Ben-Gurion University of the Negev
Faculty of Engineering Sciences
Department of Industrial Engineering and Management

A Collaborative Hierarchical Reinforcement Learning Framework

Thesis Submitted In Partial Fulfillment
of the Requirements for the M.Sc. Degree

by

Amit Gil

December 2008

Beer - Sheva

Ben-Gurion University of the Negev
Faculty of Engineering Sciences
Department of Industrial Engineering and Management

A Collaborative Hierarchical Reinforcement Learning Framework

Thesis Submitted In Partial Fulfillment
of the Requirements for the M.Sc. Degree

By: Amit Gil
Supervised by: Helman Stern
Yael Edan

Author	____________	Date	____________
Supervisor	____________	Date	____________
Supervisor	____________	Date	____________
Chairman of Graduate Studies Committee	____________	Date	____________

December 2008

Beer - Sheva

This work was carried out under the supervision of

Prof. Helman Stern

Prof. Yael Edan

In the

Department of Industrial Engineering and Management

Faculty of Engineering Sciences

Ben-Gurion University of the Negev

Acknowledgments

I would like to thank my supervisors **Prof. Helman Stern** and **Prof. Yael Edan** for their guidance and support throughout the last two years. Their ideas and insights provided me with direction and motivation, and their patience and valuable advice encouraged and promoted my research in many ways. For that I owe them my deepest gratitude.

Very special thanks go to **Dr. Uri Kartoun**, my professional advisor, but first of all a good friend. Uri is the one who introduced me to the challenging and fascinating world of machine learning, and helped me take my first steps in the area. His support, advices and amazing sense of humor made it very easy and enjoyable for me to enter the academic life and state of mind.

I would like to gratefully acknowledge the technical and professional support of **Nissim Abuhatzira**, **Yossi Zehavi, Noam Peles, Paul Erez** and **Rubi Gertner** from the *Department of Industrial Engineering and Management*.

I would also like to express my appreciation to **Dr. Sigal Berman** who helped me in the exhausting process of choosing the subject for this research.

Finally, I recognize that this research would not have been possible without the financial assistance of the *Paul Ivanier Center for Robotics Research and Production Management*, *Ben-Gurion University of the Negev* and by the *Rabbi W. Gunther Plaut Chair in Manufacturing Engineering*.

Amit Gil
Ben-Gurion University of the Negev
Beer Sheva, 2008

Abstract

To introduce robotic applications into real-world environments, robots must be constructed for a large variety of tasks and be able to adapt continuously to new and changing working conditions. Since it is impossible to model all environments and task conditions, the adaptation to new tasks cannot be achieved by regular end-user programming. Rather, the robot must be delivered with advanced capabilities to autonomously learn new tasks and new working conditions.

A common learning approach in robotics is reinforcement learning (RL). In RL the robot (agent) acts autonomously in a process guided by reinforcements from the environment, indicating how well it is performing the required task. RL is an attractive alternative for programming autonomous systems, as it allows the agent to learn behaviors on the basis of sparse, delayed reward signals. Nevertheless, RL has several drawbacks preventing it from answering the challenges presented by real-world applications, such as the necessity for extensive interaction between the robot and the environment, or the fact that it allows only one goal state for the learning task.

This thesis provides one more step in the continuing struggle to overcome these drawbacks. The framework proposed in this research, Collaborative Hierarchical Reinforcement Learning (CHRL), combines two known techniques used for addressing the drawbacks, hierarchical RL and Human-Robot collaboration, in order to scale up RL and alleviate some of its disadvantages. This combination enables both the execution of complex tasks and the improvement of the learning process. Hierarchical RL reduces the search space and allows efficient learning. Human aid can improve or expand already learned behaviors and enable the robot to handle unknown and unpredictable events that are beyond the competence of current autonomous robotic systems.

In the proposed CHRL framework the learning task is decomposed into a two-level learning hierarchy. The first level consists of learning the desired sequence of execution of a set of basic sub-tasks. The second level consists of learning how to perform each of the sub-tasks required. Human intervention is allowed at both levels, to expedite the learning process by exploiting human intelligence and expertise. The applicability of the framework is proven using an automated toast making system presenting both high and low level learning tasks.

Two RL-based algorithms were developed to support the CHRL framework: A sequencing algorithm was developed for providing a sub-task execution sequence, as part of the first level of the hierarchy. The algorithm addresses the learning task of scheduling a single transfer agent (a robot arm) through a set of sub-tasks in a sequence that will achieve optimal task execution times. In lieu of fixed inter-process job transfers, the robot allows the flexibility of job movements at any point in time. Execution of complex tasks was demonstrated using two applications – the automated toast making system and a flexible manufacturing system. The algorithm presents good results, matching and outperforming the compared methods, Monte-Carlo and random search.

A collaborative algorithm was developed to allow the introduction of an advisor. This approach is referred to as cognitive collaborative reinforcement learning (CCRL). In the CCRL algorithm an autonomous learner (RL in this case) is enabled with a self awareness cognitive skill to decide when to solicit instructions from the advisor. Furthermore, the learner is able to assess the value of any advice given and to decide whether to accept or reject it. This approach of intelligent adjustable autonomy was demonstrated and evaluated using the toast making system and a simulated three-dimensional path planning task. Tests were conducted for advisors with various skill levels from expert to novice. The algorithm expedites and improves the learning process by taking advantage of the advisor's knowledge and expertise, and learning to use advice given by an expert while discarding advice suggested by a novice.

The main contribution of this research is in the introduction of the CHRL framework and the development of the algorithms supporting its implementation, especially the cognitive collaborative reinforcement learning (CCRL) algorithm.

.

Key words: Hierarchical reinforcement learning, Human-robot collaboration, Cognitive robot learning, Path planning, Scheduling.

Table of Contents

List of Appendices V
List of Figures VI
List of Tables VII
Acronyms VII

1. Introduction 1
1.1 Problem Description and Research Motivation 1
1.2 Research Objectives 3
1.3 Research Contributions and Innovations 3

2. Scientific Background 5
2.1 Reinforcement Learning 5
2.2 Common Reinforcement Learning Algorithms 5
2.3 Hierarchical Reinforcement Learning 8
2.4 Reinforcement Learning for Scheduling 10
2.5 Robot Learning 12
2.6 Collaborative Learning 13
2.7 Summary 16

3. Methodology 18
3.1 Introduction 18
3.2 Robotic Toast Making System 19
3.3 Collaborative RL for a 3D Path Planning Task 21

4. Collaborative Hierarchical Reinforcement Learning 23
4.1 Introduction 23
4.2 Key Concepts 24
4.3 An illustrative example 25

5. RL Sequencing Algorithm (SRL) 26
5.1 Introduction 26
5.2 The SRL Algorithm 27

.6 Cognitive Collaborative Reinforcement Learning (CCRL) 30
6.1 Introduction 30
6.2 The CCRL Algorithm 31
6.3 Collaborative Learning 31
6.4 Self-Performance Assessment Capability 32
6.5 Advice Request Test 32
6.6 Advice Assessment Capability 33
6.7 Advice Rejection Test 34

7. Sub-task Sequencing for a Toast Making System 36
7.1 Introduction 36
7.2 High Level Learning Task – Toast Transition Sequencing 36
7.3 Experimental Setup and Method of Operation 36
7.4 Task Definition 38
7.5 Implementation of the SRL Algorithm 39
7.6 Analysis 40
7.7 Results and Discussion 42
7.8 Summary 50

8. Path Planning for a Toast Making System 51

8.1 Introduction 51
8.2 Task Definition 51
8.3 Experimental Setup and Method of Operation 52
8.4 Evaluation and Summary 54

9. Evaluation of CCRL using a 3D Path Planning Task 55
9.1 Introduction 55
9.2 Representation of Advisor Skill Levels 55
9.3 CCRL 57
9.4 Fully Autonomous Learning (Base-Line) 57
9.5 Introspection Approach (IA) 58
9.6 Combined Method 58
9.7 Task Definition 58
9.8 Analysis 60
9.9 Results and Discussion 61
9.10 Summary 72

10. Conclusions and Future Research 73
10.1 Conclusions 73
10.2 Future Research 76

11. References 79

List of Appendices

Appendix I. Toast Making System – Specifications and Operation 83
Appendix II. Statistical Analysis 92
Appendix III. Task Sequencing for a Flexible Manufacturing System (FMS) 94
Appendix IV. Toast Making System – Additional Results 104
Appendix V. 3D Path Planning Task – Additional Results 108
Appendix VI. Toast Making System – Source Code 120
Appendix VII. 3D Path Planning Task – Source Code 145
Appendix VIII. Task Sequencing for a FMS – Source Code 151

List of Figures

Fig. 2.1 SARSA algorithm pseudo code ... 7
Fig. 2.2 Q-learning algorithm pseudo code ... 7
Fig. 2.3 Q(λ) algorithm pseudo code ... 8
Fig. 4.1 Flowchart of the CHRL framework ... 24
Fig. 5.1 Pseudo-code of the SRL algorithm ... 29
Fig. 6.1 Scheme of the CCRL algorithm ... 34
Fig. 6.2 Pseudo-code of the CCRL algorithm ... 35
Fig. 7.1 General scheme of the toast making system ... 37
Fig. 7.2 Motoman robot and cardboard mockup of the system stations ... 37
Fig. 7.3 Convergence to the scheduling problem's solution, case II ... 43
Fig. 7.4 Toast locations for the sequence found, case II ... 43
Fig. 7.5 Action selection analysis, 3-toast problem, type A reward factor, case II ... 44
Fig. 7.6 Adaptive ε-greedy action selection ... 44
Fig. 7.7 Reward factor analysis, 3-toast problem, case I, deterministic environment ... 45
Fig. 7.8 Reward factor analysis, 4-toast problem, case I, deterministic environment ... 45
Fig. 7.9 Performance comparison – 3-toast problem, type B reward factor, case III, deterministic environment ... 46
Fig. 7.10 Performance comparison – 3-toast problem, type B reward factor, case I, stochastic environment ... 47
Fig. 7.11 Performance comparison – 4-toast problem, type B reward factor, case III, deterministic ... 47
Fig. 7.12 Performance comparison – 4-toast problem, type B reward factor, case I, stochastic environment ... 48
Fig. 8.1 Experimental setup – side and overhead views ... 52
Fig. 8.2 Simulated environment ... 53
Fig. 8.3 User interface - low level learning task ... 53
Fig. 8.4 State-action value map ... 54
Fig. 9.1 Human advisor representation ... 57
Fig. 9.2 The grid-worlds ... 59
Fig. 9.3 Convergence in a learning session ... 62
Fig. 9.4 Autonomous learning - influence of β ... 62
Fig. 9.5 CCRL - collaboration level switching, example I ... 63
Fig. 9.6 CCRL – collaboration level switching, example II ... 64
Fig. 9.7 CCRL - influence of Λ ... 64
Fig. 9.8 CCRL - learning without advice assessment capabilities ... 65
Fig. 9.9 CCRL - learning with advice assessment capabilities ... 65
Fig. 9.10 CCRL - influence of Ω ... 66
Fig. 9.11 IA - influence of the width parameter Ψ ... 67
Fig. 9.12 IA - learning with various advisors ... 68
Fig. 9.13 Combined method - learning with various advisors ... 68

List of Tables

Table 2.1 RL notations 6
Table 2.2 Summary of related work 17
Table 7.1 System stations and machine processing times 38
Table 7.2 Robot transition times (sec.) 38
Table 7.3 Summary of tests 42
Table 7.4 Summary of case I results 48
Table 7.5 Summary of case II results 49
Table 7.6 Summary of case III results 49
Table 9.1 Summary of tests 61
Table 9.2 Scores for world I with full view 70
Table 9.3 Scores for world I with limited view 70
Table 9.4 Scores for world II with full view 71
Table 9.5 Scores for world II with limited view 71

Acronyms

AGV	Automated Guided Vehicle
CCRL	Cognitive Collaborative Reinforcement Learning Algorithm
CHRL	Collaborative Hierarchical Reinforcement Learning
CQ(λ)	Collaborative *Q(λ)*
FIFO	First In First Out
FMS	Flexible Manufacturing System
GUI	Graphical User Interface
HO	Human Operator
HRI	Human-Robot Interaction
HRL	Hierarchical Reinforcement Learning
IA	Introspection Approach
MC	Monte-Carlo
ML	Machine Learning
RL	Reinforcement Learning
SRL	Sequencing Reinforcement Learning Algorithm

1. Introduction

Chapter Overview

This chapter describes the problem addressed in this work and presents the research objectives, contributions and innovations.

1.1 Problem Description and Research Motivation

To expand robotic applications into real-world environments, robots must be constructed for a large variety of tasks and be able to adapt continuously to new and changing working conditions. Since it is impossible to model all environments and task conditions, the adaptation to new tasks cannot be achieved by regular end-user programming. Rather, the robot must be delivered with advanced capabilities to autonomously learn new tasks and new working conditions.

A common learning approach in robotics, which can answer some of these challenges, is reinforcement learning (RL) [Watkins, 1989; Peng and Williams, 1996; Sutton and Barto, 1998; Ribeiro, 2002]. RL is an unsupervised learning method in which an agent (a robot[1]) learns autonomously through direct interaction with the environment, using trial and error. The basic notion is of a learning process in which an agent observes its current state (s) and chooses an action to perform (a) from a set of all possible actions, with the ultimate objective of reaching a defined goal state. The agent's actions can change both its state and the environment's state. Throughout the process, the agent receives reinforcements from the environment (r), indicating how well it is performing the required task. The robot's goal is to optimize system responses by maximizing a reward function suited for the desired task, *i.e.*, maximize the rewards received during the entire process.

RL is an attractive alternative for programming autonomous systems (agents), as it allows the agent to learn behaviors on the basis of sparse, delayed reward signals provided only when the agent reaches desired goals [Bakker and Schmidhuber, 2004]. Furthermore, RL does not require a detailed model of the environment or training examples, as it creates its own model and examples during the learning process. However, standard RL methods do not scale well for larger, more complex tasks. The extensive interaction between the robot and the environment, necessary for determining an effective policy, implies expensive computability and long learning times in large state-action spaces. Another problem, directly derived from the fundamental characteristics of RL, is the fact that it allows only one goal state for the learning task. These drawbacks present significant difficulties when developing autonomous learning robotic systems, which are characterized by large state-action spaces and typically consists of several goal states.

[1] The terms "agent" and "robot" will be used interchangeably throughout this work.

One promising approach to scaling up RL is hierarchical reinforcement learning (HRL) [Watkins, 1989; Dietterich, 1999]. Rather than attempting to solve the whole problem at once, decomposition is performed to create a hierarchical structure of sub-problems. Low-level policies, which emit the actual actions, solve only parts of the overall task. Higher-level policies solve the overall task, considering only a few abstract, high-level observations and actions. This reduces each level's search space and facilitates temporal credit assignment [Bakker and Schmidhuber, 2004]. Moreover, HRL allows a learning process to consist of more than one goal. When assuming the low-level policies are known (*i.e.*, we know how to perform them in an optimal way), what is left is to schedule (or sequence) their execution in an order that will lead to optimal execution of the overall task.

Another way of addressing the drawbacks described is to allow a human advisor or a training agent to intervene and provide guidance in the learning process. So, instead of relying solely on reinforcements provided by the environment, the learning agent also has access to supervised instruction supplied by a training agent or human advisor. A human may aid the robot in its learning process by showing it how to solve new tasks and how to improve or expand already learned behaviors. This can enable the robot to handle unknown and unpredictable events that are beyond the competence of current autonomous robotic systems. Previous research indicated that human-robot collaboration is essential to improve the learning and reduce the amount of time it takes a robot to accomplish a learning task (*e.g.*, [Papudesi and Huber, 2003; Mihalkova and R. Mooney, 2006; Kartoun, 2008]).

Nevertheless, many of the previous works assumed that human assistance is available at all times. Indeed, human intervention can improve the learning process and accelerate the robot learning, but if it is required too frequently, the autonomous sense of the learning is lost, along with the initial purpose of a robot replacing the human. Hence, a central issue in human-robot collaboration, addressed in this research, is **adjustable autonomy**, the determination of **whether** and **when** human intervention is required.

Another deficiency in prior works is the assumption that the human advisor is an expert providing only optimal advice. This might not be the case when the instructor is tired for example, or if it is a child instructing a service robot performing daily household chores. Hence, this assumption of expert advisors is relaxed in this research, so that non-expert instructors are also considered.

The framework proposed in this research, Collaborative Hierarchical Reinforcement Learning (CHRL), combines the two techniques described in the above paragraphs, hierarchical reinforcement learning and human-robot collaboration. The framework aims to enable the execution of complex tasks and to accelerate the learning process by decomposing the tasks into a two-level learning hierarchy. The high level consists of learning the desired sequence of execution of the basic sub-tasks, and the low level consists of learning how to perform each of the sub-tasks required. Since the sub-tasks may be performed many times, the reuse of learned sub-task problems provides a definite advantage. Human intervention is allowed at both levels, to expedite the learning process by exploiting human intelligence and expertise.

In this research two RL-based algorithms were developed to support the CHRL framework: (i) a sequencing RL algorithm (SRL), and (ii) a cognitive collaborative RL algorithm (CCRL). The SRL algorithm was developed [Gil *et al.*, 2008] for providing a sub-task execution sequence, as part of the high level of the hierarchy. The algorithm addresses the learning task of scheduling a single transfer agent (a robot arm) through a set of sub-tasks in a sequence that will achieve optimal task execution times. In lieu of fixed inter-process job transfers, the robot allows the flexibility of job movements at any point in time and to any location. Execution of complex tasks was demonstrated using two applications – an automated toast making system and a flexible manufacturing system. The CCRL algorithm was developed to allow the introduction of an advisor through intelligent adjustable autonomy. The autonomous learner is enabled with two cognitive capabilities: a self awareness skill to assess its own performance and decide when it is not sufficient hence advisor guidance should be solicited, and the ability to judge the value of the advice given and decide whether to accept or reject it. This approach was demonstrated and evaluated using the toast making system and a simulated three-dimensional path planning task.

1.2 Research Objectives

The fundamental research objective of this work is to introduce a new reinforcement learning framework, noted as Collaborative Hierarchical Reinforcement Learning (CHRL), designed to enable learning and execution of partially modeled complex tasks by a self learning agent, while allowing human collaboration in the process. The specific objectives are to describe the development and evaluation of the two novel algorithms supporting the implementation of the CHRL framework - the sequencing RL algorithm (SRL) and the cognitive collaborative RL algorithm (CCRL).

1.3 Research Contributions and Innovations

RL is a common learning method, widely used in the world of robotics. Although RL has many advantages over other learning methods, it has several drawbacks preventing it from answering the challenges presented by real-world applications. This thesis provides one more step in the continuing struggle to overcome these drawbacks. The CHRL framework proposed in this research combines two known techniques used for addressing the drawbacks, hierarchical RL and Human-Robot collaboration, in order to scale up RL and alleviate some of its disadvantages. This combination enables both the execution of complex tasks and the improvement of the learning process.

Two new algorithmic tools are introduced to support the new framework: The first algorithm is the SRL, a RL-based sequencing algorithm aimed to solve various problems and produce effective learning and execution under time and resource limitations, without the requirement of a detailed model of the problem or predefined scheduling rules. This work also presents an alternative view of the sequencing problem, referring to the robotic transfer agent as the limited resource, and to the tasks it has to perform as the “jobs” waiting in its queue. This view can simplify the formulation of such problems.

The second algorithm is a cognitive collaborative RL model (CCRL) which allows the learning agent not only to decide when to solicit advice, but also to recognize a less capable advisor and decide to stop the interaction, returning to autonomous operation. The CCRL algorithm improves the interaction between the learner and the advisor by finding the right balance between independent and guided learning, taking into account the advisor’s skills. In this last context, the research also suggests a new method of representing various advisor skill levels, allowing the evaluation of collaboration algorithms under realistic conditions of imperfect guidance.

Finally, this work demonstrates the applicability of RL-based methods for a real-world scenario, presenting encouraging results to support future research in this area.

2. Scientific Background

Chapter Overview

This chapter reviews the literature of the relevant research topics, in particular reinforcement learning methods and applications. Current human-robot collaboration and robot learning applications are also presented.

2.1 Reinforcement Learning

"Reinforcement learning (RL) is a computational approach to understanding and automating goal-directed learning and decision-making. It is distinguished from other computational approaches by its emphasis on learning by the individual from direct interaction with its environment, without relying on exemplary supervision or complete models of the environment" [Sutton and Barto, 1998].

RL does not assume the existence of a teacher that provides training examples. The learning agent receives signals (reinforcements) from the environment indicating how well it is performing the required task. These signals are usually associated to some dramatic condition - *e.g.*, accomplishment of a sub-task (reward) or complete failure (punishment), and the agent's goal is to optimize its behavior based on some performance measures (maximization of a reward function) [Kartoun, 2008]. "The learning agent learns the associations between observed states and chosen actions that lead to rewards or punishments, *i.e.*, it learns how to assign credit to past actions and states by correctly estimating costs associated to these events" [Ribeiro, 2002].

RL algorithms make explorative and exploitative traverses in the state-space trying to find a "path" that is highly rewarded. The benefit of the algorithm is its capability of exploration, *i.e.*, traversing through states that are not well-rewarded but may yield higher reward in the long run, bypassing local maxima this way. It is important to pay attention to the exploration and exploitation balancing problem [Sutton and Barto, 1998]. [Stefan, 2003] notes that exploration is interpreted as an operation mode of the learning agent where it makes experiments and tries to discover its environment. Exploitation, on the other hand, is a mode in which the agent has gathered enough knowledge and makes real decisions.

2.2 Common Reinforcement Learning Algorithms[1]

The central idea in reinforcement learning is *Temporal Difference* (TD) learning. TD learning is a combination of Monte Carlo (MC) ideas and dynamic programming (DP) ideas. Like MC methods, TD methods can learn directly from raw experience without a model of the environment's dynamics.

[1] This review is based on material from [Sutton and Barto, 1998].

Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap).

The basic assumption in reinforcement learning studies is that any state s_{t+1} made by the agent must be a function only of its last state and action: $s_{t+1} = f(s_t, a_t)$, where $s_t \in S$ and $a_t \in A$ are the state and chosen action at time step t. *Q* is the system's estimate of the optimal action-value function. The system estimates the optimal action-value function $Q(s_t, a_t)$ directly and then uses it to derive a control policy.

$Q(s_t, a_t)$ represents the expected discounted cost for taking action a_t when visiting state s_t and following an optimal policy thereafter. These characteristics allow an iterative process for calculating an optimal action. The first step is to initialize the system's action-value function, *Q*. Since no prior knowledge is available, the initial values can be arbitrary (*e.g.*, uniformly zero). Next, the system's initial control policy is established. This control policy will chose the action to be taken from the current state. The control policy is usually derived from *Q*, and can change during the process. The next action can be chosen such that it will lead the agent to the state with the highest *Q* value (*i.e.*, greedy action selection), or using other methods, such as *ε-greedy* or *softmax* (described in Sections 5.2 and 9.2, respectively), which have a probabilistic feature, allowing better exploration of the state-space.

At time-step t, the agent visits state $s_t \in S$ and selects an action $a_t \in A$, receives from the process the reinforcement $r(s_t, a_t) \in R$ and observes the next state s_{t+1}. Then it updates the action value $Q(s_t, a_t)$ according to the used algorithm (*e.g.*, SARSA, *Q*-learning), thus completing one step. The RL notations are described in Table 2.1.

Table 2.1 RL notations

S	State space
A	Action space
$s_t \in S$	State at time step t
$s_{t+1} \in S$	State at time step $t+1$
$a_t \in A$	Action at time step t
$a_{t+1} \in A$	Action at time step $t+1$
$r(s_t, a_t)$	Reward at time step t
α	Learning rate. Controls the weight given to the new *Q* estimate, as opposed to the old one
γ	Discount factor. Determines the present value of future rewards
$e(s_t, a_t)$	Eligibility trace
λ	Eligibility trace factor
δ	Temporal difference error
$Q(s_t, a_t)$	State-action value estimate

2.2.1 SARSA

SARSA is an On-Policy TD control algorithm, meaning that the evaluated policy is the policy used for control. The Q value estimates are updated after each step. The pseudo code of the SARSA algorithm is described in Fig. 2.1.

```
Initialize $Q(s,a)$ arbitrarily
**Repeat** (for each learning episode *n*):
        Initialize state $s_t$, pick initial action $a_t$
        **Repeat** (for each step *t* of episode):
                Take action $a_t$, observe $r_t$, $s_{t+1}$
                Choose $a_{t+1}$ for $s_{t+1}$ using a policy derived from $Q$
                $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
                $s_t \leftarrow s_{t+1}$;   $a_t \leftarrow a_{t+1}$
        **Until** $s_t$ is terminal
**Until** $n = N$ (reached the desired number of learning episodes)
```

Fig. 2.1 SARSA algorithm pseudo code

2.2.2 Q-Learning

Q-learning is an Off-Policy TD control algorithm, meaning that the evaluated policy is not necessarily the policy used for control - the action chosen to be taken, a_t, isn't necessarily the one that will locally maximize the action-value, although the Q updating equation assumes the optimal expected cost. This is done in order to encourage exploration of the state-space, and avoid from converging to a local optimum. A way to achieve this is to use action selection methods such as *ε-greedy* or *softmax*. The pseudo code of the *Q*-learning algorithm is described in Fig. 2.2.

```
Initialize $Q(s,a)$ arbitrarily
**Repeat** (for each learning episode *n*):
        Initialize state $s_t$
        **Repeat** (for each step *t* of episode):
                Choose $a_t$ for $s_t$ using a policy derived from $Q$ (*e.g., ε-greedy*)
                Take action $a_t$, observe $r_t$, $s_{t+1}$
                $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
                $s_t \leftarrow s_{t+1}$
        **Until** $s_t$ is terminal
**Until** $n = N$ (reached the desired number of learning episodes)
```

Fig. 2.2 Q-learning algorithm pseudo code

2.2.3 *Q(λ)*

$Q(\lambda)$ is a generalization of Q-learning. $Q(\lambda)$ uses eligibility traces, $e(s_t, a_t)$: the one-step Q-learning is a particular case with $\lambda = 0$. The Q-learning algorithm might learn slowly since only one time-step is traced for each action. To boost learning, a multi-step tracing mechanism, the eligibility trace, is used in which the Q values of a sequence of actions can be updated simultaneously according to the respective lengths of the eligibility traces. The pseudo code of the *Q(λ)* algorithm is described in Fig. 2.3.

```
Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s,a$
Repeat (for each learning episode $n$):
    Initialize state $s_t$, $a_t$
    Repeat (for each step $t$ of episode):
        Take action $a_t$, observe $r_t$, $s_{t+1}$
        Choose $a_{t+1}$ for $s_{t+1}$ using a policy derived from $Q$ (e.g., ε-greedy)
        $a^* \leftarrow \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$ (if $a_{t+1}$ ties for the max, than $a^* = a_{t+1}$)
        $\delta_t \leftarrow r_t + \gamma Q(s_{t+1}, a^*) - Q(s_t, a_t)$
        $e(s_t, a_t) \leftarrow e(s_t, a_t) + 1$
        For all $s,a$
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
            If $a_t = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
                    else $e(s,a) \leftarrow 0$
        $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$
    Until $s_t$ is terminal
Until $n = N$ (reached the desired number of learning episodes)
```

Fig. 2.3 Q(λ) algorithm pseudo code

2.3 Hierarchical Reinforcement Learning

The notion of hierarchical reinforcement learning (HRL) presented in this work has been applied for various problems. [Dietterich, 1999] states that "the ideal hierarchical RL method would provide the benefits of hierarchy (faster learning and sub-task sharing and reuse) while maintaining the benefits of RL (optimality, online learning from the environment and autonomy)." Specifically, an ideal method should satisfy the following requirements [Dietterich, 1999]: (i) support state abstraction, *i.e.*, make it possible for individual sub-tasks to ignore irrelevant aspects of the state space, (ii) sharing/reuse of sub-tasks – the method should make it possible to learn an optimal policy for a sub-task and then reuse the learned policy for different parent tasks, (iii) efficient learning – obviously, hierarchical RL will be useless if it does not provide better performance than non-hierarchical methods, (iv) optimally – the method should learn optimal or near-optimal policies, and (v) online learning – The method should be able to learn online working with the entire task. [Sun and Sessions, 2000] separate the many existing models of HRL into two cases: (i) the use of structurally pre-determined domain-specific hierarchies and (ii) automatic building of hierarchies. Furthermore, they distinguish two directions in automatically building hierarchies: upward and downward.

An application of hierarchical RL to the problem of negotiating obstacles with a quadruped (four legged robot) is described in [Honglak *et al.*, 2006]. The algorithm is based on a two-level hierarchical decomposition of the task, in which the high-level controller selects the sequence of foot placement positions, and the low-level controller generates the continuous motions to move each foot to the specified positions. The high-level controller uses an estimate of the value function to guide its search. Then, a beam search is used to look multiple steps ahead, and try to find a sequence of actions to move the robot towards the goal. The low-level controller is obtained via policy search. A reward function penalizes undesirable behaviors such as taking a long time to complete the foot movement, passing too close to of an obstacle, or failing to move the foot to the desired goal location. After learning the parameters for both the low and the high level controllers, the resulting hierarchical policy was tested, in simulation, and later using the real robot, for a large variety of obstacles. The experiment demonstrates that the robot can successfully climb over a variety of obstacles which were not encountered at the training stage.

In [Jeni *et al.*, 2007], HRL is employed to a mobile robot navigation task. The environment is decomposed into separate sections, and an interconnection function describes the prior knowledge of how various parts of the environment are connected. In the first part of the learning process the algorithm explores the state space by starting several trajectories. If it reaches an interconnecting state it creates a new abstract state, which contains the states of the trajectory. If it finds a path between two abstract states, which does not contain a connecting state, it merges the two states. The result of this stage is a set of abstract states and each state represents a partition of the original state space. In the second part of the learning process the algorithm learns partial policies on the partitions represented by the abstract states. The results of simulations performed illustrated that the algorithm performs much better than the flat learner algorithm, but it requires some prior knowledge about the problem.

A method called HASSLE (Hierarchical Assignment of Sub-goals to Sub-policies Learning algorithm) is presented in [Bakker and Schmidhuber, 2004]. As in other HRL methods, the high-level value functions cover the state space at a coarse level, and the low-level value functions cover only parts of the state space at a fine-grained level, with the aim of reaching the sub-goals assigned by the high-level policy. The difference is that unlike other methods, at HASSLE the high-level policies not only select the next sub-goal to be reached by a lower-level policy, but also autonomously discover and define sub-goals. Both high-level policies and low-level policies use essentially standard value function-based reinforcement learning algorithms. The issue of how to autonomously arrive at abstract high-level observations is addressed using an unsupervised clustering algorithm. The main requirement is that a clustering of "primitive", low-level observations is accomplished, such that neighboring low-level states tend to be clustered together. The HASSLE algorithm was tested using a navigation task in a simulated "office" grid world. The agent had to learn to move from any possible start position to a fixed goal position. Experiments showed that HASSLE outperformed standard, "flat" RL methods in deterministic and stochastic tasks, and learned significantly faster. Nevertheless, the system has some limitations, such as large number of parameters, lack of strict convergence guarantees and the dependence on identifying reasonable high-level observations.

Similarly to HRL, Compositional *Q*-Learning (*CQ-L*) [Singh, 1992] is a modular approach to learning to perform composite tasks made up of several elemental tasks by RL. In *CQ-L*, skills acquired while performing elemental tasks are also applied to solve composite tasks. Individual skills compete for the right to act and only winning skills are included in the decomposition of the composite task. [Tham and Prager, 1995] extend the original *CQ-L* concept in two ways: (i) a more general reward function, and (ii) the agent can have more than one actuator. They use the *CQ-L* architecture to acquire skills for performing composite tasks with a simulated two-linked manipulator having large state and action spaces. The agent is required to drive the manipulator from an arbitrary starting arm configuration to one where its end-effector is brought to a fixed destination in the case of elemental tasks, or to several destinations, one after another, for composite tasks. Results indicated that the *CQ-L* architecture can be successfully applied to the learning of complex composite tasks with large state and action spaces.

2.4 Reinforcement Learning for Scheduling

Production scheduling is one of the most important processes in manufacturing systems, and when properly executed can provide such benefits as increased throughput, enhanced customer satisfaction, lower inventory levels, and increased utilization of resources [Wang and Usher, 2005]. Scheduling problems essentially involve completing a set of jobs with a limited number of manufacturing resources and under various constraints, with the objective of optimizing performance measures such as makespan (total completion time), mean flow time and mean tardiness [Wang and Usher, 2005].

[Stefan, 2003] describes the three main scheduling concepts: mathematically grounded algorithms, heuristic approaches and algorithms supported by machine learning (ML). The first concept can be adapted to small-sized scheduling problems. The advantage of the concept is that it is well defined, exact and can be generally applied to the wide range of two-machine scheduling tasks.

The price is lack of scalability, *i.e.*, no mathematical proof can be given for a larger number of machines. He states that "there are two directions of research to overcome the restrictions of mathematical formulations: using heuristics and/or machine-learning. While heuristic approaches provide direct rules of thumb to follow, but no algorithm to find the solution in a modified decision environment, ML methods give a model of a mental process itself. As the knowledge of the learning agent improves the method results in solutions that are more and more close to the optimal solution, even in a changing environment." In his research, [Stefan, 2003] propose an RL-based algorithm designed to give a quasi-optimal solution to the m-machine flow-shop scheduling problem. Namely, given a set of parts to be processed and a set of machines to carry out the process, each part should have the same technological path on all machines and the order of jobs can be arbitrary. The goal is to find an appropriate sequence of jobs that minimizes the sum of machining idle times. States are defined as job sequences, or more precisely job precedence relations. State-changes (or actions) are defined as changes in relations. Results indicated that the RL-scheduler was able to find close-to-optimal solutions.

In their study, [Wei and Zhao, 2004] developed an adaptive rule selection method for dynamic job-shop scheduling. A *Q*-learning agent performs dynamic scheduling based on information provided by the scheduling system. The learning agent's decision on the rule to be employed for selecting a job from the buffer is based on the status of the system's buffer. The agent was trained by the *Q*-learning algorithm, entailing the capabilities of selecting the appropriate rules in real time based on changes in the state of the system. The action selection was performed using an adaptive *ε-greedy* strategy[1], and the goal was to minimize mean tardiness. The *Q*-learning algorithm showed superiority over most of the conventional rules compared for a simulated environment. [Creighton and Nahavandi, 2002] present an intelligent agent-based scheduling system for solving the Economic Lot Scheduling Problem (ELSP). This problem refers to the production of multiple parts on a single machine, with the restriction that no two parts may he produced at the same time. The production facility studied was a multi-product serial line subject to stochastic failure. The agent goal was to minimize total production costs, through selection of job sequence and batch size. By applying an independent inventory control policy for each product, the agent successfully identified optimal operating policies for a real production facility.

[Gabel and Riedmiller, 2007] note that "most approaches to tackle job-shop scheduling problems assume complete task knowledge and search for a centralized solution." In their work they adopt an alternative view where each resource is equipped with an adaptive agent that, independent of other agents, makes job dispatching decisions based on its local view on the plant and employs

[1] The adaptive *ε-greedy* action selection method is described in Section 5.2.

reinforcement learning to improve its dispatching strategy. This decentralized approach is particularly suitable for environments where unexpected events may occur, such as the arrival of new tasks or machine breakdowns, hence frequent re-planning would be required. The empirical evaluation in the research leads to the conclusion that problems of current standards of difficulty can very well be effectively solved by the learning method they suggest.

2.5 Robot Learning

"Robotics is one of the most challenging applications of machine learning techniques. It is characterized by direct interaction with a real world, sensory feedback and complex control tasks" [Kreuziger, 1992]. Learning should lead to faster and more reliable solution executions, and to development of the ability to solve problems the robot was not able to solve before. [Connell and Mahadevan, 1993] state "building robotic systems that learn to perform a task has been acknowledged as one of the major challenges facing artificial intelligence. Self-improving robots can relieve humans from much of the drudgery of programming and potentially allow their operation in unknown and dynamic environments." Progress towards this goal can contribute to intelligent systems by advancing the understanding of how to successfully integrate disparate abilities such as perception, planning, learning, and action.

Common robot learning tasks, such as navigation in an environment, include: (i) localization - the process of determining the robot's location; (ii) mapping - the process of building a model of the environment, and (iii) planning - the process of planning the robot's movements [Howard, 1999]. A navigation learning task of a miniature mobile robot equipped with vision capabilities using several RL-based algorithms is described in [Bhanu *et al.*, 2001]. Comparison between the Q and $Q(\lambda)$ algorithms for a 6 x 6 maze show only a few significant differences between the two learning algorithms. Overall, the $Q(\lambda)$ algorithm takes fewer actions during the entire experiment, suggesting it is faster in finding the shortest path. A RL algorithm for accelerating acquisition of new skills by real mobile robot is presented in [Martínez-Marín and Duckett, 2005]. The algorithm, tested using a docking task, speeds up Q-learning by applying memory-based sweeping [Touzet, 2003].

Autonomous object approaching with an arm-hand robot is a very difficult problem since the possible configurations are numerous. [Wang *et al.*, 2006] propose a modified RL algorithm for solving the problem of how a multi-fingered robotic hand should approach objects before grasping. Learning is divided into two phases, heuristic learning and autonomous learning. In the first phase, the heuristic search (a function of A^* search) is utilized to help the robot reach the goal quickly, while updating a Q table. Once the table has been modified enough to effectively control the robot, the second learning phase starts. In this phase, the robot is trained using a standard RL learning method, which impels the robot to find the local optimal policy. The experimental results demonstrate the effectiveness of the proposed algorithm.

It is stated in [Kartoun, 2008] "although *Q*-learning and *Q(λ)* were used in many fields of robotics, the issue of acceleration of learning towards finding an optimal or close to optimal solution is still significant."

2.6 Collaborative Learning

Human-robot interaction (HRI) can be defined as the study of humans, robots, and the manner in which they influence each other. Sheridan describes a ten-level formulation of robot autonomy, viewing the robot as a highly intelligent system that is capable of performing a task by itself in a given context [Sheridan, 1987]. The degree of robot autonomy is scaled accordingly based on human decisions when performing the task. Through this, a balance of control between the robot and the human is achieved. On the one hand, to ensure that highest-quality decisions are made, a robot should transfer control and collaborate with a human operator (HO) when it has superior decision-making expertise. On the other hand, interrupting a user might cause delays or acquiring information that is not necessarily beneficial; thus such transfers of control should be optimized.

Fong and his co-researchers [Fong *et al.*, 2001] determined that there are four key issues that must be addressed when constructing a collaborative control system. First, the robot must have self-awareness, not in the sense of being fully sentient, but merely in having the capabilities for detecting and determining if it should ask for help, and recognizing when it has to solve problems on its own. Second, the robot must be self-reliant. Since the robot cannot rely on the human to always be available or to provide accurate information, it must be able to maintain its own safety. Specifically, the robot should be capable of avoiding hazards, when necessary. Third, the system must support dialogue, allowing the robot and the human to communicate effectively with each other. Each participant must be able to convey information, to ask questions and to judge the quality of responses received. Finally, the system must be adaptive. By design, collaborative control provides a framework for integrating users with varied skills, knowledge, and experience. [Kartoun, 2008] points that collaboration between a robot and a human during learning is beneficial since humans have superior intelligence and skills such as perception, intuition and awareness, to direct policy adjustments in the most suitable direction. These skills are especially important in real-world applications which are characterized by unknown and unstructured environments.

[Breazeal and Thomaz, 2008] indicate that past work that incorporate human input into a Machine Learning process tend to maintain a constant level of human involvement. Several are highly dependent on guidance, learning nothing without human interaction, while other approaches are almost entirely exploration based, using limited input from a teacher. They posit that a social learner must be able to move flexibly along this guidance-exploration spectrum, explore and learn on its own, but also take full advantage of a human partner's guidance when available.

Human-robot collaboration research deals with collaboration between the HO and the system and the level of automation in aspects of data acquisition, data and information analysis, decision making, action selection and action implementation, in accordance to specific task or sub-task goals and parameters [Bechar *et al.*, 2006]. Shifting from one collaboration level to another during task performance is required in cases in which the robot or the HO parameters change [Bechar *et al.*, 2003]. [Kartoun, 2008] states "a robot system performing a learning task has to be designed in such a way to consider how to achieve optimal cooperation via appropriate degrees of sharing and trading between human and robot." Human-robot collaboration is unnecessary as long as the robot learns policies autonomously, and adapts to new states. The collaboration with a HO should be triggered when a robot reports to the human that its learning performance is low. Then the human is required to intervene and suggest alternative solutions [Kartoun, 2008].

Sliding scale autonomy is defined in [Yanco *et al.*, 2005] as the ability to create new levels of autonomy between existing, pre-programmed autonomy levels. The suggested sliding scale autonomy system allows dynamical combination of human and robot inputs, using a small set of variables such as user and robot speeds, speed limitations, and obstacle avoidance. An experimental environment called EVIPRO (Virtual Environment for Prototyping and Robotic) was developed allowing the assistance of autonomous robots during the realization of a teleoperation mission [Heguy *et al.*, 2001]. This project studied man-machine cooperation in a system using virtual reality and adaptive tools. The goal of the human users and autonomous robots was to achieve a global task in virtual environment. Thanks to virtual reality, the project could have natural and intuitive interface, and allowed mixing of different information to increase user perception. A collaborative process enabling a robotic learner to acquire concepts and skills from human examples is presented in [Lockerd and Breazeal, 2004]. During the teaching process, the robot performs tasks based on human instructions. Using a Q-learning approach, the robot learns a button pushing task.

Mobile robot optimal navigation to a specific target in a two-dimensional world, is achieved by changes in rewards and Q-value functions, performed by the user [Papudesi and Huber, 2003; Papudesi *et al.*, 2003]. In [Wang *et al.*, 2003], a variable autonomy approach is used. User commands serve as training inputs for the robot learning component, which optimizes the autonomous control for its task. This is achieved by employing user commands for modifying the robot's reward function. Similarly, [Thomaz and Breazeal, 2006] describe a new RL-based approach for giving reward signals by human. The signals depend not only on past actions but also on future rewards. The experimental platform, called "Sophie's Kitchen", simulates a cake baking learning process. The platform was developed for investigating how human prefer to interact with a robotic learner. One feature of “Sophie’s Kitchen” is called the “Interactive Rewards Interface” in which humans can give rewards in two ways: (i) rewarding a whole state of the world, and (ii) rewarding a state of a particular object. This distinction was made in order to examine the hypothesis that people prefer to communicate feedback about particular aspects of a state rather than an entire world state. Results achieved indicate that people use the reward signal not only to provide feedback about past actions, but also to provide future directed rewards to guide subsequent actions. Given this, and after making specific modifications to the simulated RL robot to incorporate guidance, the results show significant improvements on several measures. The work demonstrates the importance of understanding the human-teacher / robot-learner system as a whole in order to design algorithms that support how people want to teach while simultaneously improving the robot’s learning performance.

The steps involved in taking advice from external entities were defined [Hayes-Roth *et al.*, 1980] as: (i) request the advice, (ii) convert the advice to an internal representation, (iii) convert the advice into a usable form, (iv) integrate the reformulated advice into the agent’s current knowledge base, and (v) judge the value of the advice. The potential of learning from environmental reinforcement and human advice is illustrated in [Papudesi and Huber, 2003]. By incorporating advice into an additional reward function, the advisor is provided with high degree of freedom in shaping the control policy, but cannot prevent the achievement of the overall task. Furthermore, strategic advice can accelerate the learning process, while incorrect advice is ultimately ignored, as its effects diminish over time.

[Cetina, 2007] introduce a supervised reinforcement learning architecture for robot control problems with high dimensional state spaces. A supervisor is used to dynamically generate subsets of relevant actions at each state of the environment. The use of these subsets of actions leads the agent to exploit relevant parts of the action space, avoiding the selection of irrelevant actions, and accelerate its learning rate very early in the learning process. Once the agent has exploited the information provided by the behavior model, it keeps improving its value function without any help, by selecting the next actions to be performed from the complete action space. The algorithms were tested with the robot dribbling problem, in the framework of the RoboCup simulation league. Such problem involves a continuous state space with high dimensionality. Experimental work shows how the approach can dramatically speed up the learning process.

A learning mechanism, Socially Guided Exploration, in which a robot learns new tasks through a combination of self-exploration and social interaction, is presented in [Breazeal and Thomaz, 2008]. The system’s motivational drives (novelty, mastery), along with social scaffolding from a human partner, bias behavior to create learning opportunities for a RL. The system is able to learn on its own, but can flexibly use the guidance of a human partner to improve performance, through attention direction, action suggestion, labeling of goal states, and feedback. The research platform is Leonardo ("Leo"), a 65 degree of freedom robot specifically designed for human social interaction. Leo has speech and vision sensory inputs and uses gestures and facial expressions for social communication. An experiment with non-expert human subjects shows a human is able to shape the learning process through suggesting actions and drawing attention to goal states. Human guidance results in a task set that is significantly more focused and efficient, while self exploration results in a broader set.

Another implementation of human-robot collaborative learning process is described in [Kartoun, 2008]. An ER-1 mobile robot was required to navigate toward a target location in a two-dimensional world containing undesirable navigation areas. The robot, located remotely from the HO, used environmental sensing capabilities. Learning was achieved by sharing knowledge with the HO, using a Collaborative *Q(λ)* learning algorithm, noted as *CQ(λ)*. Two levels of collaboration where defined: (i) autonomous - the robot decides which actions to take, acting autonomously according to its *Q(λ)* learning function, and (ii) semi-autonomous - HO suggests actions remotely and the robot combines this knowledge into its *CQ(λ)* learning function. Evaluating robot performance for the navigation task revealed the superiority of the collaborative algorithm, *CQ(λ)*, over the standard *Q(λ)* algorithm for various parameter combinations. Results show that the human collaboration accelerated robot learning performance for different collaboration threshold values (the threshold values determine the balance between autonomous and collaborative learning). On the other hand, the human intervention rate was not consistent with the improvement level of the robot. The Introspection Approach (IA) [Clouse, 1996] is a similar method by which the learning agent determines when it requires aid from a training agent. In IA the agent asks for instruction when it is confused or otherwise unable to decide upon a course of action. To implement IA, a test was developed to determine whether the learner is unsure of its choices, indicating the need for help in novel situations. The test examines the two extreme values of possible actions: if they are close to each other it implies that the learner has not experienced this state often enough to produce a clear choice, thus should ask for advice. The IA approach was evaluated using two-dimensional maze problems in which the agent is required to traverse optimally from a starting cell to a goal cell, and compared against an approach in which the learning agent requests help randomly. Guidance received via IA was shown to be more informative than random guidance, thus reducing the interaction that the learning agent has with the training agent without reducing the speed with which the learner develops its policy.

2.7 Summary

Significant work related to reinforcement learning applied for robot learning, human-robot interaction and scheduling problems is summarized in Table 2.2.

Table 2.2 Summary of related work

Robotic Applications		
Method	Application	Reference
Socially Guided Exploration (HRI)	human social interaction (Leonardo)	Breazeal and Thomaz, 2008
Collaborative $Q(\lambda)$-learning	Mobile robot navigation	Kartoun, 2008
Hierarchical RL	Mobile robot navigation	Jeni *et al.*, 2007
Supervised RL (HRI)	robot dribbling (RoboCup)	Cetina, 2007
Q-learning	"Relocation" of mobile robots	Mihalkova and Mooney, 2006
A^* and Q-learning	Object approaching with multi-fingered robotic hand	Wang *et al.*, 2006
Human-computer interaction and future directed rewards	Sophie's Kitchen	Thomaz and Breazeal, 2006
Hierarchical RL	Quadruped robot obstacle negotiation	Honglak *et al.*, 2006
Q-learning	Mobile robot	Martínez-Marín and Duckett, 2005
HRI and sliding scale autonomy	Robot speed control and obstacle avoidance	Yanco *et al.*, 2005
Q-learning and human instructions	Robot button pushing task	Lockerd and Breazeal, 2004
Hierarchical RL	autonomously discover and define subgoals (HASSLE)	Bakker and Schmidhuber, 2004
HRI and Q-learning	Mobile robot navigation	Papudesi *et al.*, 2003
HRI and variable autonomy	Modifying mobile robot reward function	Wang *et al.*, 2003
Q and $Q(\lambda)$-learning	Mobile robot navigation	Bhanu *et al.*, 2001
Virtual reality and behavior simulation	Cooperative assistance in teleoperation (EVIPRO)	Heguy *et al.*, 2001
RL-based approach involving human interaction	Human teacher to guide exploration during learning	Clouse, 1996
Compositional Q-learning	Simulated two-linked manipulator	Tham and Prager, 1995

Scheduling Applications		
Method	Application	Reference
Q-learning	Large scale job-shop problems	Gabel and Riedmiller, 2007
Q-learning	Dynamic job-shop scheduling	Wei and Zhao, 2004
Q-learning	m-machine flow-shop scheduling	Stefan, 2003
RL and Simulation	Economic Lot Scheduling	Creighton and Nahavandi, 2002

3. Methodology

Chapter Overview

This chapter describes the methods used in this research. An overview of the algorithms and learning systems developed is introduced first, followed by problem definitions and notations, experiments and performance measures.

3.1 Introduction

The collaborative hierarchical reinforcement learning (CHRL) framework presented in this work was developed in order to allow the execution of complex tasks by a self learning agent, and to improve and accelerate the learning through the use of advisor guidance.

Two algorithms were developed in order to support the CHRL approach: (i) a sequencing algorithm (SRL) was developed to address the high level learning task of the hierarchical reinforcement learning approach, a task of determining the optimal order of execution of low level sub-tasks; (ii) a cognitive collaborative reinforcement learning algorithm (CCRL) was developed in order to allow the introduction of an instructor into the learning process, endowing the learner with the abilities to decide when to ask for guidance and to evaluate the quality of the guidance.

A robotic toast making system was used to demonstrate the applicability of the CHRL framework and to evaluate the two algorithms. Toast making is a complex multi-goal task since it is composed of many sub-tasks (such as grasping a toast, inserting it to the toaster or applying butter over it), each having its own goal state. In this system, the SRL algorithm was used to create a sequence of required sub-tasks (robot movements)[1] and the CCRL algorithm was used for learning how to perform one of those sub-tasks. The CCRL algorithm was thoroughly evaluated[2] using a simulated 3D path planning task.

In all applications the learning phase of the task is performed using MATLAB simulations, in which the algorithms (SRL and CCRL) are implemented. The use of simulations allows fast learning, due to the fact that no real robot manipulations are required. Furthermore, the simulation constitutes a convenient and powerful tool for analyzing the performance of the algorithms, by conducting various virtual experiments off-line.

[1] The SRL algorithm was also applied for a Flexible Manufacturing System for further evaluation - see Appendix III.
[2] The CCRL algorithm was employed both for the toast making system (solving a 2D path planning task) and for a simulated 3D path planning task. Since the 3D task is more complex and poses greater challenges, a thorough statistical evolution was performed only for it.

3.2 Robotic Toast Making System

3.2.1 Problem Definitions and Notations

The robotic toast making system includes six stations (two processing stations and four storage stations) and a transfer "agent" (a fixed-arm robot), utilized to advance the toasts through the system, one toast at a time. There are predefined process and transferring times. The complex task of toast making is addressed by decomposing it into a two-level learning hierarchy to be solved by CHRL. The high-level consists of learning the desired sequence of execution of basic sub-tasks (the sequencing of the robot's toast transfers) and the low-level consists of learning how to perform each of the sub-tasks (*i.e.*, learn actual robot movements).

High-level learning task (sub-task sequencing):

In this application the SRL algorithm is used to generate a sequence of toast (robot) transitions through the system stations that will result in the completion of toast making in minimum time. The sequencing of the robot's transitions can be viewed as a job sequencing problem, where the robot is the "machine", and the toasts transitions are the "jobs" waiting in its queue, each requiring a different "process time" (robot transition time). Here, as in conventional job sequencing problems, there is a need to prioritize the job execution (toast transfers) using a certain policy (*i.e.*, determine which toast will be transferred first).

Learning the high-level sequencing task is performed off-line using an event-based MATLAB simulation. On-line fixed-arm robot motions are performed only after the simulation supplies the desired sequence.

To solve the sequencing problem using the SRL algorithm, it is formulated as a RL problem[1]. The system's overall state at time step *t*, denoted as $s_t \in S$, is defined by the current locations of the toasts. A solution is a specific sequence of toast transfers: "move toast 1 to its next station, move toast 3 to its next station, move toast 1 to its next station, move toast 3…". The goal state of the learning task is the state where all the toasts have reached the finished plate. An action at step *t* is denoted as $a_t \in A(s_t)$, where *A* is the action space of all possible actions (the action space is state dependent). The execution of an action constitutes the advancement of a toast to its next station in the processing sequence. Rewards are assigned according to the performance, as explained in Section 5.2.

[1] A detailed formulation is presented in Section 7.5.

Low-level learning task (path planning):

This application includes an example for learning the execution of one of the required low-level sub-tasks, a task of path planning for the insertion of a slice of bread into the toaster. In this task the robot-arm is required to transfer the bread slice in the shortest path from a starting position to a target position above the toaster's hatch, while avoiding obstacles. During the learning phase a human advisor guides the robot when requested. The optimal path is learned using a simulation employing the CCRL algorithm integrated with a standard *Q(λ)* algorithm [Watkins, 1989] and a human advisor. Once the path is obtained it is sent to the robot's controller to carry out the actual robot motions accordingly.

