*, pages 565–588, 1991.

[5] E. Cervera and A. del Pobil. Sensor-based learning for practical planning of the fine motion in robotics. *Information Sciences*, 145:147–168, 2002.

[6] E. Cervera, A. del Pobil, E. Marta, and M. Serna. A sensor-based approach to motion in contact in task planning. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 468–473, 1995.

[7] J. Ferrman and D. Skapura. *Neural networks. Algorithms, applications and programming techniques*. Computation and neural systems series, 1991. ISBN: 0–201–51376–5.

[8] S. Grossberg. Self-organizing neural networks for stable control of autonomous behaviour in a changing world. *Elsevier Science Publishers*, 1993.

[9] V. Gullapalli. Skillful control under uncertainty via direct reinforcement learning. *Robotics and autonomous systems*, pages 237–246, 1995.

[10] M. Howarth. *An Investigation Of Task Level Programming For Robotic Assembly*. PhD thesis, The Nottingham Trent University, January 1998.

[11] I. Lopez-Juarez. *On-line Learning for Robotic Assembly Using Artificial Neural Networks and Contact Force Sensing*. PhD thesis, The Nottingham Trent University, April 2000.

[12] N. Simunovic, S. Force information in assembly processes. In *Proceedings of 5th International Symposium Industrial robots*, pages 415–431, 1975.

[13] R. Sutton and A. Barto. *Reinforcement learning. An introduction*. The MIT Press, 1998. ISBN: 0-262-19398-1.