The robot's state at time step *t*, $s_t \in S$, is represented by its location in a 2D grid world, defined by two coordinates. An action $a_t \in A$, taken at each step, is traveling left, right, forward, or backward. Rewards are defined as $r(s_t, a_t)$. If the robot reaches the target, the reward is positive. If it passes through an undesirable area (obstacle), the reward is negative. Furthermore, a small negative reward is assigned after each step in order to facilitate minimal number of steps.

3.2.2 *Analysis*

High-level learning task (sub-task sequencing):

The sequencing algorithm's performance is tested using two problems: a "3-toast" problem and a "4-toast" problem, requiring the generation of a sequence for the optimal production of 3 and 4 toasts, respectively. Based on a MATLAB simulation, tests are conducted to examine the influence of the action selection parameters and the reward factors on the performance, and to evaluate the performance by comparison to the Monte-Carlo method [Sutton and Barto, 1998] and to a random search algorithm. Three sets of machine processing times and robot transfer times are considered, one adjusted to suit the transition times of the real robot and two with increased processing times and modified robot transition times.

Low-level learning task (path planning):

In this experiment the robot-arm is required to transfer the bread slice in the shortest path from a starting position to a target position above the toaster's hatch, while avoiding obstacles. The optimal path is learned using a simulation, employing the CCRL algorithm, and the actual robot motions are performed later, according to that path. Human advice is introduced during the path learning process, using a dedicated interface.

3.2.3 Performance Measures

System performance is evaluated using the following measures:

High-level learning task (sub-task sequencing):

1) Average number of learning episodes required to reach convergence[1].
2) Average percentage of learning sessions reaching the optimal solution, for deterministic times.
3) Average best result achieved in the learning session, for stochastic times, where an optimal solution cannot be defined due to the probabilistic feature.

Low-level learning task (path planning):

The low level task implementation is only intended to prove the applicability of the CHRL framework, hence the performance measure is a successful implementation, as described in Chapter 8 (See further explanation in Section 8.4).

3.3 Collaborative RL for a 3D Path Planning Task

3.3.1 Problem Definitions and Notations

The CCRL algorithm is applied for a simulated 3D path planning task[2]. Results are compared to those achieved by: (i) a fully autonomous learner, (ii) the Introspection Approach[3] and (iii) a combined method, integrating the advice request rules of both CCRL and IA. A simulated adviser with various skill levels is used in the evaluations, to examine the performance achieved with sub-optimal advice.

Evaluation is conducted for a simulated mobile robot path planning problem in a three-dimensional grid environment of size 10×10×10 (1000 states). Two grid-world instances are considered, one with a relatively low obstacle density, and another with a higher density. In order to evaluate performance in the case of human teleoperated guidance, a limited region of assistance is also considered for each world. The objective of the robot is to traverse from a starting state to a goal state through the shortest path, while avoiding obstacles. At each state $s_t \in S$, defined by three coordinates, the robot can choose one of six actions $a_t \in A$ (up, down, left, right, forward, or backward). The RL reward structure is such that the robot receives a positive reward for reaching the

[1] Detailed definitions of learning episodes, learning sessions, convergence etc. are found in Chapters 7, 8 and 9.
[2] The CCRL algorithm is applied here for a simulated environment and not a real one due to time constraints. However, the simulated environment is sufficient for the algorithmic analysis and in this aspect there is not much added value for a real robot implementation.
[3] The Introspection Approach (IA) is described in details in Sections 6.1 and 9.5.

target, a small negative reward for each step performed and a large negative reward for colliding with an obstacle (similar to the reward structure of the low-level task described in Section 3.2.1).

3.3.2 Analysis

Extensive analysis is performed using MATLAB simulations in order to examine the suggested CCRL algorithm and compare its performance to the base-line fully autonomous learning, the IA method, and the combined method integrating CCRL and IA.

All methods are evaluated using four environments – worlds I and II, each with full and limited views, and four different tests are performed for each environment: The *first* test examines the base-line fully autonomous learning, the *second* test evaluates the performance of the CCRL algorithm, the *third* test implements IA for solving the path planning problem and the *fourth* and final test explores the combined method, integrating both CCRL and IA into one algorithm.

3.3.3 Performance Measures

Performance is evaluated using the following measures:

1) Average number of requests for advice during the learning session (used only for CCRL, in which advice is requested for a whole episode).

2) Average number of steps performed using advice during the learning session (used for the IA method in which advice is requested per step, and for the comparison).

3) Average percentage of learning sessions reaching the optimal solution (minimal path length).

4) Weighted normalized scoring based on the 2nd and 3rd measures.

4. Collaborative Hierarchical Reinforcement Learning

Chapter Overview

This chapter describes the CHRL framework suggested in this research. The framework combines two known techniques used for addressing the RL drawbacks, hierarchical RL and Human-Robot collaboration.

4.1 Introduction

The Collaborative Hierarchical Reinforcement Learning framework (CHRL), illustrated in Fig. 4.1, aims to enable the execution of complex tasks and to accelerate the learning process. This is achieved by decomposing the task into a two-level learning hierarchy, while allowing human collaboration at both levels. The high level consists of learning the desired sequence of execution of basic sub-tasks, and the low level consists of learning how to perform each of the sub-tasks required. Human intervention is allowed at both levels, to expedite the learning process and to improve[1] it by exploiting human intelligence and expertise. The innovation is in combining two known techniques, hierarchical RL and human-robot collaboration, into one framework.

CHRL allows the use of accumulated knowledge gathered in previous learning sessions. Each complex task is composed of a set of basic sub-tasks. These sub-tasks can be saved in a “toolbox”, ready to be reused by the agent without the need to learn them from the start. The sub-tasks saved in the “toolbox” should be general-purposed, allowing them to be used for various high level tasks (picking up a tray for example, can be used in a hospital or in a restaurant). They may vary from delicate operations such as picking up a glass without spilling its content or performing a precise cut during a surgical operation, to obstacle-avoiding navigation and other complex tasks. In the end-state, the agent will have no need for learning new tasks, but only to learn how to use a set of available “tools” suitable for performing a specific task.

The inherent modularity and agility of the framework can simplify the formation and execution of new complex tasks: various tasks can be composed by creating a desired sequence of already known sub-tasks, available in the system’s toolbox. The SRL algorithm, presented in Chapter 5 was developed for this purpose.

As both the high and low levels of the hierarchy might still present large-scale, unstructured, unpredictable problems, human guidance and assistance should be made available for the agent, in order to improve and accelerate the learning process. The CCRL algorithm presented in Chapter 6 suggests a method of achieving this interaction in an efficient and intelligent way.

[1] Improvement means achieving better results than a fully autonomous learning, as demonstrated in Chapter 9.

It is important to emphasize that this research lays the foundations for the use of the CHRL framework, introducing the necessary tools for its application, but does not present a complete implementation. This remains open for future research[1].

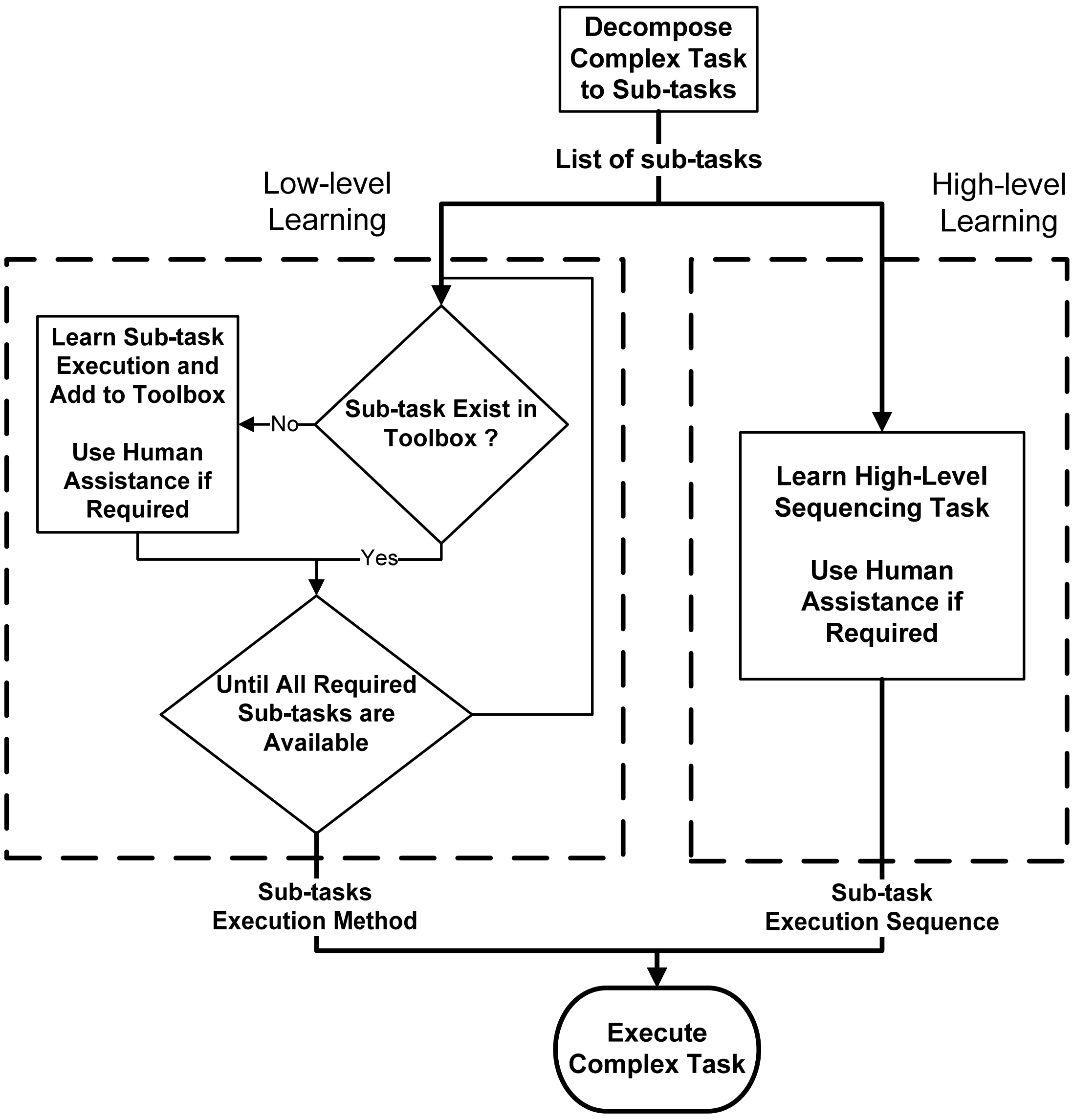

Decompose Complex Task to Sub-tasks
List of sub-tasks
Low-level Learning
High-level Learning
Learn Sub-task Execution and Add to Toolbox
Use Human Assistance if Required
No
Sub-task Exist in Toolbox ?
Yes
Until All Required Sub-tasks are Available
Learn High-Level Sequencing Task
Use Human Assistance if Required
Sub-tasks Execution Method
Sub-task Execution Sequence
Execute Complex Task

Fig. 4.1 Flowchart of the CHRL framework

4.2 Key Concepts

CHRL has four key concepts:

1) The complex task is decomposed into a sequence of low level sub-tasks, thus creating a two-level learning hierarchy: learning the desired sequence of execution (high level) and learning how to perform each of the basic sub-tasks required (low level).

[1] Areas for future expansion of this work can be found in details in Section 10.2.

2) The high level sequence can be constructed by a human operator (HO), when the full task is composed of a simple and straightforward set of steps, or by autonomous or collaborative learning sessions aimed to optimize a more complex task. The low level tasks can also be acquainted to the system by a set of predefined steps supplied by the HO (supervised control), or by autonomous or collaborative learning.

3) Learning the sequence and some of the sub-tasks (such as path planning tasks) can be performed using a simulated environment, fitted to the specific system, thus saving time spent while learning using real robot moves, and preventing safety issues. After the simulated learning sessions generate the desired set of steps, the data will be downloaded to the robot's controller for execution, allowing fast learning and implementation.

4) Sub-tasks are saved in a toolbox, available for reuse, either within the same application or in other applications.

4.3 An illustrative example

Consider a trash disposing robot required to take a trash can from a certain room in an office building, dispose of the trash in a central trash container located in another room, and return the empty can to the original room. First, the robot needs to learn how to navigate inside the room to reach the trash can and grasp it. Then, it needs to navigate out of the room and through the corridor to the container room. Finally, it will need to find the way to the container, empty the can, and return to the original room with the empty can. Now, consider emptying all trash cans of the entire floor; instead of letting the robot learn the entire task from scratch, it is suggested to create a correct sequence of sub-tasks, using the ones the robot had already learned and stored in its toolbox. Relevant general purpose sub-tasks could be: grasping a can, navigating in the corridor (each corridor is relevant for different office rooms), navigating inside the central container room, emptying the can, returning to the original room etc. By composing these general sub-tasks together in the right order, the robot's learning and task execution could be achieved much faster.

5. RL Sequencing Algorithm (SRL)

Chapter Overview

This chapter presents the RL-based sequencing algorithm developed for providing a sub-task execution sequence.

5.1 Introduction

In real-time control of dynamic manufacturing systems, scheduling decisions are usually implemented through a policy that assigns priorities to the jobs waiting at a machine - the job with the highest priority is selected for imminent processing [Park *et al.*, 1997]. These problems are also referred to as job sequencing problems, where decision makers must determine the production sequence of the jobs awaiting their next process in the machine queue. A common approach to address such problems is to adopt dispatching rules - priority rules used to determine the order in which the jobs are to be processed as soon as a machine becomes available. However, a dispatching rule often favors one performance measure at the expense of other measures [Wang and Usher, 2005]. The relative effectiveness of any rule depends upon the current state of the system. Therefore, there should be flexibility in selecting a dispatching rule employed in such a dynamic environment.

[Park *et al.*, 1997] present an adaptive scheduling policy for dynamic manufacturing systems that tailors the dispatching rule to be used at a given point in time according to the state of the system. The rule selection logic is embedded in a decision tree that is generated by applying an inductive learning algorithm on a set of training examples. Experimental studies indicated the superiority of the suggested approach over the alternative approach involving the repeated application of a single dispatching rule.

However, in order to implement dispatching rules, a complete system model is required. Furthermore, for dynamically assigning dispatching rules there is a need to continually compute system parameters, such as flow allowance (the lead time permitted to any job), system utilization, relative machine workloads (points system bottlenecks), and machine homogeneity.

RL provides a relatively easy way to model scheduling problems. With RL there is no need for predefining desirable or undesirable intermediate states, which is very hard to do in such problems. All that must be done is to construct a fairly simple reward policy (*e.g.*, higher reward for shorter completion times) and the algorithm will supply a solution.

5.2 The SRL Algorithm

The sequencing RL algorithm (Fig. 5.1) is developed to solve the job sequencing problem. The objective is to sequence the jobs so as to minimize the makespan (total completion time) of the desired task.

Problem states, denoted as $s_t \in S$, are defined as system's overall state at time step *t*. Problem actions, $a_t \in A$, which convert the system from state to state, are defined in accordance to the specific problem. A value *Q*, associated with a state-action pair, (s_t,a_t), represents how "good" it is to perform action a_t when the system is in state s_t.

A *learning episode* is defined as a finite sequence of time steps, during which the system traverses from a starting state to a goal state, according to the agent's actions. A *learning session* is a series of *N* learning episodes.

Action selection in the SRL algorithm is performed using an adaptive *ε-greedy* method [Sutton and Barto, 1998], in which the agent behaves greedily by selecting an action according to Max *Q* most of the time with probability *1- ε*, but with a small probability *ε*, selects a random action instead. The probability *ε* starts with a relatively high value, and is adaptively reduced over time using exponential decay as a function of the number of episodes, *n*, as shown in (5.1). The rate of decay is controlled by *β*, a positive parameter specifying how fast *ε* will decrease towards zero.

$$\varepsilon = \frac{1}{n^{\beta}} \qquad (5.1)$$

At the beginning of the learning session, when the agent has not gathered much information, a high value of *ε* encourages exploration of the state-space by allowing more random actions. As the learning session progresses, the probability *ε* decreases, reducing the number of random actions, and allowing the agent to exploit the information already gathered and perform better.

To solve the scheduling problem there is a need to consider the sequence of steps as a whole and not only step by step. The reason is that the policy's performance can only be evaluated and rewarded at the end of the learning episode, when the task completion time is known. This is also the reason why standard RL algorithms, such as *Q*-learning, updating value estimates on a step-to-step basis and assigning predefined constant rewards, cannot be applied here. Hence, the algorithm includes two updating methods.

The **first** method is performed after each step, similar to the SARSA[1] control algorithm [Sutton and Barto, 1998]. The difference is that because of the characteristics of the scheduling problem, there is no way of evaluating whether a certain action taken is good or not, from the narrow

[1] The SARSA algorithm is described in Section 2.2.

perspective of a single step. Therefore, it is impossible to assign an effective instantaneous reward. The one step update of the state-action values is described in (5.2).

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[\gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{5.2}$$

Where $Q(s_t, a_t)$ is the value of performing action a_t when the system is in state s_t, α is the learning rate which controls how much weight is given to the new *Q* estimate, as opposed to the old one, and *γ* is the discount rate, determining the present value of future rewards.

The **second** update method is performed at the end of the learning episode *n*, when it is possible to evaluate the performance of the policy used. At this stage there is an update of all the steps in the episode sequence, by multiplying their *Q* values with a reward factor indicating how good the last episode was. Two reward factor calculations are suggested, both assigning higher values to lower task completion times. A *type A* reward factor receives a value of *1* if task completion time T_n, achieved at episode *n*, was less than or equal to the best time found so far. Otherwise, the factor will be smaller than *1*, proportional to the difference between the current episode's time and the best time achieved so far. This way, *Q* values of states visited during a "good" sequence remain the same, while *Q* values of states included in worse sequences are decreased. The *type A* reward factor is calculated according to (5.3).

$$R_n = \begin{cases} 1, & if\ T_n \le T^*_{n-1} \\ 1/(T_n - T^*_{n-1} + a) + b, & if\ T_n > T^*_{n-1} \end{cases}$$
$$Where\ T^*_{n-1} = \underset{i=0,\dots,n-1}{Min}\{T_i\} \tag{5.3}$$

Where R_n is the reward factor at episode *n*, T_n is the completion time achieved at the current episode *n*, and T^*_{n-1} is the best time achieved up to episode *n-1*. The parameters *a* and *b* are used to adjust the reward factor to achieve the desired values.

The *type B* reward factor, described in (5.4), is simply set in inverse proportion to the completion time T_n, achieving the desired effect of a higher reward factor for lower completion times.

$$R_n = \frac{1}{T_n} \tag{5.4}$$

Here T_n is the time achieved at the current episode *n* and R_n is the reward factor.

```
Initialize $Q(s,a)=1$ for a learning session
**Repeat** (for each learning episode *n*):
        Initialize state $s_t$ as starting state, pick initial action $a_t$
        **Repeat** (for each step *t* of episode):
                Take action $a_t$, observe next state $s_{t+1}$
                Choose $a_{t+1}$ for $s_{t+1}$ using a certain action selection rule (*e.g.*, *ε-greedy*)
                $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[\gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
                $s_t \leftarrow s_{t+1}$ ;   $a_t \leftarrow a_{t+1}$
        **Until** $s_t$ is terminal (reached the goal state)

        Calculate $R_n$ (*type A* or *type B*)

        **For all** $(s,a)$ visited during the episode:
                $Q(s,a) \leftarrow R_n * Q(s,a)$
        **End**
**Until** *n* = *N* (reached the desired number of learning episodes)
```

Fig. 5.1 Pseudo-code of the SRL algorithm

6. Cognitive Collaborative Reinforcement Learning (CCRL)

Chapter Overview

This chapter presents the cognitive collaborative algorithm allowing the introduction of an advisor to the learning process.

6.1 Introduction

As reviewed in the Introduction chapter, a central issue in human-robot collaboration is adjustable autonomy, the determination of **whether** and **when** human intervention is required. Collaborative *Q*-learning, *CQ(λ)* [Kartoun, 2006; Kartoun, 2008], addresses the issue of accelerated learning through the concept of human-robot collaboration and adjustable autonomy. The *CQ(λ)* algorithm integrates the experience of the learning agent with the knowledge of a human operator.

A similar approach is that of the *Introspection Approach* (IA) [Clouse, 1996]. The IA is a method by which the learning agent determines when it requires aid from a training agent. The main challenge in such a method is designing a mechanism for deciding when the learner should ask for advice. The goal here is to maximize the impact of the advisor's instruction, so that the learner develops its decision policy quickly and correctly, with as little training as possible [Clouse, 1996].

When addressing the question of when the agent should ask for advice, Clouse relies on his informal perception of when human learners require instructions, noting that humans seek help when they are confused or otherwise unable to decide upon a course of action. To implement IA, he developed a test to determine whether the learner is unsure of its choices, indicating the need for help in novel situations. Clouse notes that "when discussing automated learners, it is fairly easy to specify exactly when they are unsure: one has access to the decision policy and the evaluations on which the decision is based." The test examines the two extreme values of possible actions (*Q(s,a)*): if they are close to each other it implies that the learner has not experienced this state often enough to produce a clear choice, thus should ask for advice. Guidance received via IA is shown to be more informative than random guidance, thus making better use of the training agent.

In this thesis the question is addressed by employing a result-oriented approach. We argue that another case in which humans seek aid is when they come to the understanding that their performance is not improving fast enough, or in other words, that their improvement rate is not sufficient. Thus, the decision whether assistance is required relies on the objective outcome of the learning, evaluated according to certain acceptable performance thresholds.

The following sections describe a cognitive collaborative reinforcement learning algorithm (CCRL) which extends the concept of *CQ(λ)* to include the cognitive capabilities of **performance assessment** and **advice assessment.**

6.2 The CCRL Algorithm

The cognitive collaborative reinforcement learning algorithm (CCRL), addresses the questions of **whether** and **when** the robot should solicit advice by endowing the robot with two human-like cognitive abilities: **The ability to assess its performance** and request advice when it is not sufficient, and **the ability to assess the value of the offered advice** and decide whether to continue asking for it or stop the requests and switch to fully autonomous learning.

The robot applies a result-oriented approach, seeking aid when it comes to the understanding that its performance is not sufficient. Furthermore, the robot is given the ability to judge the worth of the advice it receives. This self-awareness is achieved by performing self tests designed to evaluate its learning performance according to acceptable performance thresholds.

The CCRL algorithm, as well as the IA approach, uses the basic model of a RL learner incorporating advisor-suggested actions online. Upon receipt of an action from the advisor, the robot executes the action as if it had chosen the action with its own policy. Thus, the basic RL algorithm used (*e.g.*, *Q*-learning) does not need to be modified to handle the advisors actions. The adjustable autonomy method includes two learning modes, supervised and autonomous, following the model introduced in [Kartoun, 2006].

6.3 Collaborative Learning

Consider a collaborative learning model in which the system can be in one of two modes: (i) autonomous (unsupervised learning) and (ii) guided (supervised by an outside intelligent agent). In the autonomous mode the robot decides which actions to take according to feedback from the environment (reinforcements), using a certain action selection method (*e.g.*, *ε-greedy, softmax*). It is in this mode that the collaborative feature is added in which the learner can switch into the guided supervised mode and back. In the guided supervised mode a guidance agent such as a human advisor suggests actions. This knowledge is incorporated into the learning function if it is deemed worthy. The learning itself can be done using any RL algorithm (*e.g.*, SARSA, *Q*-learning). The advice from an outside guidance agent is unnecessary as long as the robot learns policies and adapts to new states while showing improvement. Only when the robot senses its performance is not improving at the desired rate, is the advisor solicited to intervene and suggest actions. The robot then performs the suggested action, and updates its *Q* values according to the action taken as though it had chosen the action itself.

The robot is endowed with two cognitive capabilities that allow it to decide **whether** and **when** to switch between the autonomous and guided modes. These decisions are triggered when two performance thresholds are exceeded: Λ, used to determine when to ask for advice, and Ω, used to determine whether the advice is acceptable or not. The robot performs self tests (incorporating these thresholds) based on the ability to assess its own learning performance, as detailed in the following sections.

6.4 Self-Performance Assessment Capability[1]

The robot must determine whether its performance is sufficient in order to decide when to switch between the two learning modes. Since the optimal solution (minimal number of steps to reach the goal in this case) is unknown *a priori*, the threshold for triggering a request for advice cannot be set as a constant measure, above which advisor assistance will be desired. Furthermore, even if we had some idea of the scale of the optimal solution, the robot cannot be expected to achieve it immediately, since the learning process is gradual. What can be expected from the robot is to continuously improve its performance. Therefore, the threshold used is not a constant value it has to reach, but an improvement rate. The way for the robot to sense it is not learning fast enough is by comparing its current performance with past performance. The robot wishes to achieve a certain improvement rate during the learning session[2]. When it does not achieve that rate, a request for advice is triggered. The improvement rate is defined as a ratio between moving averages of the number of steps of previous episodes, as described in (6.1).

$$IR = \frac{T_p - T_c}{T_p}$$

$$T_c = \frac{\sum_{i=n-K}^{n-1} (T_i)}{K}\ ; \ T_p = \frac{\sum_{i=n-2K}^{n-K+1} (T_i)}{K} \tag{6.1}$$

Where n is the current episode, T_i is the performance at episode i (number of steps to reach the goal in this case) and IR is the actual performance improvement rate, comparing the previous average number of steps T_p (average over previous K episodes, $n\text{-}2K$ to $n\text{-}K\text{+}1$) and the current average T_c (average over the most recent K episodes, $n\text{-}K$ to $n\text{-}1$). If the current average is smaller than the previous one (less steps required to reach the goal – better performance) IR will be positive.

6.5 Advice Request Test

The CCRL advice request self test compares IR with the threshold Λ, as shown in (6.2).

[1] The explanations and calculations described here refer to a problem in which the objective is to minimize the number of steps to reach the goal state. In maximization problems the formulations would be slightly modified.

[2] A learning episode is defined here as a finite sequence of time steps, during which the agent traverses from the starting state to the goal state. A learning session is a series of N learning episodes.

$$\begin{array}{l}\text{If } IR < \Lambda \text{, then } \textit{request advice} \\ \text{Else } \textit{learn autonomously}\end{array} \tag{6.2}$$

Here Λ is a predefined collaboration threshold, representing the desired improvement in performance. Before each learning episode begins, the actual improvement rate is compared to the threshold. If IR > Λ, meaning the actual rate is better then the desired, the robot will continue to learn autonomously and will not solicit advice. If $IR < \Lambda$, the improvement rate is not sufficient, and advisor assistance will be requested. When requested, the advisor will assist during the entire episode.

When the robot converges to the optimum, obviously there will not be any improvement in the performance, and advisor assistance will be asked recurrently without need. This problem is solved by applying the following rule: If after *2K* episodes the robot produces the same result, it assumes it has reached the optimum and stops asking for aid. Even if the optimum found was a local one, if *2K* episodes using human assistance did not help the robot escape it, then there is no sense in continuing the requests.

Another rule is that the robot can start asking for advice only after a certain number of episodes *X*. This is done in order to allow the robot to operate autonomously, since at the beginning of the session there is a lot of exploration, and it is not expected to show improvement.

It is important to note, that in the CCRL algorithm switching between the autonomous mode (action selection using the *Q* table of the RL algorithm) and the guided mode (action selection by the advisor), occurs only at the end of an episode, whereas in the IA approach this switch can take place at any step within an episode.

6.6 Advice Assessment Capability

Until here the assumption was that the advisor provides good instructions, but what happens if the advice is bad? Wrong advice will not promote learning, and might even cause deterioration in performance. By endowing the robot with the capability to assess the value of the advice, such situations may be avoided. The robot judges the advisor's suggestions by comparing its performance when using the advisor's aid with past performance. If assistance does not improve the performance, the robot learns to stop asking for it. The number of steps achieved at episodes performed with advisor assistance is compared to the average number of steps over the *K* episodes previous to the assisted episodes. When the number of steps to reach the goal in the assisted episodes, T_a, is higher (worse) than the average, it insinuates that advisor instructions are worthless and maybe even misleading. The number of times in which the episode with advisor assistance produced worse results than the average, denoted as *ML*, is counted as shown in (6.3)

$$\text{If } T_a(n) > \frac{\sum_{i=n-K}^{n-1} (T_i)}{K}, \text{ then } ML = ML + 1 \tag{6.3}$$

6.7 Advice Rejection Test

When *ML* exceeds a predefined threshold, meaning the human misled the robot too many times, the robot refuses the advice, and switches to a fully autonomous learning mode until the end of the session. The CCRL advice rejection test is elaborated in (6.4).

$$\begin{aligned}&\text{If } ML > \Omega, \text{ then } \textit{refuse advice}\\&\text{Else } \textit{continue requesting advice when } IR < \Lambda\end{aligned} \tag{6.4}$$

Here *ML* is the number of occasions in which the human misled the robot causing the episode with advisor assistance to achieve worse results than the average results of the *K* previous episodes, and Ω is a predefined advice refusal threshold for such occasions, above which collaboration is stopped.

When the human has poor expertise, the episodes performed with his assistance will result in decreased performance, *ML* will rapidly rise and exceed Ω, and the robot will stop asking for advisor aid, as it should. In this final structure of the algorithm (Fig 6.1), collaboration is defined by the two threshold parameters, Λ and Ω, determining the desired improvement rate and the acceptable number of human misleads, respectively. A pseudo-code of the algorithm is displayed in Fig. 6.2. An example for the collaboration mode switching during a learning session is presented in Fig 9.5.

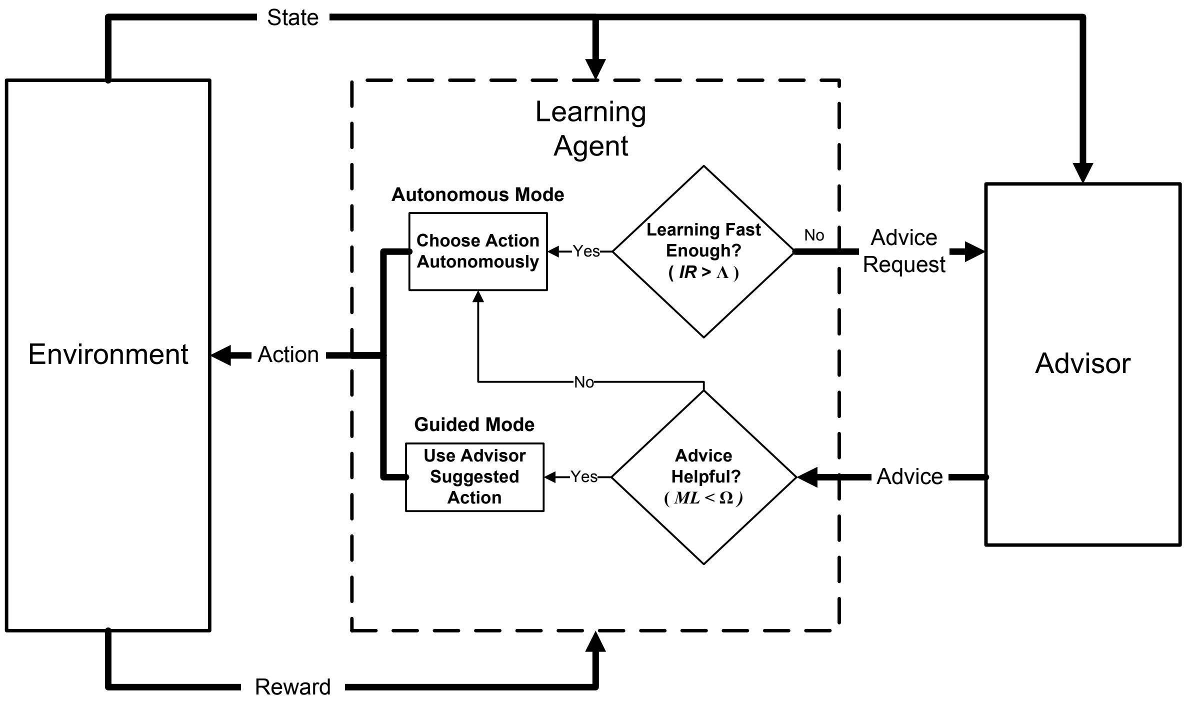

State
Environment
Action
Reward
Learning Agent
Autonomous Mode
Choose Action Autonomously
Yes
Learning Fast Enough? (IR > Λ)
No
Advice Request
Advisor
No
Guided Mode
Use Advisor Suggested Action
Yes
Advice Helpful? (ML < Ω)
Advice

Fig. 6.1 Scheme of the CCRL algorithm

```
Initialize the basic RL algorithm (e.g., Q-learning, SARSA) for the learning session
Initialize $ML = 0$
Set desired $\Lambda$, $\Omega$ (collaboration thresholds)
Repeat (for each learning episode $n$):
        Initialize state $s_t$ as starting state, pick initial action $a_t$
        If $n < X$ or $ML > \Omega$
                Use Action Selection I (learn autonomously)
        Else
                If $IR < \Lambda$
                        Use Action Selection II (request advice)
                Else
                        Use Action Selection I (learn autonomously)
                End
        End
        Repeat (for each step $t$ of episode):
                Take action $a_t$, observe reward $r_t$ and next state $s_{t+1}$
                Action Selection I: Choose $a_{t+1}$ for $s_{t+1}$ using a certain action selection rule (e.g., ε-greedy)
                Action Selection II: Advisor suggests action $a_{t+1}$
                Update basic algorithm's parameters according to $a_t, r_t, s_{t+1}$ and $a_{t+1}$
                $s_t \leftarrow s_{t+1}$ ;    $a_t \leftarrow a_{t+1}$
        Until $s_t$ is terminal (reached the goal state)
        $IR = (T_p - T_c) / T_p$
        If $T_a(n) > \sum_{i=n-K}^{n-1} (T_i) / K$
                $ML \leftarrow ML + 1$
        End
Until $n = N$ (reached the desired number of learning episodes)
```

Fig. 6.2 Pseudo-code of the CCRL algorithm

7. Sub-task Sequencing for a Toast Making System

Chapter Overview

The applicability of the CHRL framework is demonstrated using an automated toast making system, presenting both high and low level learning tasks for its operation. This chapter describes the high level learning task of sequencing toast transitions through the system's stations.

7.1 Introduction

A test-bed application, robotic toast making system, was developed to demonstrate the applicability of the CHRL framework[1]. Toast making is a complex multi-goal task since it is composed of many sub-tasks (such as grasping a toast, inserting it to the toaster or applying butter over it), each having its own goal state. The system includes six stations (two of which are processing stations) and a transfer "agent", a fixed-arm robot, advancing the toasts through the system, one toast at a time. In lieu of fixed inter-process job transfers, the robot allows the flexibility of job movements at any point in time and to any location. The complex task of toast making is addressed here by decomposition into a two-level learning hierarchy to be solved by CHRL. The high-level consists of learning the desired sequence of execution of basic sub-tasks and the low-level consists of learning how to perform the required sub-tasks. In this application the SRL algorithm is used to generate a sequence of toast transitions through the system stations, to achieve completion of toast making in minimum time, and the CCRL algorithm is employed for learning the execution of an exemplary sub-task, the insertion of a bread-slice to the toaster (See Chapter 8).

7.2 High Level Learning Task – Toast Transition Sequencing

The high level sequencing problem presented by the toast making system is addressed by the SRL algorithm. The algorithm's performance is evaluated by comparison to a Monte-Carlo method and a random search through extensive experimentation. The generation of the desired sequence is performed off-line using an event-based MATLAB simulation. This solution is implemented on-line using a Motoman fixed-arm robot operating on a toast making system in an environment consisting of a cardboard mockup of toast objects and processing units.

7.3 Experimental Setup and Method of Operation

The system consists of six stations (Fig. 7.1): 1- plate for raw slices of bread, 2- a buffer in front of a toaster, 3- toaster (with a capacity of one slice), 4- a buffer in front of a butter applier, 5- butter

[1] A detailed description of the system and its operation can be found in Appendix I.

applier (butter can be applied to only one slice at a time), and 6- finished toasts plate. Each toast has to go through all of the stations in the specified order, except for the buffer stations (2 and 4) which are used only when needed (the buffers allow bread-slice advancement while the machines are occupied, thus may save time in later loading of the machines, since the slices would be located closer). The toast transfer “agent” is a fixed-arm six degree of freedom Motoman UP-6 robot with a pneumatic gripper (Fig. 7.2), which advances the toasts through the system, one toast at a time.

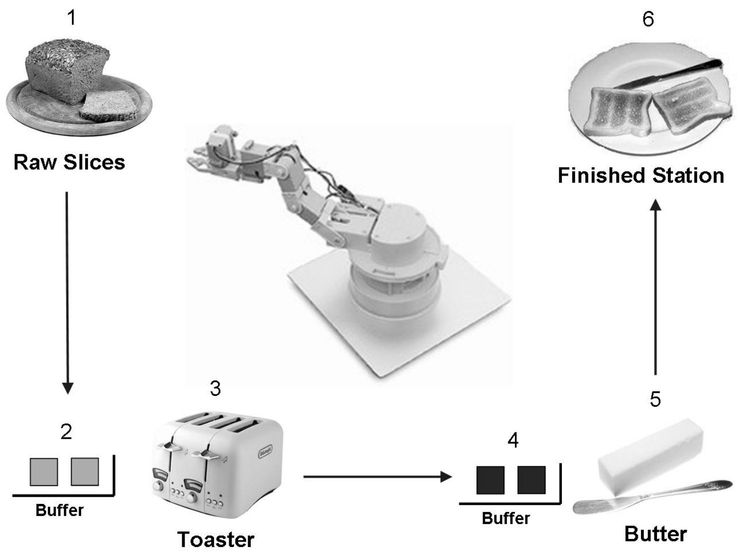


Fig. 7.1 General scheme of the toast making system

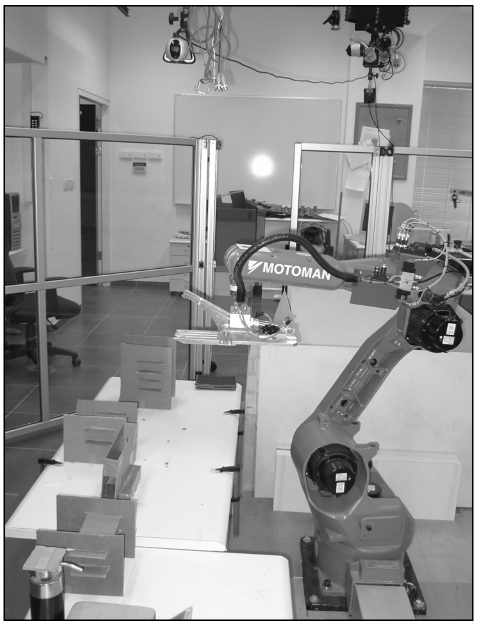


Fig. 7.2 Motoman robot and cardboard mockup of the system stations

Three instances of the robot sequencing problem are examined. We designate these as cases I, II and III. Each is described by specific robot transition and machine processing times (Tables 7.1 and

7.2). Case I is adjusted to suit the transition times of the real robot[1]. Cases II and III have increased processing times and modified robot transition times.

Table 7.1 System stations and machine processing times

No.	Station	Processing times (sec.)
1	Raw slices plate	-
2	Toaster buffer	-
3	Toaster	case I - 60; case II - 90; case III - 120
4	Butter applier buffer	-
5	Butter applier	case I - 60; case II - 90; case III - 120
6	Finished toasts plate	-

Table 7.2 Robot transition times (sec.)

Case I

From Station \ To Station	1	2	3	4	5	6
1	X	24	36	X	X	X
2	21	X	24	16	15	X
3	22	X	X	22	28	X
4	24	26	19	X	19	X
5	26	24	20	X	X	12
6	25	23	19	21	X	X

Case II

From Station \ To Station	1	2	3	4	5	6
1	X	20	30	X	X	X
2	30	X	30	20	30	X
3	30	X	X	30	50	X
4	30	20	30	X	40	X
5	30	20	30	X	X	20
6	30	20	20	30	X	X

Case III

From Station \ To Station	1	2	3	4	5	6
1	X	50	70	X	X	X
2	40	X	40	40	50	X
3	45	X	X	30	45	X
4	50	40	30	X	35	X
5	55	50	45	X	X	30
6	60	55	50	45	X	X

* Transition combinations marked with X are inapplicable.

7.4 Task Definition

The objective of the system is to produce butter covered toasts from a fixed number of raw bread slices as fast as possible, that is to find a sequence of robot (toast) moves that minimizes total job completion time. When addressing the high level learning task, it is assumed that low-level task times, achieved via optimal robot motions, are known, such that only the high-level sequencing task must be solved.

When defining the sequencing problem presented by the system in the conventional way, it can be regarded as a flow-shop scheduling problem, which requires sequencing of jobs (toasts) with different processing times through a set of machines. The difference here is that the jobs have identical processing times, and that they are not automatically transferred from station to station. Other unique characteristics are: (i) the use of dynamic buffers (unlimited) for the processing stations, which are not a part of the technological path, and are used only when a station is busy, and (ii) the fact that robot's arm movements while empty must be considered (duration depends on the source and target locations).

[1] The robot's transition times where measured in the real experimental setup.

Due to the system's unique characteristics, it is easier to approach the problem from a different point of view, as described in the Methodology chapter: the problem can be viewed as a job sequencing problem, in which the robot is the limited resource (the "machine"), and the toast transition tasks are the "jobs" waiting in its queue, requiring a different "process time" (robot transition time). Since the transfer agent has limited capacity (the robot can move only one bread slice at a time) there is a problem of how to schedule this "limited resource".

7.5 Implementation of the SRL Algorithm

To solve the sequencing problem using the SRL algorithm, it is formulated as a RL problem. The system's overall state at time step *t*, denoted as $s_t \in S$, is defined by the current locations of the toasts. For a *K* toast problem, the state will be represented as (L_1, L_2, L_3, …, L_k), where L_i is the location of the *i*-th toast ($L_i \in 1...6$). In a three toast problem for example, states can be: (1,1,1), (3,1,1), (3,2,1), (5,2,1) etc. Not all location combinations are feasible, due to system's characteristics: state (3,3,1) for example is not feasible, since the toaster (station 3) can process only one bread slice at a time.

A solution is a specific sequence of toast transfers: "move toast 1 to its next station, move toast 3 to its next station, move toast 1 to its next station, move toast 2…;" presented as a vector: [1,3,1,2,…]. Different sequences might be in different lengths, since some may use the dynamic buffer stations (2 and 4) and some may not. The goal state of the learning task is state (6,6,6), where all the toasts have reached the finished plate. In this context, it is important to understand the distinction between the goal state of the toasting system, which is, as noted, (6,6,6), and the goal of the learning task, which is to find the sequence of steps that would lead from state (1,1,1) to state (6,6,6) in minimum time[1]. The sequence's execution time is composed of transition, processing and waiting times.

An action at step *t* is denoted as $a_t \in A(s_t)$, where *A* is the action space of all possible actions (the action space is state dependent). The execution of an action constitutes the advancement of a toast to its next station in the processing sequence, causing the system to arrive at a new state. For example, at state (3,2,1) there are two possible actions: (i) advance toast number one from station 3 to station 5, arriving to state (5,2,1), and (ii) advance toast number three from station 1 to station 2, arriving to state (3,2,2). Toast number two can not be moved to station 3 (the toaster) because the station is still loaded with toast number one.

Rewards are assigned according to the performance, as explained in Section 5.2. A learning episode starts from the state (1,1,1) where all the slices lie on the raw slice plate, and ends in state

[1] An example for a state sequence is shown in Fig 7.4.

(6,6,6) when the last slice arrives to the serving plate, toasted and covered with butter. A step is the transition from one system state to another.

As a side note, it is important to understand the difference between the problem's state definition here and the definitions in other tasks. In path planning tasks for example, the state is defined as the agent's location, and during the learning it traverses from state to state. In our case, the state is defined as the system's status, and the agent's actions shift the system from state to state. Hence, not the location of the agent (robot) itself counts, but the influence of its actions on the system's state. Therefore, when at a certain state, the real decision considered is which next system state is desirable, not which robot's location. As an outcome of the final robot sequence however, one can obtain the location-time trace of the robot's activities.

Learning the high-level sequencing task is performed off-line using an event-based MATLAB simulation. On-line fixed-arm robot motions are performed only after the simulation supplies the desired sequence. The use of simulation allows fast learning, since real robot manipulations are extremely time-consuming. Furthermore, the simulation constitutes a convenient and powerful tool for analyzing the performance of the SRL algorithm, by conducting various virtual experiments off-line. The model receives robot transition times and machine processing times as input data. The system allows the user to input the number of toasts in a session.

7.6 Analysis

The SRL performance is tested with two different size problems, using demands of 3 and 4 toasts. The 3-toast problem allows better understanding of the algorithm's characteristics, and its optimal solution can be found in reasonable time and compared to the solution reached by the algorithm. The 4-toast problem is closer to real-world problems, having a larger state-space. Based on a MATLAB simulation, various simulated experiments are conducted, to examine the influence of the adaptive *ε-greedy* action selection parameter *β* and of the reward types on the SRL performance, and to evaluate the performance by comparison to the Monte-Carlo method [Sutton and Barto, 1998] and to a random search algorithm.

Monte-Carlo (MC) methods are ways of solving the RL problem based on averaging sample returns [Sutton and Barto, 1998]. It is only upon the completion of an episode that value estimates and policies are changed, thus incremental in an episode-by-episode sense, but not in a step-by-step sense (this is the reason MC methods can be applied for scheduling problems). Here the Q values are simply the average rewards received after visits to the states during the episodes. The reward for a specific episode is set to be $1/T_n$, assigning a higher reward for lower times, and is accumulated and averaged for each state-action pair encountered during the episode. Action selection here is performed using the same adaptive *ε-greedy* method of the SRL algorithm. When applying the random search method, actions are chosen with equal probability, using a uniform distribution. In all tests (Table 7.3), the RL parameters[1] are set as follows: $\alpha = 0.05$, $\gamma = 0.9$. These parameters were selected empirically.

The *first* test examines the SRL performance with various values of the parameter β, controlling the decay rate of ε (the probability of choosing random actions). The values are varied from 1.0 to 1.7. Each value is evaluated by performing 100 learning sessions with 200 learning episodes. This is done for all three cases (I, II and III).

In a *second* test the SRL algorithm is compared to the MC and random search methods for the three cases (I, II and III) and for both the 3 and 4-toast problems. Comparisons are made using various learning session lengths (N). Lengths are varied from 15 to 60 learning episodes in increments of 5 for the 3-toast problem, and 50 to 400 learning episodes in increments of 50 for the more complex 4-toast problem (requiring more episodes in order to achieve good results). Each length is evaluated by performing 10 simulation replications, each containing 100 and 30 sessions of a certain length for the 3 and 4-toast problems, respectively[2]. Each session length is evaluated four times, twice for SRL (once using *type A* reward factor and once using *type B*) and once for each of the other methods (MC and random search). In terms of equation (5.1), for the random search $\beta = 0$ ($\varepsilon = 1$ for all n) is used, while for the SRL and MC, using the adaptive *ε-greedy* method, a value of $\beta = 1$ is used for the 3-toast problem and $\beta = 0.5$ for the 4-toast problem. These values were selected since they produced the best performance in the first test.

A *third* test is conducted to examine the performance in a stochastic environment. Stochastic process times sampled from a Gaussian probability density function are used, with a mean equal to the constant process times, and a standard deviation of 10% of the mean[3]. Similar to the second test, various learning session lengths are examined, for cases I, II and III and for the 3 and 4 toast problems.

Performance is evaluated using the following measures:

1) *CE* (convergence episode) - Average number of learning episodes required to reach convergence.

2) *SP* (success percentage) - Average percentage of learning sessions reaching the optimal solution, for deterministic times.

[1] The RL parameters, α (learning rate) and γ (discount factor), are described in Section 5.2.
[2] 10 replications are performed for the statistical analysis of the performance measures. 30 sessions are performed for the 4-toasts problem, as opposed to 100 for the 3-toast problem, since the simulation running times are longer.
[3] The variations in robot transition times are negligible and therefore they are represented as deterministic.

3) *BR* (best result) - Average best result achieved in the learning session, for stochastic times, where an optimal solution cannot be defined due to the probabilistic feature.

Convergence (measure 1) means not only reaching the optimal solution, but eventually coming to the understanding it is the best solution possible, and continuing to produce it until the end of the learning session. The episode at which convergence occurs, *n**, is defined as the episode after which there is no change in the performance over the interval [*n**, *N*], meaning the algorithm supplied the same solution until the end of the session (consisting *N* episodes), as described in (7.1).

$$\begin{aligned} & n^* = Min\{n\} \; over \; all \; intervals \, [n, N] \\ & such \; that \; \Delta(n+j) = 0, \forall \; j = 0,\ldots,N-n \\ & n = 1,\ldots,N \\ & \Delta(n) = T_{n+1} - T_n \end{aligned} \tag{7.1}$$

Where $\Delta(n) = T_{n+1} - T_n$ is the change in performance at episode *n* and *N* is the number of episodes in the learning session.

Table 7.3 Summary of tests

No.	Examined Methods	Environments	Analyzed Parameters (Values)
1	SRL	3-toast problem Cases I, II and III Deterministic	Decay factor β (1.0 - 1.7)
2	SRL MC Random Search	3 and 4 toast problems Cases I, II and III Deterministic	Session length *N* (15 - 60; 50 - 400) Reward factor (*Types A* and *B*)
3	SRL MC Random Search	3 and 4 toast problems Cases I, II and III Stochastic	Session length *N* (15 - 60; 50 - 400) Reward factor (*Types A* and *B*)

7.7 Results and Discussion[1]

7.7.1 SRL Analysis

The best solutions produced by the SRL algorithm for the 3-toast problem are 513, 700 and 995 seconds for cases I, II and III respectively[2]. Fig. 7.3 shows the results for a learning session with case II times. It can be seen that the algorithm converges to a solution after 67 episodes.

[1] Additional results can be found in Appendix IV.

[2] The optimality of the solution was verified using the Branch and Bound general search technique as illustrated in Appendix IV.

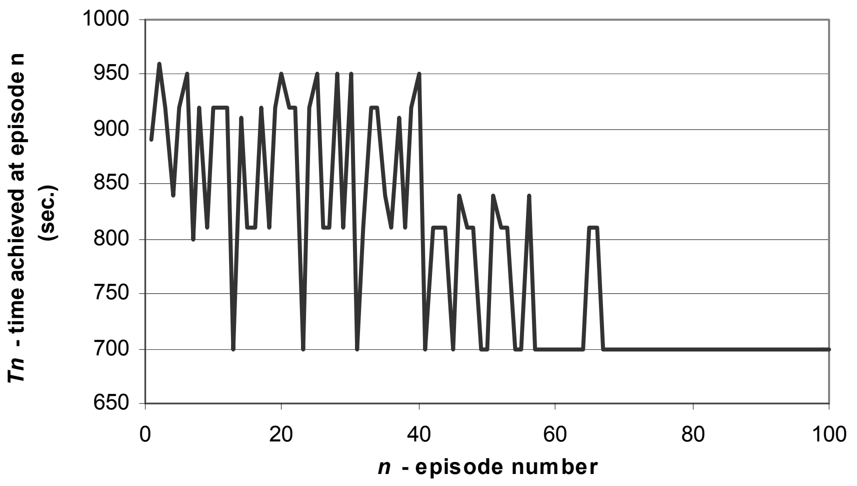


Fig. 7.3 Convergence to the scheduling problem's solution, case II

The solution achieved at this case is to schedule the toast advancements as follows (from left to right): [1,2,1,2,1,2,3,2,3,3]. Fig. 7.4 shows the toast locations, or in other words the system states during an episode, for the specified scheduling[1].

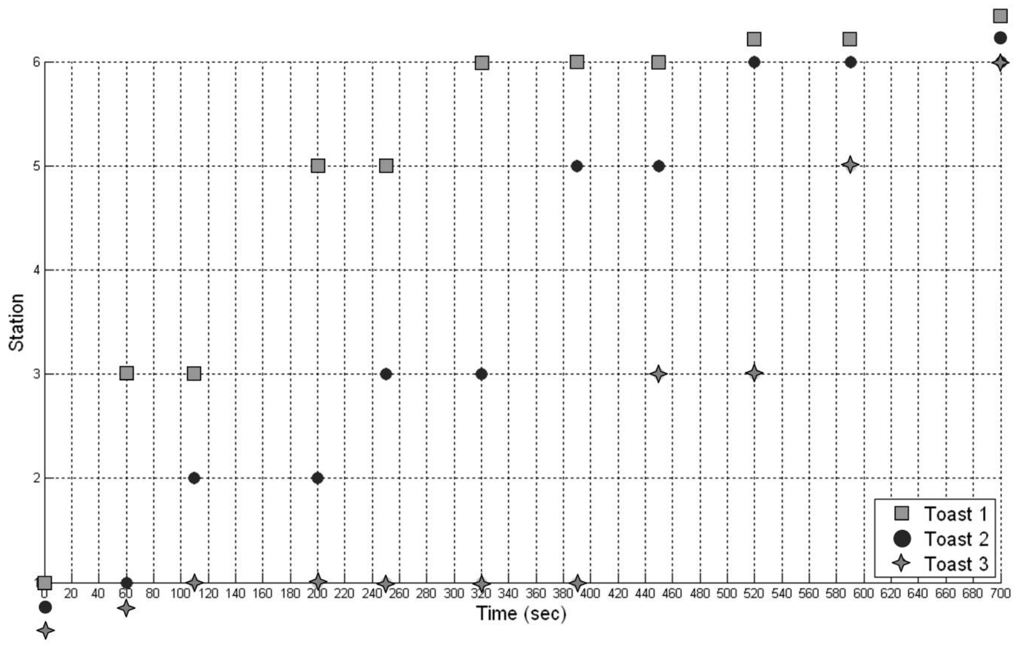


Fig. 7.4 Toast locations for the sequence found, case II

Examining the influence of the action selection parameters on the SRL performance (Fig. 7.5), reveals that when using a relatively small β ($\beta = 1$) the algorithm reaches the optimal solution with very high percentage of success, yet with the cost of a high number of episodes required for convergence. As β increases, the percentage of success in reaching the optimal solution decreases,

[1] A Gantt chart of the solution can be seen in Appendix IV.

but fewer episodes are required to achieve convergence. The reason for this behavior lies in the action selection method. As explained in Section 5.2, the algorithm uses an adaptive *ε-greedy* action selection method, allowing a balance between exploration and exploitation. *ε*, specifying the probability in which random actions are chosen, decreases as the episode number increases. At the limit $\varepsilon \rightarrow 0$ actions are always chosen greedily, meaning the best action (to the agent's knowledge) is always chosen. Fig. 7.6 shows the decrease in the probability of choosing a random action as a function of the episode number.

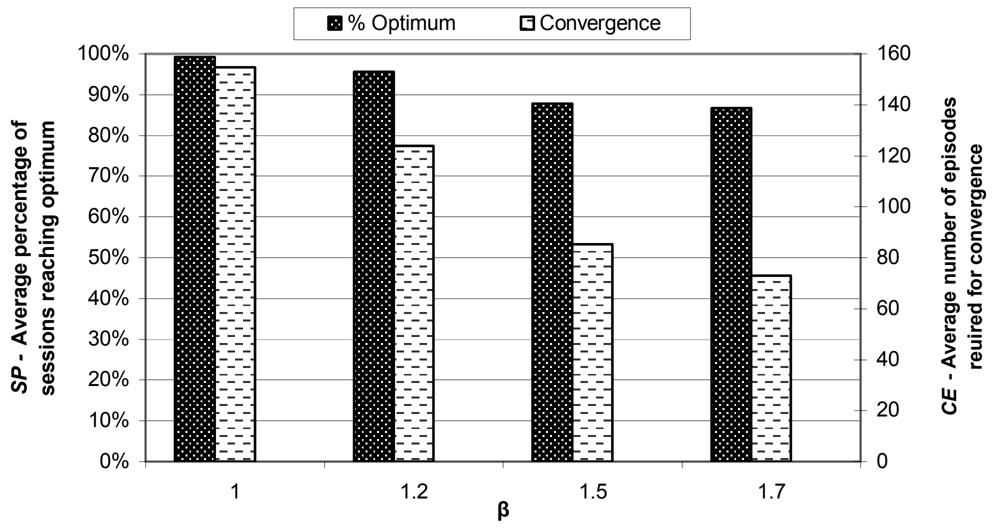


Fig. 7.5 Action selection analysis, 3-toast problem, *type A* reward factor, case II

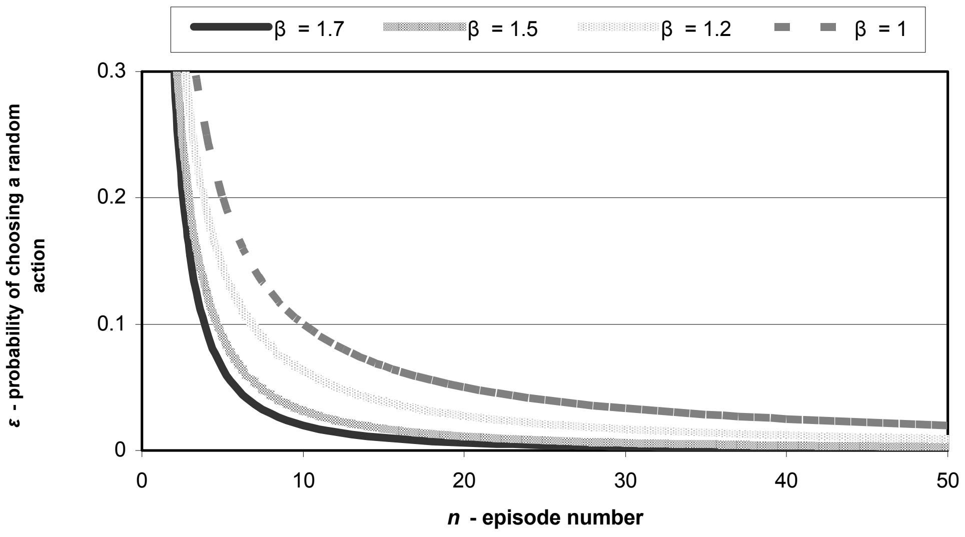


Fig. 7.6 Adaptive *ε-greedy* action selection

When using a small *β*, the probability of choosing a random action remains relatively high when the episode number rises. The action selection rule allows much exploration, resulting in a higher percentage of sessions reaching the optimal solution, but also a higher number of episodes required for convergence. When using larger values of *β*, the probability of choosing a random action decreases very fast, resulting in less exploration of the environment, and more exploitation of the information already gathered. This allows much faster convergence to a solution, but not necessarily the optimal one.

For all instances (cases I, II and III, 3 and 4 toast problems, deterministic and stochastic), the use of a *type A* reward factor achieves fast learning and good results in a low number of episodes. When using the *type B* reward factor, the algorithm requires more episodes in order to achieve good results, but ultimately it outperforms the *type A* results. Figs. 7.7 and 7.8 illustrate this trade-off for the deterministic 3 and 4 toast problems, respectively.

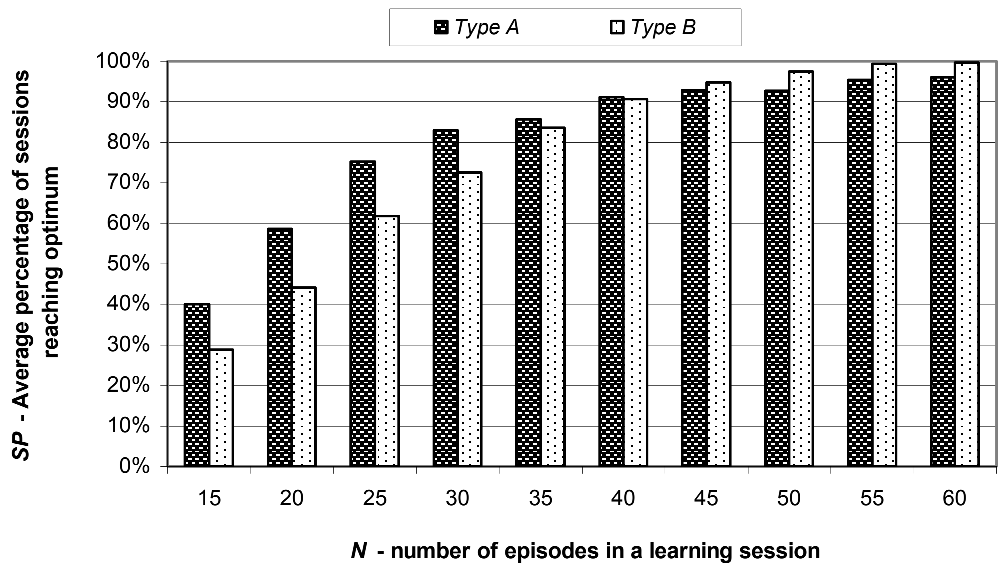


Fig. 7.7 Reward factor analysis, 3-toast problem, case I, deterministic environment

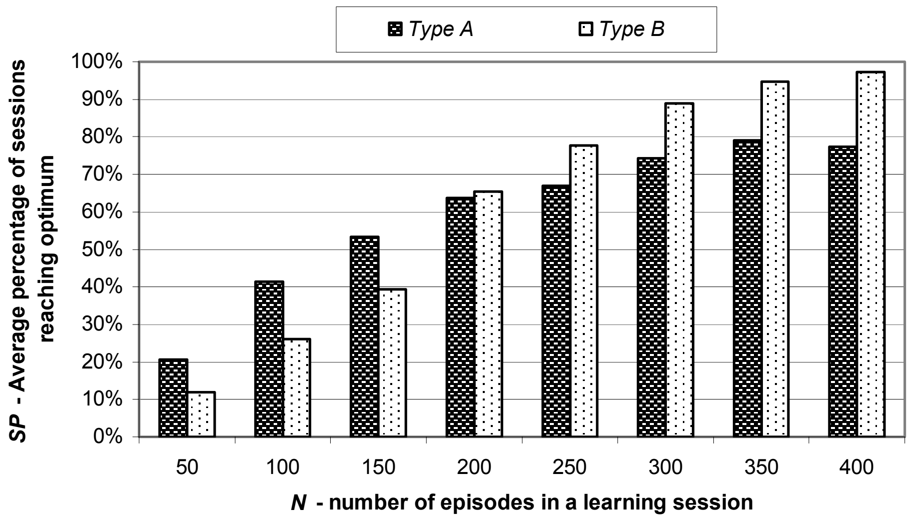


Fig. 7.8 Reward factor analysis, 4-toast problem, case I, deterministic environment

7.7.2 Comparative Analysis – 3 Toast problem

Comparison of the SRL algorithm to the MC method and random search in solving the deterministic 3-toast problem, demonstrates the superiority of the SRL algorithm over a wide range of learning conditions (25-60 episode sessions). The same results appear with all three cases: For 15 to 25 episode sessions, the agent does not achieve enough interaction with the environment therefore does not have sufficient information, and its *Q* values do not reflect the real state-action values. At this state, the algorithm acts *de facto* as a random search method, hence the performance is similar. From 25 to 60 episodes, the agent obtains sufficient information on the environment, allowing it to correctly update the *Q* values and reach optimality more times than the other methods. Fig. 7.9 shows the results for case III. For this case the SRL algorithm reaches up to 37% difference from the MC method and 22% difference from the random search method.

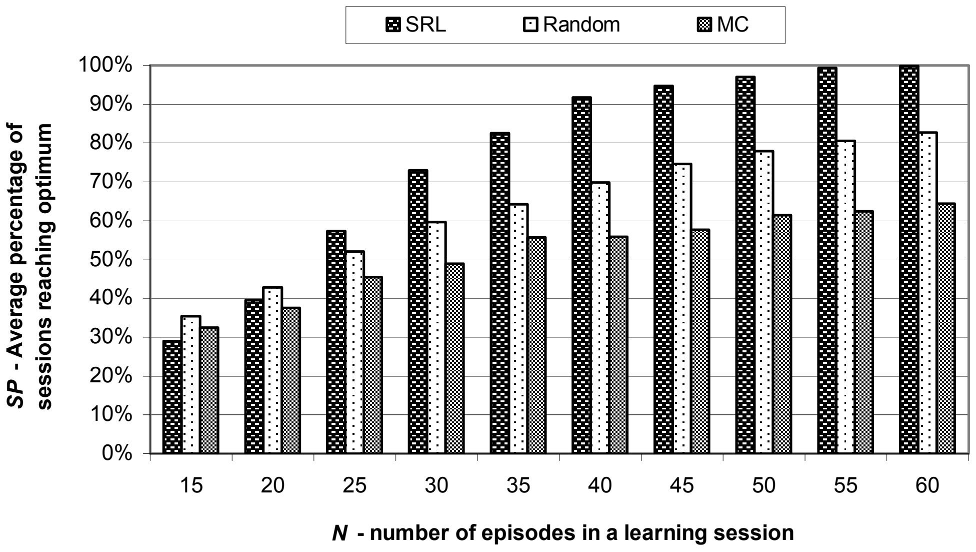


Fig. 7.9 Performance comparison – 3-toast problem, *type B* reward factor, case III, deterministic environment

The MC method acts similarly, but converges to worse solutions. For the random search on the other hand, more episodes implies a greater chance of reaching the optimal solution in one of them, hence its success percentage continues to rise along the full range.

In the stochastic environment (Fig. 7.10) the SRL algorithm again shows superiority, achieving better (lower) times over a wide range, outperforming both the MC method and the random search for all cases. Here, as in the deterministic environment, the random search produces better results than the MC method.

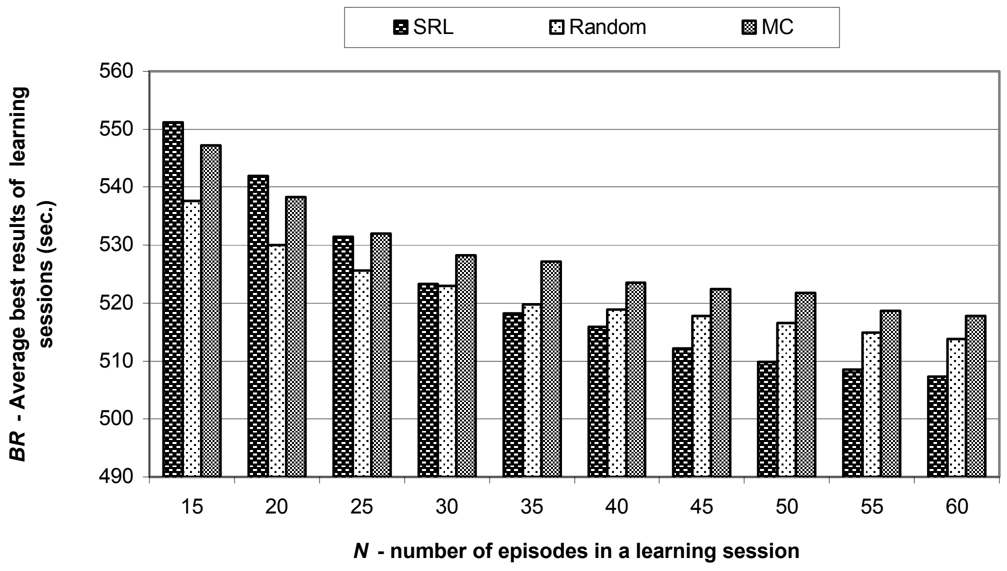


Fig. 7.10 Performance comparison – 3-toast problem, *type B* reward factor, case I, stochastic environment

7.7.3 *Comparative Analysis – 4 Toast Problem*

Comparison of performance for the deterministic 4-toast problem (Fig. 7.11) reveals the superiority of the SRL algorithm in reaching the best results of 663, 900 and 1,290 seconds (cases I, II and III respectively) in the higher range of session lengths (300-400). These results are consistent for all three cases. Due to its complexity, the 4-toast problem requires more learning episodes to reach the best solution, and the algorithm requires more experience to reach good results. Here, as opposed to the 3-toast problem, the MC method shows better results than the random search, which apparently requires even more episodes in order to deal with the increased complexity presented by the 4-toast problem.

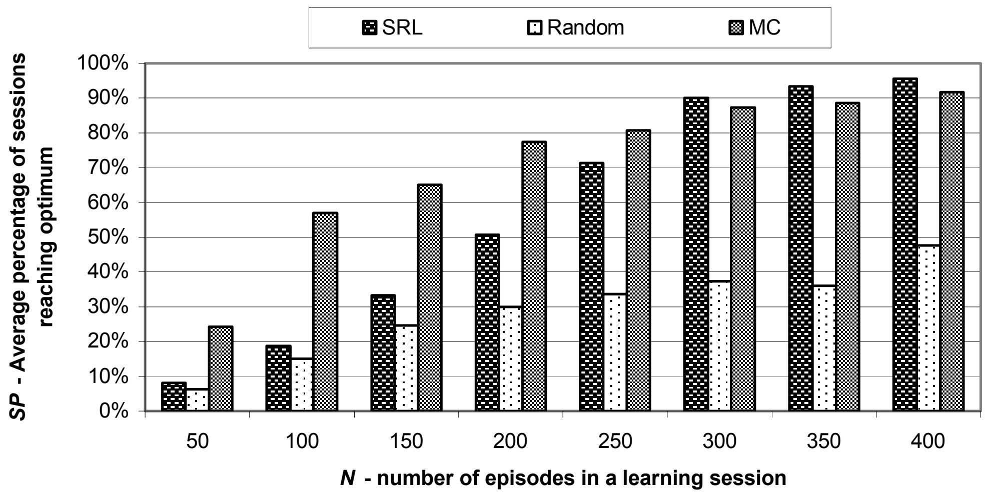


Fig. 7.11 Performance comparison – 4-toast problem, *type B* reward factor, case III, deterministic

In the stochastic environment (Fig. 7.12), as in the deterministic one, the SRL algorithm performs worse than the MC method when less experience is available (low number of episodes in a session), but matches the results when the sessions are longer. These results are again consistent for all cases.

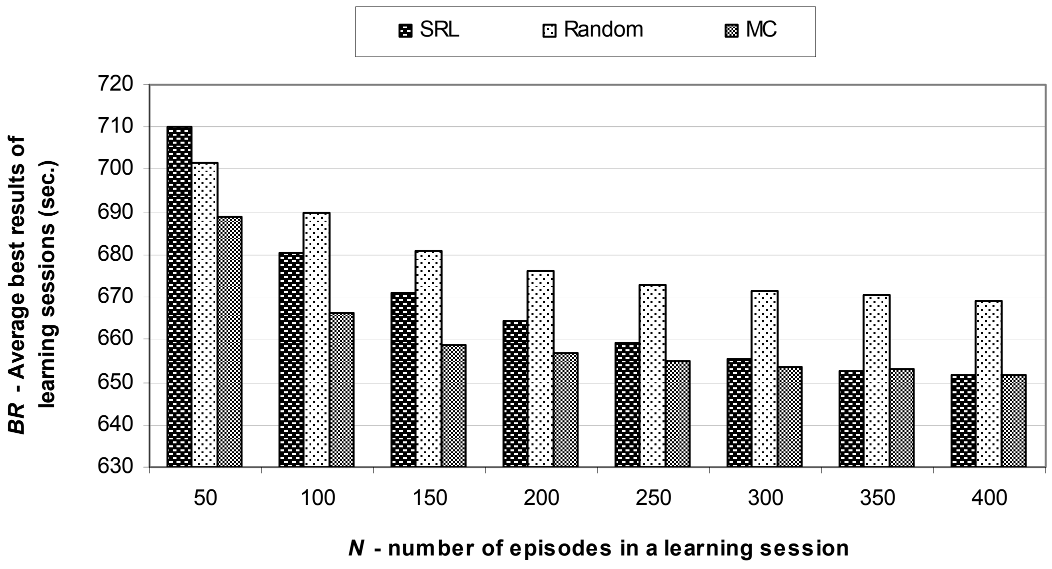


Fig. 7.12 Performance comparison – 4-toast problem, *type B* reward factor, case I, stochastic environment

7.7.4 Summary of Results

Tables 7.4 - 7.6 summarize the results for the three cases (I, II and III). The results presented for the SRL algorithm are for learning performed with the *type B* reward factor. Significant[1] best results are marked with gray shading (when two methods are significantly better than the third, both are marked). Note that for the deterministic scenarios the highest success percentage is the best, while for the stochastic scenarios the lowest average completion time is the best.

Table 7.4 Summary of case I results

Deterministic 3-toast problem

Method	Session length (number of episodes)									
	15	20	25	30	35	40	45	50	55	60
SRL	29%	44%	62%	73%	84%	91%	95%	97%	99%	100%
Random	41%	50%	62%	69%	75%	79%	81%	84%	87%	88%
MC	34%	44%	50%	50%	52%	58%	60%	62%	60%	65%

Stochastic 3-toast problem

Method	Session length (number of episodes)									
	15	20	25	30	35	40	45	50	55	60
SRL	551.15	541.89	531.44	523.29	518.18	515.92	512.17	509.87	508.49	507.26
Random	537.59	529.97	525.6	523	519.73	518.87	517.83	516.56	514.96	513.81
MC	547.24	538.3	531.97	528.24	527.11	523.49	522.46	521.74	518.67	517.74

[1] Significance is determined using one-way ANOVA analysis (F-test) and Tukey's HSD test demanding 95% confidence level, as explained in Appendix II.

Deterministic 4-toast problem

Method	Session length (number of episodes)							
	50	100	150	200	250	300	350	400
SRL	12%	26%	39%	65%	78%	89%	95%	97%
Random	16%	20%	31%	37%	47%	47%	58%	62%
MC	27%	54%	70%	79%	83%	86%	91%	93%

Stochastic 4-toast problem

Method	Session length (number of episodes)							
	50	100	150	200	250	300	350	400
SRL	709.92	680.54	670.98	664.47	659.07	655.27	652.81	651.8
Random	701.84	689.66	680.77	676.39	672.66	671.55	670.29	669.24
MC	688.79	666.23	658.66	656.99	654.99	653.68	653.22	651.83

Table 7.5 Summary of case II results

Deterministic 3-toast problem

Method	Session length (number of episodes)									
	15	20	25	30	35	40	45	50	55	60
SRL	32%	43%	59%	77%	86%	91%	94%	97%	100%	100%
Random	42%	49%	60%	66%	72%	78%	82%	86%	88%	90%
MC	34%	40%	44%	52%	54%	56%	54%	56%	61%	60%

Stochastic 3-toast problem

Method	Session length (number of episodes)									
	15	20	25	30	35	40	45	50	55	60
SRL	760.31	745.26	730.44	714.94	706.93	701.6	698.7	695.7	693.1	690.61
Random	732.61	720.76	715.33	710.88	709.21	707.12	704.75	702.77	701.5	700.97
MC	746.27	733.88	728.04	719.39	717.14	713.29	710	708.9	706.68	704.96

Deterministic 4-toast problem

Method	Session length (number of episodes)							
	50	100	150	200	250	300	350	400
SRL	8%	23%	39%	64%	79%	93%	96%	98%
Random	12%	19%	25%	36%	39%	49%	55%	60%
MC	23%	55%	69%	75%	80%	82%	87%	95%

Stochastic 4-toast problem

Method	Session length (number of episodes)							
	50	100	150	200	250	300	350	400
SRL	964.47	925.31	911.48	903.78	897.89	894.21	890.26	887.84
Random	947.14	935.82	924.9	920.02	914.5	912.81	909.63	910
MC	938.36	906.25	898.03	894.82	892.05	890.06	889.16	884.84

Table 7.6 Summary of case III results

Deterministic 3-toast problem

Method	Session length (number of episodes)									
	15	20	25	30	35	40	45	50	55	60
SRL	29%	40%	57%	73%	83%	92%	95%	97%	99%	100%
Random	35%	43%	52%	60%	64%	70%	75%	78%	81%	83%
MC	32%	38%	45%	49%	56%	56%	58%	61%	63%	64%

Stochastic 3-toast problem

Method	Session length (number of episodes)									
	15	20	25	30	35	40	45	50	55	60
SRL	1074.06	1055.66	1035.05	1016.16	1006.05	999.36	994.72	990.23	986.86	983.97
Random	1041.20	1030.56	1023.09	1017.69	1011.05	1007.46	1004.06	1003.36	998.51	999.17
MC	1062.67	1049.40	1032.95	1026.31	1020.80	1017.19	1013.74	1011.91	1006.80	1004.63

Deterministic 4-toast problem

Method	Session length (number of episodes)							
	50	100	150	200	250	300	350	400
SRL	8%	19%	33%	51%	71%	90%	93%	96%
Random	6%	15%	25%	30%	34%	37%	36%	48%
MC	24%	57%	65%	77%	81%	87%	89%	92%

Stochastic 4-toast problem

Method	Session length (number of episodes)							
	50	100	150	200	250	300	350	400
SRL	1390.44	1336.51	1309.87	1298.78	1290.06	1282.53	1278.65	1276.1
Random	1376.35	1348.41	1331.49	1328.15	1323.66	1315.18	1310.79	1307.7
MC	1353.09	1305.14	1290.78	1284.69	1280.49	1277.61	1275.03	1274.3

7.8 Summary

The applicability of the CHRL framework is shown using a robotic toast making system, requiring both low and high-level learning for its operation[1]. The SRL algorithm is used for learning high-level policies in the decomposed complex task, where there is a need to sequence the execution of a set of sub-tasks in order to optimize a target function. In such learning tasks, where there is a need to consider the sequence of steps as a whole, standard step-by-step update RL methods cannot be applied. Analyses indicate the SRL algorithm produces good results, matching or outperforming both the Monte-Carlo and the random search methods when allowed sufficient experience, in both deterministic and stochastic environments. Furthermore, the algorithm can be adjusted to achieve desired performance (in aspects of percentage of success in reaching the optimal solution and the number of episodes required to achieve convergence) by choosing the proper action selection parameters and reward factor type.

[1] The implementation of the low level learning is shown in the following chapter.

8. Path Planning for a Toast Making System

Chapter Overview

This chapter describes an exemplary low level learning task of inserting a bread slice into the toaster, as part of the toast making system developed to demonstrate the applicability of the CHRL framework.

8.1 Introduction

This application demonstrates an example for learning the execution of one of the required low-level sub-tasks, a task of path planning for the insertion of a slice of bread into the toaster. In this task the robot-arm is required to transfer the bread slice in the shortest path from a starting position to a target position above the toaster's hatch, while avoiding obstacles. The optimal path is learned off-line using the CCRL algorithm integrated with a standard *Q(λ)*[1] algorithm and a human advisor. After the learning phase it is implemented on-line with actual robot motions.

Since the low level task implementation is aimed to prove the applicability the CHRL framework, and not to assess the performance of the CCRL algorithm, the experiments are only intended to demonstrate a successful implementation[2].

8.2 Task Definition

In this task of inserting a bread-slice to the toaster, the robot is required to move its gripper, grasping a toast, from a starting location to a target location above the toaster, from which the bread will be lowered into the toaster's hatch. Along the path of traverse, the robot also must avoid obstacles found in the environment. This is a two-dimensional path planning task, where an optimal path from the starting state to the goal state is sought.

The problem formulation is as follows: The robot's state $s_t \in S$, is defined by two coordinates: $s_t = (x_i, y_j)$ where $i, j \in (1, 2, ..., 12)$. An action $a_t \in A$, taken at each state is traveling left, right, forward, or backward. Rewards are defined as $r_t \in \{-1, -0.1, +1.5\}$. If the robot reaches the target, the reward is positive (+1.5). If it collides with an obstacle, the reward is negative (-1). Furthermore, a small negative reward (-0.1) is assigned after each step in order to facilitate minimal number of steps. A learning episode comprises one event of reaching the target. The *Q(λ)* algorithm applies *softmax*

[1] The *Q(λ)* algorithm is presented in Section 2.2.

[2] As mentioned in the Methodology chapter, a thorough evaluation of the CCRL algorithm, using a 3D path planning task, is presented in Chapter 9.

action selection[1] for the autonomous learning, and its parameters[2] are set as follows: $\alpha = 0.95$, $\gamma = 0.99$ and $\lambda = 0.5$.

8.3 Experimental Setup and Method of Operation[3]

The experiment is performed using the UP-6 Motoman robot. A USB camera is used to capture the state of the system. The setup (Fig. 8.1) includes a table on which the obstacles (wooden cubes) and the toaster are located. The bread-slice is preliminary located on the corner of the table (bottom-left corner in the overhead view), to be taken by the robot during operation.

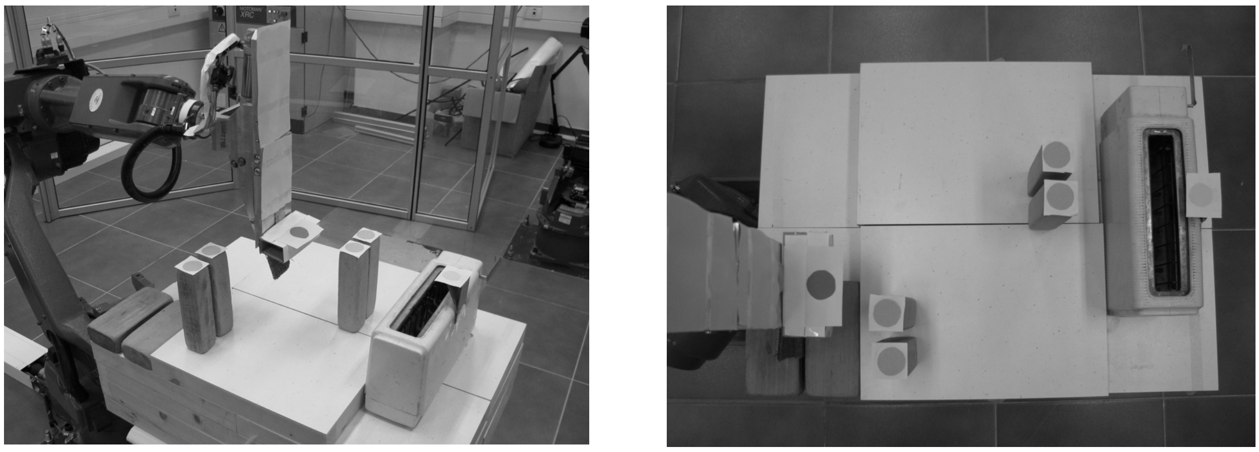


Fig. 8.1 Experimental setup – side and overhead views

The task is performed with the following steps:

1) The robot grasps the bread-slice and moves to the starting location.

2) A snapshot of the environment is taken using a USB camera situated above the table.

3) An image processing algorithm (running in MATLAB) is used locate the objects (robot's gripper, obstacls and toaster) and build a model of the environment accordingly. The objects are recognized using round markers in differnet colors. The environment is described as a 12×12 grid world.

4) A MATLAB simulation applying the CCRL algorithm (based on *Q(λ)*) is employed to learn the optimal path from the starting state to the goal state in the world's model.

[1] The *softmax* action selection method is described in Section 9.3.
[2] The *Q(λ)* parameters, α (learning rate), γ (discount factor) and λ (eligibility traces factor), are described in Section 2.2.
[3] A detailed description of the system and its operation can be found in Appendix I.

5) The robot is operated according to the generated path. Image processing is used to identify the location of the robot and syncrozine the location in the world's model with the location in the real world.

6) After arriving to the desired location above the toaster, the bread is lowered and the gripper is opened to release it into the toaster.

The simulated environment in which the learning is performed (built according to the image of the real environment) is displayed in Fig. 8.2.

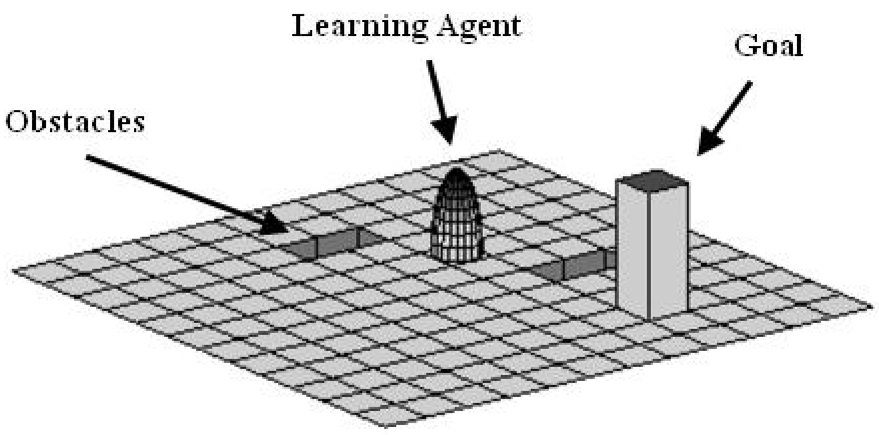
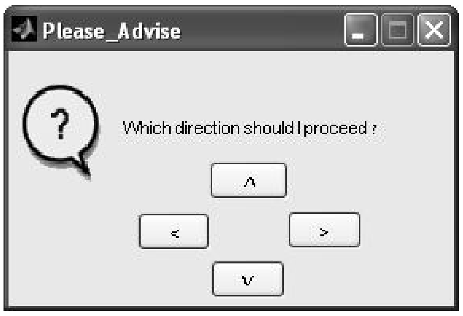
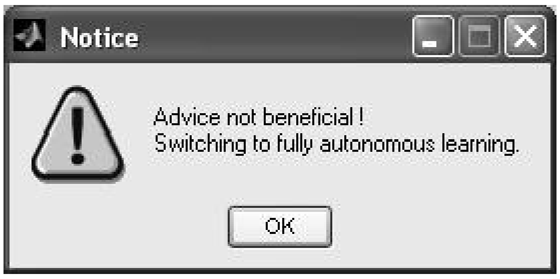


Fig. 8.2 Simulated environment

As mentioned, the CCRL algorithm is employed in the 4th step of the operation. When the robot senses that its performance does not improve fast enough, a request for advice is prompted. The human advisor is then required to guide the robot using the GUI (Graphical User Interface) shown in Fig. 8.3. If the robot concludes that the advice given is not beneficial, it switches to fully autonomous learning, and notifies the advisor.



Which direction should I proceed ?



Advice not beneficial !
Switching to fully autonomous learning.

(a) Guidance interface **(b) Autonomous learning notice**

Fig. 8.3 User interface - low level learning task

During the learning session, the robot (agent) updates the state-action values according to the employed algorithm (*Q(λ)* in this case). An example for the state-action value map is presented in Fig. 8.4. The height of the surface represents the state's value, which can be considered as the value of taking a certain action when the system is in a given state. After the off-line learning phase ends, the real robot is operated on-line according to the value map which represents the learned policy (the state-action pairs). At the on-line stage there is no learning, but only exploitation of the information gathered in the learning phase.

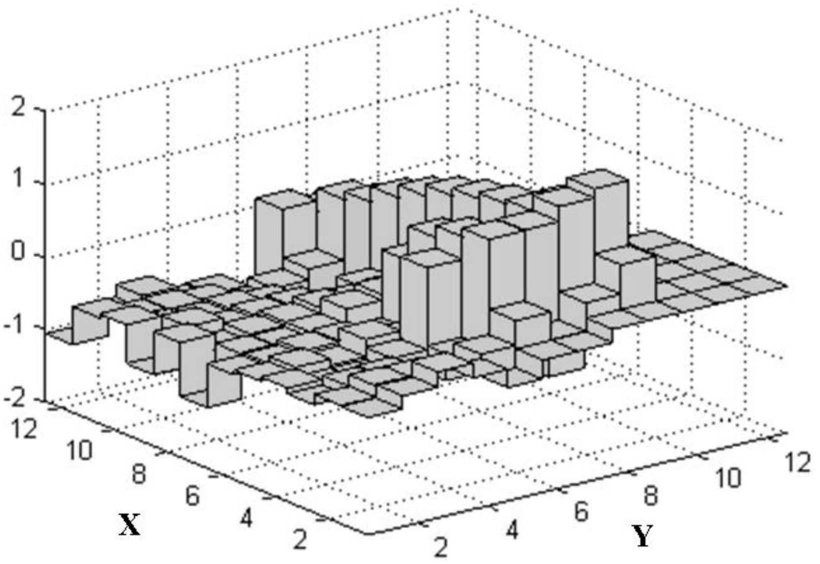


Fig. 8.4 State-action value map

8.4 Evaluation and Summary

A successful execution of a low-level task, the insertion of a bread-slice to the toaster while avoiding obstacles, is achieved using the CCRL algorithm integrated with a standard *Q(λ)* algorithm. In this case only a basic experiment was performed to prove applicability. The experiment was kept simple, since a thorough evaluation was conducted earlier with a simulated complex 3D environment (see Chapter 9), and execution of more complex experiments would not add value to the algorithmic analysis. Nonetheless, a full demonstration was conducted using several obstacle layouts to ensure a complete and feasible implementation of the CHRL framework.

9. Evaluation of CCRL using a 3D Path Planning Task

Chapter Overview

The CCRL algorithm is evaluated using a 3D path planning task. The evaluation includes a comparison of the CCRL algorithm to (i) a base-line fully autonomous RL algorithm, (ii) learning performed using the introspection approach (IA) and (iii) a combained CCRL and IA method. Various levels of human advisors are simulated, to assess the robustness of the algorithm under realistic conditions of imperfect guidance.

9.1 Introduction

The cognitive collaborative reinforcement learning algorithm (CCRL) addresses the questions of whether and when the robot should solicit advice by endowing the robot with human-like cognitive abilities. The robot applies a result-oriented approach, seeking aid when it comes to the understanding that its performance is not sufficient. Furthermore, the robot is given the ability to judge the worth of the advice it receives and to decide whether to accept or reject it. This self-awareness is achieved by performing self tests designed to evaluate its learning performance according to acceptable performance thresholds.

The CCRL algorithm is evaluated by applying it to a simulated three-dimensional path planning task, comparing the results to those achieved by (i) fully autonomous learning (a base-line used for comparison), (ii) learning using the Introspection Approach (IA) and (iii) learning with a combined CCRL and IA method. A simulated adviser with various skill levels is used in the evaluations. Advisor skill levels are represented by *softmax* temperature values varying the suggested actions from optimal to random, as described in Section 9.2.

The CCRL algorithm, as well as the IA method, uses the basic model of a RL learner incorporating advisor-suggested actions online. Upon receipt of an action from the advisor, the learning agent executes the action as if it had chosen the action with its own policy. Thus, the basic RL algorithm used (Q-learning in this case) does not need to be modified to handle the advisors actions.

9.2 Representation of Advisor Skill Levels

Since we assume that perfect guidance cannot always be provided, we analyze the effect of various skill levels of the human advisor by considering a continuum from novice to expert. When asked to give advice, the advisor simply examines the current state of the learner and provides the action that it considers best. In case of an expert advisor, this action is optimal. Lesser skilled advisors may provide either optimal or suboptimal actions. By adjusting the frequency by which the advisor responds with suboptimal actions, a wide range of problem-solving expertise can be simulated, from an expert advisor with perfect knowledge and skills to a novice with poor skills.

The skill level of the advisor is represented by the *softmax* action selection rule [Sutton and Barto, 1998], based on an optimal *Q* table[1]. In *softmax* the action probabilities are varied as a graded function of the estimated value. The greedy action is given the highest selection probability, but all the others are ranked and weighted according to their value estimates. The *softmax* method uses a Boltzmann distribution, choosing action *a* on the *t*-th step with the probability shown in (9.1).

$$\text{Prob}(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{i=1}^{n} e^{Q_t(i)/\tau}} \tag{9.1}$$

Where $Q_t(i)$ is the value of taking an action *i* from the current state and τ is a positive parameter referred to as the *temperature*. High temperatures cause the actions to be all (nearly) equiprobable. Low temperatures cause a greater difference in selection probability for actions that differ in their value estimates. In the limit, as $\tau \to 0$, *softmax* action selection becomes the same as greedy action selection.

Human skill level is adjusted using τ, the temperature parameter. Since the advisor's action selection is performed on the basis of an optimal *Q* table, the use of very small temperatures, meaning choosing actions greedily, will result in suggesting optimal actions at each state. A use of higher temperatures will result in more random action suggestions. Therefore, a human with perfect skills can be represented by using a very low temperature while a human with poor skills will be represented using a relatively high temperature.

An example for the performance of advisors with various skill levels, represented using a range of temperatures, $\tau = 0.01 - 1$, is presented in Fig. 9.1. The most skillful advisor, represented using $\tau = 0.01$, chooses actions that result in reaching the goal in the optimal number of steps (13 steps in this example) in 100% of the cases, while less skilled advisors achieve optimal solution with lower percentages and higher average number of steps. The poorest skilled advisor does not achieve the optimal solution at all and requires an average of 178 steps to reach the goal state. It is important to understand that here there is no learning process of the advisor, but only a use of the optimal *Q* table to simulate the human suggestions to the robot (As a sideline however, it is possible to use the τ representation to simulate advisor learning by dynamically modifying its value).

[1] The optimal *Q* table is assumed to be known for the advisor simulation, but is of course unknown to the learning agent.

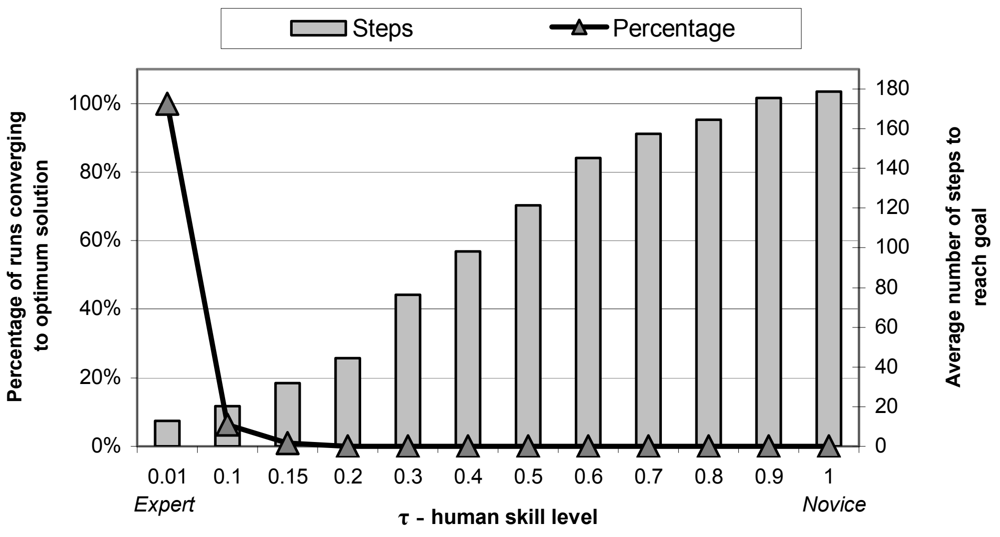


Fig. 9.1 Human advisor representation

9.3 CCRL

CCRL is implemented here based on a standard *Q*-learning algorithm. The performance assessment and advice refusal capabilities are employed as described in Chapter 6. The algorithm is configured such that the parameter *X*, defining the episode from which the robot can solicit advice, is set to 30, and the parameter *K*, used for various calculations, is set to 5. The collaboration threshold parameters Λ and Ω are varied as described in Section 9.8.

9.4 Fully Autonomous Learning (Base-Line)

The base-line robotic learner employs *Q*-learning to develop its policy[1]. A value *Q*, associated with a state-action pair, (s_t,a_t), represents how "good" it is to perform a specific action a_t when at state s_t. A learning episode is a finite sequence of time steps, during which the agent traverses from the starting state to the goal state (the episode is stopped after a predefined number of steps, even if the agent haven't reached the goal state). A learning session is a series of *N* learning episodes. Action selection is performed using the adaptive *softmax* method, in which the temperature τ is varied as a function of the learning episode. In the beginning of the session, when the agent has not gathered much information yet, a relatively high temperature encourages exploration of the state-space by allowing more non-greedy actions. As the learning session progresses, the temperature decreases, reducing the number of random actions, and allowing the agent to exploit the information already gathered and perform better. The change in τ as a function of the episode number is shown in (9.2).

$$\tau = \frac{1}{n^{\beta}} \tag{9.2}$$

[1] The *Q*-learning algorithm is described in Section 2.2.

Where n is the number of episodes already performed during the current learning session and β is a positive parameter specifying how fast τ will exponentially decrease towards zero, meaning how greedily the algorithm will act as the learning proceeds (greater β results in sooner exploitation). Note that τ is updated similarly to ε in adaptive *ε-greedy* (Eq. 5.1).

9.5 Introspection Approach (IA)

The IA method [Clouse, 1996] is used here as a benchmark for comparison. To implement IA, Clouse developed a test to determine whether the learner is unsure of its choices, indicating the need for help in novel situations. When discussing an automated learner, it is fairly easy to specify exactly when they are unsure: one has access to the decision policy and the evaluations on which the decision is based. The test examines the two extreme values of possible actions ($Q(s,a)$): if they are sufficiently close to each other it implies that the learner has not experienced this state often enough to produce a clear choice. In this case the test succeeds and the learner asks for aid. Sufficiency is determined by comparing the difference between the minimum and maximum Q values to a width parameter Ψ - if the difference is smaller than the width parameter the test succeeds. With a small width parameter, the learner rarely asks for assistance, while with a large width parameter, the learner asks for aid quite frequently. The IA advice request self test is shown in (9.3).

$$\begin{array}{l} \text{If } \mathrm{Max}_i\, Q(s,a_i) - \mathrm{Min}_i\, Q(s,a_i) < \Psi \\ \text{Then } \textit{request advice for current state } s \\ \text{Else } \textit{choose action autonomously} \end{array} \qquad (9.3)$$

Where $Q(s,a_i)$ is the value of taking action a_i when at state s, and Ψ is the width parameter.

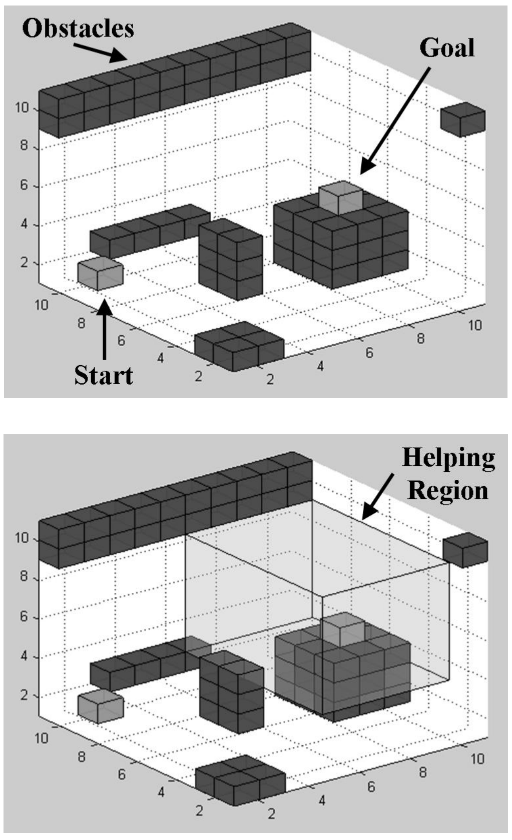
9.6 Combined Method

The combined method integrates both CCRL and IA to one algorithm. In this combined algorithm, advice is solicited only when both advice request tests (Eq. 6.2 and 9.3) are passed (meaning that the learner's rate of improvement is unsatisfactory and that the learner is unsure of its choices in its current state), and when the advise refusal threshold (Eq. 6.4) in not exceeded.

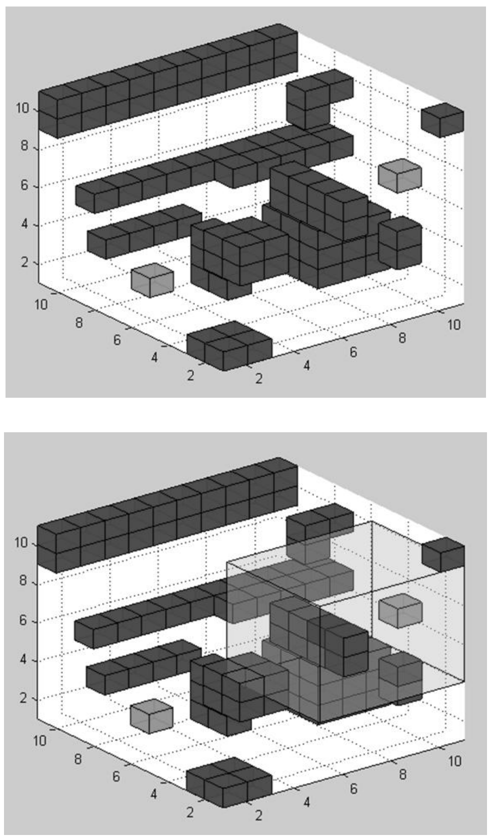
9.7 Task Definition

Evaluation is conducted using a path planning test problem comprised of a simulated mobile robot traversing a three-dimensional grid-world of size 10×10×10 (1000 states). Two grid-world instances are considered, one with a relatively low obstacle density (60 obstacle states), and another with higher density (100 obstacle states). In order to evaluate the performance for cases where the human advisor has only a limited view of the environment and can supply advice only when the robot is within that view, a limited region of assistance is also considered for each grid-world. Such situations may arise in systems employing teleoperated guidance, due to various reasons (*e.g.*, hazard materials, radiation), and the human operator is located remotely, receiving visual feedback from the robot's operation area through a camera with limited area coverage.

Fig. 9.2 shows the two grid-worlds. The starting and goal states are represented by light-gray cubes. The obstacles are shown as dark-gray cubes. The limited helping region is marked with a large transparent cube. The optimal (shortest) path from the starting state to the goal state includes 13 and 16 steps for worlds I and II, respectively.



(a) World I – full and limited views



(b) World II – full and limited views

Fig. 9.2 The grid-worlds

The objective of the robot is to traverse from a starting state to a goal state through the shortest path, while avoiding obstacles. At each state (s_t), the robot can choose one of six actions (a_t) - up, down, left, right, forward, or backward. When the robot collides with an obstacle, reaches the goal or exceeds a maximum number of steps, the learning episode is stopped, and the robot is returned to the starting point. Reinforcements (r_t) are set as follows: the robot receives a positive reward of +1.5 units for reaching the goal, a negative reward of -1.0 units for colliding with an obstacle and a negative reward of -0.1 units for each step performed. The state of the system is the position of the robot defined by its three coordinate values.

9.8 Analysis

Analysis is performed using MATLAB simulations. Four different learning methods are employed and compared: (i) fully-autonomous learning using a standard *Q*-learning algorithm (serving as base-line for the comparison), (ii) the CCRL algorithm, (iii) the IA method, and (iv) a combined method integrating the advice request rules of both CCRL and IA. All methods are evaluated using four environments – worlds I and II, each with full and limited views.

In all of the tests the *Q*-learning and action selection parameters are set as follows: learning rate $\alpha = 0.95$, discount rate $\gamma = 0.99$[1] (These parameters were selected empirically). Furthermore, all of the learning sessions includes 200 learning episodes ($N = 200$), with a maximum of 200 steps allowed at each episode (Also selected empirically). For the collaborative algorithms, human skills, represented by τ, is varied from an expert advisor ($\tau = 0.01$) to a novice ($\tau = 1$).

Four different tests are performed for each environment (Table 9.1): The *first* test examines the base-line fully autonomous learning and the differences in performance using various action selection parameters. Sensitivity analysis is performed using various values of the adaptive *softmax* parameter β (0.5 to 1.5). Each value is evaluated by performing 500 learning sessions.

To evaluate the performance of the CCRL algorithm, a *second* test is conducted. The collaboration threshold Λ is varied from 0.01 (demanding small improvement in performance) to 0.9 (demanding significant improvement during the session). The advice refusal threshold Ω, defining the number of occasions in which the results of learning with an advisor's aid are allowed to be worse than the results of the previous episodes, is varied from 1 to 7. For each combination of τ, Λ and Ω, five simulation replications are performed, each composed of 100 learning sessions. In all of the tests the parameter X, defining the episode from which the robot can solicit advice, is set to 30, and the parameter K, used for various calculations, is set to 5.

In a *third* test, IA is implemented to solve the path planning problem under consistent assumptions. The width parameter Ψ is varied from 0.1 (representing a learner which is rarely uncertain) to 1.3 (representing a learner that asks for aid quite frequently). Each value of Ψ is evaluated by performing five simulation replications, each containing 100 learning sessions.

[1] The RL parameters, α (learning rate) and γ (discount factor) are described in Section 2.2.

A *fourth* test evaluates the performance of a combined method. For each combination of τ, Λ, Ω and Ψ, five replications of 100 sessions are performed.

Performance is evaluated using the following measures:

1) *AR* (advice requests) – Average number of requests for advice during the learning session (used only for CCRL, in which advice is requested for a whole episode).

2) *HS* (helped steps) – Average number of steps performed using advice during the learning session (used for the IA method in which advice is requested per step, and for the comparison)

3) *SP* (success percentage) – Average percentage of learning sessions reaching the optimal solution (minimal path length).

4) *Score* – Weighted normalized scoring based on the *HS* and *SP* measures, described in Section 9.9.4.

Table 9.1 Summary of tests

No.	Examined Methods	Environments / Advisors	Analyzed Parameters (Values)
1	Fully Autonomous	Worlds I and II	Decay factor *β* (0.5 - 1.5)
2	CCRL	Worlds I and II Full and limited views Expert to novice advisors	Collaboration threshold Λ (0.01 - 0.9) Advise refusal threshold Ω (1 - 7) Human skill level τ (0.01 - 1)
3	IA	Worlds I and II Full and limited views Expert to novice advisors	Width parameter Ψ (0.1 - 1.3) Human skill level τ (0.01 - 1)
4	Combined Method (CCRL and IA)	Worlds I and II Full and limited views Expert to novice advisors	Collaboration threshold Λ (0.01 - 0.9) Advise refusal threshold Ω (1 - 7) Width parameter Ψ (0.1 - 1.3) Human skill level τ (0.01 - 1)

9.9 Results and Discussion

9.9.1 Fully Autonomous Learning

Fig. 9.3 shows an example of convergence to a solution in a given learning session in world I. In this example a value of *β* = 1.3 is used. Note that in order to distinguish the episodes in which the robot collided with an obstacle, they are assigned a value of 201 steps (maximal number of steps +1). As can be seen, in the beginning of the session, the robot, still unfamiliar with the environment, collides with obstacles many times, and only after about 50 episodes it starts reaching the goal regularly, eventually converging to the optimal path of 13 steps at episode number 135.

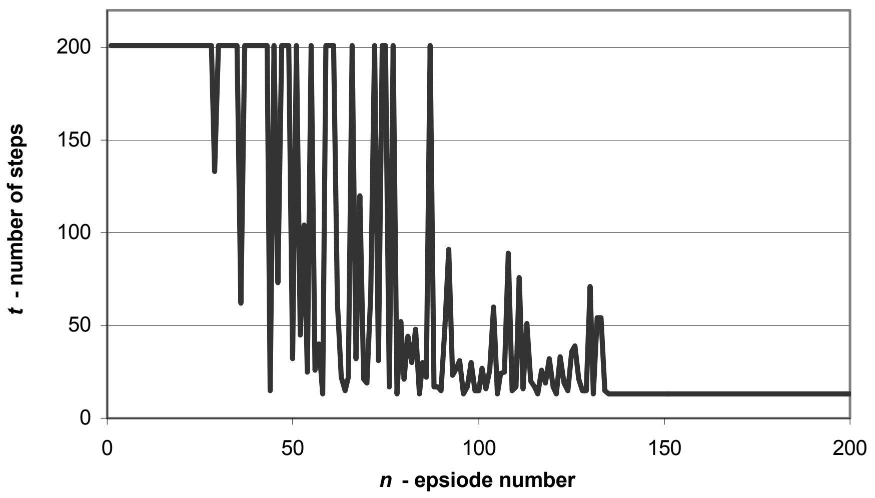


Fig. 9.3 Convergence in a learning session

For world I, the best performance is obtained with *β* = 1.3, when the learning achieves *SP* of **63%**. For world II, performance (*SP*) is **37%**, obtained with *β* = 0.7 (Fig. 9.4). With lower values of *β*, τ decreases to zero relatively slow so actions are chosen more randomly, and the algorithm does not achieve convergence fast enough, resulting in low percentage of sessions reaching the optimal solution. With higher values of *β*, τ decreases to zero too fast so actions are chosen greedily, and the algorithm converges to a local optimum, hence does not achieve the global optimum. The reason for the lower success percentage in world II lies in the fact that it has a higher obstacle density, and hence it is harder to reach the goal. This is also the reason why in world II the optimal *β* is lower – more exploration in needed in order to find the optimal path when there are more obstacles, and a lower *β* is required to allow this exploration. The *β* values achieving best results here (1.3 and 0.7 for worlds I and II, respectively) were used for the following tests.

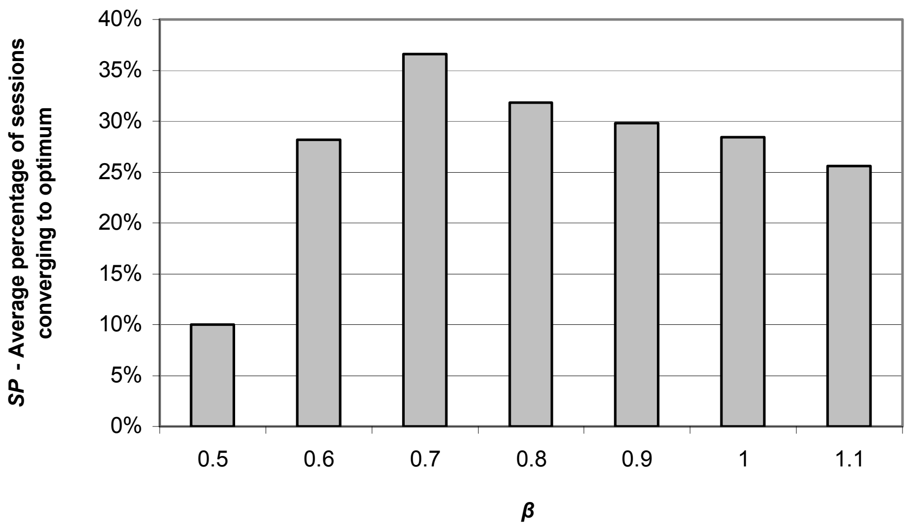


Fig. 9.4 Autonomous learning - influence of *β*

9.9.2 CCRL

When applying CCRL and introducing an expert advisor ($\tau = 0.01$), the results improve drastically as expected. The learning achieves **99%** and **96%** *SP* for worlds I and II respectively, with significant improvements of **36%** and **59%** with respect to the autonomous learning results. An example for the collaboration level switching during a learning session is shown in Fig. 9.5. When the improvement rate (*IR*) is above the threshold (Λ = 0.05 in this case) the robot learns autonomously. When it drops below the threshold, human assistance is requested and the system switches to the guided mode, until the rate again exceeds the threshold. The *IR* stabilized to 0 at episode number 162, meaning the algorithm converged to a solution.

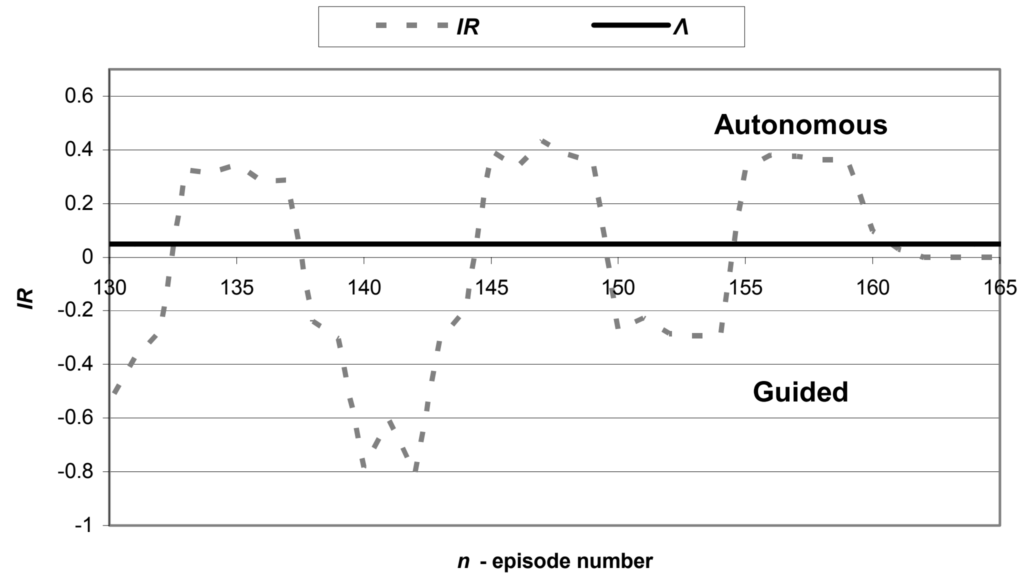


Fig. 9.5 CCRL - collaboration level switching, example I

Another example, presenting both the number of steps performed in each episode and the *IR* calculated accordingly, is shown in Fig. 9.6. A value of $K = 5$ is used, meaning that *IR* is calculated (according to Eq. 6.1) using the 10 ($2K$) previous episodes. The high average number of steps during episodes 145 - 149, in comparison to the lower average in the prior episodes, causes *IR* to drop below Λ (0.05) at episode number 148 and triggers requests for human guidance. With the supplied advice, the number of steps reduces again, and convergence is achieved at episode 157, followed by the stabilization of *IR* to 0 at episode number 166 (10 episodes later).

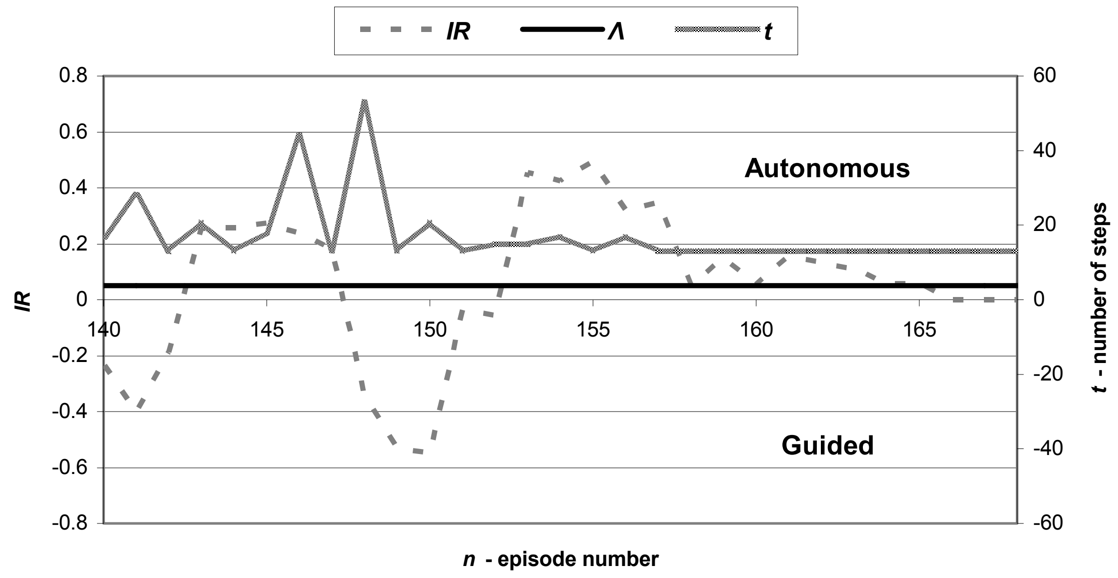


Fig. 9.6 CCRL – collaboration level switching, example II

To examine the self-assessment capability, learning was performed with various threshold values. When setting low values of Λ, the robot is expected to achieve less improvement in each episode, hence requests less advice. As Λ rises, high improvement is required and the robot asks for help more frequently. When the advisor is an expert (τ = 0.01), more advice leads to better performance, as can be seen in Fig. 9.7 (results for world II with limited view).

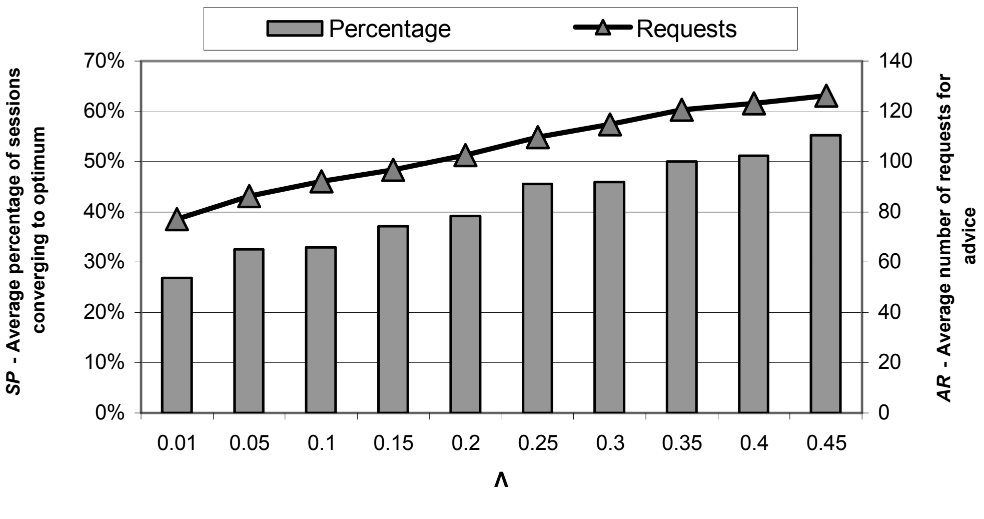


Fig. 9.7 CCRL - influence of Λ

When the advisor does not supply optimal instructions, an interesting phenomenon appears. Fig. 9.8 shows the results for the limited view case of world I, with a collaboration threshold of Λ = 0.05 and advisors of various skill levels (represented by τ), but **without advice assessment capabilities**. On the one hand, the learning agent senses its performance is not sufficient, and therefore it requests human guidance. On the other hand, when the human is not an expert, the advice may not bring the desired improvement in performance and might even cause deterioration. The learning agent keeps asking for guidance because it does not improve, and the guidance further deteriorates its performance. The situation enters a "vicious cycle" from which there is no escape, resulting in very low performance. As expected, when the human skill level is lower, the agent requests guidance more often, and the performance deteriorates faster.

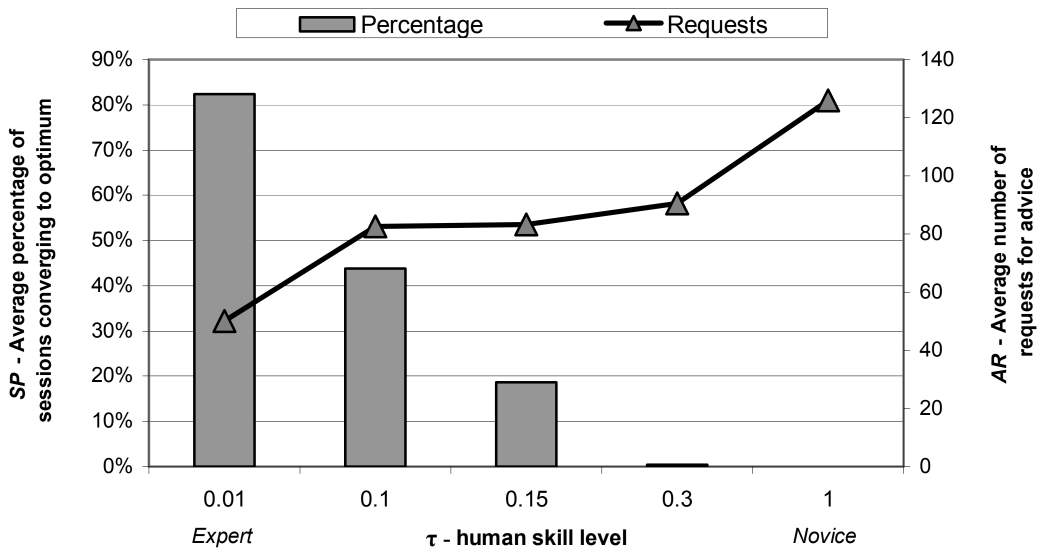


Fig. 9.8 CCRL - learning without advice assessment capabilities

The introduction of advice assessment capabilities helps to break this "vicious cycle" and improves performance significantly. Fig. 9.9 shows results for the same environment (world I, limited view), with a collaboration threshold Λ = 0.05 and an advice refusal threshold of Ω = 1, again using the aid of advisors with various skill levels (various values of τ).

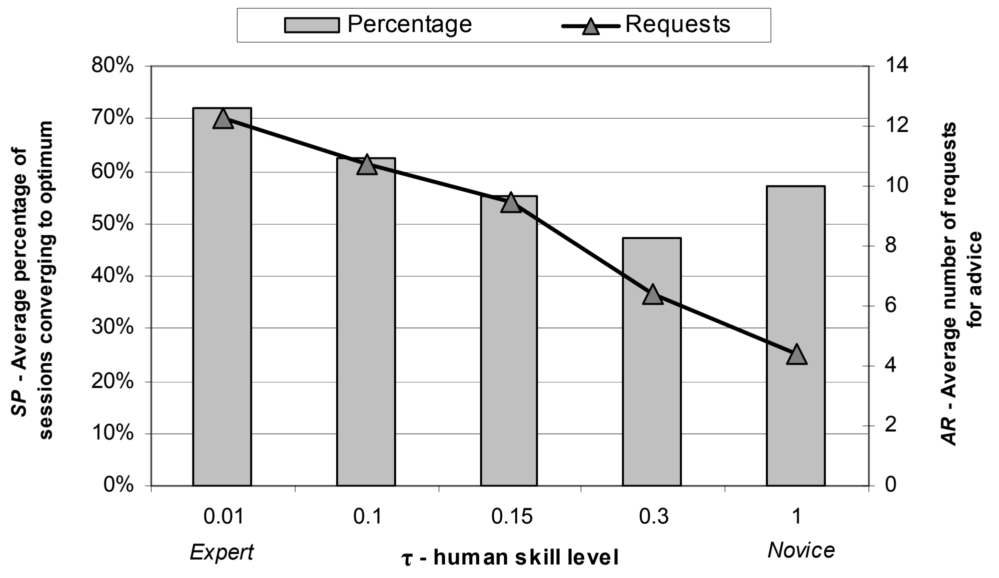


Fig. 9.9 CCRL - learning with advice assessment capabilities

Here, when human skill level is low, it is sensed by the robot which accordingly stops asking for guidance. Since the robot reverts to autonomous learning, human advice does not interfere and this results in improved performance. When the advisor is a novice, the robot understands it very fast and stops asking for advice early in the session, achieving better results. The worse performance appears when the advisor's expertise is midway ($\tau = 0.3$), not very good and not very bad, so it takes time for the robot to notice that the advice is not good enough and to stop asking for it. When the advisor is an expert, the results are slightly worse in comparison to those achieved without the advice assessment capability (72% vs. 82%). The reason is that the agent sometimes misjudges the advice and rejects it, even though the advisor is an expert. In these cases the actions chosen autonomously by the agent (who reverted to fully autonomous learning) are worse then those suggested by the advisor, leading to worse performance.

A further study of the effects of the advise refusal threshold Ω (Eq. 6.4) reveals that there is a trade-off when setting its values. When Ω is low, poor skilled advisors would be quickly recognized and discarded, but experts might be misjudged and unjustly discarded as well. When Ω is high there is a smaller chance of discarding an expert, but it also takes longer time for the robot to identify worse skilled advisors, and the prolonged use of their advices obstructs the learning. Hence, when the advisor is skilled a use of high values of Ω achieves the best performance, while when the advisor has limited or no skills lower values of Ω result in better learning. This trade-off is illustrated in Fig. 9.10, showing the results of employing CCRL with $\Lambda = 0.05$ and various values of Ω (1, 3, 5, 7) to the limitedly viewed world I.

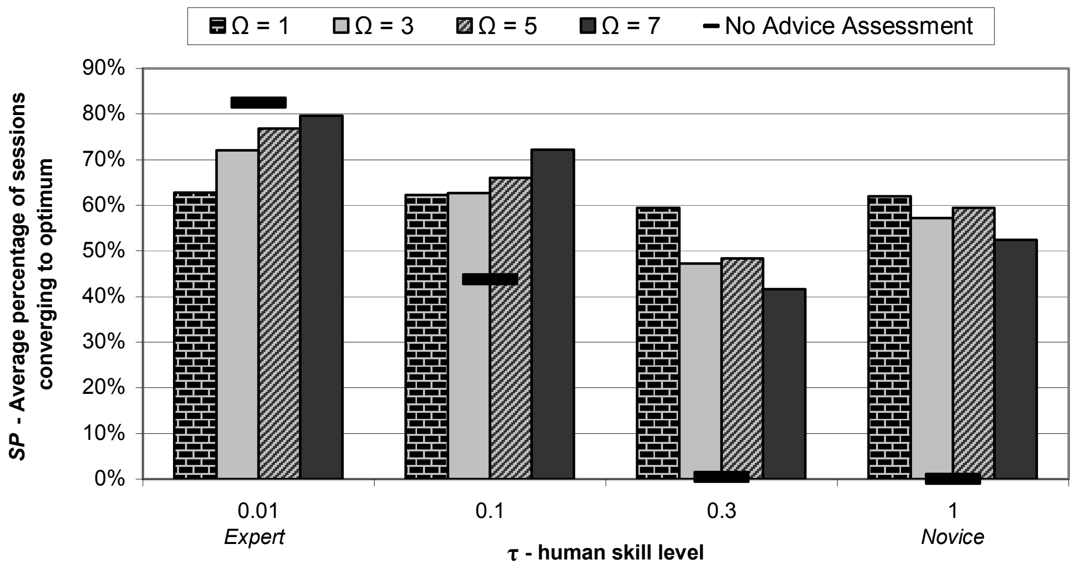


Fig. 9.10 CCRL - influence of Ω

As explained, for the skilled advisors (τ = 0.01, 0.1) higher Ω results with better performance, while with less competent advisors (τ = 0.3, 1) the trend inverts and higher Ω results with worse performance (though without statistical significance). As discussed above, when advice is given by an expert, even better results are achieved when the advice assessment capability is not used at all (no misjudgment occurs), yet for other advisors this capability is essential for obtaining good results.

9.9.3 IA

When employing IA, the width parameter Ψ influences the learning as described in Section 9.5: with a small width, the learner is rarely uncertain, asking little advice, while with a large width, the learner asks for guidance quite frequently. Fig. 9.11 presents the results of learning with various values of the width parameter Ψ, in world II with a limited view, assisted by an expert (τ = 0.01).

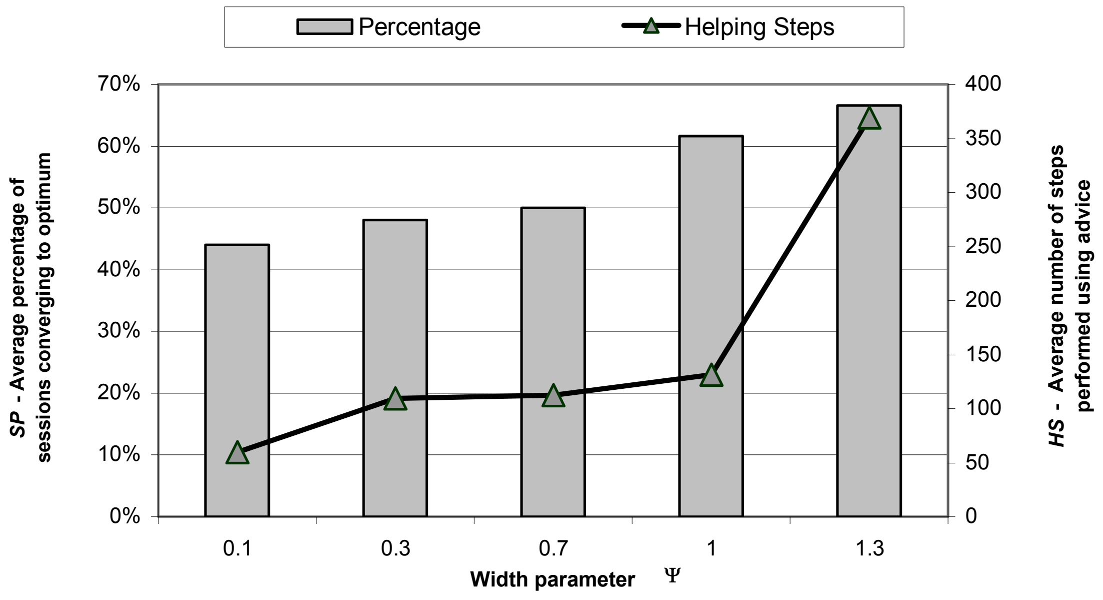


Fig. 9.11 IA - influence of the width parameter Ψ

With less competent advisors, IA suffers from the same problem described for CCRL – bad advice leaves the robot uncertain, leading it to ask for more advice, causing even more uncertainty. When advisor skill level is lower, the agent requests more guidance, and the performance is worse, as shown in Fig. 9.12 (World II, limited view, Ψ = 1). The difference from CCRL is that IA does not endow the robot with the advice assessment capability that enables it to cope with such advisors.

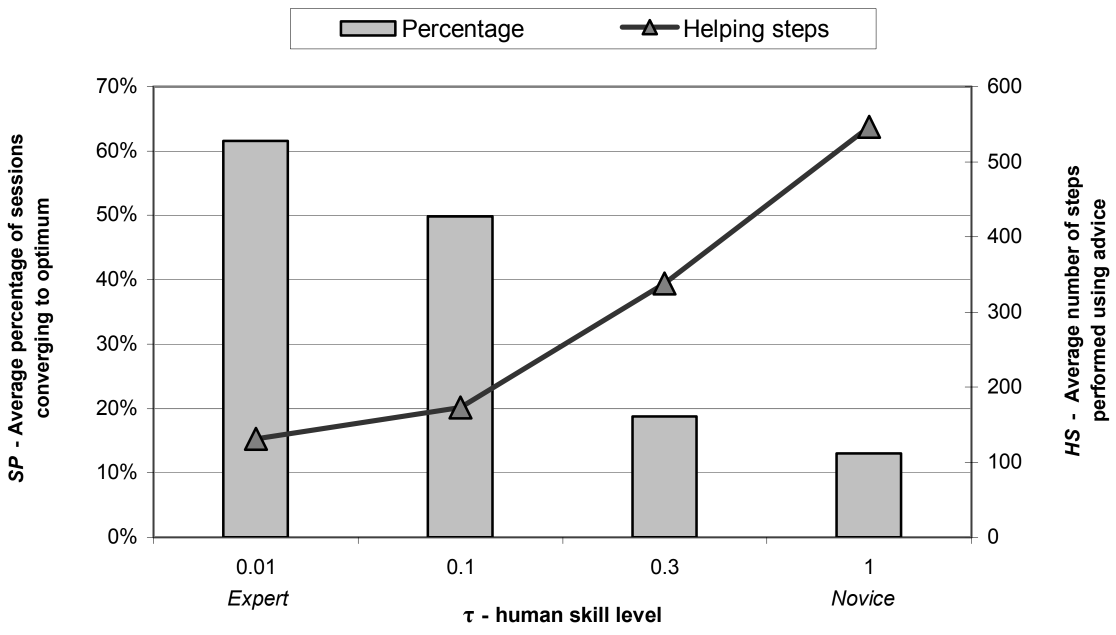


Fig. 9.12 IA - learning with various advisors

9.9.4 *Combined Method*

As explained, in the combined algorithm advice is solicited only when both advice request tests (Eq. 6.2 and 9.3) are passed, and when the advice refusal threshold (Eq. 6.4) in not exceeded. The severity of the advice request conditions promises that the agent will ask for guidance only when it is really necessary (to the agent's understanding). This difference is shown in Fig. 9.13, presenting the results for World II with limited view and collaboration parameter values of $\Psi = 1$, $\Lambda = 0.05$ and $\Omega = 5$ (similar to the conditions presented in Fig 9.12 for the IA method).

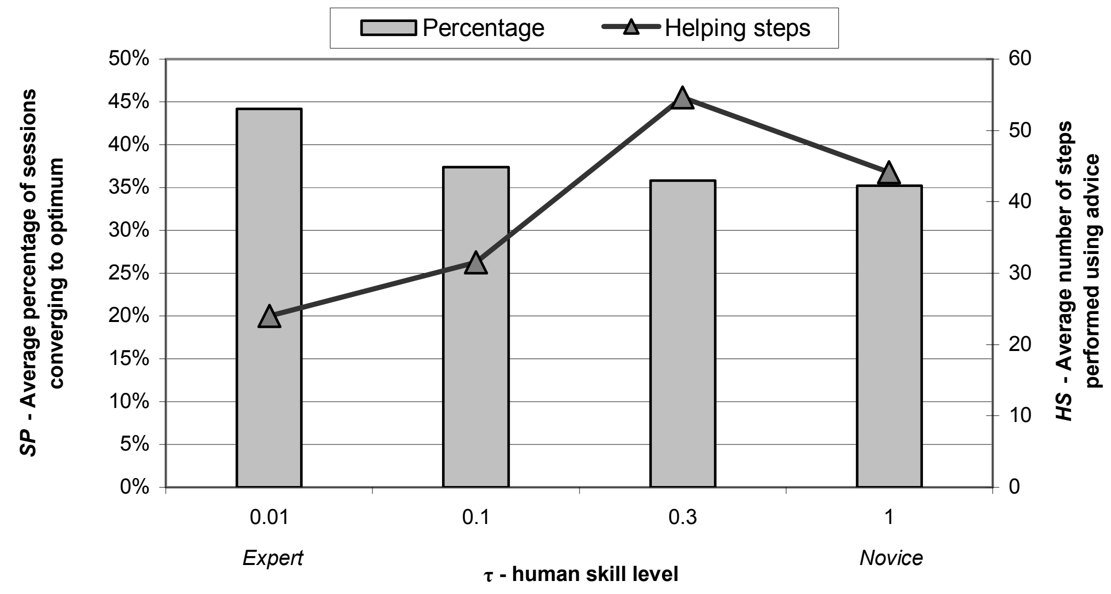


Fig. 9.13 Combined method - learning with various advisors

Here, as expected, the results (*SP*) for the skilled advisors ($\tau = 0.01, 0.1$) are worse than the ones achieved by IA or CCRL, since less guidance is requested. For the less skilled advisors ($\tau = 0.3, 1$) the behavior is similar to the one seen for CCRL (Fig. 9.9), where the agent learns to ask for less guidance and achieves better results by reverting to autonomous learning. In general, the combined method achieves lower *SP* than IA with skilled advisors or CCRL. However, it demonstrates robust performance for all the spectrum of advisor skill levels, and achieves it with much less guidance than IA and CCRL. A thorough comparison is presented in the following section.

9.9.5 Comparative Analysis

When comparing the methods it is important to notice that unlike CCRL, the assistance request in the IA method is triggered per step and not for the entire episode. Therefore, the performance measure used for comparison is *HS*, the number of steps in the entire session performed using advisor assistance.

The comparison is problematic, since we have a multi-objective problem with two performance measures, *SP* (success percentage) and *HS* (helped steps). The preferable case, higher performance with the cost of many interruptions to the advisor, or less interruption with inferior performance, depends on the specific application. A way to address this difficulty is to base the comparison on a weighted normalized scoring. The two performance measures receive a weight corresponding to their relative importance. The results of a specific combination of collaboration parameters (Λ, Ω and Ψ) are normalized to achieve a common basis for comparison. The score, representing how good was the learning using these parameters is calculated according to (9.4).

$$\begin{gathered} Score = W_1 * \frac{SP_i - \min(SP)}{\max(SP) - \min(SP)} + W_2 * \frac{\max(HS) - HS_i}{\max(HS) - \min(HS)} \\ W_1 + W_2 = 1 \end{gathered} \tag{9.4}$$

Where W_1 and W_2 are the weights assigned to *SP* and *HS*, respectively. SP_i is the average percentage achieved using the *i*-th combination of collaboration parameters and HS_i is the average number of helping steps used with that combination. The calculation is designed in a way that will result in the highest score of 1 when the evaluated combination achieves the highest *SP*, using the lowest *HS*. Lower SP_i or higher HS_i will reduce the score.

When comparing, one can seek the combination receiving the highest score for a specific advisor skill level, or the combination that shows the most robust performance, dealing well with all levels of human expertise (here the final score for a specific combination is an average of the scores received for the various human skill levels).

Tables 9.2 - 9.5 summarize the best scores achieved by each of the three methods (CCRL, IA and combined), for the full and limited view cases of worlds I and II. Equal weights are assigned to each performance measure ($W_1 = W_2 = 0.5$). The scores are compared and ranked using one-way ANOVA (F-test) and Tukey's HSD test, demanding 95% confidence level[1]. The rank column indicates the relations between the methods. If two or more methods have the same rank, it means they are not significantly different. Significant best scores are marked in gray. When two scores are marked it means they are significantly better than the third, but there is no significance between them.

Table 9.2 Scores for world I with full view

Advisor (τ)	Method	Parameters	*SP*	*HS*	*Score*	Rank
Expert (0.01)	CCRL	Λ=0.05, Ω=1	99%	292.2	0.96	2
	IA	Ψ=0.7	98%	268.6	0.96	2
	Combined	Λ=0.3, Ω=1, Ψ=1	100%	96.8	1.00	1
Moderately expert (0.1)	CCRL	Λ=0.3, Ω=1	67%	825.2	0.72	2
	IA	Ψ=1	99%	200.5	0.98	1
	Combined	Λ=0.3, Ω=1, Ψ=1	98%	156.2	0.98	1
Limited skills (0.3)	CCRL	Λ=0.9, Ω=1	11%	593.6	0.47	3
	IA	Ψ=1.3	96%	747.9	0.87	1
	Combined	Λ=0.05, Ω=1, Ψ=1	80%	623.7	0.81	2
Novice (1)	CCRL	Λ=0.9, Ω=1	55%	271.4	0.74	1
	IA	Ψ=0.1	62%	915.1	0.68	2
	Combined	Λ=0.05, Ω=1, Ψ=0.3	61%	308.8	0.77	1
All levels (average)	CCRL	Λ=0.3, Ω=1	55%	595.4	0.69	3
	IA	Ψ=0.7	84%	988.9	0.77	2
	Combined	Λ=0.05, Ω=1, Ψ=1	82%	290.9	0.88	1

Table 9.3 Scores for world I with limited view

Advisor (τ)	Method	Parameters	*SP*	*HS*	*Score*	Rank
Expert (0.01)	CCRL	Λ=0.05, Ω=7	80%	135.3	0.85	1
	IA	Ψ=1.3	82%	153.9	0.86	1
	Combined	Λ=0.05, Ω=7, Ψ=0.3	73%	51.5	0.84	1
Moderately expert (0.1)	CCRL	Λ=0.2, Ω=1	66%	25.5	0.78	1
	IA	Ψ=1.3	80%	171.9	0.82	1
	Combined	Λ=0.05, Ω=1,Ψ=0.3	67%	15.9	0.8	1
Limited skills (0.3)	CCRL	Λ=0.05, Ω=1	59%	39.9	0.69	1
	IA	Ψ=0.1	60%	136.2	0.62	2
	Combined	Λ=0.5, Ω=1, Ψ=0.3	60%	19.2	0.72	1
Novice (1)	CCRL	Λ=0.01, Ω=1	63%	28.8	0.74	1
	IA	Ψ=0.1	63%	149.6	0.63	2
	Combined	Λ=0.5, Ω=1, Ψ=0.3	61%	18.9	0.73	1
All levels (average)	CCRL	Λ=0.3, Ω=1	62%	27.9	0.73	1
	IA	Ψ=0.1	63%	121.4	0.66	2
	Combined	Λ=0.05, Ω=1,Ψ=0.3	63%	17.4	0.74	1

[1] A detailed explanation of the comparison method can be found in Appendix II.

Table 9.4 Scores for world II with full view

Advisor (τ)	Method	Parameters	*SP*	*HS*	*Score*	Rank
Expert (0.01)	CCRL	Λ=0.05, Ω=3	96%	523.0	0.95	2
	IA	Ψ=0.7	98%	522.1	0.96	2
	Combined	Λ=0.01, Ω=1, Ψ=1	98%	185.5	0.98	1
Moderately expert (0.1)	CCRL	Λ=0.5, Ω=1	47%	1852.7	0.61	3
	IA	Ψ=0.7	87%	608.1	0.90	2
	Combined	Λ=0.01, Ω=1, Ψ=1	91%	347.4	0.94	1
Limited skills (0.3)	CCRL	Λ=0.5, Ω=1	20%	736.2	0.56	2
	IA	Ψ=1	65%	2216.6	0.67	1
	Combined	Λ=0.05, Ω=1, Ψ=0.3	36%	150.4	0.68	1
Novice (1)	CCRL	Λ=0.3, Ω=1	37%	177.1	0.68	1
	IA	Ψ=0.1	40%	738.4	0.65	2
	Combined	Λ=0.05, Ω=1, Ψ=0.3	37%	115.9	0.68	1
All levels (average)	CCRL	Λ=0.5, Ω=1	49%	819.9	0.70	2
	IA	Ψ=0.7	70%	2105.5	0.71	2
	Combined	Λ=0.01, Ω=1, Ψ=1	64%	434.3	0.80	1

Table 9.5 Scores for world II with limited view

Advisor (τ)	Method	Parameters	*SP*	*HS*	*Score*	Rank
Expert (0.01)	CCRL	Λ=0.05, Ω=7	51%	43.2	0.83	1
	IA	Ψ=1.0	62%	131.1	0.84	1
	Combined	Λ=0.05, Ω=5, Ψ=1	44%	24.0	0.78	1
Moderately expert (0.1)	CCRL	Λ=0.3, Ω=3	41%	26.6	0.75	1
	IA	Ψ=0.1	41%	66.1	0.72	1
	Combined	Λ=0.05, Ω=1,Ψ=1	40%	11.0	0.76	1
Limited skills (0.3)	CCRL	Λ=0.3, Ω=1	35%	22.2	0.70	1
	IA	Ψ=0.1	34%	78.3	0.64	2
	Combined	Λ=0.3, Ω=1, Ψ=1	38%	20.0	0.73	1
Novice (1)	CCRL	Λ=0.9, Ω=1	36%	16.1	0.72	1
	IA	Ψ=0.1	34%	86.5	0.64	2
	Combined	Λ=0.05, Ω=3, Ψ=0.3	38%	22.9	0.73	1
All levels (average)	CCRL	Λ=0.5, Ω=1	36%	15.4	0.72	1,2
	IA	Ψ=0.1	39%	72.7	0.69	2
	Combined	Λ=0.01, Ω=1,Ψ=1	38%	13.7	0.74	1

Overall, the combined method achieves the best results for both worlds and both view cases. It does so for most of the advisor skill levels separately, and for the average case, demonstrating robustness in dealing with various levels of advisors. These results are achieved since the robot asks for aid only when it really requires it, under conditions of uncertainty and deficiency in performance, and stops asking for it when it is not beneficial.

When considering CCRL and IA, it can be seen that when assisted by skillful advisors (τ = 0.01, 0.1) and in average in the full view cases, IA performs better, while with lesser skilled advisors (τ = 0.3, 1) and in average in the limited cases, CHRL achieves better results, equivalent to those of the combined method. This can be attributed to the advice assessment capability employed in CCRL and in the combined method.

Generally, in the limited view cases CCRL performs relatively better in comparison to IA, while in the full view cases IA has the upper hand. This can be explained by the fact that in the full view cases, CCRL uses advice during the entire episode, even for states where the robot does not really require it, thus many help requests are issued in vain. In the limited view cases on the contrary, advice requests are prompted only on a restricted region, minimizing the described effect and making better use of the CCRL cognitive capabilities.

9.10 Summary

The CCRL algorithm allows a RL learner to intelligently decide whether and when to solicit advice from an advisor, by endowing it with the capabilities to evaluate its performance and to assess the value of the advice. When assisted by highly skilled advisors the agent learns to use them sufficiently frequently to improve its performance. When dealing with less skilled advisors it learns to discard bad advice and switch to autonomous learning. The CCRL algorithm and especially the combined method (CCRL with IA) achieved better results than the base-line fully autonomous learner and the learner employing IA in many learning scenarios, proving the expediency of the endowed cognitive capabilities. Furthermore, a method for simulating human advisors with various skill levels was presented.

10. Conclusions and Future Research

Chapter Overview

This concluding section presents the importance and implications of this work, and offers a comparison between the CHRL framework and algorithms presented here, and the current best practice in related research. The section ends with a discussion of areas for future work.

10.1 Conclusions

This work introduces a new reinforcement learning framework and the necessary tools for its application. The proposed framework, Collaborative Hierarchical Reinforcement Learning (CHRL) is targeted to provide efficient learning and execution of complex tasks otherwise inapplicable by a reinforcement learning agent, due to large state-action spaces (curse of dimensionality) and multiple goal states. The framework includes a two-level learning hierarchy, thus reducing the search space and allowing multiple goals, and a collaboration model, allowing human intervention for the improvement and acceleration of the learning process.

The framework suggested in this work allows complex tasks to be learned using RL methods, by defining the required set of sub-tasks and an appropriate sequence for their execution. A RL-based sequencing algorithm (SRL) is developed to address the high level learning task of determining the desired sequence, and a cognitive collaborative RL algorithm (CCRL) is introduced to enable adaptive use of human assistance when learning how to perform the various low level sub-tasks.

Extensive experimentation and analysis demonstrates the applicability of the CHRL framework and the strengths and weaknesses of the SRL and CCRL supporting algorithms. A robotic toast making application serves as a test-bed for CHRL, presenting an intelligent environment using a fixed-arm robot as a transfer agent. A simulated path planning task is used to evaluate the performance of the CCRL algorithm.

Analysis of the SRL algorithm reveals the superiority of the algorithm over the compared methods under various configurations. The algorithm does not require any predefined sequencing rules or specific information on the problem, hence can be adjusted to suit other sequencing problems presented by various applications, especially those employing a job transfer agent. This is demonstrated for a flexible manufacturing system (see Appendix III). Note that the algorithm can be used for general sequencing probems in a broader context, and not necessarily for robotics related problems as presented in this work.

The CCRL algorithm and the combined method integrating its logic with Clouse's Introspection Approach, demonstrated robust performance when dealing with various advisor skill levels, learning to accept advice received from an expert, while rejecting the aid from lesser skilled collaborators.

The CCRL algorithm can be integrated with practically any RL algorithm without requiring modifications to handle the advisors actions, since it is only added to the outer layer.

It is important to emphasize that the complete CHRL framework, presented in Chapter 4, was not fully demonstrated in this thesis, but we believe that the necessary foundations were placed for future research in this area. The aspects yet to be addressed are described in Section 10.2.1.

The following sections provide a fuller elaboration on the contributions of this research, in terms of a comparison of the CHRL framework and the suggested algorithms with the current best practice in RL-based robot learning.

10.1.1 Robot learning

Traditionally, robot behaviors are tailored for a specific task. This is not acceptable for a general-purpose robot learning system. It is noted in [Kartoun, 2008] that to become economically attractive, the robots of tomorrow will have to be constructed for a wide variety of tasks. As such, robots must be able to learn new tasks under new working conditions from its new user in its new environment. Intensive research has shown reinforcement learning to be a suitable tool for enabling such autonomous learning, but the execution of real-world complex tasks still presents many unanswered challenges.

The CHRL framework proposed in this research combines two techniques, hierarchical RL and Human-Robot collaboration, in order to scale up RL, and provide the infrastructure for the execution of intricate tasks. The novelty lies in the combination of the two methods into one complete learning framework, benefiting from the advantages of both approaches. Using the CHRL framework, this work demonstrates the applicability of RL-based methods for real-world scenarios, presenting encouraging results to support future research in this area. Another innovation in this research is the method suggested for representing various advisor skill levels, allowing the evaluation of collaboration algorithms under realistic conditions of imperfect guidance.

10.1.2 Human-robot interaction

It is well established that robot learning should make use of human intelligence in the learning process [*e.g.*, Ehrenmann *et al.*, 2001; Breazeal and Thomaz, 2008; Kartoun, 2008]. Human interaction increases the learning capabilities of a robot in realistically complex situations and further elevated robot intelligence in the post-processing and editing of learned behaviors will further elevate robot intelligence [Kartoun, 2008]. Many works have attempted to address the many challenges associated with adding an advisor to the learning process, such as the form of instruction and the manner in which the learner incorporates the knowledge to its learning function.

In many works, the rewards the agent receives are controlled or modified by a human [*e.g.*, Papudesi and Huber, 2003; Wang *et al.*, 2003], thus actually modifying the task by altering the reward function. [Thomaz and Breazeal, 2006] describe an approach in which rewards do not only provide feedback about past actions, but also provide future directions to guide subsequent actions. These methods might be problematic when non-expert collaborators are required to perform such modifications to the reward functions. The CCRL algorithm proposes a more intuitive way of collaboration, requiring the advisor only to suggest a certain action when at a certain state, thus guiding the exploration without a demand for any knowledge of the problem formulation and the reward policy (except a general notion of the goal of the task).

[Breazeal and Thomaz, 2008] indicate that most past work that incorporate human input into a Machine Learning process tend to maintain a constant level of human involvement. Several are highly dependent on guidance, learning nothing without human interaction, while other approaches are almost entirely exploration based, using limited input from a teacher. They posit that a social learner must be able to move flexibly along this guidance-exploration spectrum, explore and learn on its own, but also take full advantage of a human partner's guidance when available. The CCRL algorithm allows this flexibility using an adjustable autonomy approach, based on the model suggested by [Kartoun, 2006]. Human-robot collaboration is unnecessary as long as the robot learns policies and adapts to new states. Only when the robot senses its performance sufficient the advisor is solicited to intervene and suggest actions.

In his work, [Kartoun, 2006] suggests that advisor intervention should be triggered when the robot's learning performance is below a constant **predefined** threshold, set in the context of the problem (*e.g.*, number of steps in a navigation problem, accumulated rewards received during the learning process in other problems). This approach is problematic since in order to determine the appropriate threshold, one must have *a-priori* task-specific information about the solution, which obviously does not exist in practice. The CCRL algorithm resolves this issue by setting a threshold in terms of improvement rate, expecting only certain performance improvements during the learning, without the necessity for any preliminary knowledge. Furthermore, Kartoun's model refers only to the $Q(\lambda)$ algorithm, while CCRL can be used with any RL algorithm.

As mentioned in the introduction section, most previous research assumes perfect advisors, suggesting only optimal actions. Furthermore, even the few approaches found in the literature to consider less skilled advisors (*e.g.*, Clouse, 1996; Cetina, 2007; Breazeal and Thomaz, 2008), assume that though the advisor may not provide helpful advice, it at least does not interfere with the robot's learning process. However, there are certain contexts where incompetent advisors will damage the learning process. Such an example is when dealing with extremely tired workers or children instructing domestic service robots. This research confronts this problem with an innovative approach, endowing the learner with a higher cognitive level, enabling it not only to decide when to solicit advice, but also to conclude that the advisor is not promoting the learning, hence should be discarded.

10.1.3 Adaptive sequencing

Job sequencing problems are problems in which decision makers must determine the production sequence of the jobs awaiting their next process in the machine queue. A common approach to address such problems is to adopt dispatching rules - priority rules used to determine the order in which the jobs are to be processed. The drawback of using dispatching rules is that in order to implement them a complete model of the system is required. Furthermore, for dynamically assigning dispatching rules there is a need to continually compute system parameters, such as: system utilization, relative machine workloads (points system bottlenecks), machine homogeneity, etc. The SRL algorithm presented here avoids these requirements by utilizing the basic characteristics of reinforcement learning - the lack of necessity for a complete model of the environment, and the relative ease in problem formulation. When implementing it, there is no need for predefining desirable or undesirable intermediate states. All that must be done is to construct a fairly simple reward policy (*e.g.*, higher reward for shorter completion times) and the algorithm will supply a solution. Furthermore, the solution is tailored for the specific parameters of the problem.

This work also suggests an alternative view of the sequencing problem, referring to the robotic transfer agent as the limited resource, and to the tasks it has to perform as the “jobs” waiting in its queue. This view can simplify the formulation of such problems.

10.2 Future Research

Many research areas remain open for future expansion of this work:

10.2.1 Complete implementation of the CHRL framework

This work presents implementations of the fundamental aspects of the CHRL framework, but a full demonstration including all of the elements is still to be achieved. The toast making test-bed application can serve this purpose, with the following elements to be addressed:

(i) An integration of both the learning of the low level activities and the learning of the required sequence into one framework. After integration, the transferring times used by the SRL algorithm will be dynamically updated. First, the robot will learn how to perform the sub-tasks in an optimal way. Then the found operation times would be **automatically** taken as an input for the SRL algorithm, leading to the generation of the execution sequence. This sequential process will lead to optimal task execution.

(ii) The noted "toolbox" should be applied, saving low level sub-tasks, such as grasping a toast, to be used recurrently during the task execution. This should be done to demonstrate the simplicity of the formation and execution of new composite tasks, based on already learned sub-tasks.

(iii) The introduction of human advice into the high-level learning task.

10.2.2 Further evolution and development of the CCRL algorithm

The CCRL algorithm, demonstrating good results for the path planning task presented by the toasting system and for the more complex simulated task, should be implemented with other robotic systems and other machine learning methods. Further evaluation should also be conducted for various configurations of the maze environment. Furthermore, advanced human-robot interfaces could be introduced to enhance the interaction. The use of natural language control or virtual reality technologies could replace the standard interfaces (mouse, keyboard and screen) to achieve more intuitive interaction and extend human control capabilities.

As a next step, the algorithm could address an opposite direction of knowledge transfer. After the agent assess the worth of the advice given, it can notify the advisor as to the worth of his suggestions, thus allowing him to improve his knowledge and guidance for future interaction, and provide better suggestions.

10.2.3 Further assessment and development of the SRL algorithm

The SRL algorithm could be applied for other systems, for further evaluation. An initial study was conducted for a Flexible Manufacturing System (see Appendix III). The algorithm can be used to minimize machine or robot idle times, mean tardiness, number of tardy jobs etc. Furthermore, its performance could be compared to that of other soft computing methods such as genetic algorithms.

One issue not addressed in this research is the method in which the complex task is decomposed to the sub-tasks when creating the two-level hierarchy. Previous work [*e.g.*, Bakker and Schmidhuber, 2004] presented methods in which the high-level policies not only select the sequence of the sub-task execution, but also autonomously discover and define those sub-tasks. The deficiency is that these methods are appropriate only for relatively defined environments (*e.g.*, grid-worlds for path planning tasks), and cannot deal with complex tasks as toast making. This issue should be addressed in future research. Furthermore, the formulation of the minimum makespan problem as a single complex problem without the two-level decomposition could be investigated. If this succeeds, a comparison of the decomposed (two level hierarchy) and full problem results should be made.

10.2.4 Improvement of the learning algorithms

Though preliminary results for both the algorithms are promising, there are some issues that should be attended to in future research:

(i) The algorithms include many parameters which can significantly influence the performance. Evaluation methods could be employed to find the optimal RL and threshold parameters leading to the best performance.

(ii) The low level learning tasks described in this work consist of relatively small state-action spaces. In problems with extremely large state-action spaces it is infeasible to use tables for holding the Q values, due to the huge amount of memory and long running times required to maintain the tables. In these cases, function approximators employing only a representative sub-set of the entire state-space are used, to achieve practical performance. The algorithms suggested in this work should be modified to allow the use of such approximators when applied to larger problems.

(iii) Current research employs multi-agent methods to accelerate and upgrade the learning, profiting from the advantages of using parallel computation. Again, the algorithms presented here can be modified to suit multi-agent models. The application of the CCRL cognitive model for multi-agent learning, where agents could function both as learners and as advisors, should be especially interesting, addressing the issue of how to handle advice in such a way that facilitates the inclusion of advice from several sources (with possible conflicting advice because of different skill levels of the advisors). The advice could be weighted differently or organized as a hierarchical mixture of experts.

11. References

[1] Bakker B. and Schmidhuber J., Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization, *Proceedings of the 8th Conference on Intelligent Autonomous Systems*, Amsterdam, The Netherlands, pp. 438-445, 2004.

[2] Bechar A., Edan Y. and Meyer J., Performance Measurement of Collaborative Human-Robot Target Recognition Systems in Unstructured Environments, *Proceedings of the 17th International Conference on Production Research (ICPR)*, Blacksburg, Virginia, Paper No. 0321, 2003.

[3] Bechar A., Edan Y. and Meyer J., Optimal Collaboration in Human-Robot Target Recognition Systems, *IEEE Conference on Systems, Man, and Cybernetics*, Taipei, Taiwan, Paper No. 01113, 2006.

[4] Berman S., CIM-NEGEV – User Manual, *Department of Industrial Engineering and Management, Ben-Gurion University of the Negev*, 2003.

[5] Bhanu B., Leang P., Cowden C., Lin Y. and Patterson M., Real-time Robot Learning, *Proceedings of the IEEE International Conference on Robotics and Automation*, Seoul, Korea, pp. 491-498, 2001.

[6] Breazeal C. and Thomaz A., Learning from Human Teachers with Socially Guided Exploration, *IEEE International Conference on Robotics and Automation*, Pasadena, California, USA, pp. 3539-3544, 2008.

[7] Cetina V. U., Supervised Reinforcement Learning Using Behavior Models, *IEEE Computer Society 6th International Conference on Machine Learning and Application*s, Cincinnati, Ohio, USA, pp. 336-341, 2007.

[8] Clouse J. A., An Introspection Approach to Querying a Trainer, *Technical Report 96-13*, University of Massachusetts, Amherst, MA, 1996.

[9] Connell J. H. and Mahadevan S., Robot Learning, *Kluwer Academic Publishers*, Boston, 1993.

[10] Creighton D. C. and Nahavandi S., The Application of a Reinforcement Learning Agent to a Multi-Product Manufacturing Facility, *IEEE International Conference on Industrial Technology*, Bangkok, Thailand, pp. 1229-1234, 2002.

[11] Dietterich T. G., Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition, *Journal of Artificial Intelligence Research*, Vol. 13, pp. 227–303, 1999.

[12] Ehrenmann, M., Rogalla, O., Zollner, R. and Dillmann, R., Teaching Service Robots Complex Tasks: Programming by Demonstration for Workshop and Household Environments, *Proceedings of the International Conference on Field and Service Robots*, Helsinki, Finland, vol. 1, pp. 397-402, 2001.

[13] Fong T., Thorpe C., Baur C., Collaboration, Dialogue, and Human-Robot Interaction, *10th International Symposium of Robotics Research*, Lorne, Victoria, Australia, 2001.

[14] Gabel T. and Riedmiller M., Scaling Adaptive Agent-Based Reactive Job-Shop Scheduling to Large-Scale Problems, *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling*, Honolulu, Hawaii, pp. 259-266, 2007.

[15] Gil A, Stern H, Edan Y. and Kartoun U., A Scheduling Reinforcement Learning Algorithm, *Proceedings of the ASME International Symposium on Flexible Automation*, Atlanta, Georgia, USA, 2008.

[16] Glorennec P. Y., Reinforcement Learning: an Overview, *European Symposium on Intelligent Techniques*, Aachen, Germany, 2000.

[17] Hayes-Roth, F., Klahr, P. and Mostow, D. J., Advice-taking and Knowledge Refinement: An Iterative View of Skill Acquisition, *Technical Report*, Rand Corporation, 1980.

[18] Heguy O., Rodriguez N., Luga H., Jessel J. P. and Duthen Y., Virtual Environment for Cooperative Assistance in Teleoperation, *The 9th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, Plzen, Czech Republic, 2001.

[19] Honglak L., Yirong S., Chih-Han Y., Gurjeet S. and Andrew Y. N., Quadruped Robot Obstacle Negotiation via Reinforcement Learning, *Proceedings of the IEEE International Conference on Robotics and Automation*, Orlando, Florida, USA, pp. 3003-3010, 2006.

[20] Howard A., Probabilistic Navigation: Coping With Uncertainty in Robot Navigation Tasks, *Ph.D. Dissertation, Department of Computer Science and Software Engineering, University of Melbourne*, Australia, 1999.

[21] Jeni, L.A., Istenes, Z., Korondi, P. and Hashimoto, H., Hierarchical Reinforcement Learning for Robot Navigation using the Intelligent Space Concept, *11th International Conference on Intelligent Engineering Systems*, Budapest, Hungary, pp. 149 – 153, 2007.

[22] Kartoun U., Stern H., Edan Y., Feied C., Handler J., Smith M. and Gillam M, Collaborative $Q(\lambda)$ Reinforcement Learning Algorithm - A Promising Robot Learning Framework, *IASTED International Conference on Robotics and Applications*, Cambridge, U.S.A, 2005.

[23] Kartoun U., Stern H., Edan Y., Human-Robot Collaborative Learning System for Inspection, *Conference on Systems, Man, and Cybernetics*, Taipei, Taiwan, Paper No. 01191, 2006.

[24] Kartoun U., Human-Robot Collaborative Learning Methods, *Ph.D. Dissertation, Department of Industrial Engineering and Management, Ben-Gurion University*, Israel, 2008.

[25] Kreuziger J., Application of Machine Learning to Robotics - an Analysis, *Proceedings of the Second International Conference on Automation, Robotics and Computer Vision*, Singapore, 1992.

[26] Lockerd A. and Breazeal C., Tutelage and Socially Guided Robot Learning, *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, pp. 3475-3480, 2004.

[27] Martínez-Marín T. and Duckett T., Fast Reinforcement Learning for Vision-guided Mobile Robots, *Proceedings of the IEEE International Conference on Robotics and Automation*, Barcelona, Spain, pp. 4170-4175, 2005.

[28] Mihalkova L. and Mooney R., Using Active Relocation to Aid Reinforcement, *Proceedings of the 19th International FLAIRS Conference*, Melbourne Beach, Florida, USA, 2006.

[29] Papudesi V. N. and Huber M., Learning From Reinforcement and Advice Using Composite Reward Functions, *Proceedings of the 16th International FLAIRS Conference*, St. Augustine, Florida, USA, pp. 361-365, 2003.

[30] Papudesi V. N., Wang Y., Huber M. and Cook D. J., Integrating User Commands and Autonomous Task Performance in a Reinforcement Learning Framework, *AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments*, Stanford University, CA, 2003

[31] Park S. C., Raman N. and Shaw M. J., Adaptive Scheduling in Dynamic Flexible Manufacturing Systems, *IEEE Transactions on Robotics and Automation*, Vol. 13, No. 4, pp. 486-502, 1997.

[32] Peng J. and Williams R., Incremental Multi-Step Q-learning, *Machine Learning*, Vol. 22, No. 1-3, pp. 283-290, 1996.

[33] Ribeiro C., Reinforcement Learning Agents, *Artificial Intelligence Review*, Vol. 17, No. 3, pp. 223-250, 2002.

[34] Sheridan T. B., Supervisory Control, In: *Handbook of Human Factors/Ergonomics*, G. Salvevely (Ed.), Wiley, ISBN 0471116, NY, 1987.

[35] Singh S., Transfer of Learning by Composing Solutions of Elemental Sequential Tasks, *Machine Learning*, Vol. 8 No. 3-4, pp. 323–339, 1992.

[36] Stefan P., Flow-Shop Scheduling Based on Reinforcement Learning Algorithm, *Production Systems and Information Engineering*, Miskolc, Vol. 1, pp. 83-90, 2003.

[37] Sun R. and Sessions C., Self-Segmentation of Sequences: Automatic Formation of Hierarchies of Sequential Behaviors, *IEEE Transaction on Systems, Man and Cybernetics*, Vol. 30, No. 3, pp. 403-418, 2000.

[38] Sutton R. S. and Barto A. G., Reinforcement Learning: An Introduction, *Cambridge, MA: MIT Press*, 1998.

[39] Tham C. K. and Prager R.W., A Modular Q-Learning Architecture for Manipulator Task Decomposition, *International Conference on Machine Learning*, 1994.

[40] Thomaz A. and Breazeal C., Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance, *Proceedings of the 21st National Conference on Artificial Intelligence*, Boston, Massachusetts, USA, 2006.

[41] Touzet C. F., *Q*-learning for Robots, In: *Handbook of brain theory and neural networks*, Cambridge, MA: M. Arbib editor, MIT Press, pp. 934-937, 2003.

[42] Wang B., Li J. W. and Liu H., A heuristic Reinforcement Learning for Robot Approaching Objects, *IEEE Conference on Robotics, Automation and Mechatronics*, pp. 1-5, 2006.

[43] Wang Y. and Usher J.M., Application of Reinforcement Learning for Agent-Based Production Scheduling, *Engineering Applications of Artificial Intelligence*, Elsevier, Vol. 18, pp. 73–82, 2005.

[44] Wang Y., Huber M., Papudesi V. N. and Cook D. J., User-Guided Reinforcement Learning of Robot Assistive Tasks for an Intelligent Environment, *Proceedings of the IEEE/RJS International Conference on Intelligent Robots and Systems*, pp. 424-429, 2003.

[45] Watkins C. J. C. H., Learning from Delayed Rewards, *Ph.D. Dissertation, Cambridge University*, 1989.

[46] Watkins C. J. C. H. and Dayan P., *Q*-learning, *Machine Learning*, Vol. 8, pp. 279-292, 1992.

[47] Wei Y. and Zhao M., Composite Rules Selection Using Reinforcement Learning for Dynamic Job-Shop Scheduling, *Proceedings of the IEEE Conference on Robotics, Automation and Mechatronics*, Singapore, pp. 1083-1088, 2004.

[48] Yanco H. A., Baker M., Casey R., Chanler A., Desai M., Hestand D., Keyes B. and Thoren P., Improving Human-Robot Interaction for Remote Robot Operation, Robot Competition and Exhibition Abstract, *National Conference on Artificial Intelligence*, Pittsburgh, Pennsylvania, USA, 2005.

Appendix I. Toast Making System – Specifications and Operation

This appendix reviews the toasting system's components, methods of operation and user interfaces.

System Components

The system includes several software and hardware components, exchanging data as described in Fig. I.1.

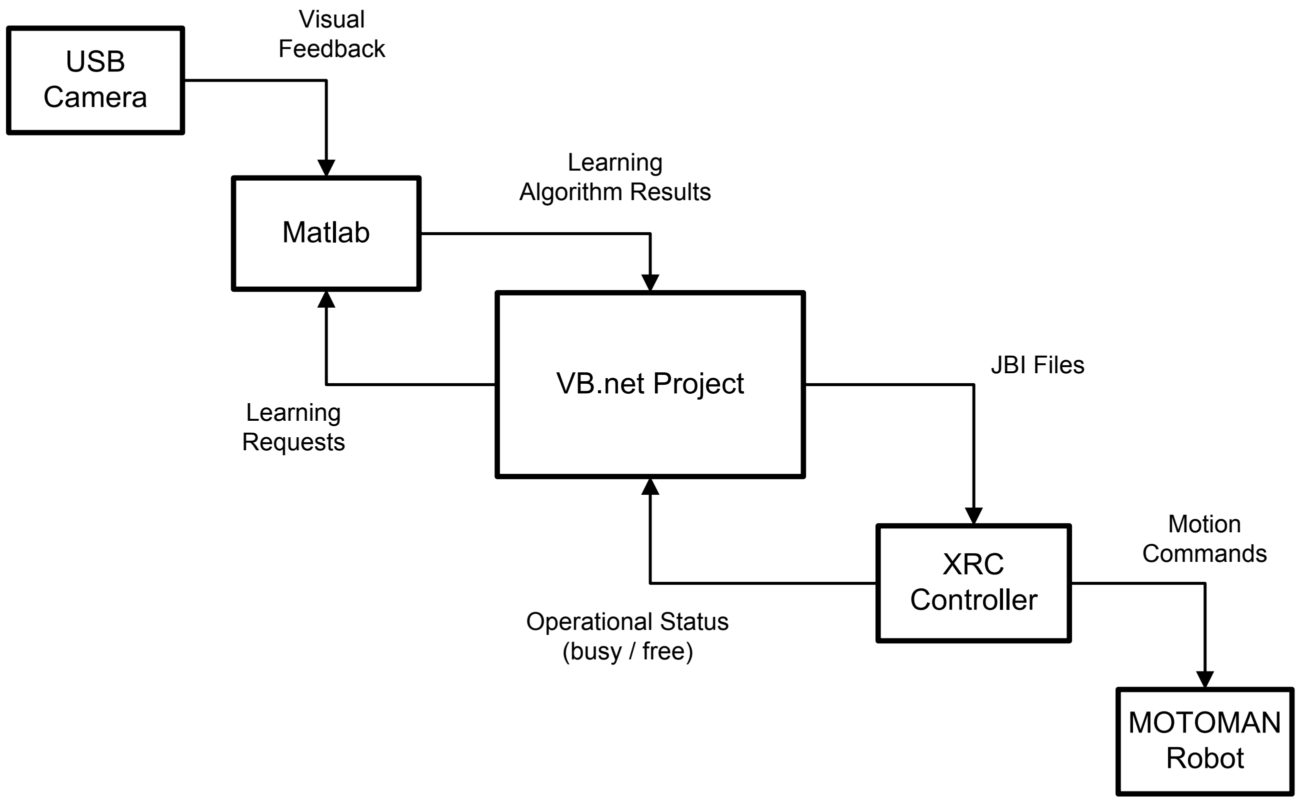


Fig. I.1 System components and data exchange

- **VB.net application** – this is the system's framework, managing and synchronizing other system components. The VB.net project[1] manages the communication with robot's controller: it creates the motion command files for robot (JBI files), downloaded to the XRC controller, and receives indication of the operation status from the controller. It also sends the learning requests to the MATLAB application employing the learning algorithms, and receives the desired results (robot movements for the high-level sequencing task or path for the low-level bread insertion task). Finally, the VB.net project includes the user interfaces.

[1] The VB.net source code is presented in Appendix VII.

- **MATLAB application** – the MATLAB application employs the SRL and CCRL learning algorithms[1] according to the requests received from the VB.net application. Furthermore, the MATLAB application applies the image processing algorithm and displays parts of the user interface for the low-level task.
- **Motoman six degrees of freedom UP-6 Fixed-Arm Robot** – this is the “transfer agent” of the system. The robot (Fig. I.2) is operated according to movement commands received from the XRC controller.
- **XRC controller –** the robot’s controller (Fig I.3), executing the programs generated by the VB.net project (JBI files), sending motion commands to robot.
- **USB camera –** a simple USB camera (Fig. I.4), located above the experimental setup, supplies the visual feedback for the image processing algorithm employed in MATLAB.
- **Experimental setup –** a cardboard mockup of system stations for the sequencing task (Fig I.5), and a table with obstacles and the toaster for the bread insertion task (Fig. I.6).

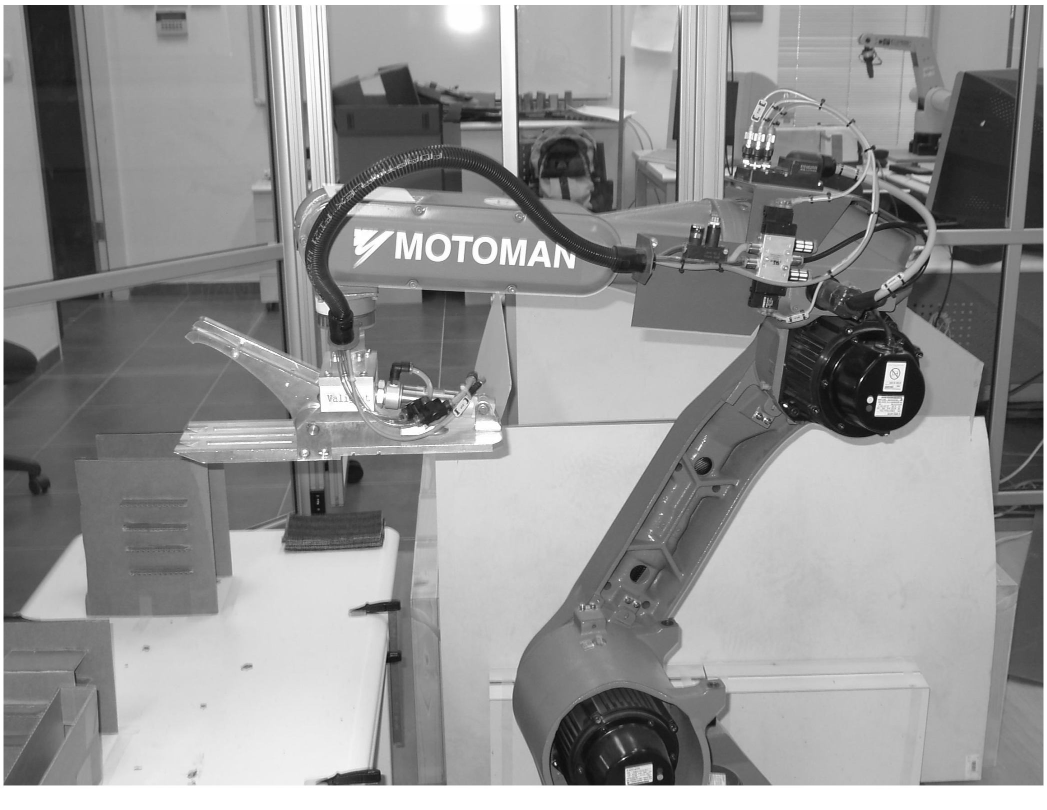


Fig. I.2 The Motoman UP-6 fixed-arm robot

[1] The Matlab source code is presented in Appendix VII.

Fig. I.3 The XRC controller

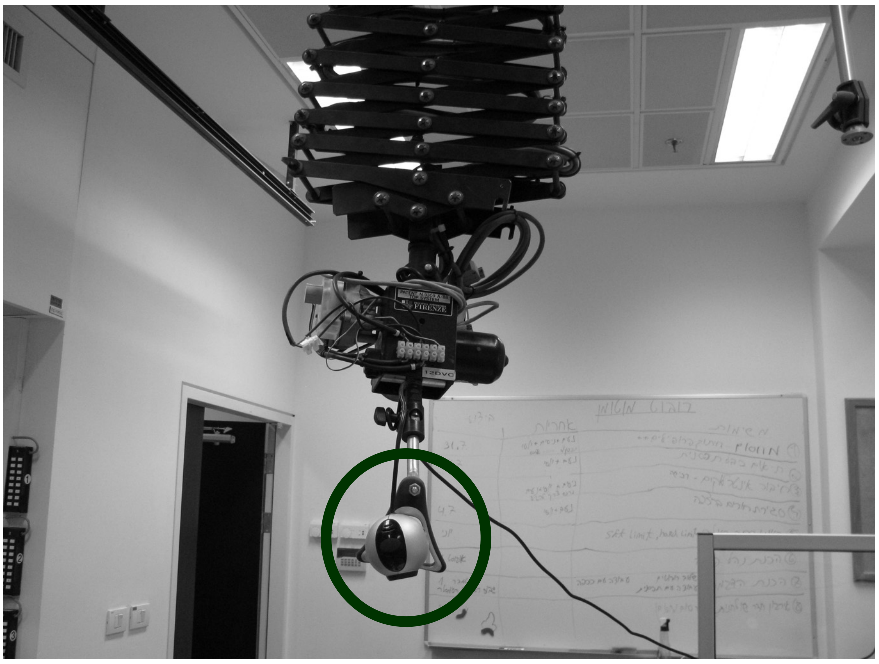


Fig. I.4 The USB camera located above the experimental setup

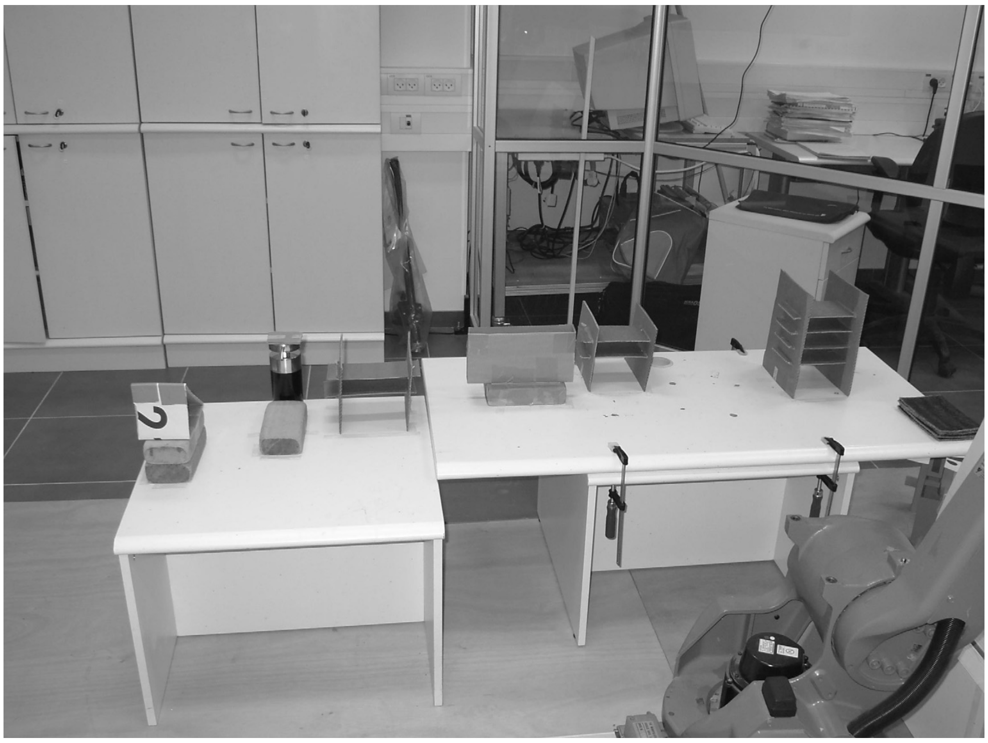


Fig. I.5 The sequencing task experimental setup

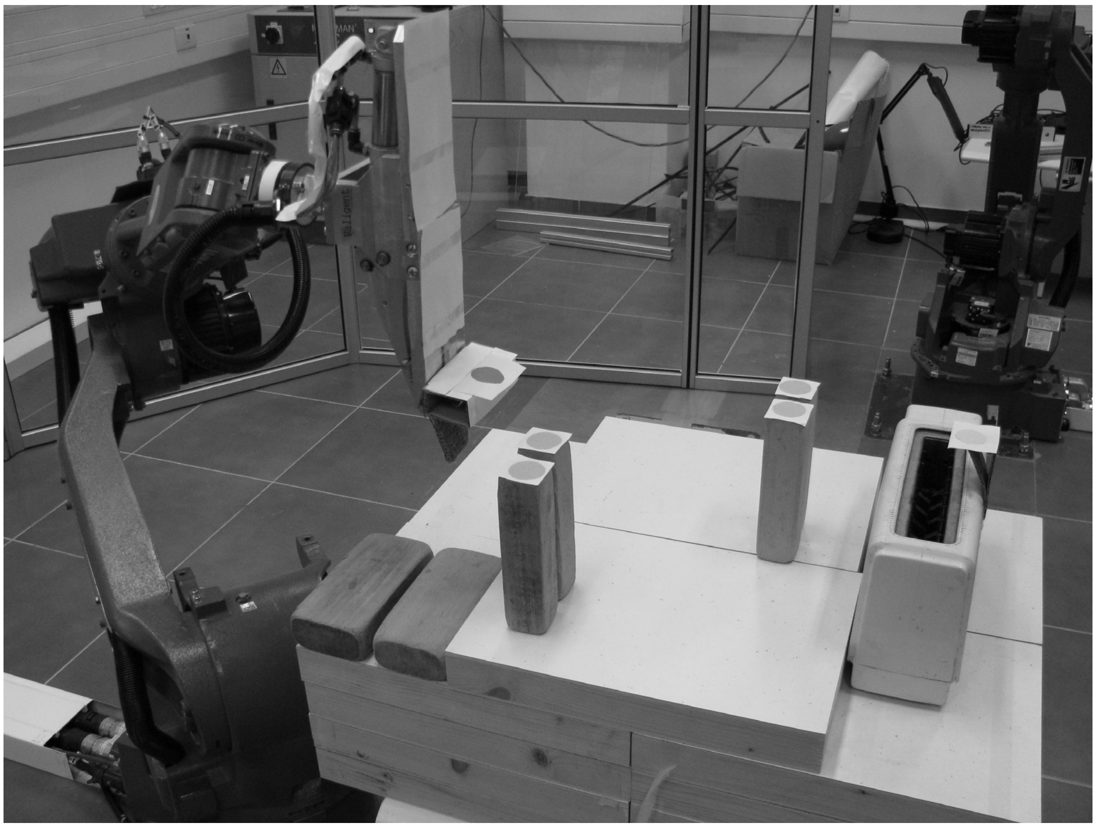
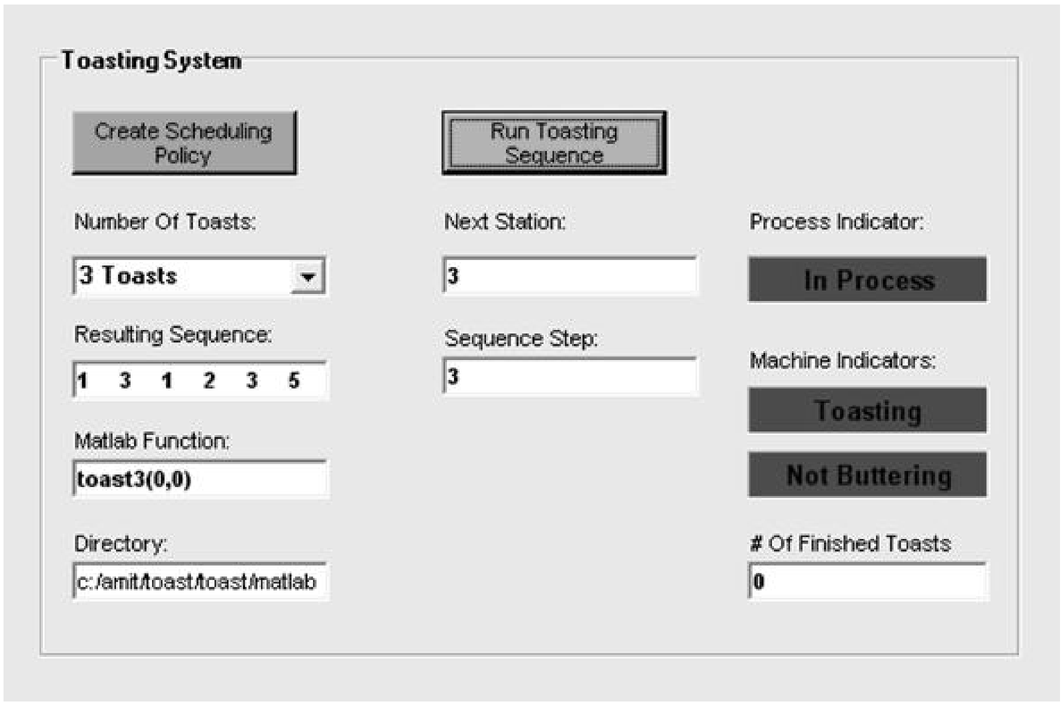


Fig. I.6 The bread insertion task experimental setup

Toast Transfer Sequencing Task

The system's user interface is displayed in Fig. I.7.



Number Of Toasts: 3 Toasts
Resulting Sequence: 1 3 1 2 3 5
Matlab Function: toast3(0,0)
Directory: c:/amit/toast/toast/matlab
Next Station: 3
Sequence Step: 3
Of Finished Toasts 0

Fig. I.7 The sequencing task user interface

The setup (Fig. I.5) includes the six cardboard stations situated on two tables. The bread-slices are preliminary located in station 1 (starting buffer), to be transferred by the robot during operation.
The system is operated in the following order:

1) The user chooses the desired number of toasts (1-4) and presses the "Create Scheduling Policy" button.
2) The MATLAB application runs the SRL algorithm and generates the sequence of robot movements suitable for the desired number of toasts.
3) The VB.net application receives the sequence.
4) When the user presses the "Run Toasting Sequence" button, the application dynamically creates the robot movement command files (JBI files) suitable for the sequence.

During the system's operation, the machines status, robot's next location (station), step of the sequence and number of finished toasts are presented to the user.

A flowchart of the process is presented in Fig. I.8.

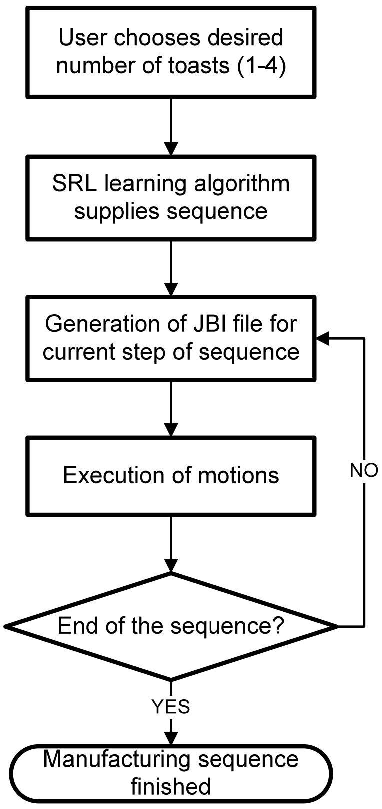

User chooses desired number of toasts (1-4)
SRL learning algorithm supplies sequence
Generation of JBI file for current step of sequence
Execution of motions
End of the sequence?
NO
YES
Manufacturing sequence finished

Fig. I.8 Flowchart of the sequencing task operation

Bread Insertion Task

The system's user interface is displayed in Fig. I.9.

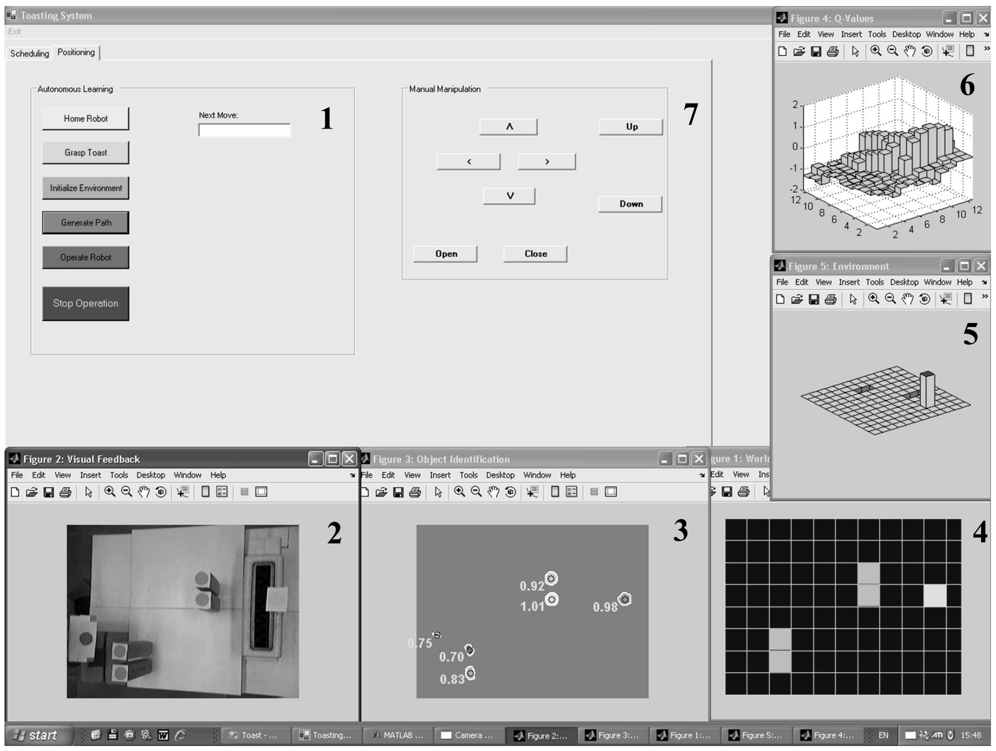


Fig. I.9 The bread insertion task user interface

The setup (Fig. I.6) includes a table on which the obstacles (wooden cubes) and the toaster are located. The bread-slice is preliminary located on the corner of the table, to be taken by the robot during operation. The task is performed with the following steps:

1) The robot grasps the bread-slice and moves to the starting location.

2) A snapshot of the environment is taken using a simple USB camera (displayed to the user in window 2 in Fig. I.9).

3) An image proceccing algorithm (running in MATLAB) is used locate the objects (robot's gripper, obstacls and toaster) and build a simulated model of environment accordingly. The objects are recognized using round markers in differnet colors (displayed in window 3). The simulated environment is a 12×12 grid world (displayed in windows 4 and 5).

4) A MATLAB simulation applying the CCRL algorithm (based on *Q(λ)*) is employed to learn the optimal path from the starting state to the goal state in the simulated world. During the learning process, the agent traversing the simulaetd environment and the *Q* table are displayed (Fig I.10). When required, user interaction messages are prompted: when the agent senses that its performance does not improve fast enough, a request for advice is prompted. The human advisor is then required to guide the agent using the user interface shown in Fig I.11. If the agent concludes the advice given is not beneficial, it switches to fully autonomous learning, and notifies the advisor. The resulting *Q* table is displayed to the user in window 6.

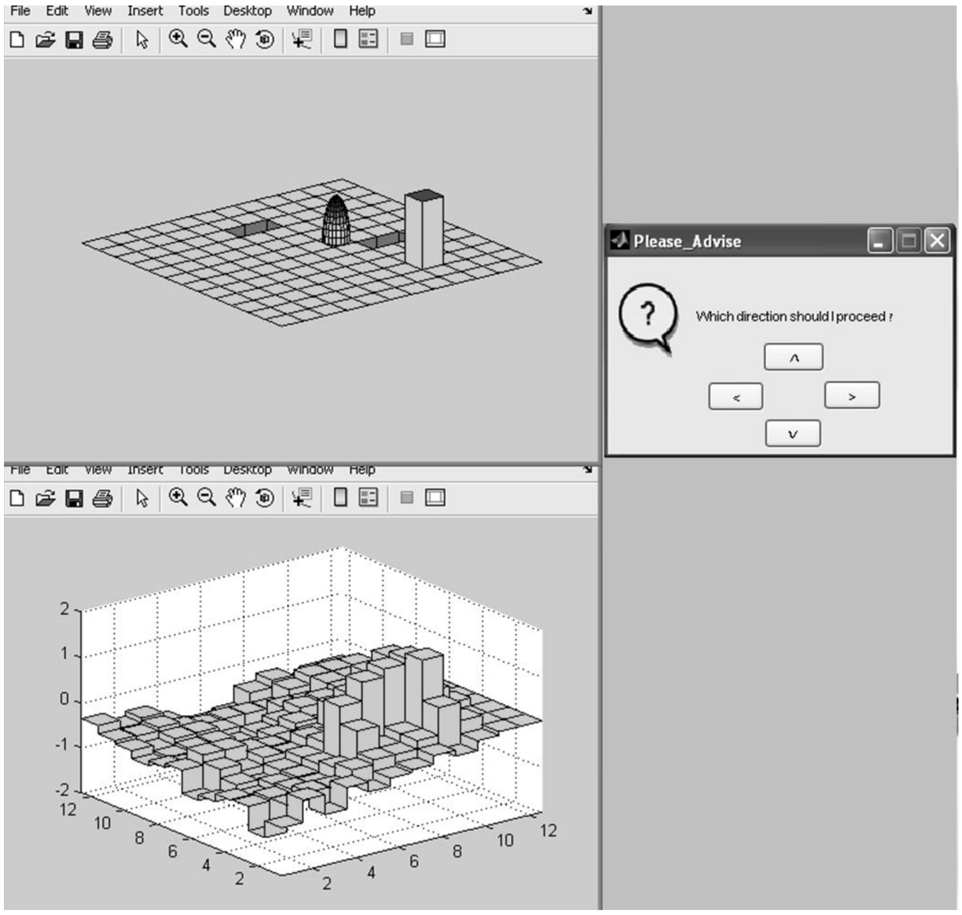
5) The robot is operated according to the generated path. Image processing is used to identify the location of the robot and syncrozine the location in the simulated environment to the location in the real world (displayed in windows 3 and 4).

6) After arriving to the desired location above the toaster, the bread is lowered and the gripper is opened to release it into the toaster.

The buttons for operating the various steps are located in window 1. Furthermore, the system allows manual control over the robot's movements and gripper's status (open / close) using the interface displayed in window 7.

A flowchart of the process is presented in Fig. I.12.

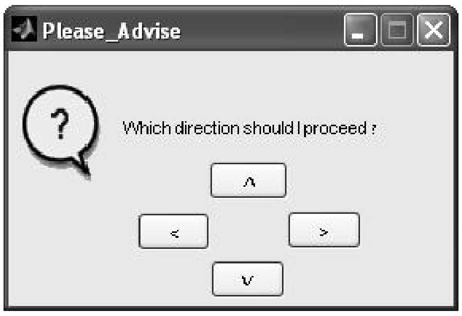
.



Which direction should I proceed ?

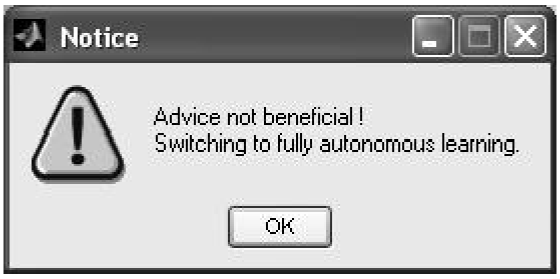
^ < > v

Fig. I.10 Simulated learning environment



Which direction should I proceed ?

^ < > v



Advice not beneficial !
Switching to fully autonomous learning.

OK

(a) Guidance request (b) Autonomous learning notice

Fig. I.11 User interaction messages

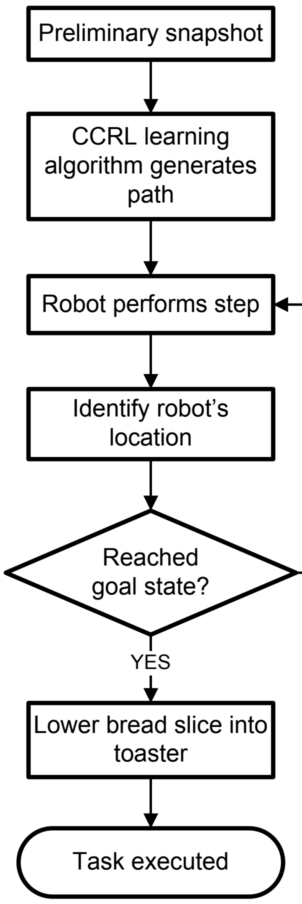

Preliminary snapshot
CCRL learning algorithm generates path
Robot performs step
Identify robot's location
NO
Reached goal state?
YES
Lower bread slice into toaster
Task executed

Fig. I.12 Flowchart of the bread insertion task operation

Appendix II. Statistical Analysis[1]

For the analysis of the results for both the sequencing task of the toast making system (Section 7.7) and the simulated path planning task (Section 9.9) there is a necessity to compare the means of three groups - SRL, MC and random search for the sequencing task, and CCRL, IA and a combined method for the path planning task.

The comparisons are performed using one-way ANOVA (analysis of variance) and Tukey's HSD test. The ANOVA analysis employs an F-test to determine whether there is a significant difference between two or more of the means. Tukey's HSD is a post hoc multiple comparisons test, performed after the F-test determines that the means aren't equal. The HSD test separates and ranks the groups demanding 95% confidence level ($\alpha = 0.05$) for the entire comparison.

Fig. II.1 shows an example for the results of the tests for the deterministic 3-toasts sequencing problem (time set I, 30 episode sessions).

ANOVA

Percentage

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	2938.867	2	1469.433	93.794	.000
Within Groups	423.000	27	15.667		
Total	3361.867	29			

Percentage

			Subset for alpha = .05	
	Method	N	1	2
Tukey HSD[a]	MC	10	49.9000	
	Random	10		68.8000
	SRL	10		72.5000
	Sig.		1.000	.111

Means for groups in homogeneous subsets are displayed.

a. Uses Harmonic Mean Sample Size = 10.000.

Fig. II.1 ANOVA and Tukey's HSD test results

The null hypothesis of the ANOVA F-test is that the means are equal. The P-value of the test is 0 ("Sig." column), meaning that the null hypothesis is rejected, and that there is a significant difference between the means. Tukey's HSD test separates the means into two separate groups: SRL and the random search in one group, and MC on the other. Both SRL and the random search achieved

[1] All the statistical tests presented here are performed using SPSS 15.0 software.

significantly better success percentages than the MC method, but there is no significant different between them.

The ANOVA's F-test has 3 assumptions: (i) Independence of the groups, (ii) Normality of the distributions and (iii) Equal variances.

The independence of the groups results from the design of the experiments. The Normality of the distributions is validated using one-way Kolmogorov-Smirnov (K-S) test. Fig II.2 shows the test's result for the SRL group of the above example.

One-Sample Kolmogorov-Smirnov Test

		Percentage
N		10
Normal Parameters[a,b]	Mean	72.5000
	Std. Deviation	3.86580
Most Extreme Differences	Absolute	.217
	Positive	.151
	Negative	-.217
Kolmogorov-Smirnov Z		.687
Asymp. Sig. (2-tailed)		.732

a. Test distribution is Normal.

b. Calculated from data.

Fig. II.2 Kolmogorov-Smirnov test results

The test's null hypothesis is that the sample distribution is Normal. The high P-value (0.732) indicates that the null hypothesis is not rejected, meaning that the distribution is indeed Normal.

The equality of the groups' variances in examined using Leven's test for homogeneity of variances. Fig II.3 shows the test's result for the groups of the above example.

Test of Homogeneity of Variances

Percentage

Levene Statistic	df1	df2	Sig.
.007	2	27	.993

Fig. II.3 Leven's test results

Leven's test null hypothesis is that the variances are equal, and the high P-value (0.993) indicates that the null hypothesis is not rejected, and that the variances are equal.

Appendix III. Task Sequencing for a Flexible Manufacturing System (FMS)

This appendix describes the application of the SRL algorithm to a second system, a flexible manufacturing system performing machining operations, to further examine its performance.

Introduction

The CIM-NEGEV system in Ben-Gurion University [Berman, 2003] is a decentralized manufacturing system composed of a central on-line data base, an Automated Storage and Retrieval System (ASRS), a Flexible Manufacturing System (FMS), an assembly station, a quality control station and a material handling system based on Automated Guided Vehicles (AGVs). Here we shall focus only on the FMS station which performs machining operations. In this station a fixed-arm robot transfers the material from location to location, executing such tasks as unloading raw material from the AGVs, loading and unloading the various machine-tools, moving pallets from in-buffers to out-buffers etc.

In conventional terms, the FMS presents an *n* job flow-shop problem with several types of jobs (parts), deterministic and stochastic job arrival times, three machines, limited buffers, and a single job transfer agent (robot). However, as explained in the Methodology chapter, the problem can also be viewed as a much simpler job sequencing problem, in which the robot is the limited resource, and the part transfer tasks are the “jobs” waiting in its queue, requiring a different “process time” (robot transition time).

The SRL algorithm is applied here with the learning objective of finding task execution policies (sequences) that will minimize the completion time of production of various manufacturing orders. The performance of algorithm is compared to that of the currently employed FIFO policy.

Experimental Setup and Task Definition

The FMS (Figs. III.1, III.2) consists of three machines - one Mill (Emco VMC-100) and two lathes (Emco Compact 5 CNC), a 5 degree-of-freedom fixed-arm robot (Intelitek ER-IX) situated on a linear slide-base and a docking station for Automated Guided Vehicles (AGVs). Raw material arrives to the docking station on pallets carried by an AGV. They are loaded to the machines by the robot, processed by the machines, and finally taken by the robot back to the AGV in the docking station as finished products, to be transferred to their next destination in the CIM system.

Each machine has an in-buffer and an out-buffer with a capacity of one pallet. The docking station also has a capacity of one incoming pallet and one outgoing pallet (two spaces on the AGV itself). System stations are listed in Table III.1. Another feature of the system is that after milling, there is a necessity to vacuum the mill area, in order to allow proper loading of the next part to be processed. The vacuuming is done using a designated vacuum cleaner operated by the robot.

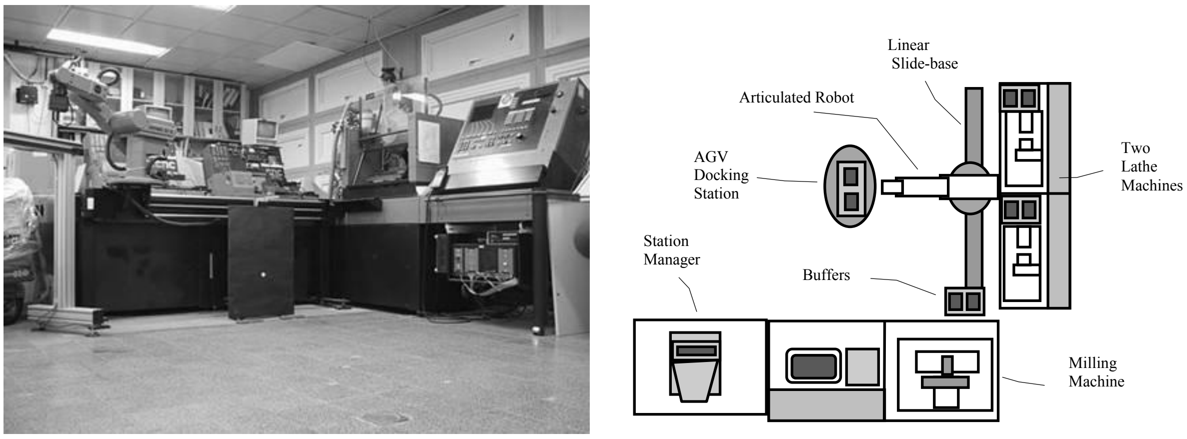


Fig. III.1 Experimental setup - FMS layout and general scheme

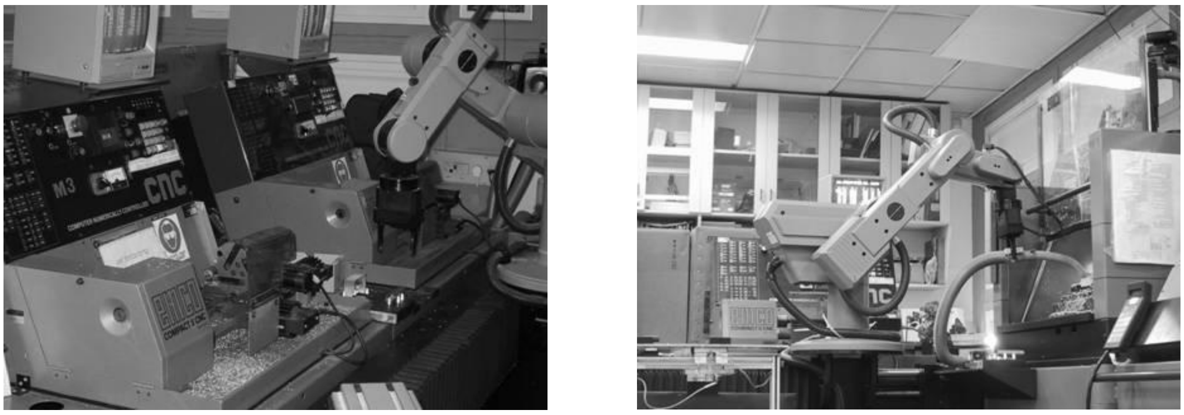


Fig. III.2 Experimental setup – Lathes, mill and robot

The FMS produces four kinds of products: “Sign”, “Hole-Axis”, “Box” and “Rook”. In this work we consider the manufacturing of only two of the products (Fig. III.3): Sign, manufactured from a Perspex box by a milling process, and Rook, manufactured from a brass cylinder by a turning process. The processing times are 605 seconds for the Sign and 185 seconds for the Rook. The robot transition times are specified in Table III.2.

A general process flow for a part in the system: pallet with raw material arrives to the docking station on the AGV; the pallet is transferred by the robot to the machine in-buffer; the raw material is loaded to the machine by the robot; the pallet is transferred from the in-buffer to the out-buffer; after the machine finished processing the part is unloaded by the robot and placed back on the pallet in the out-buffer; the pallet carrying the finished part is transferred by the robot to the docking station departure point.

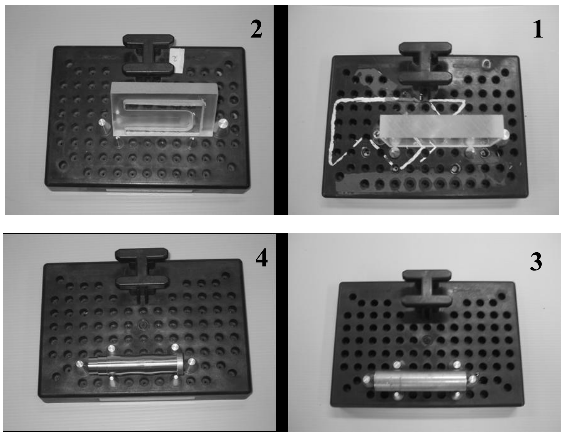


Fig. III.3 FMS products - Sign (1-raw material, 2-processed); Rook (3-raw material, 4-processed)

The FMS receives orders for the manufacturing of a certain product mix. The raw materials for the products of each manufacturing order arrive in a known order and with certain inter-arrival times. The process flow of each part is decomposed to several tasks, such as: pallet transfer from docking station to machine in-buffer, material loading to machine, pallet transfer from in-buffer to out-buffer, unloading part from machine, etc. Currently, these tasks are executed by the robot according to a task queue managed by a first-in-first-out (FIFO) policy. Each time the robot finishes a task related to a certain part, the next task of the part's process flow is inserted to the queue, and the robot is assigned a new task from the head of the queue. The FIFO policy assures that simultaneous production of several parts will be possible without collisions in robot action requests, but it is not directly concerned with minimizing makespan. The SRL algorithm is used here with the objective of finding task execution policies (sequences) that will minimize the makespan of production of various manufacturing orders.

Table III.1 System stations

No.	Station
1	Incoming raw material buffer
2	Mill in-buffer
3	Mill (machine)
4	Mill out-buffer
5	Lathe I in-buffer
6	Lathe I (machine)
7	Lathe I out-buffer
8	Lathe II in-buffer
9	Lathe II (machine)
10	Lathe II out-buffer
11	Outgoing finished products buffer

Table III.2 Robot transition times (Sec.)

To Station

From Station	1	2	3	4	5	6	7	8	9	10	11
1	X	65	15	45	45	45	25	25	25	40	X
2	25	X	20	45	45	45	25	25	25	40	X
3	23	45	X	45	45	45	25	25	25	40	40
4	45	45	45	X	15	50	50	50	50	60	X
5	45	45	45	25	X	20	50	50	50	60	X
6	45	45	45	25	45	X	50	50	50	60	60
7	25	25	25	50	50	50	X	50	15	30	X
8	25	25	25	50	50	50	25	X	25	30	X
9	25	25	25	50	50	50	25	25	X	30	30
10	40	X	X	60	X	X	30	X	X	X	X
11	40	40	40	60	60	60	30	30	30	20	X

* Transition combinations marked with X are inapplicable.

Implementation of the SRL Algorithm

Unlike the current FIFO policy, which assigns priorities to the movement tasks on-line, the SRL algorithm is implemented off-line. When a manufacturing order is issued, the algorithm takes it as an input, along with robot transition times and machine processing times, and generates a sequence of robot transitions (fitting the desired task execution sequence) as an output[1]. This way, the algorithm tailors a unique policy for each order.

To solve the sequencing problem using the SRL algorithm, it is formulated as a RL problem. The system's overall state at time step *t*, denoted as $s_t \in S$, is defined by the current state of the buffers (unoccupied, occupied with pallet and part, occupied with pallet only) and the machines (free, in process, idle with part after process). An action at step *t* is denoted as $a_t \in A(s_t)$, where A is the action space of all possible actions (the action space is state dependent). The execution of an action constitutes the execution of a certain task that changes the system's state (*e.g.*, loading a part to the mill, changing the state of the in-buffer from "occupied with pallet and part" to "occupied with pallet only", and the state of the Mill from "free" to "in process").

[1] Each system task, such as "pallet transfer from docking station to machine in-buffer", "material loading to machine" etc., can be translated to the appropriate robot movements.

A solution is a specific sequence of robot transitions (part and pallet transfers and moving empty from station to station), that results in the production of the manufacturing order. The learning task, as mentioned, is to find the sequence that would achieve the completion of a manufacturing order in minimum time.

A learning episode starts from the state where all the buffers and machines are free and the system waits for the arrival of the first part, and ends when the last part of the manufacturing order has arrived to the docking station as a finished product. A step is the transition from one system state to another generated by execution of a task.

Note that in this case, as in the toasting system's case, the state is defined as the system's status, and the agent's actions shift the system from state to state. Hence, not the location of the agent (robot) itself counts, but the influence of its actions on the system's state.

Analysis

Analysis is performed using event-based MATLAB simulations. Three modules are constructed: (i) a module simulating the operation of the FMS with the current FIFO dispatching policy, (ii) a module implementing the SRL algorithm to produce robot transition sequences and (iii) a module simulating the FMS operation using those learned sequences.

The learning algorithm's performance is evaluated by comparison to the current FIFO policy. The test case is to schedule orders of three sizes: 3, 5 and 8 parts, containing a mix of the two products - Sign and Rook. For the 3-part orders, all of the eight possible product combinations are examined (*e.g.* Rook-Rook-Rook; Rook-Rook-Sign; Rook-Sign-Rook; etc.). For the 5 and 8-part orders, 10 random product mixes are examined. For each order size, three distributions of part inter-arrival times are examined, with two mean values for each distribution: Constant with t = 100 / 200 seconds, Normal with a mean μ = 100 / 200 and standard deviation σ = 10 / 20 (10% of the mean), and Exponential with a rate parameter λ = 100 / 200 seconds.

For the FIFO policy, one simulation run is performed for each product mix to find the makespan in the Constant cases (since these cases are deterministic all runs would yield the same result, hence one is sufficient), and 50 simulation replications are performed for each product mix, to find an average makespan in the stochastic cases (Normal and Exponential inter-arrival times).

For the SRL algorithm, learning is done based on the deterministic (Constant) cases, by performing one learning session containing 200 learning episodes for each product mix. The resulting sequence (policy) is then used for operation in the stochastic cases, and the average makespan is calculated by performing 50 simulation replications for each product mix.

In terms of equation (5.1), a value of $\beta = 1$ is used for the 3-part orders and $\beta = 0.5$ for 5 and 8-part orders. A *type B* reward factor is used. These setting were selected since they produced the best performance in preliminary experiments. In all experiments the RL parameters[1] are set as follows: $\alpha = 0.05$, $\gamma = 0.9$. These parameters were selected empirically.

Performance is evaluated using the following measures:

1) *Makespan* - Average total completion time of manufacturing orders.

2) *IP* (improvement percentage) - Percentage of improvement achieved by the SRL algorithm (in comparison to the FIFO policy).

Results and Discussion

Fig. III.4 shows the makespans achieved by the FIFO policy and by the policies generated by the SRL algorithm for three example manufacturing orders. For all three examples the algorithm achieves shorter (better) makespans. Furthermore, it can be seen that the makespans are not equal, and depend on the product mix and on the order of part arrival.

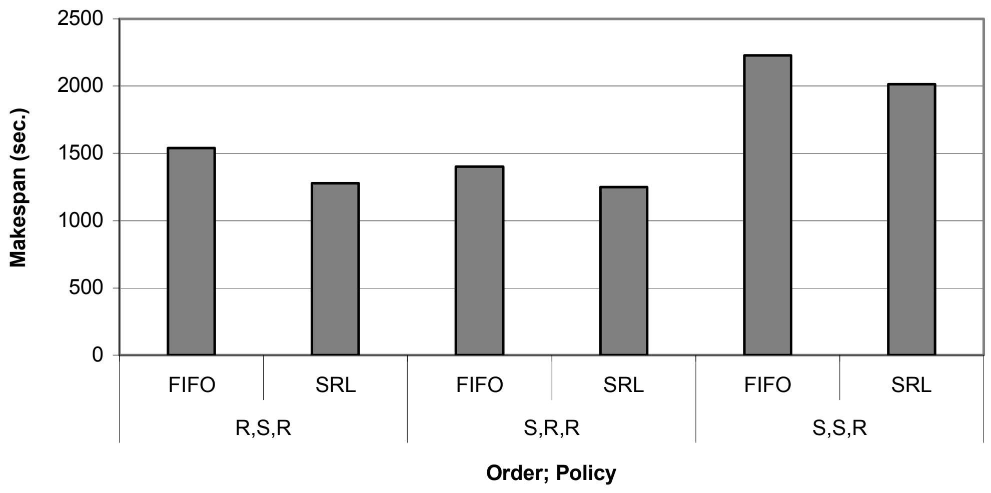


Fig. III.4 Makespans for various 3-part orders. R – Rook. S – Sign. Raw materials for the parts arrive in order from left to right

Comparison between the results of the FIFO policy and the results of the policy generated by the SRL algorithm was performed using a paired t-test for the mean makespans (examining the significance of the difference between the means). Fig. III.5 displays the average differences in

[1] The RL parameters, α (learning rate) and γ (discount factor), are described in Section 5.2.

makespans for 5-part orders, with various inter-arrival times. The black square marks the average, while the gray line indicates a 95% confidence interval (calculated using the paired t-test).

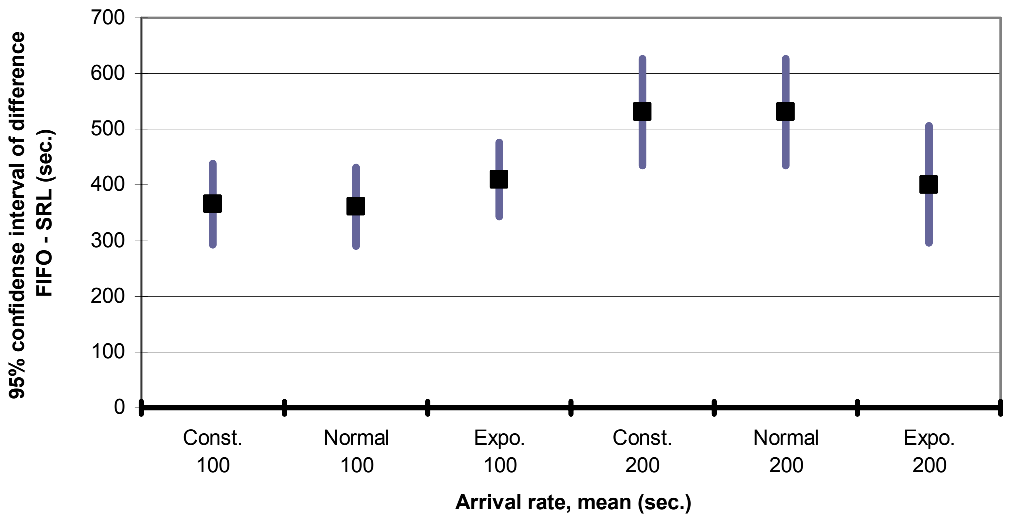


Fig. III.5 Makespan differences – 5-part orders

The positive confidence intervals indicate that the algorithm's makespans are significantly shorter (better), and the higher the average difference is, the better the algorithm's performs is in comparison to the FIFO policy. With Constant and Normal inter-arrival times, the results are approximately the same, and it can be seen that the makespan difference is higher when a mean of 200 seconds is set for the inter-arrival time. With Exponential arrivals, the differences are approximately the same for the 100 and 200 second cases. In average, the differences with a 200 second mean appear to be higher than the differences with 100 seconds (not statistically significant).

The results for the 3-part orders are similar to the ones of the 5-part orders shown here, while for the 8-part orders, the Exponential arrival results also show a higher difference with a 200 second mean (as do the results for the Constant and Normal arrivals).

Fig. III.6 shows another view of the results - the average differences in makespan between the FIFO policy and the policy generated by the SRL algorithm, for Normal inter-arrival times and various order sizes. The average difference ranges from 211 seconds for 3-part orders with mean inter-arrival time of 100 seconds, to 576 seconds for 8-part orders with a mean of 200 seconds.

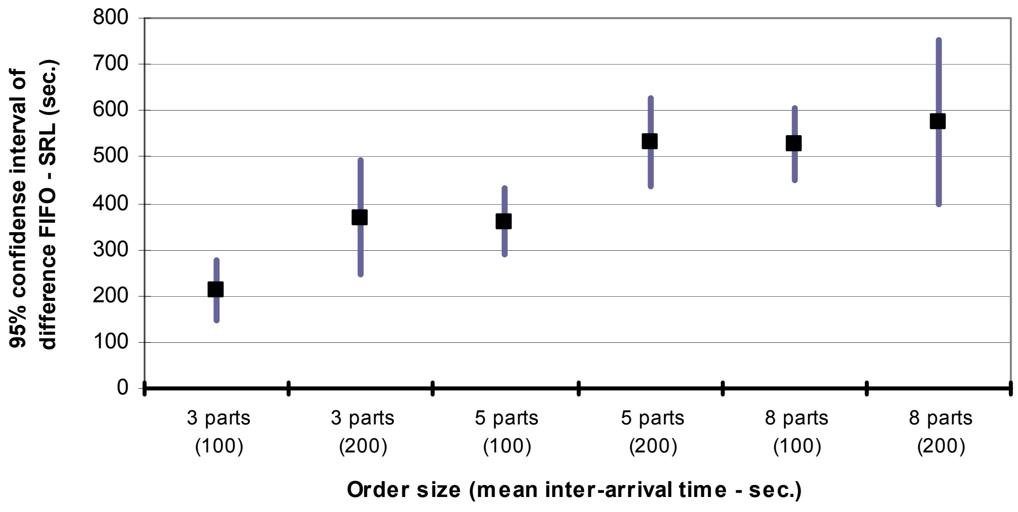


Fig. III.6 Makespan differences – Normal inter-arrival times

Two trends can be seen here. The first is that the difference is larger when the mean inter-arrival time is 200, for all of the orders sizes (especially for 3 and 5-part orders). When the mean is set to be 200 seconds, the standard deviation (which is set to be 10% of the mean) is also higher, introducing more variance to the process. The results insinuate that the SRL algorithm may deal with increased variance better than the FIFO policy.

The second trend is that the difference appears to grow larger as the number of parts in an order increases (similar results appear for Constant and Exponential inter-arrival times). This can be explained by the fact that when the order is larger and the makespan is longer, there is more room for improvement. Furthermore, for a given percentage of improvement gained by the algorithm, the longer the makespan is, the greater the relative difference between the results would be (*e.g.*, 10% improvement for 1,000 second makespan would yield 100 second difference, while for 2,000 second makespan it would yield 200 second difference).

Thus, in order to properly asses the performance of the SRL algorithm, and have the ability to infer from the examined product mixes to others, a better measure would be the percentage of improvement achieved by the algorithm, calculated as described in (III.1).

$$IP = \frac{T_f - T_a}{T_f} \tag{III.1}$$

Where IP is the improvement percentage, T_f is the makespan time achieved by the FIFO policy, and T_a is the makespan time achieved by the policy generated by the SRL algorithm.

Figs. III.7 - III.9 display the average percentage of improvement achieved by the use of the SRL algorithm for Normal, Constant and Exponential inter-arrival times. As can be seen, the algorithm achieves 11-17% improvement for the various combinations of orders sizes and mean inter-arrival times.

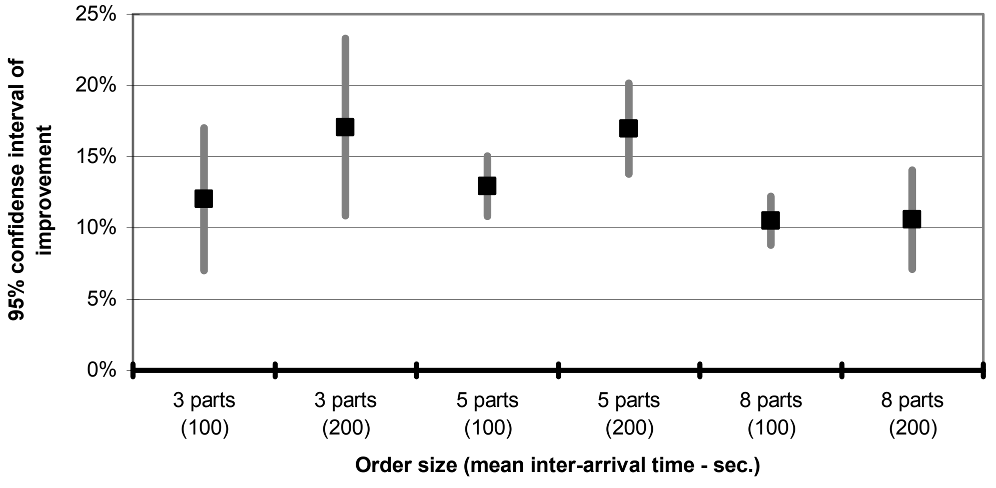


Fig. III.7 Improvement percentage - Normal inter-arrival times

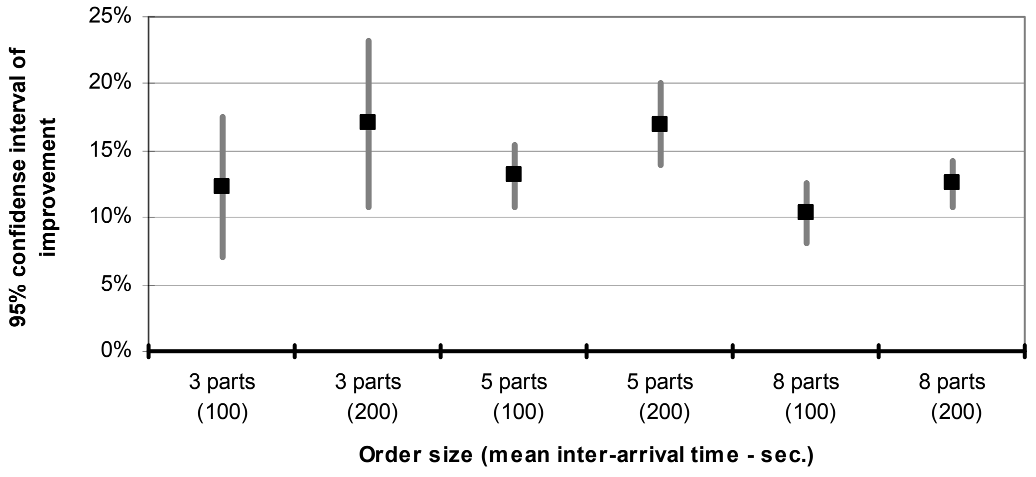


Fig. III.8 Improvement percentage - Constant inter-arrival times

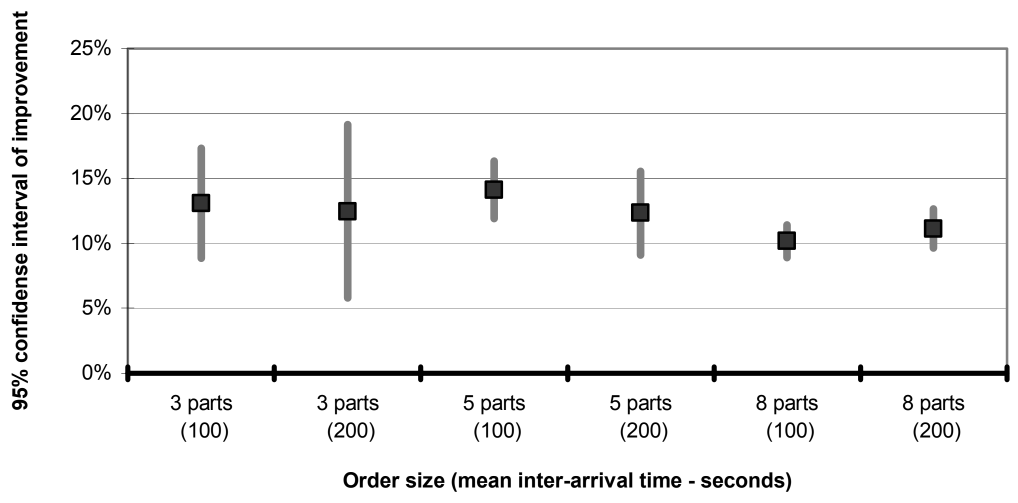


Fig. III.9 Improvement percentage - Exponential inter-arrival times

As seen also in the previous charts, for the 3 and 5-part orders with Normal inter-arrival times, the improvement appears to be greater when the mean is 200 seconds (though there is no statistical significance). For the 8-part orders, the improvement is 11% for both 100 and 200 seconds. Similar results appear for Constant inter-arrival times, while for Exponential inter-arrival times the results are generally the same for both 100 and 200 seconds.

Summary

The SRL algorithm is applied to the problem of sequencing tasks executed by a single transfer agent in a flow-shop system, with the objective of achieving minimal completion times of manufacturing orders. Analysis indicates that the SRL algorithm outperforms the FIFO policy currently employed for various combinations of order sizes and part inter-arrival times (deterministic and stochastic), achieving up to 17% improvement in performance. This is achieved by tailoring a unique sequence for every manufacturing order according to its specific characteristics.

Appendix IV. Toast Making System – Additional Results

Gantt Chart

Fig. IV.1 presents a Gantt chart of the solution achieved for the 3-toasts deterministic problem with case II times.

Toast transferred
Toast in process station
Toast in buffer station
Robot transferring toast
Robot moving empty
Robot Idle

Toast 1				Transferred			Toaster - Process								
Toast 2										Transferred		Toaster Buffer			
Toast 3															
Robot	2->1			1->3			3->1			1->2		2->3			Idle
Time (sec)	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150

Toast 1	Transferred					Butter Applier - Process									Idle
Toast 2	Toaster Buffer							Transferred			Toaster - Process				
Toast 3															
Robot	3->5					5->2		2->3			3->5				
Time (sec)	160	170	180	190	200	210	220	230	240	250	260	270	280	290	300

Toast 1	Transferred		Finished												
Toast 2	Process				Transferred					Butter Applier - Process					
Toast 3													Transferred		
Robot	5->6		6->3		3->5					5->1			1->3		
Time (sec)	310	320	330	340	350	360	370	380	390	400	410	420	430	440	450

Toast 1															
Toast 2	Process			Idle		Transferred		Finished							
Toast 3	Toaster - Process									Transferred					
Robot	3->5					5->6		6->3		3->5					Idle
Time (sec)	460	470	480	490	500	510	520	530	540	550	560	570	580	590	600

Toast 1												
Toast 2												
Toast 3	Butter Applier - Process								Transferred		Finished	
Robot	Idle								5->6			
Time (sec)	610	620	630	640	650	660	670	680	690	700		

Fig IV.1 Gantt chart of the toasting process

Proof of Optimality

The solution's optimality is proven using "Branch and Bound", a general search method for finding optimal solutions of various problems. It is basically an enumeration approach in a fashion that prunes the non-promising search space. System states are represented by nodes, while each node is branched to other nodes, representing the possible states following the state of the root node. Each node receives a value indicating the time passing until reaching the state that node represents. If the value of a node exceeds the value of an identical node, or a node representing a more advanced state, then the node is bounded, and that part of the state-space is pruned. This way there is a large part of the search space which is removed from consideration, allowing a faster and feasible search. Fig. IV.2 presents an illustration of the "tree" produced in the search process for the 3-toasts deterministic problem with case II times..

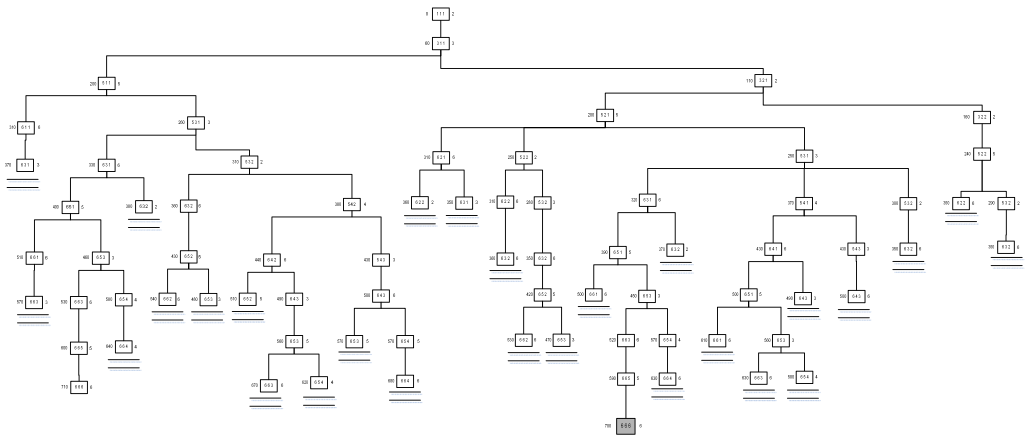


Fig IV.2 Branch and Bound solution tree

The optimal solution reached by the Branch and Bound method is 700 seconds, matching the solution produced by the algorithm, and proving it is indeed optimal. Fig. IV.3 shows a small part of the search space, ending with the optimal solution.

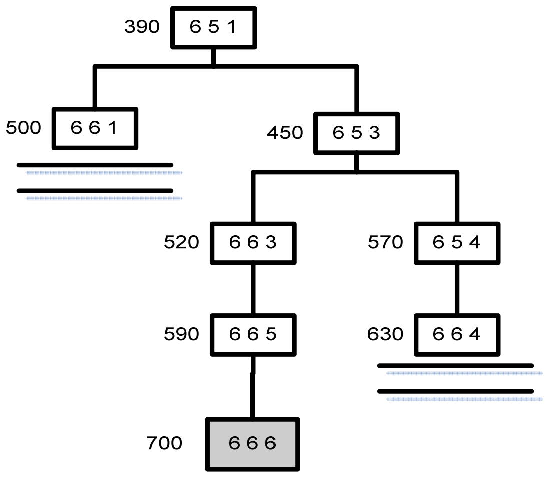


Fig. IV.3 Branch and Bound example

β Analysis

Table IV.1 summarize the results of the analysis of the influence of *β* on the performance, for the 3-toasts deterministic problem. As described in Section 7.7.1, when using a relatively small *β* (*β* = 1) the algorithm reaches the optimal solution with the highest percentage of success (*SP*), yet with the cost of a high number of episodes required for convergence (*CE*). As *β* increases, the percentage of success in reaching the optimal solution decreases, but fewer episodes are required to achieve convergence.

Table 11IV.1 Summary of *β* analysis

Case	Measure	*β*			
		1	**1.2**	**1.5**	**1.7**
I	***SP* (Success Percentage)**	99%	98%	95%	94%
	***CE* (Convergence Episode)**	155.7	126.3	83.8	69.1
II	***SP* (Success Percentage)**	99%	96%	88%	87%
	***CE* (Convergence Episode)**	154.7	123.8	85.2	72.9
III	***SP* (Success Percentage)**	98%	96%	92%	91%
	***CE* (Convergence Episode)**	153.7	126.7	82.4	73.1

Reward Factor Analysis

Table IV.2 summarize the results of the reward factor analysis. As described in Section 7.7.1, for all instances the use of a *type A* reward factor achieves fast learning and good results in a low number of episodes. When using the *type B* reward factor, the algorithm requires more episodes in order to achieve good results, but ultimately it outperforms the *type A* results.

Table IV.2 Summary of reward factor analysis

Deterministic 3-toast problem

Case	Reward Type	Session length (number of episodes)									
		15	**20**	**25**	**30**	**35**	**40**	**45**	**50**	**55**	**60**
I	***A***	40%	59%	75%	83%	86%	91%	93%	93%	96%	96%
	B	29%	44%	62%	73%	84%	91%	95%	97%	99%	100%
II	***A***	42%	61%	73%	83%	88%	91%	92%	94%	95%	96%
	B	32%	43%	59%	77%	86%	91%	94%	97%	100%	100%
III	***A***	38%	56%	71%	80%	85%	90%	92%	93%	94%	95%
	B	29%	40%	57%	73%	83%	92%	95%	97%	99%	100%

Deterministic 4-toast problem

Case	Reward Type	Session length (number of episodes)							
		15	20	25	30	35	40	45	50
I	*A*	21%	41%	53%	64%	67%	74%	79%	77%
	B	12%	26%	39%	65%	78%	89%	95%	97%
II	*A*	23%	33%	48%	52%	61%	70%	71%	78%
	B	8%	23%	39%	64%	79%	93%	96%	98%
III	*A*	18%	34%	46%	50%	59%	65%	74%	74%
	B	8%	19%	33%	51%	71%	90%	93%	96%

Stochastic 3-toast problem

Case	Reward Type	Session length (number of episodes)									
		15	20	25	30	35	40	45	50	55	60
I	*A*	544.22	530.26	522.48	517.13	513.66	511.95	510.48	509.37	508.65	508.58
	B	551.15	541.89	531.44	523.29	518.18	515.92	512.17	509.87	508.49	507.26
II	*A*	746.75	727.28	713.34	705.95	701.43	697.79	696.18	694.27	693.67	692.56
	B	760.31	745.26	730.44	714.94	706.93	701.6	698.7	695.7	693.1	690.61
III	*A*	1041.8	1030.2	1021.9	1017.3	1012.9	1009.9	1005.5	1002.8	999.95	999.26
	B	1074.1	1055.7	1035.1	1016.2	1006.1	999.36	994.72	990.23	986.86	983.97

Stochastic 4-toast problem

Case	Reward Type	Session length (number of episodes)							
		15	20	25	30	35	40	45	50
I	*A*	692.42	676.63	668.26	662.87	661.58	658.6	656.63	654.75
	B	709.92	680.54	670.98	664.47	659.07	655.27	652.81	651.8
II	*A*	945.71	926.46	918.16	906.09	906.42	903.3	895.91	895.55
	B	964.47	925.31	911.48	903.78	897.89	894.21	890.26	887.84
III	*A*	1369.1	1335.9	1317.6	1310.5	1301.7	1299.1	1295.3	1293.3
	B	1390.4	1336.5	1309.9	1298.8	1290.1	1282.5	1278.6	1276.1

Appendix V. 3D Path Planning Task – Additional Results

The following tables summarize the scores for the various methods and parameter combinations examined.

Scores for the IA Method

World I, Full view

Advisor (τ)	**Ψ**	***SP***	***HS***	***Score***
Expert (0.01)	0.1	67%	479.36	0.77
	0.3	94%	154.92	0.96
	0.7	98%	268.62	0.96
	1	100%	707.11	0.90
	1.3	100%	1149.01	0.83
Moderately expert (0.1)	0.1	67%	595.43	0.75
	0.3	83%	349.4	0.88
	0.7	96%	215	0.96
	1	99%	200.53	0.98
	1.3	49%	1098.84	0.58
Limited skills (0.3)	0.1	65%	808.46	0.71
	0.3	62%	1180	0.64
	0.7	80%	829.83	0.78
	1	88%	704.06	0.84
	1.3	96%	747.87	0.87
Novice (1.0)	0.1	62%	915.15	0.68
	0.3	50%	2143.39	0.42
	0.7	61%	2642.53	0.39
	1	57%	3214.58	0.28
	1.3	66%	3250.7	0.33
All levels (average)	0.1	65%	699.6	0.73
	0.3	70%	956.9	0.72
	0.7	84%	988.9	0.77
	1	86%	1206.57	0.75
	1.3	78%	1561.6	0.65

World I, Limited view

Advisor (τ)	**Ψ**	***SP***	***HS***	***Score***
Expert (0.01)	0.1	65%	92.88	0.71
	0.3	78%	98.9	0.85
	0.7	78%	101.18	0.85
	1	78%	105.45	0.85
	1.3	82%	153.95	0.86
Moderately expert (0.1)	0.1	64%	106.77	0.69
	0.3	75%	127.75	0.80
	0.7	75%	131.78	0.79
	1	76%	134.9	0.80
	1.3	80%	171.91	0.82
Limited skills (0.3)	0.1	60%	136.23	0.62
	0.3	59%	240.05	0.51
	0.7	65%	265.71	0.57
	1	67%	277.55	0.58
	1.3	68%	317.51	0.57
Novice (1.0)	0.1	63%	149.554	0.63
	0.3	48%	329.76	0.32
	0.7	53%	440.72	0.28
	1	50%	473.8	0.22
	1.3	53%	513.48	0.23
All levels (average)	0.1	63%	121.36	0.66
	0.3	65%	199.12	0.62
	0.7	68%	234.85	0.63
	1	68%	247.93	0.61
	1.3	71%	289.21	0.62

World II, Full view

Advisor (τ)	Ψ	*SP*	*HS*	*Score*
Expert (0.01)	0.1	43%	483.95	0.69
	0.3	59%	553.8	0.76
	0.7	98%	522.11	0.96
	1	100%	954.99	0.94
	1.3	100%	1611.92	0.89
Moderately expert (0.1)	0.1	39%	563.84	0.66
	0.3	63%	959.62	0.75
	0.7	87%	608.11	0.90
	1	61%	1097.8	0.73
	1.3	27%	2260.21	0.48
Limited skills (0.3)	0.1	40%	670.91	0.66
	0.3	50%	2104.91	0.61
	0.7	60%	2063.29	0.66
	1	65%	2216.56	0.67
	1.3	64%	3606.93	0.57
Novice (1.0)	0.1	40%	738.4	0.65
	0.3	51%	3047.25	0.55
	0.7	37%	5228.54	0.32
	1	30%	6387.81	0.21
	1.3	38%	7205.69	0.19
All levels (average)	0.1	41%	614.28	0.67
	0.3	56%	1666.40	0.67
	0.7	70%	2105.51	0.71
	1	64%	2664.29	0.64
	1.3	57%	3671.19	0.54

World II, Limited view

Advisor (τ)	**Ψ**	***SP***	***HS***	***Score***
Expert (0.01)	0.1	48%	59.97	0.78
	0.3	44%	109.55	0.70
	0.7	50%	112.23	0.76
	1	62%	131.11	0.84
	1.3	67%	369.08	0.68
Moderately expert (0.1)	0.1	41%	66.128	0.72
	0.3	41%	144.95	0.65
	0.7	37%	154.76	0.60
	1	50%	172.57	0.70
	1.3	27%	329.54	0.36
Limited skills (0.3)	0.1	34%	78.34	0.64
	0.3	35%	227.9	0.52
	0.7	33%	293.09	0.44
	1	19%	338.08	0.27
	1.3	12%	412.57	0.15
Novice (1.0)	0.1	34%	86.52	0.64
	0.3	31%	295.35	0.43
	0.7	21%	438.13	0.20
	1	13%	546.05	0.04
	1.3	11%	565.06	0.00
All levels (average)	0.1	39%	72.74	0.69
	0.3	38%	194.44	0.57
	0.7	35%	249.55	0.50
	1	36%	296.95	0.46
	1.3	29%	419.06	0.30

Scores for the CCRL Algorithm

World I, Full view

Advisor (τ)	Λ	Ω	*SP*	*HS*	*Score*
Expert (0.01)	0.05	1	99%	292.2	0.96
	0.3	1	98%	311.64	0.95
	0.5	1	98%	309.68	0.96
	0.3	3	97%	317.81	0.95
	0.9	1	82%	142.42	0.90
Moderately expert (0.1)	0.05	1	61%	768.62	0.70
	0.3	1	67%	825.19	0.72
	0.5	1	52%	770.25	0.65
	0.3	3	65%	911.06	0.69
	0.9	1	27%	210.03	0.61
Limited skills (0.3)	0.05	1	2%	1476.89	0.28
	0.3	1	4%	959.38	0.38
	0.5	1	7%	756.97	0.43
	0.3	3	1%	1573.2	0.26
	0.9	1	11%	593.59	0.47
Novice (1.0)	0.05	1	49%	280.11	0.71
	0.3	1	50%	285.36	0.72
	0.5	1	55%	278.12	0.74
	0.3	3	44%	562.26	0.64
	0.9	1	55%	271.42	0.74
All levels (average)	0.05	1	53%	704.46	0.66
	0.3	1	55%	595.39	0.69
	0.5	1	53%	528.76	0.69
	0.3	3	52%	841.08	0.64
	0.9	1	44%	304.37	0.68

World I, Limited view

Advisor (τ)	Λ	Ω	*SP*	*HS*	*Score*
Expert (0.01)	0.05	5	77%	88.03	0.85
	0.05	3	72%	50.31	0.83
	0.3	5	73%	66.58	0.82
	0.05	1	63%	20.52	0.74
	0.3	1	69%	19.12	0.82
	0.15	1	67%	20.01	0.79
	0.15	3	69%	46.65	0.79
	0.05	7	80%	135.35	0.85
	0.3	3	71%	40.8	0.82
	0.01	1	61%	20.48	0.73
Moderately expert (0.1)	0.05	5	66%	112.49	0.70
	0.05	3	63%	62.83	0.71
	0.3	5	66%	86.45	0.72
	0.05	1	62%	26.64	0.73
	0.3	1	59%	24.71	0.69
	0.15	1	64%	26.93	0.76
	0.15	3	64%	58.27	0.72
	0.05	7	72%	178.87	0.72
	0.3	3	63%	52.93	0.72
	0.01	1	61%	30.47	0.71
Limited skills (0.3)	0.05	5	48%	136.83	0.48
	0.05	3	47%	90.4	0.50
	0.3	5	50%	122.98	0.51
	0.05	1	59%	39.92	0.69
	0.3	1	58%	37.98	0.67
	0.15	1	54%	36.94	0.63
	0.15	3	51%	89.15	0.55
	0.05	7	42%	205.6	0.34
	0.3	3	52%	80.14	0.57
	0.01	1	53%	41.41	0.61
Novice (1.0)	0.05	5	59%	92.22	0.64
	0.05	3	57%	57.32	0.65
	0.3	5	57%	86.11	0.62
	0.05	1	62%	29.17	0.73
	0.3	1	62%	29.63	0.72
	0.15	1	58%	27.97	0.68
	0.15	3	58%	57.58	0.65
	0.05	7	52%	125.19	0.53
	0.3	3	58%	57.81	0.66
	0.01	1	63%	28.85	0.74
All levels (average)	0.05	5	63%	107.39	0.67
	0.05	3	60%	65.22	0.67
	0.3	5	61%	90.53	0.67
	0.05	1	62%	29.06	0.72
	0.3	1	62%	27.86	0.73
	0.15	1	61%	27.96	0.71
	0.15	3	60%	62.91	0.68
	0.05	7	61%	161.25	0.61
	0.3	3	61%	57.92	0.69
	0.01	1	60%	30.30	0.70

World II, Full view

Advisor (τ)	Λ	Ω	*SP*	*HS*	*Score*
Expert (0.01)	0.05	1	94%	507	0.94
	0.3	1	93%	505.82	0.94
	0.05	3	96%	522.97	0.95
	0.3	3	94%	531.41	0.94
	0.05	5	95%	541.72	0.94
	0.05	7	94%	541.61	0.94
	0.5	1	93%	510.95	0.94
	0.01	1	96%	556.94	0.95
Moderately expert (0.1)	0.05	1	36%	1999.93	0.55
	0.3	1	36%	2541.8	0.51
	0.05	3	33%	2065.69	0.52
	0.3	3	31%	2555.03	0.48
	0.05	5	28%	2110.39	0.50
	0.05	7	24%	2173.08	0.47
	0.5	1	47%	1852.72	0.61
	0.01	1	36%	1826.38	0.56
Limited skills (0.3)	0.05	1	9%	1591.23	0.44
	0.3	1	14%	950.89	0.51
	0.05	3	5%	2901.74	0.33
	0.3	3	8%	1578.94	0.44
	0.05	5	2%	4012.02	0.23
	0.05	7	0%	5016.02	0.15
	0.5	1	20%	736.25	0.56
	0.01	1	11%	1635.27	0.44
Novice (1.0)	0.05	1	33%	185.16	0.66
	0.3	1	37%	177.12	0.68
	0.05	3	33%	362.47	0.65
	0.3	3	27%	348.6	0.62
	0.05	5	32%	537.45	0.63
	0.05	7	28%	705.84	0.60
	0.5	1	37%	179.8	0.68
	0.01	1	34%	176.73	0.67
All levels (average)	0.05	1	43%	877.84	0.65
	0.3	1	45%	2063.25	0.66
	0.05	3	42%	3778.91	0.61
	0.3	3	40%	1400.97	0.62
	0.05	5	39%	1565.77	0.58
	0.05	7	37%	1308.11	0.54
	0.5	1	49%	1348.03	0.70
	0.01	1	44%	1192.07	0.65

World II, Limited view

Advisor (τ)	**Λ**	**Ω**	***SP***	***HS***	***Score***
Expert (0.01)	0.01	1	38%	10.03	0.74
	0.05	1	36%	9.57	0.72
	0.3	1	39%	8.57	0.75
	0.5	1	41%	9.96	0.77
	0.01	3	40%	18.84	0.75
	0.05	3	37%	18.96	0.73
	0.3	3	39%	19.09	0.74
	0.05	5	43%	30.55	0.77
	0.05	7	51%	43.16	0.83
Moderately expert (0.1)	0.01	1	37%	11.98	0.73
	0.05	1	37%	12.43	0.73
	0.3	1	37%	11.54	0.73
	0.5	1	38%	12.76	0.73
	0.01	3	35%	28.67	0.70
	0.05	3	38%	25.86	0.72
	0.3	3	41%	26.59	0.75
	0.05	5	42%	40.83	0.75
	0.05	7	41%	62.23	0.72
Limited skills (0.3)	0.01	1	32%	24.53	0.68
	0.05	1	32%	25.28	0.68
	0.3	1	35%	22.25	0.70
	0.5	1	33%	23.45	0.68
	0.01	3	34%	49.39	0.67
	0.05	3	34%	48.31	0.67
	0.3	3	31%	45.6	0.64
	0.05	5	30%	74.92	0.61
	0.05	7	30%	103.75	0.58
Novice (1.0)	0.01	1	35%	17.1	0.70
	0.05	1	36%	14.94	0.71
	0.3	1	32%	16.23	0.68
	0.5	1	34%	15.41	0.70
	0.01	3	39%	33.06	0.72
	0.05	3	32%	31.38	0.67
	0.3	3	33%	33.53	0.68
	0.05	5	35%	49.23	0.68
	0.05	7	38%	65.76	0.69
All levels (average)	0.01	1	35%	15.91	0.71
	0.05	1	35%	15.56	0.71
	0.3	1	36%	14.65	0.71
	0.5	1	36%	15.40	0.72
	0.01	3	37%	32.49	0.71
	0.05	3	35%	31.13	0.70
	0.3	3	36%	31.20	0.70
	0.05	5	38%	48.88	0.70
	0.05	7	40%	68.73	0.71

Scores for the Combined Method (CCRL and IA)

World I, Full view

Advisor (τ)	**Λ**	**Ω**	**Ψ**	***SP***	***HS***	***Score***
Expert (0.01)	0.05	1	1	99%	91.59	0.99
	0.05	1	0.3	94%	120.51	0.96
	0.3	1	1	100%	96.83	1.00
	0.05	3	0.3	93%	123.49	0.96
	0.05	1	0.7	97%	107.67	0.98
Moderately expert (0.1)	0.05	1	1	97%	147.46	0.98
	0.05	1	0.3	78%	237.25	0.87
	0.3	1	1	98%	156.19	0.98
	0.05	3	0.3	85%	247.9	0.90
	0.05	1	0.7	94%	188.43	0.95
Limited skills (0.3)	0.05	1	1	80%	623.69	0.81
	0.05	1	0.3	53%	428.63	0.71
	0.3	1	1	75%	634.81	0.79
	0.05	3	0.3	53%	641.5	0.68
	0.05	1	0.7	62%	629.89	0.72
Novice (1.0)	0.05	1	1	54%	300.78	0.74
	0.05	1	0.3	61%	308.83	0.77
	0.3	1	1	53%	296.16	0.73
	0.05	3	0.3	54%	585.6	0.69
	0.05	1	0.7	55%	496.18	0.71
All levels (average)	0.05	1	1	82%	290.88	0.88
	0.05	1	0.3	71%	273.81	0.83
	0.3	1	1	81%	296.00	0.87
	0.05	3	0.3	71%	399.62	0.81
	0.05	1	0.7	77%	355.54	0.84

World I, Limited view

Advisor (τ)	Λ	Ω	Ψ	*SP*	*HS*	*Score*
Expert (0.01)	0.05	1	0.3	66%	12.96	0.79
	0.05	3	0.3	67%	26.51	0.79
	0.05	5	0.3	68%	39.53	0.79
	0.05	7	0.3	73%	51.47	0.84
	0.05	1	0.7	62%	12.65	0.73
	0.05	1	1	63%	14.01	0.75
	0.05	5	1	69%	43.01	0.80
	0.3	3	1	66%	26.91	0.77
	0.3	3	0.3	66%	24.13	0.77
	0.5	1	0.3	63%	12.45	0.75
Moderately expert (0.1)	0.05	1	0.3	67%	15.91	0.80
	0.05	3	0.3	62%	29.32	0.73
	0.05	5	0.3	64%	45.11	0.73
	0.05	7	0.3	69%	59.44	0.79
	0.05	1	0.7	61%	15.62	0.73
	0.05	1	1	62%	18.57	0.74
	0.05	5	1	68%	49.85	0.77
	0.3	3	1	65%	31.85	0.76
	0.3	3	0.3	62%	30.43	0.73
	0.5	1	0.3	63%	15.52	0.75
Limited skills (0.3)	0.05	1	0.3	58%	20.45	0.68
	0.05	3	0.3	59%	41.1	0.69
	0.05	5	0.3	59%	60	0.66
	0.05	7	0.3	60%	82.74	0.66
	0.05	1	0.7	58%	23.64	0.69
	0.05	1	1	58%	26.28	0.68
	0.05	5	1	63%	75.49	0.70
	0.3	3	1	57%	55.3	0.64
	0.3	3	0.3	58%	41.09	0.67
	0.5	1	0.3	60%	19.21	0.72
Novice (1.0)	0.95	1	0.3	60%	20.18	0.71
	0.95	3	0.3	58%	39.08	0.67
	0.95	5	0.3	63%	57.17	0.71
	0.95	7	0.3	59%	78.71	0.65
	0.95	1	0.7	61%	22.12	0.72
	0.95	1	1	60%	23.84	0.71
	0.05	5	1	57%	77.1	0.63
	0.3	3	1	60%	47.51	0.68
	0.3	3	0.3	61%	38	0.71
	0.5	1	0.3	61%	18.89	0.73
All levels (average)	0.05	1	0.3	63%	17.38	0.74
	0.05	3	0.3	62%	34.00	0.72
	0.05	5	0.3	63%	50.45	0.72
	0.05	7	0.3	65%	68.09	0.73
	0.05	1	0.7	61%	18.51	0.72
	0.05	1	1	61%	20.68	0.72
	0.05	5	1	64%	61.36	0.73
	0.3	3	1	62%	40.39	0.71
	0.3	3	0.3	62%	33.41	0.72
	0.5	1	0.3	62%	16.52	0.74

World II, Full view

Advisor (τ)	**Λ**	**Ω**	**Ψ**	***SP***	***HS***	***Score***
Expert (0.01)	0.01	1	1	98%	185.48	0.98
	0.05	1	0.3	43%	111.44	0.71
	0.05	3	0.3	47%	215.05	0.73
	0.05	1	0.7	91%	186.29	0.95
	0.05	1	1	97%	179.73	0.98
	0.3	1	1	97%	195.1	0.98
Moderately expert (0.1)	0.01	1	1	91%	347.37	0.94
	0.05	1	0.3	43%	125.78	0.71
	0.05	3	0.3	45%	260.93	0.71
	0.05	1	0.7	74%	370.66	0.85
	0.05	1	1	90%	360.03	0.93
	0.3	1	1	90%	403	0.93
Limited skills (0.3)	0.01	1	1	32%	1019.47	0.60
	0.05	1	0.3	36%	150.42	0.68
	0.05	3	0.3	34%	282.39	0.66
	0.05	1	0.7	32%	610.96	0.62
	0.05	1	1	33%	1056.98	0.60
	0.3	1	1	29%	807.62	0.59
Novice (1.0)	0.01	1	1	36%	184.96	0.67
	0.05	1	0.3	37%	115.94	0.68
	0.05	3	0.3	35%	229.8	0.67
	0.05	1	0.7	34%	218.36	0.66
	0.05	1	1	33%	176.92	0.66
	0.3	1	1	36%	166.23	0.67
All levels (average)	0.01	1	1	64%	434.32	0.80
	0.05	1	0.3	40%	125.895	0.70
	0.05	3	0.3	41%	247.0425	0.69
	0.05	1	0.7	58%	346.5675	0.77
	0.05	1	1	63%	443.415	0.79
	0.3	1	1	63%	392.9875	0.79

World II, Limited view

Advisor (τ)	**Λ**	**Ω**	**Ψ**	***SP***	***HS***	***Score***
Expert (0.01)	0.01	1	1	41%	7.59	0.77
	0.05	1	0.3	38%	7.25	0.74
	0.05	3	0.3	40%	14.51	0.75
	0.05	1	0.7	39%	7.44	0.76
	0.05	1	1	40%	8.56	0.76
	0.05	5	1	44%	23.96	0.78
	0.3	1	1	38%	8.09	0.74
	0.5	1	0.3	39%	7.14	0.75
Moderately expert (0.1)	0.01	1	1	39%	11.67	0.75
	0.05	1	0.3	38%	8.69	0.74
	0.05	3	0.3	37%	18.45	0.72
	0.05	1	0.7	38%	10	0.74
	0.05	1	1	40%	11	0.76
	0.05	5	1	37%	31.52	0.72
	0.3	1	1	35%	9.57	0.72
	0.5	1	0.3	35%	9.67	0.72
Limited skills (0.3)	0.01	1	1	37%	19.13	0.72
	0.05	1	0.3	35%	12.96	0.71
	0.05	3	0.3	34%	25.44	0.69
	0.05	1	0.7	33%	17.4	0.69
	0.05	1	1	32%	22.36	0.68
	0.05	5	1	36%	54.63	0.68
	0.3	1	1	38%	20.03	0.73
	0.5	1	0.3	34%	14.19	0.70
Novice (1.0)	0.01	1	1	35%	16.38	0.71
	0.05	1	0.3	33%	11.85	0.69
	0.05	3	0.3	38%	22.88	0.73
	0.05	1	0.7	37%	15.19	0.72
	0.05	1	1	33%	15.02	0.69
	0.05	5	1	35%	44.16	0.68
	0.3	1	1	36%	13.67	0.72
	0.5	1	0.3	33%	10.33	0.69
All levels (average)	0.01	1	1	38%	13.69	0.74
	0.05	1	0.3	36%	10.19	0.72
	0.05	3	0.3	37%	20.32	0.72
	0.05	1	0.7	37%	12.51	0.72
	0.05	1	1	36%	14.24	0.72
	0.05	5	1	38%	38.57	0.72
	0.3	1	1	37%	12.84	0.73
	0.5	1	0.3	35%	10.33	0.71

Appendix VI. Toast Making System – Source Code

VB.net Code

Form1.vb

```
Option Strict Off
Option Explicit On
Imports Microsoft.Win32
Imports System.IO
'Imports System.Security.Permissions
'Imports System.Math
'Imports System.Data.SqlClient
'Imports System.Data.OleDb

Imports System
Imports System.Drawing
Imports System.Windows.Forms
Imports vb = Microsoft.VisualBasic



Public Class Form1
    Inherits System.Windows.Forms.Form
    Dim nCid As Integer

    ' Global Declarations
    Dim MatLab As Object
    Dim Sequence(2) As Integer
    Dim ToasterFree As Boolean
    Dim ButtererFree As Boolean
    Dim ToasterFinished As Boolean
    Dim ButtererFinished As Boolean
    Dim StopRun As Boolean 'for stoping learning episode
    Dim NumOfEpisods As Integer ' counting number of learning
episodes
    Dim t As New System.Timers.Timer(50000) '106000
    Dim b As New System.Timers.Timer(48000) '94000
    Dim p As New System.Timers.Timer(5000)

    Public Const SND_ASYNC = &H1 ' play asynchronously
    Public Const SND_LOOP = &H8 ' loop the sound until next
sndPlaySound
    Public Const SND_NOSTOP = &H10 ' don't stop any currently
playing sound
    Public Const SND_NOWAIT = &H2000 ' don't wait if the driver is
busy

    Private Declare Function PlaySound Lib "winmm.dll" Alias
"PlaySoundA" (ByVal lpszName As String, ByVal hModule As
Long, ByVal dwFlags As Long) As Long



#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form
Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents GroupBox1 As
System.Windows.Forms.GroupBox
    Friend WithEvents CheckBox2 As
System.Windows.Forms.CheckBox
    Friend WithEvents Button1 As System.Windows.Forms.Button
    Friend WithEvents Label10 As System.Windows.Forms.Label
    Public WithEvents Label11 As System.Windows.Forms.Label
    Friend WithEvents Label12 As System.Windows.Forms.Label
    Friend WithEvents Label13 As System.Windows.Forms.Label
    Friend WithEvents Label14 As System.Windows.Forms.Label
    Friend WithEvents Label15 As System.Windows.Forms.Label
    Friend WithEvents CheckBox1 As
System.Windows.Forms.CheckBox
    Friend WithEvents GroupBox2 As
System.Windows.Forms.GroupBox
    Public WithEvents Button7 As System.Windows.Forms.Button
    Friend WithEvents TextBox4 As System.Windows.Forms.TextBox
    Public WithEvents Button5 As System.Windows.Forms.Button
    Public WithEvents Button6 As System.Windows.Forms.Button
    Friend WithEvents TextBox6 As System.Windows.Forms.TextBox
    Public WithEvents CmdDownLoad As
System.Windows.Forms.Button
    Public WithEvents Button9 As System.Windows.Forms.Button
    Friend WithEvents GroupBox6 As
System.Windows.Forms.GroupBox
    Friend WithEvents Label7 As System.Windows.Forms.Label
    Friend WithEvents Label8 As System.Windows.Forms.Label
    Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
    Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents Label3 As System.Windows.Forms.Label
    Friend WithEvents Label4 As System.Windows.Forms.Label
    Friend WithEvents Label5 As System.Windows.Forms.Label
    Friend WithEvents Label6 As System.Windows.Forms.Label
    Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
    Friend WithEvents Label9 As System.Windows.Forms.Label
    Friend WithEvents Button4 As System.Windows.Forms.Button
    Friend WithEvents GroupBox11 As
System.Windows.Forms.GroupBox
    Friend WithEvents TextBox11 As
System.Windows.Forms.TextBox
    Friend WithEvents Label18 As System.Windows.Forms.Label
    Friend WithEvents Label25 As System.Windows.Forms.Label
    Friend WithEvents CheckBox5 As
System.Windows.Forms.CheckBox
    Friend WithEvents Label17 As System.Windows.Forms.Label
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents CheckBox4 As
System.Windows.Forms.CheckBox
    Friend WithEvents TextBox10 As
System.Windows.Forms.TextBox
    Friend WithEvents Label23 As System.Windows.Forms.Label
    Friend WithEvents Label24 As System.Windows.Forms.Label
    Friend WithEvents Label42 As System.Windows.Forms.Label
    Friend WithEvents TextBox76 As
System.Windows.Forms.TextBox
    Friend WithEvents Label39 As System.Windows.Forms.Label
    Friend WithEvents TextBox75 As
System.Windows.Forms.TextBox
    Friend WithEvents Button3 As System.Windows.Forms.Button
    Public WithEvents Label103 As System.Windows.Forms.Label
    Friend WithEvents TextBox148 As
System.Windows.Forms.TextBox
    Friend WithEvents Button8 As System.Windows.Forms.Button
    Friend WithEvents TextBox8 As System.Windows.Forms.TextBox
    Public WithEvents Label16 As System.Windows.Forms.Label
    Friend WithEvents TextBox13 As System.Windows.Forms.TextBox
    Public WithEvents Label19 As System.Windows.Forms.Label
    Friend WithEvents GroupBox3 As System.Windows.Forms.GroupBox
    Public WithEvents Label20 As System.Windows.Forms.Label
    Public WithEvents Label21 As System.Windows.Forms.Label
    Public WithEvents Label22 As System.Windows.Forms.Label
    Friend WithEvents ComboBox1 As System.Windows.Forms.ComboBox
    Public WithEvents Label26 As System.Windows.Forms.Label
    Public WithEvents Label27 As System.Windows.Forms.Label
    Public WithEvents Label28 As System.Windows.Forms.Label
    Friend WithEvents TextBox5 As System.Windows.Forms.TextBox
    Public WithEvents Label29 As System.Windows.Forms.Label
    Public WithEvents Label30 As System.Windows.Forms.Label
    Friend WithEvents MainMenu1 As System.Windows.Forms.MainMenu
    Friend WithEvents MenuItem1 As System.Windows.Forms.MenuItem
    Friend WithEvents TabControl1 As System.Windows.Forms.TabControl
    Friend WithEvents TabPage1 As System.Windows.Forms.TabPage
    Friend WithEvents TabPage2 As System.Windows.Forms.TabPage
    Friend WithEvents Button2 As System.Windows.Forms.Button
    Friend WithEvents Button10 As System.Windows.Forms.Button
    Friend WithEvents Button11 As System.Windows.Forms.Button
    Friend WithEvents Button12 As System.Windows.Forms.Button
    Friend WithEvents Button13 As System.Windows.Forms.Button
    Friend WithEvents Button14 As System.Windows.Forms.Button
    Friend WithEvents Button15 As System.Windows.Forms.Button
    Public WithEvents Label34 As System.Windows.Forms.Label
    Friend WithEvents GroupBox4 As System.Windows.Forms.GroupBox
    Friend WithEvents GroupBox5 As System.Windows.Forms.GroupBox
    Friend WithEvents Label33 As System.Windows.Forms.Label
    Friend WithEvents TextBox7 As System.Windows.Forms.TextBox
    Friend WithEvents Button16 As System.Windows.Forms.Button
    Friend WithEvents Button17 As System.Windows.Forms.Button
    Friend WithEvents Button18 As System.Windows.Forms.Button
    Friend WithEvents TextBox15 As System.Windows.Forms.TextBox
    Friend WithEvents Button19 As System.Windows.Forms.Button
    Friend WithEvents Button21 As System.Windows.Forms.Button
    Friend WithEvents Button20 As System.Windows.Forms.Button
    Friend WithEvents Button22 As System.Windows.Forms.Button
    Friend WithEvents Button23 As System.Windows.Forms.Button
    Friend WithEvents Button24 As System.Windows.Forms.Button
    Friend WithEvents Button25 As System.Windows.Forms.Button
    Friend WithEvents Button27 As System.Windows.Forms.Button
    Friend WithEvents Button28 As System.Windows.Forms.Button
    Friend WithEvents Button30 As System.Windows.Forms.Button
    Friend WithEvents Button32 As System.Windows.Forms.Button
    Friend WithEvents Button26 As System.Windows.Forms.Button
    Friend WithEvents Button29 As System.Windows.Forms.Button
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        Me.GroupBox1 = New System.Windows.Forms.GroupBox
        Me.CheckBox2 = New System.Windows.Forms.CheckBox
        Me.Button1 = New System.Windows.Forms.Button
        Me.Label10 = New System.Windows.Forms.Label
        Me.Label11 = New System.Windows.Forms.Label
        Me.Label12 = New System.Windows.Forms.Label
        Me.Label13 = New System.Windows.Forms.Label
        Me.Label14 = New System.Windows.Forms.Label
        Me.Label15 = New System.Windows.Forms.Label
        Me.CheckBox1 = New System.Windows.Forms.CheckBox
        Me.GroupBox2 = New System.Windows.Forms.GroupBox
        Me.Button7 = New System.Windows.Forms.Button
        Me.TextBox4 = New System.Windows.Forms.TextBox
        Me.Button5 = New System.Windows.Forms.Button
        Me.Button6 = New System.Windows.Forms.Button
        Me.TextBox6 = New System.Windows.Forms.TextBox
        Me.CmdDownLoad = New System.Windows.Forms.Button
        Me.Button9 = New System.Windows.Forms.Button
        Me.GroupBox6 = New System.Windows.Forms.GroupBox
        Me.Label7 = New System.Windows.Forms.Label
        Me.Label8 = New System.Windows.Forms.Label
        Me.TextBox1 = New System.Windows.Forms.TextBox
        Me.TextBox2 = New System.Windows.Forms.TextBox
        Me.Label2 = New System.Windows.Forms.Label
        Me.Label3 = New System.Windows.Forms.Label
        Me.Label4 = New System.Windows.Forms.Label
        Me.Label5 = New System.Windows.Forms.Label
        Me.Label6 = New System.Windows.Forms.Label
        Me.TextBox3 = New System.Windows.Forms.TextBox
        Me.Label9 = New System.Windows.Forms.Label
        Me.Button4 = New System.Windows.Forms.Button
        Me.GroupBox11 = New System.Windows.Forms.GroupBox
        Me.TextBox11 = New System.Windows.Forms.TextBox
        Me.Label18 = New System.Windows.Forms.Label
        Me.Label25 = New System.Windows.Forms.Label
        Me.CheckBox5 = New System.Windows.Forms.CheckBox
        Me.Label17 = New System.Windows.Forms.Label
        Me.Label1 = New System.Windows.Forms.Label
        Me.CheckBox4 = New System.Windows.Forms.CheckBox
        Me.TextBox10 = New System.Windows.Forms.TextBox
        Me.Label23 = New System.Windows.Forms.Label
        Me.Label24 = New System.Windows.Forms.Label
        Me.Label42 = New System.Windows.Forms.Label
        Me.TextBox76 = New System.Windows.Forms.TextBox
        Me.Label39 = New System.Windows.Forms.Label
        Me.TextBox75 = New System.Windows.Forms.TextBox
        Me.Button3 = New System.Windows.Forms.Button
        Me.Label103 = New System.Windows.Forms.Label
        Me.TextBox148 = New System.Windows.Forms.TextBox
        Me.Button8 = New System.Windows.Forms.Button
        Me.TextBox8 = New System.Windows.Forms.TextBox
        Me.Label16 = New System.Windows.Forms.Label
        Me.TextBox13 = New System.Windows.Forms.TextBox
        Me.Label19 = New System.Windows.Forms.Label
        Me.GroupBox3 = New System.Windows.Forms.GroupBox
        Me.Label30 = New System.Windows.Forms.Label
        Me.Label29 = New System.Windows.Forms.Label
        Me.TextBox5 = New System.Windows.Forms.TextBox
        Me.Label28 = New System.Windows.Forms.Label
        Me.Label27 = New System.Windows.Forms.Label
        Me.Label26 = New System.Windows.Forms.Label
        Me.ComboBox1 = New System.Windows.Forms.ComboBox
        Me.Label22 = New System.Windows.Forms.Label
        Me.Label21 = New System.Windows.Forms.Label
        Me.Label20 = New System.Windows.Forms.Label
        Me.Button17 = New System.Windows.Forms.Button
        Me.MainMenu1 = New System.Windows.Forms.MainMenu
        Me.MenuItem1 = New System.Windows.Forms.MenuItem
        Me.TabControl1 = New System.Windows.Forms.TabControl
        Me.TabPage1 = New System.Windows.Forms.TabPage
        Me.Button25 = New System.Windows.Forms.Button
        Me.Button24 = New System.Windows.Forms.Button
        Me.Button23 = New System.Windows.Forms.Button
        Me.Button22 = New System.Windows.Forms.Button
        Me.Button20 = New System.Windows.Forms.Button
        Me.Button21 = New System.Windows.Forms.Button
        Me.Button19 = New System.Windows.Forms.Button
        Me.Button18 = New System.Windows.Forms.Button
        Me.TextBox15 = New System.Windows.Forms.TextBox
        Me.TabPage2 = New System.Windows.Forms.TabPage
        Me.GroupBox5 = New System.Windows.Forms.GroupBox
        Me.Button32 = New System.Windows.Forms.Button
        Me.Button30 = New System.Windows.Forms.Button
        Me.Label33 = New System.Windows.Forms.Label
        Me.TextBox7 = New System.Windows.Forms.TextBox
        Me.Button13 = New System.Windows.Forms.Button
        Me.Button15 = New System.Windows.Forms.Button
        Me.Button14 = New System.Windows.Forms.Button
        Me.Label34 = New System.Windows.Forms.Label
        Me.Button16 = New System.Windows.Forms.Button
        Me.GroupBox4 = New System.Windows.Forms.GroupBox
        Me.Button28 = New System.Windows.Forms.Button
        Me.Button27 = New System.Windows.Forms.Button
        Me.Button10 = New System.Windows.Forms.Button
        Me.Button11 = New System.Windows.Forms.Button
        Me.Button2 = New System.Windows.Forms.Button
        Me.Button12 = New System.Windows.Forms.Button
        Me.Button26 = New System.Windows.Forms.Button
        Me.Button29 = New System.Windows.Forms.Button
        Me.GroupBox1.SuspendLayout()
        Me.GroupBox2.SuspendLayout()
        Me.GroupBox6.SuspendLayout()
        Me.GroupBox11.SuspendLayout()
        Me.GroupBox3.SuspendLayout()
        Me.TabControl1.SuspendLayout()
        Me.TabPage1.SuspendLayout()
        Me.TabPage2.SuspendLayout()
        Me.GroupBox5.SuspendLayout()
        Me.GroupBox4.SuspendLayout()
        Me.SuspendLayout()
        '
        'GroupBox1
        '
        Me.GroupBox1.Controls.Add(Me.CheckBox2)
        Me.GroupBox1.Controls.Add(Me.Button1)
        Me.GroupBox1.Controls.Add(Me.Label10)
        Me.GroupBox1.Controls.Add(Me.Label11)
        Me.GroupBox1.Controls.Add(Me.Label12)
        Me.GroupBox1.Controls.Add(Me.Label13)
        Me.GroupBox1.Controls.Add(Me.Label14)
        Me.GroupBox1.Controls.Add(Me.Label15)
        Me.GroupBox1.Controls.Add(Me.CheckBox1)
        Me.GroupBox1.Font = New System.Drawing.Font("Arial",
8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.GroupBox1.Location = New System.Drawing.Point(32, 24)
        Me.GroupBox1.Name = "GroupBox1"
        Me.GroupBox1.Size = New System.Drawing.Size(240, 184)
        Me.GroupBox1.TabIndex = 70
        Me.GroupBox1.TabStop = False
        Me.GroupBox1.Text = "Communication"
        '
        'CheckBox2
        '
        Me.CheckBox2.Location = New System.Drawing.Point(80, 160)
        Me.CheckBox2.Name = "CheckBox2"
        Me.CheckBox2.Size = New System.Drawing.Size(16, 16)
        Me.CheckBox2.TabIndex = 26
        Me.CheckBox2.Text = "Servo"
        '
        'Button1
        '
        Me.Button1.BackColor =
System.Drawing.Color.FromArgb(CType(255, Byte), CType(255,
Byte), CType(192, Byte))
        Me.Button1.Cursor = System.Windows.Forms.Cursors.Default
        Me.Button1.Font = New System.Drawing.Font("Arial", 8.0!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button1.ForeColor =
System.Drawing.SystemColors.ControlText
        Me.Button1.Location = New System.Drawing.Point(8, 24)
        Me.Button1.Name = "Button1"
        Me.Button1.RightToLeft =
System.Windows.Forms.RightToLeft.No
        Me.Button1.Size = New System.Drawing.Size(104, 40)
        Me.Button1.TabIndex = 4
        Me.Button1.Text = "Open Communication"
        '
        'Label10
        '
        Me.Label10.Location = New System.Drawing.Point(104, 136)
        Me.Label10.Name = "Label10"
        Me.Label10.Size = New System.Drawing.Size(80, 16)
        Me.Label10.TabIndex = 31
        '
        'Label11
        '
        Me.Label11.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label11.Location = New System.Drawing.Point(40, 136)
        Me.Label11.Name = "Label11"
        Me.Label11.Size = New System.Drawing.Size(36, 16)
        Me.Label11.TabIndex = 32
        Me.Label11.Text = "Mode:"
        '
        'Label12
        '
        Me.Label12.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label12.Location = New System.Drawing.Point(40, 160)
        Me.Label12.Name = "Label12"
        Me.Label12.Size = New System.Drawing.Size(36, 16)
        Me.Label12.TabIndex = 33
        Me.Label12.Text = "Servo:"
        '
        'Label13
        '
        Me.Label13.Location = New System.Drawing.Point(104, 160)
        Me.Label13.Name = "Label13"
        Me.Label13.Size = New System.Drawing.Size(80, 16)
        Me.Label13.TabIndex = 34
        '
        'Label14
        '
        Me.Label14.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label14.Location = New System.Drawing.Point(8, 104)
        Me.Label14.Name = "Label14"
        Me.Label14.Size = New System.Drawing.Size(128, 16)
        Me.Label14.TabIndex = 35
        Me.Label14.Text = "Communication Status:"
        '
        'Label15
        '
        Me.Label15.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label15.Location = New System.Drawing.Point(136, 104)
        Me.Label15.Name = "Label15"
        Me.Label15.Size = New System.Drawing.Size(96, 16)
        Me.Label15.TabIndex = 36
        Me.Label15.Text = "Disconnected"
        '
        'CheckBox1
        '
        Me.CheckBox1.Checked = True
        Me.CheckBox1.CheckState =
System.Windows.Forms.CheckState.Checked
        Me.CheckBox1.Location = New System.Drawing.Point(80, 136)
        Me.CheckBox1.Name = "CheckBox1"
        Me.CheckBox1.Size = New System.Drawing.Size(16, 16)
        Me.CheckBox1.TabIndex = 25
        Me.CheckBox1.Text = "Teach / Play"
        '
        'GroupBox2
        '
        Me.GroupBox2.Controls.Add(Me.Button7)
        Me.GroupBox2.Controls.Add(Me.TextBox4)
        Me.GroupBox2.Controls.Add(Me.Button5)
        Me.GroupBox2.Controls.Add(Me.Button6)
        Me.GroupBox2.Controls.Add(Me.TextBox6)
        Me.GroupBox2.Controls.Add(Me.CmdDownLoad)
        Me.GroupBox2.Controls.Add(Me.Button9)
        Me.GroupBox2.Font = New System.Drawing.Font("Arial",
8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.GroupBox2.Location = New System.Drawing.Point(32, 238)
        Me.GroupBox2.Name = "GroupBox2"
        Me.GroupBox2.Size = New System.Drawing.Size(240, 192)
        Me.GroupBox2.TabIndex = 71
        Me.GroupBox2.TabStop = False
        Me.GroupBox2.Text = "Download / Upload"
        '
        'Button7
        '
        Me.Button7.BackColor =
System.Drawing.Color.FromArgb(CType(255, Byte), CType(255,
Byte), CType(192, Byte))
        Me.Button7.Cursor = System.Windows.Forms.Cursors.Default
```

```
    Me.Button7.Font = New System.Drawing.Font("Arial", 8.0!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.Button7.ForeColor =
System.Drawing.SystemColors.ControlText
    Me.Button7.Location = New System.Drawing.Point(128, 48)
    Me.Button7.Name = "Button7"
    Me.Button7.RightToLeft =
System.Windows.Forms.RightToLeft.No
    Me.Button7.Size = New System.Drawing.Size(104, 40)
    Me.Button7.TabIndex = 27
    Me.Button7.Text = "Upload Job"
    '
    'TextBox4
    '
    Me.TextBox4.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.TextBox4.Location = New System.Drawing.Point(128, 24)
    Me.TextBox4.Name = "TextBox4"
    Me.TextBox4.Size = New System.Drawing.Size(104, 20)
    Me.TextBox4.TabIndex = 28
    Me.TextBox4.Text = "BAGS1.JBI"
    '
    'Button5
    '
    Me.Button5.BackColor =
System.Drawing.Color.FromArgb(CType(255, Byte), CType(255,
Byte), CType(192, Byte))
    Me.Button5.Cursor = System.Windows.Forms.Cursors.Default
    Me.Button5.Font = New System.Drawing.Font("Arial", 8.0!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.Button5.ForeColor =
System.Drawing.SystemColors.ControlText
    Me.Button5.Location = New System.Drawing.Point(8, 96)
    Me.Button5.Name = "Button5"
    Me.Button5.RightToLeft =
System.Windows.Forms.RightToLeft.No
    Me.Button5.Size = New System.Drawing.Size(104, 40)
    Me.Button5.TabIndex = 23
    Me.Button5.Text = "Delete Job"
    '
    'Button6
    '
    Me.Button6.BackColor =
System.Drawing.Color.FromArgb(CType(255, Byte), CType(255,
Byte), CType(192, Byte))
    Me.Button6.Cursor = System.Windows.Forms.Cursors.Default
    Me.Button6.Font = New System.Drawing.Font("Arial", 8.0!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.Button6.ForeColor =
System.Drawing.SystemColors.ControlText
    Me.Button6.Location = New System.Drawing.Point(8, 144)
    Me.Button6.Name = "Button6"
    Me.Button6.RightToLeft =
System.Windows.Forms.RightToLeft.No
    Me.Button6.Size = New System.Drawing.Size(104, 40)
    Me.Button6.TabIndex = 24
    Me.Button6.Text = "Run Job"
    '
    'TextBox6
    '
    Me.TextBox6.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.TextBox6.Location = New System.Drawing.Point(8, 24)
    Me.TextBox6.Name = "TextBox6"
    Me.TextBox6.Size = New System.Drawing.Size(104, 20)
    Me.TextBox6.TabIndex = 63
    Me.TextBox6.Text = "POLICY1.JBI"
    '
    'CmdDownLoad
    '
    Me.CmdDownLoad.BackColor =
System.Drawing.Color.FromArgb(CType(255, Byte), CType(255,
Byte), CType(192, Byte))
    Me.CmdDownLoad.Cursor =
System.Windows.Forms.Cursors.Default
    Me.CmdDownLoad.Font = New System.Drawing.Font("Arial",
8.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.CmdDownLoad.ForeColor =
System.Drawing.SystemColors.ControlText
    Me.CmdDownLoad.Location = New System.Drawing.Point(8,
48)
    Me.CmdDownLoad.Name = "CmdDownLoad"
    Me.CmdDownLoad.RightToLeft =
System.Windows.Forms.RightToLeft.No
    Me.CmdDownLoad.Size = New System.Drawing.Size(104, 40)
    Me.CmdDownLoad.TabIndex = 1
    Me.CmdDownLoad.Text = "Download Job"
    '
    'Button9
    '
    Me.Button9.BackColor = System.Drawing.Color.Red
    Me.Button9.Cursor = System.Windows.Forms.Cursors.Default
    Me.Button9.Font = New System.Drawing.Font("Arial", 8.0!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.Button9.ForeColor = System.Drawing.Color.Yellow
    Me.Button9.Location = New System.Drawing.Point(128, 96)
    Me.Button9.Name = "Button9"
    Me.Button9.RightToLeft =
System.Windows.Forms.RightToLeft.No
    Me.Button9.Size = New System.Drawing.Size(104, 40)
    Me.Button9.TabIndex = 30
    Me.Button9.Text = "Emergency Stop"
    '
    'GroupBox6
    '
    Me.GroupBox6.Controls.Add(Me.Label7)
    Me.GroupBox6.Controls.Add(Me.Label8)
    Me.GroupBox6.Controls.Add(Me.TextBox1)
    Me.GroupBox6.Controls.Add(Me.TextBox2)
    Me.GroupBox6.Controls.Add(Me.Label2)
    Me.GroupBox6.Controls.Add(Me.Label3)
    Me.GroupBox6.Controls.Add(Me.Label4)
    Me.GroupBox6.Controls.Add(Me.Label5)
    Me.GroupBox6.Controls.Add(Me.Label6)
    Me.GroupBox6.Controls.Add(Me.TextBox3)
    Me.GroupBox6.Controls.Add(Me.Label9)
    Me.GroupBox6.Controls.Add(Me.Button4)
    Me.GroupBox6.Font = New System.Drawing.Font("Arial",
8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.GroupBox6.Location = New System.Drawing.Point(304, 26)
    Me.GroupBox6.Name = "GroupBox6"
    Me.GroupBox6.Size = New System.Drawing.Size(208, 176)
    Me.GroupBox6.TabIndex = 72
    Me.GroupBox6.TabStop = False
    Me.GroupBox6.Text = "Messeges"
    '
    'Label7
    '
    Me.Label7.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.Label7.Location = New System.Drawing.Point(152, 72)
    Me.Label7.Name = "Label7"
    Me.Label7.Size = New System.Drawing.Size(32, 16)
    Me.Label7.TabIndex = 18
    Me.Label7.Text = "(1)"
    '
    'Label8
    '
    Me.Label8.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.Label8.Location = New System.Drawing.Point(152, 40)
    Me.Label8.Name = "Label8"
    Me.Label8.Size = New System.Drawing.Size(48, 16)
    Me.Label8.TabIndex = 19
    Me.Label8.Text = "(not -1)"
    '
```

```
        'TextBox1
        '
        Me.TextBox1.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox1.Location = New System.Drawing.Point(88, 32)
        Me.TextBox1.Name = "TextBox1"
        Me.TextBox1.Size = New System.Drawing.Size(64, 20)
        Me.TextBox1.TabIndex = 8
        Me.TextBox1.Text = ""
        '
        'TextBox2
        '
        Me.TextBox2.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox2.Location = New System.Drawing.Point(88, 72)
        Me.TextBox2.Name = "TextBox2"
        Me.TextBox2.Size = New System.Drawing.Size(64, 20)
        Me.TextBox2.TabIndex = 9
        Me.TextBox2.Text = ""
        '
        'Label2
        '
        Me.Label2.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label2.Location = New System.Drawing.Point(96, 16)
        Me.Label2.Name = "Label2"
        Me.Label2.Size = New System.Drawing.Size(32, 16)
        Me.Label2.TabIndex = 11
        Me.Label2.Text = "nCid"
        '
        'Label3
        '
        Me.Label3.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label3.Location = New System.Drawing.Point(96, 56)
        Me.Label3.Name = "Label3"
        Me.Label3.Size = New System.Drawing.Size(32, 16)
        Me.Label3.TabIndex = 12
        Me.Label3.Text = "rc"
        '
        'Label4
        '
        Me.Label4.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label4.Location = New System.Drawing.Point(8, 32)
        Me.Label4.Name = "Label4"
        Me.Label4.Size = New System.Drawing.Size(64, 16)
        Me.Label4.TabIndex = 13
        Me.Label4.Text = "BscOpen"
        '
        'Label5
        '
        Me.Label5.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label5.Location = New System.Drawing.Point(8, 72)
        Me.Label5.Name = "Label5"
        Me.Label5.Size = New System.Drawing.Size(64, 16)
        Me.Label5.TabIndex = 14
        Me.Label5.Text = "BscConnect"
        '
        'Label6
        '
        Me.Label6.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label6.Location = New System.Drawing.Point(8, 104)
        Me.Label6.Name = "Label6"
        Me.Label6.Size = New System.Drawing.Size(80, 16)
        Me.Label6.TabIndex = 16
        Me.Label6.Text = "BscDownLoad"
        '
        'TextBox3
        '
        Me.TextBox3.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox3.Location = New System.Drawing.Point(88, 104)
        Me.TextBox3.Name = "TextBox3"
        Me.TextBox3.Size = New System.Drawing.Size(64, 20)
        Me.TextBox3.TabIndex = 15
        Me.TextBox3.Text = ""
        '
        'Label9
        '
        Me.Label9.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label9.Location = New System.Drawing.Point(152, 112)
        Me.Label9.Name = "Label9"
        Me.Label9.Size = New System.Drawing.Size(32, 16)
        Me.Label9.TabIndex = 20
        Me.Label9.Text = "(0)"
        '
        'Button4
        '
        Me.Button4.BackColor =
System.Drawing.Color.FromArgb(CType(255, Byte), CType(255,
Byte), CType(192, Byte))
        Me.Button4.Cursor = System.Windows.Forms.Cursors.Default
        Me.Button4.Font = New System.Drawing.Font("Arial", 8.0!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button4.ForeColor =
System.Drawing.SystemColors.ControlText
        Me.Button4.Location = New System.Drawing.Point(72, 136)
        Me.Button4.Name = "Button4"
        Me.Button4.RightToLeft =
System.Windows.Forms.RightToLeft.No
        Me.Button4.Size = New System.Drawing.Size(56, 32)
        Me.Button4.TabIndex = 21
        Me.Button4.Text = "Clear"
        '
        'GroupBox11
        '
        Me.GroupBox11.Controls.Add(Me.TextBox11)
        Me.GroupBox11.Controls.Add(Me.Label18)
        Me.GroupBox11.Controls.Add(Me.Label25)
        Me.GroupBox11.Controls.Add(Me.CheckBox5)
        Me.GroupBox11.Controls.Add(Me.Label17)
        Me.GroupBox11.Controls.Add(Me.Label1)
        Me.GroupBox11.Controls.Add(Me.CheckBox4)
        Me.GroupBox11.Controls.Add(Me.TextBox10)
        Me.GroupBox11.Controls.Add(Me.Label23)
        Me.GroupBox11.Controls.Add(Me.Label24)
        Me.GroupBox11.Location = New System.Drawing.Point(552,
27)
        Me.GroupBox11.Name = "GroupBox11"
        Me.GroupBox11.Size = New System.Drawing.Size(168, 176)
        Me.GroupBox11.TabIndex = 83
        Me.GroupBox11.TabStop = False
        Me.GroupBox11.Text = "Operational Mode"
        '
        'TextBox11
        '
        Me.TextBox11.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox11.Location = New System.Drawing.Point(16, 152)
        Me.TextBox11.Name = "TextBox11"
        Me.TextBox11.Size = New System.Drawing.Size(72, 20)
        Me.TextBox11.TabIndex = 88
        Me.TextBox11.Text = "5"
        '
        'Label18
        '
        Me.Label18.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label18.Location = New System.Drawing.Point(96, 152)
        Me.Label18.Name = "Label18"
        Me.Label18.Size = New System.Drawing.Size(24, 16)
        Me.Label18.TabIndex = 90
        Me.Label18.Text = "cm"
        '
        'Label25
        '
        Me.Label25.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label25.Location = New System.Drawing.Point(16, 128)
        Me.Label25.Name = "Label25"
        Me.Label25.Size = New System.Drawing.Size(88, 16)
        Me.Label25.TabIndex = 89
        Me.Label25.Text = "Wrist Step Size:"
        '
        'CheckBox5
        '
        Me.CheckBox5.Checked = True
        Me.CheckBox5.CheckState =
System.Windows.Forms.CheckState.Checked
        Me.CheckBox5.Location = New System.Drawing.Point(32, 48)
        Me.CheckBox5.Name = "CheckBox5"
        Me.CheckBox5.Size = New System.Drawing.Size(16, 16)
        Me.CheckBox5.TabIndex = 87
        '
        'Label17
        '
        Me.Label17.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Underline,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label17.Location = New System.Drawing.Point(16, 24)
        Me.Label17.Name = "Label17"
        Me.Label17.Size = New System.Drawing.Size(80, 16)
        Me.Label17.TabIndex = 86
        Me.Label17.Text = "Incremental"
        '
        'Label1
        '
        Me.Label1.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Underline,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label1.Location = New System.Drawing.Point(96, 24)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(64, 16)
        Me.Label1.TabIndex = 85
        Me.Label1.Text = "Continious"
        '
        'CheckBox4
        '
        Me.CheckBox4.Location = New System.Drawing.Point(112, 48)
        Me.CheckBox4.Name = "CheckBox4"
        Me.CheckBox4.Size = New System.Drawing.Size(16, 16)
        Me.CheckBox4.TabIndex = 84
        '
        'TextBox10
        '
        Me.TextBox10.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox10.Location = New System.Drawing.Point(16, 96)
        Me.TextBox10.Name = "TextBox10"
        Me.TextBox10.Size = New System.Drawing.Size(72, 20)
        Me.TextBox10.TabIndex = 84
        Me.TextBox10.Text = "10"
        '
        'Label23
        '
        Me.Label23.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label23.Location = New System.Drawing.Point(96, 96)
        Me.Label23.Name = "Label23"
        Me.Label23.Size = New System.Drawing.Size(24, 16)
        Me.Label23.TabIndex = 86
        Me.Label23.Text = "cm"
        '
        'Label24
        '
        Me.Label24.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label24.Location = New System.Drawing.Point(16, 72)
        Me.Label24.Name = "Label24"
        Me.Label24.Size = New System.Drawing.Size(88, 16)
        Me.Label24.TabIndex = 85
        Me.Label24.Text = "Arm Step Size:"
        '
        'Label42
        '
        Me.Label42.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label42.Location = New System.Drawing.Point(16, 189)
        Me.Label42.Name = "Label42"
        Me.Label42.Size = New System.Drawing.Size(88, 16)
        Me.Label42.TabIndex = 294
        Me.Label42.Text = "Matlab Function:"
        '
        'TextBox76
        '
        Me.TextBox76.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox76.Location = New System.Drawing.Point(16, 205)
        Me.TextBox76.Name = "TextBox76"
        Me.TextBox76.Size = New System.Drawing.Size(128, 20)
        Me.TextBox76.TabIndex = 293
        Me.TextBox76.Text = "toast18(0,0)"
        '
        'Label39
        '
        Me.Label39.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label39.Location = New System.Drawing.Point(16, 136)
        Me.Label39.Name = "Label39"
        Me.Label39.Size = New System.Drawing.Size(112, 16)
        Me.Label39.TabIndex = 292
        Me.Label39.Text = "Resulting Sequence:"
        '
        'TextBox75
        '
        Me.TextBox75.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox75.Location = New System.Drawing.Point(16, 156)
        Me.TextBox75.Name = "TextBox75"
        Me.TextBox75.Size = New System.Drawing.Size(128, 20)
        Me.TextBox75.TabIndex = 291
        Me.TextBox75.Text = ""
        '
        'Button3
        '
        Me.Button3.BackColor = System.Drawing.Color.SpringGreen
        Me.Button3.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button3.Location = New System.Drawing.Point(16, 32)
        Me.Button3.Name = "Button3"
        Me.Button3.Size = New System.Drawing.Size(112, 32)
        Me.Button3.TabIndex = 290
        Me.Button3.Text = "Create Scheduling  Policy"
        '
        'Label103
        '
        Me.Label103.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label103.Location = New System.Drawing.Point(16, 240)
        Me.Label103.Name = "Label103"
        Me.Label103.Size = New System.Drawing.Size(120, 16)
        Me.Label103.TabIndex = 299
        Me.Label103.Text = "Directory:"
        '
        'TextBox148
        '
        Me.TextBox148.Font = New System.Drawing.Font("Arial", 8.25!, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox148.Location = New System.Drawing.Point(16, 256)
        Me.TextBox148.Name = "TextBox148"
        Me.TextBox148.Size = New System.Drawing.Size(128, 20)
        Me.TextBox148.TabIndex = 298
        Me.TextBox148.Text = "c:/amit/toast/toast/matlab/temp"
        '
        'Button8
        '
        Me.Button8.BackColor = System.Drawing.Color.Aqua
        Me.Button8.Font = New System.Drawing.Font("Arial", 8.25!, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button8.Location = New System.Drawing.Point(200, 32)
        Me.Button8.Name = "Button8"
        Me.Button8.Size = New System.Drawing.Size(112, 32)
        Me.Button8.TabIndex = 300
        Me.Button8.Text = "Run Toasting Sequence"
        '
        'TextBox8
        '
        Me.TextBox8.Font = New System.Drawing.Font("Arial", 8.25!, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox8.Location = New System.Drawing.Point(200, 104)
        Me.TextBox8.Name = "TextBox8"
        Me.TextBox8.Size = New System.Drawing.Size(128, 20)
        Me.TextBox8.TabIndex = 301
        Me.TextBox8.Text = ""
        '
        'Label16
        '
        Me.Label16.Font = New System.Drawing.Font("Arial", 8.25!, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label16.Location = New System.Drawing.Point(200, 80)
        Me.Label16.Name = "Label16"
        Me.Label16.Size = New System.Drawing.Size(120, 16)
        Me.Label16.TabIndex = 302
        Me.Label16.Text = "Next Station:"
        '
        'TextBox13
        '
        Me.TextBox13.Font = New System.Drawing.Font("Arial", 8.25!, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox13.Location = New System.Drawing.Point(200, 154)
        Me.TextBox13.Name = "TextBox13"
        Me.TextBox13.Size = New System.Drawing.Size(128, 20)
        Me.TextBox13.TabIndex = 307
        Me.TextBox13.Text = ""
        '
        'Label19
        '
        Me.Label19.Font = New System.Drawing.Font("Arial", 8.25!, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label19.Location = New System.Drawing.Point(200, 138)
        Me.Label19.Name = "Label19"
        Me.Label19.Size = New System.Drawing.Size(120, 16)
        Me.Label19.TabIndex = 308
        Me.Label19.Text = "Sequence Step:"
        '
        'GroupBox3
        '
        Me.GroupBox3.Controls.Add(Me.Label30)
        Me.GroupBox3.Controls.Add(Me.Label29)
        Me.GroupBox3.Controls.Add(Me.TextBox5)
        Me.GroupBox3.Controls.Add(Me.Label28)
        Me.GroupBox3.Controls.Add(Me.Label27)
        Me.GroupBox3.Controls.Add(Me.Label26)
        Me.GroupBox3.Controls.Add(Me.ComboBox1)
        Me.GroupBox3.Controls.Add(Me.Label22)
        Me.GroupBox3.Controls.Add(Me.Label21)
        Me.GroupBox3.Controls.Add(Me.Label20)
        Me.GroupBox3.Controls.Add(Me.Button3)
        Me.GroupBox3.Controls.Add(Me.Label42)
        Me.GroupBox3.Controls.Add(Me.TextBox76)
        Me.GroupBox3.Controls.Add(Me.Label39)
        Me.GroupBox3.Controls.Add(Me.TextBox75)
        Me.GroupBox3.Controls.Add(Me.Label103)
        Me.GroupBox3.Controls.Add(Me.TextBox148)
        Me.GroupBox3.Controls.Add(Me.Button8)
        Me.GroupBox3.Controls.Add(Me.TextBox8)
        Me.GroupBox3.Controls.Add(Me.Label16)
        Me.GroupBox3.Controls.Add(Me.TextBox13)
        Me.GroupBox3.Controls.Add(Me.Label19)
        Me.GroupBox3.Font = New System.Drawing.Font("Microsoft Sans Serif", 8.25!, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(177, Byte))
        Me.GroupBox3.Location = New System.Drawing.Point(309, 239)
        Me.GroupBox3.Name = "GroupBox3"
        Me.GroupBox3.Size = New System.Drawing.Size(488, 304)
        Me.GroupBox3.TabIndex = 309
        Me.GroupBox3.TabStop = False
        Me.GroupBox3.Text = "Toasting System"
        '
        'Label30
        '
        Me.Label30.BackColor = System.Drawing.Color.Orange
        Me.Label30.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.Label30.Font = New System.Drawing.Font("Arial", 9.75!, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label30.Location = New System.Drawing.Point(200, 200)
        Me.Label30.Name = "Label30"
        Me.Label30.Size = New System.Drawing.Size(120, 40)
        Me.Label30.TabIndex = 320
        Me.Label30.Text = "Policy Creation Completed"
        Me.Label30.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        Me.Label30.Visible = False
        '
        'Label29
        '
        Me.Label29.Font = New System.Drawing.Font("Arial", 8.25!, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label29.Location = New System.Drawing.Point(352, 240)
        Me.Label29.Name = "Label29"
        Me.Label29.Size = New System.Drawing.Size(120, 16)
        Me.Label29.TabIndex = 319
        Me.Label29.Text = "# Of Finished Toasts"
        '
        'TextBox5
        '
        Me.TextBox5.Font = New System.Drawing.Font("Arial", 8.25!, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox5.Location = New System.Drawing.Point(352, 256)
        Me.TextBox5.Name = "TextBox5"
        Me.TextBox5.Size = New System.Drawing.Size(120, 20)
        Me.TextBox5.TabIndex = 318
        Me.TextBox5.Text = ""
        '
        'Label28
        '
        Me.Label28.BackColor = System.Drawing.Color.Red
        Me.Label28.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.Label28.Font = New System.Drawing.Font("Arial", 9.75!, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label28.Location = New System.Drawing.Point(352, 104)
        Me.Label28.Name = "Label28"
        Me.Label28.Size = New System.Drawing.Size(120, 24)
        Me.Label28.TabIndex = 317
        Me.Label28.Text = "Not Processing"
        Me.Label28.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'Label27
        '
        Me.Label27.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label27.Location = New System.Drawing.Point(352, 149)
        Me.Label27.Name = "Label27"
        Me.Label27.Size = New System.Drawing.Size(120, 16)
        Me.Label27.TabIndex = 316
        Me.Label27.Text = "Machine Indicators:"
        '
        'Label26
        '
        Me.Label26.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label26.Location = New System.Drawing.Point(16, 80)
        Me.Label26.Name = "Label26"
        Me.Label26.Size = New System.Drawing.Size(120, 16)
        Me.Label26.TabIndex = 315
        Me.Label26.Text = "Number Of Toasts:"
        '
        'ComboBox1
        '
        Me.ComboBox1.Items.AddRange(New Object() {"1 Toast", "2
Toasts", "3 Toasts", "4 Toasts"})
        Me.ComboBox1.Location = New System.Drawing.Point(16,
104)
        Me.ComboBox1.Name = "ComboBox1"
        Me.ComboBox1.Size = New System.Drawing.Size(128, 21)
        Me.ComboBox1.TabIndex = 312
        Me.ComboBox1.Text = "1 Toast"
        '
        'Label22
        '
        Me.Label22.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label22.Location = New System.Drawing.Point(352, 80)
        Me.Label22.Name = "Label22"
        Me.Label22.Size = New System.Drawing.Size(120, 16)
        Me.Label22.TabIndex = 311
        Me.Label22.Text = "Process Indicator:"
        '
        'Label21
        '
        Me.Label21.BackColor = System.Drawing.Color.Red
        Me.Label21.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.Label21.Font = New System.Drawing.Font("Arial", 9.75!,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label21.Location = New System.Drawing.Point(352, 201)
        Me.Label21.Name = "Label21"
        Me.Label21.Size = New System.Drawing.Size(120, 24)
        Me.Label21.TabIndex = 310
        Me.Label21.Text = "Not Buttering"
        Me.Label21.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'Label20
        '
        Me.Label20.BackColor = System.Drawing.Color.Red
        Me.Label20.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.Label20.Font = New System.Drawing.Font("Arial", 9.75!,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label20.Location = New System.Drawing.Point(352, 169)
        Me.Label20.Name = "Label20"
        Me.Label20.Size = New System.Drawing.Size(120, 24)
        Me.Label20.TabIndex = 309
        Me.Label20.Text = "Not Toasting"
        Me.Label20.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'Button17
        '
        Me.Button17.BackColor = System.Drawing.Color.Aqua
        Me.Button17.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button17.Location = New System.Drawing.Point(728, 136)
        Me.Button17.Name = "Button17"
        Me.Button17.Size = New System.Drawing.Size(112, 32)
        Me.Button17.TabIndex = 321
        Me.Button17.Text = "Run Toasting Sequence"
        Me.Button17.Visible = False
        '
        'MainMenu1
        '
        Me.MainMenu1.MenuItems.AddRange(New
System.Windows.Forms.MenuItem() {Me.MenuItem1})
        '
        'MenuItem1
        '
        Me.MenuItem1.Index = 0
        Me.MenuItem1.Text = "Exit"
        '
        'TabControl1
        '
        Me.TabControl1.Controls.Add(Me.TabPage1)
        Me.TabControl1.Controls.Add(Me.TabPage2)
        Me.TabControl1.Location = New System.Drawing.Point(0, 8)
        Me.TabControl1.Name = "TabControl1"
        Me.TabControl1.SelectedIndex = 0
        Me.TabControl1.Size = New System.Drawing.Size(976, 608)
        Me.TabControl1.TabIndex = 310
        '
        'TabPage1
        '
        Me.TabPage1.Controls.Add(Me.Button25)
        Me.TabPage1.Controls.Add(Me.Button24)
        Me.TabPage1.Controls.Add(Me.Button23)
        Me.TabPage1.Controls.Add(Me.Button22)
        Me.TabPage1.Controls.Add(Me.Button20)
        Me.TabPage1.Controls.Add(Me.Button21)
        Me.TabPage1.Controls.Add(Me.Button19)
        Me.TabPage1.Controls.Add(Me.GroupBox2)
        Me.TabPage1.Controls.Add(Me.GroupBox1)
        Me.TabPage1.Controls.Add(Me.GroupBox6)
        Me.TabPage1.Controls.Add(Me.GroupBox11)
        Me.TabPage1.Controls.Add(Me.GroupBox3)
        Me.TabPage1.Controls.Add(Me.Button18)
        Me.TabPage1.Controls.Add(Me.TextBox15)
        Me.TabPage1.Controls.Add(Me.Button17)
        Me.TabPage1.Location = New System.Drawing.Point(4, 22)
        Me.TabPage1.Name = "TabPage1"
        Me.TabPage1.Size = New System.Drawing.Size(968, 582)
        Me.TabPage1.TabIndex = 0
        Me.TabPage1.Text = "Scheduling"
        '
        'Button25
        '
        Me.Button25.Location = New System.Drawing.Point(216, 440)
        Me.Button25.Name = "Button25"
        Me.Button25.Size = New System.Drawing.Size(72, 24)
        Me.Button25.TabIndex = 330
        Me.Button25.Text = "Button25"
        Me.Button25.Visible = False
        '
        'Button24
        '
        Me.Button24.BackColor = System.Drawing.Color.Aqua
        Me.Button24.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button24.Location = New System.Drawing.Point(840, 424)
        Me.Button24.Name = "Button24"
        Me.Button24.Size = New System.Drawing.Size(112, 32)
        Me.Button24.TabIndex = 329
        Me.Button24.Text = "Grasp2"
        Me.Button24.Visible = False
        '
        'Button23
        '
        Me.Button23.BackColor = System.Drawing.Color.Aqua
        Me.Button23.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button23.Location = New System.Drawing.Point(808, 64)
        Me.Button23.Name = "Button23"
        Me.Button23.Size = New System.Drawing.Size(112, 32)
        Me.Button23.TabIndex = 328
        Me.Button23.Text = "CLOSE"
        Me.Button23.Visible = False
        '
        'Button22
        '
        Me.Button22.BackColor = System.Drawing.Color.Aqua
        Me.Button22.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button22.Location = New System.Drawing.Point(808, 16)
        Me.Button22.Name = "Button22"
        Me.Button22.Size = New System.Drawing.Size(112, 32)
        Me.Button22.TabIndex = 327
        Me.Button22.Text = "OPEN"
        Me.Button22.Visible = False
        '
        'Button20
        '
        Me.Button20.BackColor = System.Drawing.Color.Aqua
        Me.Button20.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button20.Location = New System.Drawing.Point(832, 272)
        Me.Button20.Name = "Button20"
        Me.Button20.Size = New System.Drawing.Size(112, 32)
        Me.Button20.TabIndex = 326
        Me.Button20.Text = "HOME4"
        Me.Button20.Visible = False
        '
        'Button21
        '
        Me.Button21.BackColor = System.Drawing.Color.Aqua
        Me.Button21.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button21.Location = New System.Drawing.Point(832, 224)
        Me.Button21.Name = "Button21"
        Me.Button21.Size = New System.Drawing.Size(112, 32)
        Me.Button21.TabIndex = 325
        Me.Button21.Text = "HOME2"
        Me.Button21.Visible = False
        '
        'Button19
        '
        Me.Button19.BackColor = System.Drawing.Color.Aqua
        Me.Button19.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button19.Location = New System.Drawing.Point(832, 184)
        Me.Button19.Name = "Button19"
        Me.Button19.Size = New System.Drawing.Size(112, 32)
        Me.Button19.TabIndex = 323
        Me.Button19.Text = "HOME1"
        Me.Button19.Visible = False
        '
        'Button18
        '
        Me.Button18.BackColor = System.Drawing.Color.Aqua
        Me.Button18.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button18.Location = New System.Drawing.Point(832, 312)
        Me.Button18.Name = "Button18"
        Me.Button18.Size = New System.Drawing.Size(112, 32)
        Me.Button18.TabIndex = 322
        Me.Button18.Text = "Grasp1"
        Me.Button18.Visible = False
        '
        'TextBox15
        '
        Me.TextBox15.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TextBox15.Location = New System.Drawing.Point(824,
376)
        Me.TextBox15.Name = "TextBox15"
        Me.TextBox15.Size = New System.Drawing.Size(128, 20)
        Me.TextBox15.TabIndex = 322
        Me.TextBox15.Text = "4"
        Me.TextBox15.Visible = False
        '
        'TabPage2
        '
        Me.TabPage2.Controls.Add(Me.GroupBox5)
        Me.TabPage2.Controls.Add(Me.GroupBox4)
        Me.TabPage2.Location = New System.Drawing.Point(4, 22)
        Me.TabPage2.Name = "TabPage2"
        Me.TabPage2.Size = New System.Drawing.Size(968, 582)
        Me.TabPage2.TabIndex = 1
        Me.TabPage2.Text = "Positioning"
        '
        'GroupBox5
        '
        Me.GroupBox5.Controls.Add(Me.Button32)
        Me.GroupBox5.Controls.Add(Me.Button30)
        Me.GroupBox5.Controls.Add(Me.Label33)
        Me.GroupBox5.Controls.Add(Me.TextBox7)
        Me.GroupBox5.Controls.Add(Me.Button13)
        Me.GroupBox5.Controls.Add(Me.Button15)
        Me.GroupBox5.Controls.Add(Me.Button14)
        Me.GroupBox5.Controls.Add(Me.Label34)
        Me.GroupBox5.Controls.Add(Me.Button16)
        Me.GroupBox5.Location = New System.Drawing.Point(32, 32)
        Me.GroupBox5.Name = "GroupBox5"
        Me.GroupBox5.Size = New System.Drawing.Size(448, 376)
        Me.GroupBox5.TabIndex = 323
        Me.GroupBox5.TabStop = False
        Me.GroupBox5.Text = "Autonomous Learning"
        '
        'Button32
        '
        Me.Button32.BackColor                                    =
System.Drawing.Color.FromArgb(CType(128,    Byte),    CType(255,
Byte), CType(255, Byte))
        Me.Button32.Location = New System.Drawing.Point(16, 79)
        Me.Button32.Name = "Button32"
        Me.Button32.Size = New System.Drawing.Size(120, 32)
        Me.Button32.TabIndex = 335
        Me.Button32.Text = "Grasp Toast"
        '
        'Button30
        '
        Me.Button30.BackColor                                    =
System.Drawing.Color.FromArgb(CType(192,    Byte),    CType(255,
Byte), CType(255, Byte))
        Me.Button30.Location = New System.Drawing.Point(16, 32)
        Me.Button30.Name = "Button30"
        Me.Button30.Size = New System.Drawing.Size(120, 32)
        Me.Button30.TabIndex = 334
        Me.Button30.Text = "Home Robot"
        '
        'Label33
        '
        Me.Label33.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label33.Location = New System.Drawing.Point(231, 36)
        Me.Label33.Name = "Label33"
        Me.Label33.Size = New System.Drawing.Size(112, 16)
        Me.Label33.TabIndex = 323
        Me.Label33.Text = "Next Move:"
        '
        'TextBox7
        '
        Me.TextBox7.Font = New System.Drawing.Font("Arial", 8.25!,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
```

```
        Me.TextBox7.Location = New System.Drawing.Point(231, 54)
        Me.TextBox7.Name = "TextBox7"
        Me.TextBox7.Size = New System.Drawing.Size(128, 20)
        Me.TextBox7.TabIndex = 322
        Me.TextBox7.Text = ""
        '
        'Button13
        '
        Me.Button13.BackColor = System.Drawing.Color.Aqua
        Me.Button13.Location = New System.Drawing.Point(16, 128)
        Me.Button13.Name = "Button13"
        Me.Button13.Size = New System.Drawing.Size(120, 32)
        Me.Button13.TabIndex = 5
        Me.Button13.Text = "Initialize Environment"
        '
        'Button15
        '
        Me.Button15.BackColor =
System.Drawing.Color.FromArgb(CType(0, Byte), CType(192,
Byte), CType(192, Byte))
        Me.Button15.Location = New System.Drawing.Point(16, 176)
        Me.Button15.Name = "Button15"
        Me.Button15.Size = New System.Drawing.Size(120, 32)
        Me.Button15.TabIndex = 9
        Me.Button15.Text = "Generate Path"
        '
        'Button14
        '
        Me.Button14.BackColor =
System.Drawing.Color.FromArgb(CType(0, Byte), CType(192,
Byte), CType(0, Byte))
        Me.Button14.Location = New System.Drawing.Point(16, 224)
        Me.Button14.Name = "Button14"
        Me.Button14.Size = New System.Drawing.Size(120, 32)
        Me.Button14.TabIndex = 6
        Me.Button14.Text = "Operate Robot"
        '
        'Label34
        '
        Me.Label34.BackColor = System.Drawing.Color.Orange
        Me.Label34.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.Label34.Font = New System.Drawing.Font("Arial", 9.75!,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label34.Location = New System.Drawing.Point(231, 136)
        Me.Label34.Name = "Label34"
        Me.Label34.Size = New System.Drawing.Size(128, 80)
        Me.Label34.TabIndex = 321
        Me.Label34.Text = "Reached Goal"
        Me.Label34.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        Me.Label34.Visible = False
        '
        'Button16
        '
        Me.Button16.BackColor = System.Drawing.Color.Red
        Me.Button16.Font = New System.Drawing.Font("Microsoft Sans
Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(177, Byte))
        Me.Button16.ForeColor = System.Drawing.Color.Yellow
        Me.Button16.Location = New System.Drawing.Point(16, 280)
        Me.Button16.Name = "Button16"
        Me.Button16.Size = New System.Drawing.Size(120, 48)
        Me.Button16.TabIndex = 324
        Me.Button16.Text = "Stop Operation"
        '
        'GroupBox4
        '
        Me.GroupBox4.Controls.Add(Me.Button29)
        Me.GroupBox4.Controls.Add(Me.Button26)
        Me.GroupBox4.Controls.Add(Me.Button28)
        Me.GroupBox4.Controls.Add(Me.Button27)
        Me.GroupBox4.Controls.Add(Me.Button10)
        Me.GroupBox4.Controls.Add(Me.Button11)
        Me.GroupBox4.Controls.Add(Me.Button2)
        Me.GroupBox4.Controls.Add(Me.Button12)
        Me.GroupBox4.Location = New System.Drawing.Point(544, 32)
        Me.GroupBox4.Name = "GroupBox4"
        Me.GroupBox4.Size = New System.Drawing.Size(368, 272)
        Me.GroupBox4.TabIndex = 322
        Me.GroupBox4.TabStop = False
        Me.GroupBox4.Text = "Manual Manipulation"
        '
        'Button28
        '
        Me.Button28.Font = New System.Drawing.Font("Microsoft Sans
Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(177, Byte))
        Me.Button28.Location = New System.Drawing.Point(271, 155)
        Me.Button28.Name = "Button28"
        Me.Button28.Size = New System.Drawing.Size(88, 23)
        Me.Button28.TabIndex = 5
        Me.Button28.Text = "Down"
        '
        'Button27
        '
        Me.Button27.Font = New System.Drawing.Font("Microsoft Sans
Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(177, Byte))
        Me.Button27.Location = New System.Drawing.Point(272, 48)
        Me.Button27.Name = "Button27"
        Me.Button27.Size = New System.Drawing.Size(88, 23)
        Me.Button27.TabIndex = 4
        Me.Button27.Text = "Up"
        '
        'Button10
        '
        Me.Button10.Font = New System.Drawing.Font("Microsoft Sans
Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(177, Byte))
        Me.Button10.Location = New System.Drawing.Point(160, 96)
        Me.Button10.Name = "Button10"
        Me.Button10.Size = New System.Drawing.Size(80, 23)
        Me.Button10.TabIndex = 1
        Me.Button10.Text = ">"
        '
        'Button11
        '
        Me.Button11.Font = New System.Drawing.Font("Microsoft Sans
Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(177, Byte))
        Me.Button11.Location = New System.Drawing.Point(107, 48)
        Me.Button11.Name = "Button11"
        Me.Button11.Size = New System.Drawing.Size(80, 23)
        Me.Button11.TabIndex = 2
        Me.Button11.Text = "/\"
        '
        'Button2
        '
        Me.Button2.Font = New System.Drawing.Font("Microsoft Sans
Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(177, Byte))
        Me.Button2.Location = New System.Drawing.Point(48, 96)
        Me.Button2.Name = "Button2"
        Me.Button2.Size = New System.Drawing.Size(88, 23)
        Me.Button2.TabIndex = 0
        Me.Button2.Text = "<"
        '
        'Button12
        '
        Me.Button12.Font = New System.Drawing.Font("Microsoft Sans
Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(177, Byte))
        Me.Button12.Location = New System.Drawing.Point(112, 144)
        Me.Button12.Name = "Button12"
        Me.Button12.Size = New System.Drawing.Size(72, 23)
        Me.Button12.TabIndex = 3
        Me.Button12.Text = "\/"
        '
        'Button26
        '
        Me.Button26.Font = New System.Drawing.Font("Microsoft Sans
Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(177, Byte))
        Me.Button26.Location = New System.Drawing.Point(16, 224)
        Me.Button26.Name = "Button26"
        Me.Button26.Size = New System.Drawing.Size(88, 23)
        Me.Button26.TabIndex = 6
        Me.Button26.Text = "Open"
        '
        'Button29
        '
        Me.Button29.Font = New System.Drawing.Font("Microsoft Sans
Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(177, Byte))
        Me.Button29.Location = New System.Drawing.Point(140, 224)
        Me.Button29.Name = "Button29"
        Me.Button29.Size = New System.Drawing.Size(88, 23)
        Me.Button29.TabIndex = 7
        Me.Button29.Text = "Close"
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(968, 606)
        Me.Controls.Add(Me.TabControl1)
        Me.Menu = Me.MainMenu1
        Me.Name = "Form1"
        Me.Text = "Toasting System"
        Me.GroupBox1.ResumeLayout(False)
        Me.GroupBox2.ResumeLayout(False)
        Me.GroupBox6.ResumeLayout(False)
        Me.GroupBox11.ResumeLayout(False)
        Me.GroupBox3.ResumeLayout(False)
        Me.TabControl1.ResumeLayout(False)
        Me.TabPage1.ResumeLayout(False)
        Me.TabPage2.ResumeLayout(False)
        Me.GroupBox5.ResumeLayout(False)
        Me.GroupBox4.ResumeLayout(False)
        Me.ResumeLayout(False)

    End Sub

#End Region

    'mode:  0...RS-232C   1...Ethernet
    Function Ms_BscOpenComm(ByVal mode%) As Integer
        '       Dim nCid As Integer
        Dim rc As Integer
        Dim IPAddrress As String
        Ms_BscOpenComm = -1
        If mode = 0 Then
            'Open the port.
            nCid = BscOpen(CurDir$, 1)

            If nCid < 0 Then GoTo Ms_BscOpenComm_Exit

            'Set serial communications parameters. ' Port, Rate, Parity,
Bits, Stop
            rc = BscSetCom(nCid, 1, 9600, 0, 8, 0)

        Else
            'Open the Ethernet line.
            nCid = BscOpen(CurDir$, PACKETETHERNET)
            If nCid < 0 Then GoTo Ms_BscOpenComm_Exit

        End If
        If rc <> 1 Then
            rc = BscClose(nCid)
            nCid = -1
            GoTo Ms_BscOpenComm_Exit
        End If

        'Connect communications line.
        rc = BscConnect(nCid)
        If rc <> 1 Then
            rc = BscClose(nCid)
            nCid = -1
            GoTo Ms_BscOpenComm_Exit
        End If

Ms_BscOpenComm_Exit:
        Ms_BscOpenComm = nCid
        TextBox1.Text = nCid
        TextBox2.Text = rc

    End Function

    Function Ms_BscCloseComm(ByRef nCid As Short) As Short
        Dim rc As Short
        'Cut the communications line.
        rc = BscDisConnect(nCid)
        'Close the port.
        rc = BscClose(nCid)
        rc = BscEnforcedClose(nCid) ' New
        Ms_BscCloseComm = rc
        TextBox1.Text = nCid
        TextBox2.Text = rc
    End Function

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button1.Click
        TextBox1.Text = Ms_BscOpenComm(0)
        If TextBox1.Text <> "-1" And TextBox2.Text = "1" Then
            Label15.Text = "Connected"
        Else
            Label15.Text = "Disconnected"
        End If
        CheckBox1.Checked = False
        CheckBox2.Checked = True
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button4.Click
        TextBox1.Text = ""
        TextBox2.Text = ""
        TextBox3.Text = ""
    End Sub

    Private Sub CmdDownLoad_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CmdDownLoad.Click
        TextBox3.Text = BscSelectJob(nCid, TextBox6.Text)
        TextBox3.Text = BscDeleteJob(nCid)
        TextBox3.Text = ""
        TextBox3.Text = BscDownLoad(nCid, TextBox6.Text)
    End Sub

    Private Sub Button7_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button7.Click
        TextBox3.Text = BscSelectJob(nCid, TextBox4.Text)
        TextBox3.Text = BscUpLoad(nCid, TextBox4.Text)
    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button5.Click
        TextBox3.Text = BscSelectJob(nCid, TextBox6.Text)
        TextBox3.Text = BscDeleteJob(nCid)
    End Sub

    Private Sub Button9_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button9.Click
        Disconnect_Robot()
    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button6.Click
        TextBox3.Text = BscSelectJob(nCid, TextBox6.Text)
        TextBox3.Text = BscDeleteJob(nCid)
        TextBox3.Text = ""
        TextBox3.Text = BscDownLoad(nCid, TextBox6.Text)
        TextBox3.Text = BscSelOneCycle(nCid)
        BscHoldOff(nCid)
        BscSetMasterJob(nCid)
        BscSelectMode(nCid, 2)
        BscServoOn(nCid)
        BscStartJob(nCid)
    End Sub
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        AddHandler t.Elapsed, AddressOf ToasterTimerFired ' for toast timer
        AddHandler b.Elapsed, AddressOf ButtererTimerFired ' for butterer
        AddHandler p.Elapsed, AddressOf PolicyTimerFired ' for policy indicator
        t.Enabled = False
        b.Enabled = False
        p.Enabled = False
        StopRun = False 'for stoping learning episode
        NumOfEpisods = 0 ' counting number of learning episodes

    End Sub

    Public Function Disconnect_Robot()
        CheckBox1.Checked = True
        BscSelectMode(nCid, 1)
        BscServoOff(nCid)
        TextBox2.Text = Ms_BscCloseComm(0)
        '   TextBox2.Text = BscEnforcedClose(0)
        Label10.Text = "Teach"
        Label13.Text = "Off"
        CheckBox2.Checked = False
        If TextBox1.Text <> "-1" And TextBox2.Text = "1" Then
            Label15.Text = "Connected"
        Else
            Label15.Text = "Disconnected"
        End If
    End Function

    Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
        If CheckBox1.Checked = True Then
            BscSelectMode(nCid, 1)
            Label10.Text = "Teach"
            Label13.Text = "Off"
        End If
        If CheckBox1.Checked = False Then
            BscSelectMode(nCid, 2)
            Label10.Text = "Play"
            BscHoldOff(nCid)
        End If
    End Sub

    Private Sub CheckBox2_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox2.CheckedChanged
        If CheckBox1.Checked = False Then
            If CheckBox2.Checked = False Then
                BscServoOff(nCid)
                Label13.Text = "Off"
            End If
            If CheckBox2.Checked = True Then
                BscServoOn(nCid)
                Label13.Text = "On"
            End If
        End If
    End Sub

    Public Function Run_Program(ByVal e As String) As Integer
        ' TextBox143.Text = ""
        TextBox3.Text = BscSelectJob(nCid, e)
        TextBox3.Text = BscDeleteJob(nCid)
        TextBox3.Text = ""
        TextBox3.Text = BscDownLoad(nCid, e)
        If ((e = "CLOSE.JBI") Or (e = "OPEN.JBI")) Then

            BscHoldOff(nCid)
            TextBox3.Text = BscSelLoopCycle(nCid)

            BscSetMasterJob(nCid)
            BscSelectMode(nCid, 2)
            BscServoOn(nCid)
            'If Finish_Flag = 0 Then
            BscStartJob(nCid)
            '   TextBox143.Text = ""
            'Else
            '   TextBox143.Text = BscStartJob(nCid)
            'End If
            BscHoldOn(nCid)
        Else
            If CheckBox4.Checked = False Then
                TextBox3.Text = BscSelOneCycle(nCid)
                BscSetMasterJob(nCid)
                BscSelectMode(nCid, 2)
                BscServoOn(nCid)
                'If Finish_Flag = 0 Then
                BscStartJob(nCid)
                '   TextBox143.Text = ""
                'Else
                '   TextBox143.Text = BscStartJob(nCid)
                'End If
                BscHoldOff(nCid)
            Else
                TextBox3.Text = BscSelLoopCycle(nCid)
                BscSetMasterJob(nCid)
                BscSelectMode(nCid, 2)
                BscServoOn(nCid)
                'If Finish_Flag = 0 Then
                BscStartJob(nCid)
                '   TextBox143.Text = ""
                'Else
                '   TextBox143.Text = BscStartJob(nCid)
                'End If
                BscHoldOff(nCid)
            End If
        End If
    End Function



    Private Sub Write2File(ByVal msg As String, ByVal filePath As String)
        Dim fs As FileStream = New FileStream(filePath, FileMode.Append, FileAccess.Write)
        Dim sw As StreamWriter = New StreamWriter(fs)
        sw.WriteLine(msg)
        sw.Flush()
        sw.Close()
        fs.Close()
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click

        Dim Robot_Sequence As String
        Dim Matlab_Func As String
        Dim i As Integer
        Dim NumOfMoves As Integer


        Select Case ComboBox1.SelectedIndex
            Case 0
                TextBox76.Text = "toast1(1)"
            Case 1
                TextBox76.Text = "toast2(0,0)"
            Case 2
                TextBox76.Text = "toast3(0,0)"
            Case 3
                TextBox76.Text = "toast4(0,0)"
        End Select

        Matlab_Func = TextBox76.Text ' matlab function name
        File.Delete(TextBox148.Text + "/sequence.csv") ' deleting old csv file

        MatLab = CreateObject("Matlab.Application") ' create matlab object
        Robot_Sequence = MatLab.Execute("cd " + TextBox148.Text) ' specifing path to .m file
        Robot_Sequence = MatLab.Execute(Matlab_Func) ' calling function
        Robot_Sequence = Mid(Robot_Sequence, 13) ' trimming
beginning
        Write2File(Robot_Sequence, TextBox148.Text +
"/sequence.csv") ' writing result to csv file

        NumOfMoves = Mid(Robot_Sequence, 1, 2) ' number of moves
at Robot_Sequence
        Robot_Sequence = Mid(Robot_Sequence, 9)

        ReDim Preserve Sequence(NumOfMoves)

        TextBox75.Text = Robot_Sequence.ToString ' displaying
sequence

        i = 1
        Do Until Robot_Sequence.Length < 3
            Sequence(i) = Mid(Robot_Sequence, 1, 1)
            Robot_Sequence = Mid(Robot_Sequence, 8)
            i = i + 1
        Loop

        'for interface
        Label30.Visible = True
        p.Enabled = True


    End Sub


    Private Sub TextBox77_TextChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs)

    End Sub

    Private Sub Label42_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Label42.Click

    End Sub

    Private Sub Button8_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button8.Click

        Dim i As Integer
        Dim rf As Short ' to check if robot finished
        Dim StartShift As Integer ' shifts for grasping a toast
        Dim Queue2A As Integer
        Dim Queue2B As Integer
        Dim Queue4A As Integer
        Dim Queue4B As Integer
        Dim Shift As Integer

        StartShift = 0
        Queue2A = 0
        Queue2B = 0
        Queue4A = 0
        Queue4B = 0

        Dim Positions(12) As String
        Positions(1) = "-72547,37586,36316,-1761,1583,19688"
        Positions(2) = "-60007,17796,10045,-1628,5693,16403"
        Positions(3) = "-32661,-2931,-11835,-1207,6782,9085"
        Positions(4) = "1942,-9898,-18530,-515,6924,-203"
        Positions(5) = "32203,-3025,-10859,157,6500,-8327"
        Positions(6) = "55551,12919,4204,665,6932,-14596"
        Positions(7) = "-72549,44593,61186,-1795,-9146,19497"
        Positions(8) = "-60007,21454,29236,-1621,-3638,16250"
        Positions(9) = "-32661,-208,4629,-1195,-1475,8985"
        Positions(10) = "1942,-6205,1685,-511,-3007,-254"
        Positions(11) = "32203,1659,12570,156,-4767,-8309"
        Positions(12) = "55551,16831,25180,659,-3324,-14528"


        Dim OpenClose As Integer
        OpenClose = 0 ' 0-Open, 1-Close

        Dim NumFinToasts As Integer
        NumFinToasts = 0

        ' for interface
        ' process indicator
        Label28.Text = "In Process"
        Label28.BackColor =
System.Drawing.Color.FromName("Green")


        ToasterFree = True ' init
        ButtererFree = True ' init
        ToasterFinished = False ' init
        ButtererFinished = False ' init

        TextBox6.Text = "OPEN.JBI"
        Button6_Click(sender, e)
        TextBox6.Text = "HOME1.JBI"
        Button6_Click(sender, e)
        ' Pause(10)

        TextBox5.Text = NumFinToasts.ToString


        For i = 1 To Sequence.Length - 1


            TextBox13.Text = i.ToString

            rf = BscJobWait(nCid, -1)
            If rf = 0 Then ' 0-robot finished former job. we can download
next job

                TextBox8.Text = Sequence(i).ToString



                If (Sequence(i) = 1) Or (Sequence(i) = 2) Or (Sequence(i) =
4) Then ' stations 1,2,4

                    TextBox6.Text = "HOME" + Sequence(i).ToString +
".JBI"
                    Button6_Click(sender, e)
                    OpenClose = 1 - OpenClose
                    Select Case Sequence(i)
                        Case 1
                            Shift = StartShift
                            StartShift = StartShift + 1
                        Case 2 ' A upper, B lower
                            If OpenClose = 0 And Queue2A = 0 And Queue2B
= 0 Then ' putting in empty Queue
                                Shift = 1
                                Queue2B = Queue2B + 1
                            ElseIf OpenClose = 0 And Queue2A = 0 And
Queue2B = 1 Then ' puting in queue with toast at B
                                Shift = 0
                                Queue2A = Queue2A + 1
                            ElseIf OpenClose = 1 And Queue2A = 1 Then '
taking when toasts at A and B
                                Shift = 2
                                Queue2A = Queue2A - 1
                            ElseIf OpenClose = 1 And Queue2A = 0 And
Queue2B = 1 Then ' taking when toast at B and A empty
                                Shift = 3
                                Queue2B = Queue2B - 1
                            End If


                        Case 4
                            If OpenClose = 0 And Queue4A = 0 And Queue4B
= 0 Then ' putting in empty Queue
                                Shift = 1
                                Queue4B = Queue4B + 1
                            ElseIf OpenClose = 0 And Queue4A = 0 And
Queue4B = 1 Then ' puting in queue with toast at B
                                Shift = 0
                                Queue4A = Queue4A + 1
                            ElseIf OpenClose = 1 And Queue4A = 1 Then '
taking when toasts at A and B
                                Shift = 2
                Queue4A = Queue4A - 1
             ElseIf OpenClose = 1 And Queue4A = 0 And
Queue4B = 1 Then ' taking when toast at B and A empty
                Shift = 3
                Queue4B = Queue4B - 1
             End If
          End Select

          If OpenClose = 0 Then
             grasping2(sender, e, Shift, OpenClose)
          Else
             grasping1(sender, e, Shift, OpenClose)
          End If



       ElseIf (Sequence(i) = 6) Then ' station 6

          rf = BscJobWait(nCid, -1)
          While rf <> 0 ' 0-robot finished former job. we can
download next job
             rf = BscJobWait(nCid, -1)
          End While

          OpenClose = 1 - OpenClose
          TextBox6.Text = "FNSHPLC.JBI"
          Button6_Click(sender, e)

          ' for interface
          ' Number of finished toasts
          NumFinToasts = NumFinToasts + 1
          TextBox5.Text = NumFinToasts.ToString


       ' if inserting to free station
    Else ' station 3 or 5

          If ((Sequence(i) = 3) And (ToasterFree = True)) Or
((Sequence(i) = 5) And (ButtererFree = True)) Or ((Sequence(i) = 3)
And (ToasterFinished = True)) Or ((Sequence(i) = 5) And
(ButtererFinished = True)) Then

             OpenClose = 1 - OpenClose

             If Sequence(i) = 3 Then ' toaster is free

                TextBox6.Text = "HOME3.JBI"
                Button6_Click(sender, e)

                If OpenClose = 0 Then
                   TextBox6.Text = "TSTRPLC.JBI"
                Else
                   TextBox6.Text = "TSTRPICK.JBI"

                End If

                rf = BscJobWait(nCid, -1)
                While rf <> 0 ' 0-robot finished former job. we can
download next job
                   rf = BscJobWait(nCid, -1)
                End While
                Button6_Click(sender, e)

             End If

             If Sequence(i) = 5 Then ' toaster is free


                If OpenClose = 0 Then
                   TextBox6.Text = "BTRPLC.JBI"
                Else
                   TextBox6.Text = "BTRPICK.JBI"

                End If

                Button6_Click(sender, e)

             End If
```

```
             If Sequence(i) = 3 Then ' update toaster to be busy
                If ToasterFree = True Then
                   ToasterFree = False
                   ToasterFinished = False
                   t.Enabled = True
                Else
                   ToasterFree = True
                   ToasterFinished = False
                   If Sequence(i + 1) = 3 Then
                      i = i + 1
                   End If
                End If


             End If


             If Sequence(i) = 5 Then ' update butterer to be busy
                If ButtererFree = True Then
                   ButtererFree = False
                   ButtererFinished = False
                   b.Enabled = True
                Else
                   ButtererFree = True
                   ButtererFinished = False
                   If Sequence(i + 1) = 5 Then
                      i = i + 1
                   End If
                End If


             End If

          Else ' if going to toaster or butterer and didn't finish

             ' if not already at position
             If Sequence(i) <> Sequence(i - 1) Then

                TextBox6.Text = "HOME" + Sequence(i).ToString
+ ".JBI"
                Button6_Click(sender, e)

             End If

             i = i - 1


          End If


          End If ' if of all stations


       Else ' if robot still moving...
       i = i - 1

       End If ' robot finished



       ' for interface.
       ' for toaster indicator


       If ToasterFree = True Then
          Label20.Text = "Not Toasting"
          Label20.BackColor =
System.Drawing.Color.FromName("Red")
       Else
          If ToasterFinished = True Then
             Label20.Text = "Not Toasting"
             Label20.BackColor =
System.Drawing.Color.FromName("Red")
          Else
             Label20.Text = "Toasting"
```

```
                Label20.BackColor                                    =
System.Drawing.Color.FromName("Green")
            End If
        End If

        ' for butterer indicator
        If ButtererFree = True Then
            Label21.Text = "Not Buttering"
            Label21.BackColor                                        =
System.Drawing.Color.FromName("Red")
        Else
            If ButtererFinished = True Then
                Label21.Text = "Not Buttering"
                Label21.BackColor                                    =
System.Drawing.Color.FromName("Red")
            Else
                Label21.Text = "Buttering"
                Label21.BackColor                                    =
System.Drawing.Color.FromName("Green")
            End If
        End If



    Next

    ' for interface
    ' process indicator

    Label28.Text = "Process Finished"
    Label28.BackColor = System.Drawing.Color.FromName("Red")


End Sub

Public Sub ToasterTimerFired(ByVal sender As Object, ByVal e
As System.Timers.ElapsedEventArgs)
    '  TextBox9.Text = "1"
    ToasterFinished = True
    t.Enabled = False


End Sub
Public Sub ButtererTimerFired(ByVal sender As Object, ByVal e
As System.Timers.ElapsedEventArgs)
    ButtererFinished = True
    b.Enabled = False

End Sub

Public Sub PolicyTimerFired(ByVal sender As Object, ByVal e As
System.Timers.ElapsedEventArgs)
    Label30.Visible = False
    p.Enabled = False

End Sub


Private   Sub   TextBox8_TextChanged(ByVal   sender   As
System.Object,   ByVal   e   As   System.EventArgs)   Handles
TextBox8.TextChanged

End Sub


Private Sub MenuItem1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MenuItem1.Click
    Disconnect_Robot()
    Close()
End Sub

Private Sub Button11_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button11.Click
    Dim direction As String
    '/\
    direction = "1"
    moving_manual(sender, e, direction)
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button2.Click
    Dim direction As String
    '<
    direction = "2"
    moving_manual(sender, e, direction)
End Sub

Private Sub Button12_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button12.Click
    Dim direction As String
    '\/
    direction = "3"
    moving_manual(sender, e, direction)
End Sub

Private Sub Button10_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button10.Click
    Dim direction As String
    '>
    direction = "4"
    moving_manual(sender, e, direction)
End Sub

Private Sub Button27_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button27.Click
    Dim direction As String
    'Up
    direction = "5"
    moving_manual(sender, e, direction)
End Sub

Private Sub Button28_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button28.Click
    Dim direction As String
    'Down
    direction = "6"
    moving_manual(sender, e, direction)
End Sub

Private Sub moving(ByVal sender As System.Object, ByVal e As
System.EventArgs, ByVal direction As String)
    Dim rf As Short ' to check if robot finished
    Dim point As String
    Dim Success As String

    Select Case direction
        Case "1"
            point = "P0001=000.000,20,000.000,00.00,00.00,00.00"
        Case "2"
            point = "P0001=-20,000.000,000.000,00.00,00.00,00.00"
        Case "3"
            point = "P0001=000.000,-20,000.000,00.00,00.00,00.00"
        Case "4"
            point = "P0001=20,000.000,000.000,00.00,00.00,00.00"
    End Select



    rf = BscJobWait(nCid, -1)
    If rf = 0 Then ' 0-robot finished former job. we can download
next job

        If StopRun = False Then



            File.Delete("POLICY3.JBI")
            Dim    fs    As    New    FileStream("POLICY3.JBI",
FileMode.OpenOrCreate, FileAccess.Write)
            Dim s As New StreamWriter(fs)
            s.WriteLine("/JOB")
            s.WriteLine("//NAME POLICY3")
            s.WriteLine("//POS")
            s.WriteLine("///NPOS 0,0,0,1,0,0")
            s.WriteLine("///TOOL 0")
            s.WriteLine("///POSTYPE ROBOT")
            s.WriteLine("///RECTAN")
            s.WriteLine("///RCONF 0,0,0,0,0,0,0,0")
            s.WriteLine(point)

            s.WriteLine("//INST")
            s.WriteLine("///DATE 2055/03/01 22:24")
            s.WriteLine("///ATTR SC,RW")
            s.WriteLine("///GROUP1 RB1")
            s.WriteLine("NOP")
            s.WriteLine("IMOV P001 V=1000")
            s.WriteLine("END")
            s.Close()

            TextBox6.Text = "POLICY3.JBI"
            Button6_Click(sender, e)
            '       Pause(10)

            '      End If

            rf = BscJobWait(nCid, -1)
            While rf <> 0 ' 0-robot finished former job. we can
download next job
                rf = BscJobWait(nCid, -1)
            End While


        End If

      End If

   End Sub


   Private Sub moving_manual(ByVal sender As System.Object,
ByVal e As System.EventArgs, ByVal direction As String)
      Dim rf As Short ' to check if robot finished
      Dim point As String
      Dim Success As String

      Select Case direction
         Case "1"
            point = "P0001=000.000,20,000.000,00.00,00.00,00.00"
         Case "2"
            point = "P0001=-20,000.000,000.000,00.00,00.00,00.00"
         Case "3"
            point = "P0001=000.000,-20,000.000,00.00,00.00,00.00"
         Case "4"
            point = "P0001=20,000.000,000.000,00.00,00.00,00.00"
         Case "5"
            point = "P0001=000.000,000.000,20,00.00,00.00,00.00"
         Case "6"
            point = "P0001=000.000,000.000,-20,00.00,00.00,00.00"
      End Select



      rf = BscJobWait(nCid, -1)
      If rf = 0 Then ' 0-robot finished former job. we can download
next job

         If StopRun = False Then



            File.Delete("POLICY3.JBI")
            Dim fs As New FileStream("POLICY3.JBI",
FileMode.OpenOrCreate, FileAccess.Write)
            Dim s As New StreamWriter(fs)
            s.WriteLine("/JOB")
            s.WriteLine("//NAME POLICY3")
            s.WriteLine("//POS")
            s.WriteLine("///NPOS 0,0,0,1,0,0")
            s.WriteLine("///TOOL 0")
            s.WriteLine("///POSTYPE ROBOT")
            s.WriteLine("///RECTAN")
            s.WriteLine("///RCONF 0,0,0,0,0,0,0,0")
            s.WriteLine(point)
            s.WriteLine("//INST")
            s.WriteLine("///DATE 2055/03/01 22:24")
            s.WriteLine("///ATTR SC,RW")
            s.WriteLine("///GROUP1 RB1")
            s.WriteLine("NOP")
            s.WriteLine("IMOV P001 V=1000")
            s.WriteLine("END")
            s.Close()

            TextBox6.Text = "POLICY3.JBI"
            Button6_Click(sender, e)
            '       Pause(10)

            '      End If

            rf = BscJobWait(nCid, -1)
            While rf <> 0 ' 0-robot finished former job. we can
download next job
                rf = BscJobWait(nCid, -1)
            End While

         End If

      End If

   End Sub

   Private Sub Button30_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button30.Click
      Dim rf As Short ' to check if robot finished
      Dim Success As String

      ' for interface
      TextBox7.Text = ""
      Label34.Visible = False
      StopRun = False 'for stoping learning episode

      NumOfEpisods = 1

      TextBox6.Text = "HOME8.JBI"
      Button6_Click(sender, e)

      rf = BscJobWait(nCid, -1)
      While rf <> 0 ' 0-robot finished former job. we can download
next job
         rf = BscJobWait(nCid, -1)
      End While

      MatLab = CreateObject("Matlab.Application") ' create matlab
object
      MatLab.Execute("cd " + TextBox148.Text) ' specifing path to .m
file
      MatLab.Execute("clear_variables") ' calling function

   End Sub


   Private Sub Button32_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button32.Click
      Dim rf As Short ' to check if robot finished

      ' for interface
      TextBox7.Text = ""
      Label34.Visible = False
      StopRun = False 'for stoping learning episode

      NumOfEpisods = 1

      TextBox6.Text = "BCA.JBI"
      Button6_Click(sender, e)

      rf = BscJobWait(nCid, -1)
      While rf <> 0 ' 0-robot finished former job. we can download
next job
         rf = BscJobWait(nCid, -1)
      End While

   End Sub
```

```
    Private Sub Button13_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button13.Click
        MatLab = CreateObject("Matlab.Application") ' create matlab object
        MatLab.Execute("cd " + TextBox148.Text) ' specifing path to .m file
        MatLab.Execute("environ_init") ' calling function

    End Sub


    Private Sub Button14_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button14.Click
        Dim rf As Short ' to check if robot finished
        Dim point As String
        Dim Success As String
        Dim nextMove As String
        Dim EpisodeSteps As Integer

        nextMove = 4 ' init. will be changed anyway
        'for interface
        EpisodeSteps = 0


        While Val(nextMove) <> 0 ' 0-robot finished former job. we can download next job

            If StopRun = True Then
                Exit While
            End If

            MatLab = CreateObject("Matlab.Application") ' create matlab object
            MatLab.Execute("cd " + TextBox148.Text) ' specifing path to .m file
            MatLab.Execute("identify") ' calling function

            MatLab = CreateObject("Matlab.Application") ' create matlab object
            nextMove = MatLab.Execute("cd " + TextBox148.Text) ' specifing path to .m file
            nextMove = MatLab.Execute("run") ' calling function
            nextMove = Mid(nextMove, 14, 1) ' trimming beginning and end


            Select Case nextMove
                Case "0"
                    TextBox7.Text = "Stop"
                Case "1"
                    TextBox7.Text = "/\"
                Case "2"
                    TextBox7.Text = "<"
                Case "3"
                    TextBox7.Text = "\/"
                Case "4"
                    TextBox7.Text = ">"
            End Select


            rf = BscJobWait(nCid, -1)
            While rf <> 0 ' 0-robot finished former job. we can download next job
                rf = BscJobWait(nCid, -1)
            End While
            If Val(nextMove) <> 0 Then ' val - to parse to integer
                moving(sender, e, nextMove)

            Else
                ' for interface
                Label34.Visible = True

                Dim direction As String
                direction = "4"
                moving_manual(sender, e, direction)

                rf = BscJobWait(nCid, -1)
                While rf <> 0 ' 0-robot finished former job. we can download next job
                    rf = BscJobWait(nCid, -1)
                End While

                TextBox6.Text = "FINISH.JBI"
                Button6_Click(sender, e)

                rf = BscJobWait(nCid, -1)
                While rf <> 0 ' 0-robot finished former job. we can download next job
                    rf = BscJobWait(nCid, -1)
                End While

            End If

            'for interface
            If Val(nextMove) <> 0 Then
                EpisodeSteps = EpisodeSteps + 1
            End If


        End While

    End Sub

    Private Sub Button15_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button15.Click

        MatLab = CreateObject("Matlab.Application") ' create matlab object
        MatLab.Execute("cd " + TextBox148.Text) ' specifing path to .m file
        MatLab.Execute("gw('new')") ' calling function

        MatLab = CreateObject("Matlab.Application") ' create matlab object
        MatLab.Execute("cd " + TextBox148.Text) ' specifing path to .m file
        MatLab.Execute("gw('try')") ' calling function

    End Sub

    Private Sub Button16_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button16.Click
        StopRun = True
    End Sub


    Private Sub Button17_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button17.Click

        Dim i As Integer
        Dim rf As Short ' to check if robot finished

        Dim Positions(12) As String
        Positions(1) = "-72547,37586,36316,-1761,1583,19688"
        Positions(2) = "-60007,17796,10045,-1628,5693,16403"
        Positions(3) = "-32661,-2931,-11835,-1207,6782,9085"
        Positions(4) = "1942,-9898,-18530,-515,6924,-203"
        Positions(5) = "32203,-3025,-10859,157,6500,-8327"
        Positions(6) = "55551,12919,4204,665,6932,-14596"
        Positions(7) = "-72549,44593,61186,-1795,-9146,19497"
        Positions(8) = "-60007,21454,29236,-1621,-3638,16250"
        Positions(9) = "-32661,-208,4629,-1195,-1475,8985"
        Positions(10) = "1942,-6205,1685,-511,-3007,-254"
        Positions(11) = "32203,1659,12570,156,-4767,-8309"
        Positions(12) = "55551,16831,25180,659,-3324,-14528"


        Dim OpenClose As Integer
        OpenClose = 0 ' 0-Open, 1-Close

        Dim NumFinToasts As Integer
        NumFinToasts = 0


        ' for interface
    ' process indicator
    Label28.Text = "In Process"
    Label28.BackColor =
System.Drawing.Color.FromName("Green")


    ToasterFree = True ' init
    ButtererFree = True ' init
    ToasterFinished = False ' init
    ButtererFinished = False ' init


    TextBox6.Text = "HOME.JBI"
    Button6_Click(sender, e)
    ' Pause(10)

    TextBox5.Text = NumFinToasts.ToString



    For i = 1 To Sequence.Length - 1



      ' for interface
      ' Number of finished toasts
      If Sequence(i) = 6 Then
        NumFinToasts = NumFinToasts + 1
        TextBox5.Text = NumFinToasts.ToString
      End If


      TextBox13.Text = i.ToString

      rf = BscJobWait(nCid, -1)
      If rf = 0 Then ' 0-robot finished former job. we can download
next job
        TextBox8.Text = Sequence(i).ToString

        ' if inserting to free station
        If ((Sequence(i) <> 3) And (Sequence(i) <> 5)) Or
((Sequence(i) = 3) And (ToasterFree = True)) Or ((Sequence(i) = 5)
And (ButtererFree = True)) Or ((Sequence(i) = 3) And
(ToasterFinished = True)) Or ((Sequence(i) = 5) And
(ButtererFinished = True)) Then

          File.Delete("POLICY1.JBI")
          Dim fs As New FileStream("POLICY1.JBI",
FileMode.OpenOrCreate, FileAccess.Write)
          Dim s As New StreamWriter(fs)
          s.WriteLine("/JOB")
          s.WriteLine("//NAME POLICY1")
          s.WriteLine("//POS")
          s.WriteLine("///NPOS 3,0,0,0,0,0")
          s.WriteLine("///TOOL 0")
          s.WriteLine("///POSTYPE PULSE")
          s.WriteLine("///PULSE")
          s.WriteLine("C00000=" + Positions(Sequence(i) +
6).ToString)
          s.WriteLine("C00001=" +
Positions(Sequence(i)).ToString)
          s.WriteLine("C00002=" + Positions(Sequence(i) +
6).ToString)
          s.WriteLine("//INST")
          s.WriteLine("///DATE 2055/03/01 22:24")
          s.WriteLine("///ATTR SC,RW")
          s.WriteLine("///GROUP1 RB1")
          s.WriteLine("NOP")
          s.WriteLine("MOVJ C00000 VJ=10.00")
          s.WriteLine("MOVJ C00001 VJ=10.00")

          If OpenClose = 0 Then
            s.WriteLine("CALL JOB:CLOSE")
          Else
            s.WriteLine("CALL JOB:OPEN")
          End If
          OpenClose = 1 - OpenClose
          s.WriteLine("TIMER T=1.00")
          s.WriteLine("MOVJ C00002 VJ=10.00")
          s.WriteLine("END")
          s.Close()

          TextBox6.Text = "POLICY1.JBI"
          Button6_Click(sender, e)
          '       Pause(10)


          If Sequence(i) = 3 Then ' update toaster to be busy
            If ToasterFree = True Then
              ToasterFree = False
              ToasterFinished = False
              t.Enabled = True
            Else
              ToasterFree = True
              ToasterFinished = False
              If Sequence(i + 1) = 3 Then
                i = i + 1
              End If
            End If


          End If


          If Sequence(i) = 5 Then ' update butterer to be busy
            If ButtererFree = True Then
              ButtererFree = False
              ButtererFinished = False
              b.Enabled = True
            Else
              ButtererFree = True
              ButtererFinished = False
              If Sequence(i + 1) = 5 Then
                i = i + 1
              End If
            End If


          End If



        Else
          ' if going to toaster and didn't finish

          ' if not already at position
          If Sequence(i) <> Sequence(i - 1) Then

            File.Delete("POLICY2.JBI")
            Dim fs1 As New FileStream("POLICY2.JBI",
FileMode.OpenOrCreate, FileAccess.Write)
            Dim s1 As New StreamWriter(fs1)

            s1.WriteLine("/JOB")
            s1.WriteLine("//NAME POLICY2")
            s1.WriteLine("//POS")
            s1.WriteLine("///NPOS 1,0,0,0,0,0")
            s1.WriteLine("///TOOL 0")
            s1.WriteLine("///POSTYPE PULSE")
            s1.WriteLine("///PULSE")
            s1.WriteLine("C00000=" + Positions(Sequence(i) +
6).ToString)
            s1.WriteLine("//INST")
            s1.WriteLine("///DATE 2055/03/01 22:24")
            s1.WriteLine("///ATTR SC,RW")
            s1.WriteLine("///GROUP1 RB1")
            s1.WriteLine("NOP")
            s1.WriteLine("MOVJ C00000 VJ=10.00")
            s1.WriteLine("END")
            s1.Close()

            TextBox6.Text = "POLICY2.JBI"
            Button6_Click(sender, e)
            '       Pause(8)
            End If
            i = i - 1


            '          If Sequence(i) = 3 Then
            '         Do Until ToasterFinished = True
            '        TextBox9.Text = "2"
            '       Loop
            '      Else
            '     Do Until ButtererFinished = True
            '    Loop
            '   End If

            '      If ((Sequence(i) = 3) And (ToasterFinished = True))
Or ((Sequence(i) = 5) And (ButtererFinished = True)) Then



            '     File.Delete("POLICY3.JBI")
            '          Dim fs2 As New FileStream("POLICY3.JBI",
FileMode.OpenOrCreate, FileAccess.Write)
            '   Dim s2 As New StreamWriter(fs2)

            '   s2.WriteLine("/JOB")
            '  s2.WriteLine("//NAME POLICY3")
            ' s2.WriteLine("//POS")
            's2.WriteLine("///NPOS 2,0,0,0,0,0")
            's2.WriteLine("///TOOL 0")
            's2.WriteLine("///POSTYPE PULSE")
            's2.WriteLine("///PULSE")
            's2.WriteLine("C00000=" +
Positions(Sequence(i)).ToString)
            's2.WriteLine("C00001=" + Positions(Sequence(i) +
6).ToString)
            's2.WriteLine("//INST")
            's2.WriteLine("///DATE 2055/03/01 22:24")
            's2.WriteLine("///ATTR SC,RW")
            's2.WriteLine("///GROUP1 RB1")
            's2.WriteLine("NOP")
            's2.WriteLine("MOVJ C00000 VJ=10.00")

            'If OpenClose = 0 Then
            's2.WriteLine("CALL JOB:CLOSE")
            'Else
            '   s2.WriteLine("CALL JOB:OPEN")
            'End If
            'OpenClose = 1 - OpenClose
            's2.WriteLine("TIMER T=1.00")
            's2.WriteLine("MOVJ C00001 VJ=10.00")
            's2.WriteLine("END")
            's2.Close()

            ' If Sequence(i) = 3 Then
            'ToasterFree = True ' freeing toaster
            'ToasterFinished = False
            'Else
            '   ButtererFree = True ' freeing butterer
            '   ButtererFinished = False
            ' End If


            ' TextBox6.Text = "POLICY3.JBI"
            ' Button6_Click(sender, e)
            '      Pause(7)
            ' Else
            ' i = i - 1
            ' End If

          End If

        Else ' if robot still moving...
          i = i - 1
        End If ' robot finished



        ' for interface.
        ' for toaster indicator
            If ToasterFree = True Then
              Label20.Text = "Not Toasting"
              Label20.BackColor =
System.Drawing.Color.FromName("Red")
            Else
              If ToasterFinished = True Then
                Label20.Text = "Not Toasting"
                Label20.BackColor =
System.Drawing.Color.FromName("Red")
              Else
                Label20.Text = "Toasting"
                Label20.BackColor =
System.Drawing.Color.FromName("Green")
              End If
            End If

            ' for butterer indicator
            If ButtererFree = True Then
              Label21.Text = "Not Buttering"
              Label21.BackColor =
System.Drawing.Color.FromName("Red")
            Else
              If ButtererFinished = True Then
                Label21.Text = "Not Buttering"
                Label21.BackColor =
System.Drawing.Color.FromName("Red")
              Else
                Label21.Text = "Buttering"
                Label21.BackColor =
System.Drawing.Color.FromName("Green")
              End If
            End If



          Next


          ' for interface
          ' process indicator

          Label28.Text = "Process Finished"
          Label28.BackColor = System.Drawing.Color.FromName("Red")


        End Sub

        Private Sub Button18_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button18.Click
          Dim Shift As Integer
          Shift = Val(TextBox15.Text)
          grasping1(sender, e, Shift, 1)
        End Sub

        Private Sub Button19_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button19.Click
          TextBox6.Text = "HOME1.JBI"
          Button6_Click(sender, e)
        End Sub

        Private Sub Button21_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button21.Click
          TextBox6.Text = "HOME2.JBI"
          Button6_Click(sender, e)
        End Sub

        Private Sub Button20_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button20.Click
          TextBox6.Text = "HOME4.JBI"
          Button6_Click(sender, e)
        End Sub

        Private Sub Button22_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button22.Click
          TextBox6.Text = "OPEN.JBI"
          Button6_Click(sender, e)
```

```
    End Sub

    Private Sub Button23_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button23.Click
        TextBox6.Text = "CLOSE.JBI"
        Button6_Click(sender, e)
    End Sub

    Private Sub grasping1(ByVal sender As System.Object, ByVal e
As System.EventArgs, ByVal Shift As Integer, ByVal OpenClose As
Integer)
        Dim StartToasts As Integer ' number of toasts in start station
        Dim rf As Short ' to check if robot finished
        Dim OC As String
        Dim ShiftUp As Integer

        '       StartToasts = Val(TextBox15.Text)

        rf = BscJobWait(nCid, -1)
        If rf = 0 Then ' 0-robot finished former job. we can download
next job

            If OpenClose = 0 Then
                OC = "OPEN"
            Else
                OC = "CLOSE"
            End If

            Shift = -240 - (Shift * 40)
            ShiftUp = -40 - Shift

            File.Delete("GRASP1.JBI")
            Dim fs As New FileStream("GRASP1.JBI",
FileMode.OpenOrCreate, FileAccess.Write)
            Dim s As New StreamWriter(fs)
            s.WriteLine("/JOB")
            s.WriteLine("//NAME GRASP1")
            s.WriteLine("//POS")
            s.WriteLine("///NPOS 0,0,0,5,0,0")
            s.WriteLine("///TOOL 0")
            s.WriteLine("///POSTYPE ROBOT")
            s.WriteLine("///RECTAN")
            s.WriteLine("///RCONF 0,0,0,0,0,0,0,0")
            s.WriteLine("P0001=00.00,00.00," + Shift.ToString +
",000.000,000.000,00.00")

s.WriteLine("P0002=00.00,145,000.000,000.000,00.00,00.00")
            s.WriteLine("P0003=00.00,00.00,40,000.000,000.000,00.00")
            s.WriteLine("P0004=00.00,-
145,000.000,000.000,00.00,00.00")
            s.WriteLine("P0005=00.00,00.00," + ShiftUp.ToString +
",000.000,000.000,00.00")
            s.WriteLine("//INST")
            s.WriteLine("///DATE 2055/03/01 22:24")
            s.WriteLine("///ATTR SC,RW")
            s.WriteLine("///GROUP1 RB1")
            s.WriteLine("NOP")
            s.WriteLine("IMOV P001 V=100")
            s.WriteLine("IMOV P002 V=100")
            s.WriteLine("IMOV P003 V=20")
            s.WriteLine("TIMER T=1.00")
            s.WriteLine("CALL JOB:" + OC)
            s.WriteLine("TIMER T=1.00")
            s.WriteLine("IMOV P004 V=100")
            s.WriteLine("IMOV P005 V=100")
            s.WriteLine("END")
            s.Close()

            TextBox6.Text = "GRASP1.JBI"
            Button6_Click(sender, e)

            '   StartToasts = StartToasts - 1
        Else
            grasping1(sender, e, Shift, OpenClose)
        End If


    End Sub

    Private Sub grasping2(ByVal sender As System.Object, ByVal e
As System.EventArgs, ByVal Shift As Integer, ByVal OpenClose As
Integer)
        Dim StartToasts As Integer ' number of toasts in start station
        Dim rf As Short ' to check if robot finished
        Dim OC As String
        Dim ShiftUp As Integer

        '       StartToasts = Val(TextBox15.Text)

        rf = BscJobWait(nCid, -1)
        If rf = 0 Then ' 0-robot finished former job. we can download
next job

            If OpenClose = 0 Then
                OC = "OPEN"
            Else
                OC = "CLOSE"
            End If

            Shift = -240 - (Shift * 40)
            ShiftUp = 80 - Shift

            File.Delete("GRASP2.JBI")
            Dim fs As New FileStream("GRASP2.JBI",
FileMode.OpenOrCreate, FileAccess.Write)
            Dim s As New StreamWriter(fs)
            s.WriteLine("/JOB")
            s.WriteLine("//NAME GRASP2")
            s.WriteLine("//POS")
            s.WriteLine("///NPOS 0,0,0,7,0,0")
            s.WriteLine("///TOOL 0")
            s.WriteLine("///POSTYPE ROBOT")
            s.WriteLine("///RECTAN")
            s.WriteLine("///RCONF 0,0,0,0,0,0,0,0")
            s.WriteLine("P0001=00.00,00.00," + Shift.ToString +
",000.000,000.000,00.00")
            s.WriteLine("P0002=00.00,85,000.000,000.000,00.00,00.00")
            s.WriteLine("P0003=00.00,00.00,-40,000.000,000.000,00.00")
            s.WriteLine("P0004=00.00,60,000.000,000.000,00.00,00.00")
            s.WriteLine("P0005=00.00,00.00,-40,000.000,000.000,00.00")
            s.WriteLine("P0006=00.00,-
145,000.000,000.000,00.00,00.00")
            s.WriteLine("P0007=00.00,00.00," + ShiftUp.ToString +
",000.000,000.000,00.00")
            s.WriteLine("//INST")
            s.WriteLine("///DATE 2055/03/01 22:24")
            s.WriteLine("///ATTR SC,RW")
            s.WriteLine("///GROUP1 RB1")
            s.WriteLine("NOP")
            s.WriteLine("IMOV P001 V=100")
            s.WriteLine("IMOV P002 V=100")
            s.WriteLine("TIMER T=1.00")
            s.WriteLine("CALL JOB:" + OC)
            s.WriteLine("TIMER T=1.00")
            s.WriteLine("IMOV P003 V=20")
            s.WriteLine("IMOV P004 V=20")
            s.WriteLine("IMOV P005 V=20")
            s.WriteLine("IMOV P006 V=100")
            s.WriteLine("IMOV P007 V=100")
            s.WriteLine("END")
            s.Close()

            TextBox6.Text = "GRASP2.JBI"
            Button6_Click(sender, e)

            '   StartToasts = StartToasts - 1
        Else
            grasping2(sender, e, Shift, OpenClose)
        End If

    End Sub

    Private Sub Button24_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button24.Click
        'Dim Shift As Integer
        'Shift = Val(TextBox15.Text)
        'grasping2(sender, e, Shift, 0)


    End Sub

    Private Sub Button25_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button25.Click
        Dim rc As Long
        rc =
PlaySound(System.AppDomain.CurrentDomain.BaseDirectory &
"shaking_a_bag.wav", 0, SND_NOSTOP)
    End Sub

    Private Sub Button26_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button26.Click
        TextBox6.Text = "OPEN.JBI"
        Button6_Click(sender, e)
    End Sub

    Private Sub Button29_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button29.Click
        TextBox6.Text = "CLOSE.JBI"
        Button6_Click(sender, e)
    End Sub
End Class
```

MATLAB Code

SRL.m

```
function SRL(arg,arg2,arg3)

% Variable declarations
global Q
global tnow_array
global iteration
global min_tnow;
global convergence_iteration

% ------------------------------------------------------------------------

switch( arg)

% ------------------------------------------------------------------------

case 0

Q = ones(6,6,6);
min_tnow = 10000;
iterations = arg3;
for iteration=1:iterations
   SRL(1,0);
end

% ------------------------------------------------------------------------

case 1

% Variable declarations
global tnow
global toaster % 0-free, 1-busy
global butterer % 0-free, 1-busy
global robot % 0-free, 1-busy
global toasts_free % 0-free, 1-busy
global toast
global Gamma
global Alpha
global event_stack
global state %robot state
global next_state
global desired_toast
global impossible_pos

oper_times= zeros(6);
oper_times(1,2) = 24;
oper_times(1,3) = 36;
oper_times(2,1) = 21;
oper_times(2,3) = 24;
oper_times(2,4) = 26;
oper_times(2,5) = 15;
oper_times(3,1) = 22;
oper_times(3,4) = 22;
oper_times(3,5) = 28;
oper_times(4,1) = 24;
oper_times(4,2) = 26;
oper_times(4,3) = 19;
oper_times(4,5) = 19;
oper_times(5,1) = 26;
oper_times(5,2) = 24;
oper_times(5,3) = 20;
oper_times(5,6) = 12;
oper_times(6,1) = 25;
oper_times(6,2) = 23;
oper_times(6,3) = 19;
oper_times(6,4) = 21;
toast_pos = ones(1,3); % toasts
robot_pos=0;
toating_time = 60;
buttering_time = 60;
Rew = zeros(6,6,6);
Rew(6,6,6) = 1.5;
Gamma = 0.9;
Alpha = 0.05;
% event_stack[event,state,time, toast(0=robot)]
%events:
% 1: robot move empty
% 2: robot move full
% 3: finished toasting
% 4: finished buttering
tnow = 0;
tmax = 20000;
impossible_pos =
[112;114;121;122;124;133;134;141;142;143;144;155;211;212;214;22
1;222;224;233;234;241;242;243;244;255;313;314;323;324;331;332;3
33;334;335;336;341;342;343;344;353;355;363;411;412;413;414;421;
422;423;424;431;432;433;434;441;442;443;444;454;455;511;515;525
;533;535;545;551;552;553;554;555;556;565;633;655];
[impossible_pos,ind] = sort(impossible_pos);
toaster =0;
butterer = 0;
toasts_free = [0,0,0];
toast_taken = true;
robot=0;
state=2; % starting point
seq = toast_pos;
toast=1;
next_state = toast_pos(1);
next_event = [1,next_state, tnow+oper_times(state,next_state),toast];
event_stack = [next_event];
toast_taken=false;
robot=1;

% --------------------------------------------------------------------------------
------------------------------

while tnow<tmax
cur_event = event_stack(1,:);
event = cur_event(1);
tnow = cur_event(3);
if event<3
   toast = cur_event(4);
   state = cur_event(2);
   robot =0;% robot is free after reaching state, empty or full
end
```

```
if event==2
   % updating toast position after arrival
    toast_taken=true;
   toast_pos(toast) = state;
   % updating position sequence
   seq = [toast_pos;seq];
   % updating Q of previous step
   delta = Rew(seq(2,1),seq(2,2),seq(2,3)) + Gamma * Q
(toast_pos(1), toast_pos(2),toast_pos(3)) - Q
(seq(2,1),seq(2,2),seq(2,3));
   Q (seq(2,1),seq(2,2),seq(2,3)) = Q (seq(2,1),seq(2,2),seq(2,3)) +
Alpha * delta;
   % finished if all toasts are finished...
   test = find (toast_pos==6);
   if size(test)==size(toast_pos)
      Q (toast_pos(1), toast_pos(2),toast_pos(3)) = Rew (toast_pos(1),
toast_pos(2),toast_pos(3)) + Gamma * Q (toast_pos(1),
toast_pos(2),toast_pos(3));
      break;
  end
   %cheking optional next steps
   optional_next_pos = [0,0,0];
   for i = toast_pos(1)+1:toast_pos(1)+2
      if (i<7)
         temp_pos = [i,toast_pos(2),toast_pos(3)];
         num_temp_pos =
100*temp_pos(1)+10*temp_pos(2)+temp_pos(3);
         if bsearch(impossible_pos,num_temp_pos) == -1
            good=true;
         else
            good=false;
         end
         if good
            if optional_next_pos(1,1)==0
               optional_next_pos  = temp_pos;
            else
                  optional_next_pos = [optional_next_pos;temp_pos];
            end
         end
      end
   end
   for i = toast_pos(2)+1:toast_pos(2)+2
      if (i<7)
         temp_pos = [toast_pos(1),i,toast_pos(3)];
          num_temp_pos =
100*temp_pos(1)+10*temp_pos(2)+temp_pos(3);
         if bsearch(impossible_pos,num_temp_pos) == -1
            good=true;
         else
            good=false;
         end
         if good
            if optional_next_pos(1,1)==0
               optional_next_pos  = temp_pos;
            else
                  optional_next_pos = [optional_next_pos;temp_pos];
            end
         end
      end
   end
   for i = toast_pos(3)+1:toast_pos(3)+2
      if (i<7)
         temp_pos = [toast_pos(1),toast_pos(2),i];
          num_temp_pos =
100*temp_pos(1)+10*temp_pos(2)+temp_pos(3);
         if bsearch(impossible_pos,num_temp_pos) == -1
            good=true;
         else
            good=false;
         end
         if good
            if optional_next_pos(1,1)==0
               optional_next_pos  = temp_pos;
            else
                  optional_next_pos = [optional_next_pos;temp_pos];
            end
         end
      end
    end
    %finding  best next step
    max = -10000;
   for i=1:size(optional_next_pos,1)
      if
Q(optional_next_pos(i,1),optional_next_pos(i,2),optional_next_pos(i,
3))>=max
         max =
Q(optional_next_pos(i,1),optional_next_pos(i,2),optional_next_pos(i,
3));
           ind = i;
      end
   end
   best_next_pos = optional_next_pos(ind,:);
   %Choosing next step
    epsilon = 1  / iteration;
    % epsilon = 1;
    if rand>epsilon
       next_pos = best_next_pos;
    else
       pos_ind = ceil(rand * size(optional_next_pos,1));
       next_pos = optional_next_pos(pos_ind,:);
    end
    delta = next_pos - toast_pos;
    next_toast = find(delta ~=0);
    % freeing toaster and butterer after toast moved from them
    if (state == 4) || (state == 5)
       toaster = 0;
    elseif state == 6
       butterer=0;
    end
end % if event ==2
if event ==3
   toasts_free(cur_event(4))=0;
end
if event == 4
   toasts_free(cur_event(4))=0;
end
if (robot==0) && (toast_taken)
   toast=next_toast;
   next_state = toast_pos(toast);
   next_event = [1,next_state,
tnow+oper_times(state,next_state),toast];
   event_stack = [event_stack;next_event];
   toast_taken=false;
   robot=1;
end
% checking toast next station
if (robot==0) && (toasts_free(toast)==0)
    if ((state~=2) && (state~=4) && (state ~=6))
       next_state = toast_pos(toast)+1;
    else
       next_state = state;
    end
    % toaster is free - no need to go to queue
    if (next_state == 2) && (toaster==0)
       next_state = 3;
       toaster = 1;
       toasts_free(toast)=1;
       next_event = [2,next_state,
tnow+oper_times(toast_pos(toast),next_state),toast];
       event_stack = [event_stack;next_event];
       % next_event =
[3,0,tnow+oper_times(toast_pos(toast),next_state)+toating_time,toast
];
       next_event =
[3,0,tnow+oper_times(toast_pos(toast),next_state)+normrnd(toating_t
ime,toating_time/10),toast];
       event_stack = [event_stack;next_event];
       robot=1;
       % butterer is free - no need to go to queue
    elseif (next_state == 4) && (butterer==0)
       next_state = 5;
       butterer=1;
       toasts_free(toast)=1;
       next_event = [2,next_state,
tnow+oper_times(toast_pos(toast),next_state),toast];
       event_stack = [event_stack;next_event];
```

```
        % next_event =
[4,0,tnow+oper_times(toast_pos(toast),next_state)+buttering_time,toa
st];
        next_event =
[4,0,tnow+oper_times(toast_pos(toast),next_state)+normrnd(buttering
_time,buttering_time/10),toast];
        event_stack = [event_stack;next_event];
        robot=1;
    else
        next_event = [2,next_state,
tnow+oper_times(toast_pos(toast),next_state),toast];
        event_stack = [event_stack;next_event];
        robot=1;
    end
end
[sorted_stack, ind] = sort(event_stack(:,3));
for i = 1:size(ind)
if i==1
    temp = event_stack(ind(i),:);
else
    temp = [temp;event_stack(ind(i),:)];
end
end
event_stack=temp(2:1:end,:);
end %while
%updating min_tnow
if tnow<min_tnow
    min_tnow = tnow;
end
%factor = 1/(tnow-min_tnow+5) + 0.8;
factor  = 1/tnow;
for i=1:size(seq,1)-1
    Q(seq(i,1),seq(i,2),seq(i,3)) = Q(seq(i,1),seq(i,2),seq(i,3))*factor;
end
if iteration == 1
    tnow_array = tnow;
else
% check when convergence is achieved
if tnow_array(size(tnow_array)) ~=tnow
    convergence_iteration = iteration;
end
% update tnow_array
tnow_array = [tnow_array;tnow];
end

end%case
```

MC.m

```
function MC(arg,arg2,arg3)

% Variable declarations
global Q
global tnow_array
global iteration
global min_tnow;
global convergence_iteration
global avgR
global visits

% ------------------------------------------------------------------------

switch( arg)

% ------------------------------------------------------------------------

case 0

Q = ones(6,6,6);
min_tnow = 10000;
iterations = arg3;
for iteration=1:iterations
    MC(1,0);
 end

% ------------------------------------------------------------------------

case 1

% Variable declaration
global tnow
global toaster % 0-free, 1-busy
global butterer % 0-free, 1-busy
global robot % 0-free, 1-busy
global toasts_free % 0-free, 1-busy
global toast
global Gamma
global Alpha
global event_stack
global state %robot state
global next_state
global desired_toast
global impossible_pos
oper_times= zeros(6);
oper_times(1,2) = 24;
oper_times(1,3) = 36;
oper_times(2,1) = 21;
oper_times(2,3) = 24;
oper_times(2,4) = 26;
oper_times(2,5) = 15;
oper_times(3,1) = 22;
oper_times(3,4) = 22;
oper_times(3,5) = 28;
oper_times(4,1) = 24;
oper_times(4,2) = 26;
oper_times(4,3) = 19;
oper_times(4,5) = 19;
oper_times(5,1) = 26;
oper_times(5,2) = 24;
oper_times(5,3) = 20;
oper_times(5,6) = 12;
oper_times(6,1) = 25;
oper_times(6,2) = 23;
oper_times(6,3) = 19;
oper_times(6,4) = 21;
toast_pos = ones(1,3); % toasts
robot_pos=0;
toating_time = 60;
buttering_time = 60;
Rew = zeros(6,6,6);
Rew(6,6,6) = 1.5;
Gamma = 0.9;
Alpha = 0.05;
% For monte carlo
avgR = zeros(6,6,6);
visits = zeros(6,6,6);
% event_stack[event,state,time, toast(0=robot)]
%events:
% 1: robot move empty
% 2: robot move full
% 3: finished toasting
% 4: finished buttering
tnow = 0;
tmax = 20000;
impossible_pos =
[112;114;121;122;124;133;134;141;142;143;144;155;211;212;214;22
1;222;224;233;234;241;242;243;244;255;313;314;323;324;331;332;3
33;334;335;336;341;342;343;344;353;355;363;411;412;413;414;421;
422;423;424;431;432;433;434;441;442;443;444;454;455;511;515;525
;533;535;545;551;552;553;554;555;556;565;633;655];
[impossible_pos,ind] = sort(impossible_pos);
toaster =0;
butterer = 0;
toasts_free = [0,0,0];
toast_taken = true;
robot=0;
state=2;
seq = toast_pos;
toast=1;
next_state = toast_pos(1);
next_event = [1,next_state, tnow+oper_times(state,next_state),toast];
event_stack = [next_event];
toast_taken=false;
robot=1;
```

```
% ---------------------------------------------------------------------------------
------------------------------

while tnow<tmax
cur_event = event_stack(1,:);
event = cur_event(1);
tnow = cur_event(3);
if event<3
   toast = cur_event(4);
   state = cur_event(2);
   robot =0;% robot is free after reaching state, empty or full
end
if event==2
   % updating toast position after arrival
    toast_taken=true;
   toast_pos(toast) = state;
   % updating position sequence
   seq = [toast_pos;seq];
  % finished if all toasts are finished...
   test = find (toast_pos==6);
   if size(test)==size(toast_pos)
     break;
   end
   %cheking optional next steps
   optional_next_pos = [0,0,0];
   for i = toast_pos(1)+1:toast_pos(1)+2
      if (i<7)
         temp_pos = [i,toast_pos(2),toast_pos(3)];
         num_temp_pos =
100*temp_pos(1)+10*temp_pos(2)+temp_pos(3);
         if bsearch(impossible_pos,num_temp_pos) == -1
            good=true;
         else
            good=false;
         end
         if good
            if optional_next_pos(1,1)==0
               optional_next_pos  = temp_pos;
            else
                  optional_next_pos = [optional_next_pos;temp_pos];
            end
         end
      end
   end
   for i = toast_pos(2)+1:toast_pos(2)+2
      if (i<7)
         temp_pos = [toast_pos(1),i,toast_pos(3)];
          num_temp_pos =
100*temp_pos(1)+10*temp_pos(2)+temp_pos(3);
         if bsearch(impossible_pos,num_temp_pos) == -1
            good=true;
         else
            good=false;
         end
         if good
            if optional_next_pos(1,1)==0
               optional_next_pos  = temp_pos;
            else
                  optional_next_pos = [optional_next_pos;temp_pos];
            end
         end
      end
   end
   for i = toast_pos(3)+1:toast_pos(3)+2
      if (i<7)
         temp_pos = [toast_pos(1),toast_pos(2),i];
          num_temp_pos =
100*temp_pos(1)+10*temp_pos(2)+temp_pos(3);
         if bsearch(impossible_pos,num_temp_pos) == -1
            good=true;
         else
            good=false;
         end
         if good
            if optional_next_pos(1,1)==0
               optional_next_pos  = temp_pos;
            else
                  optional_next_pos = [optional_next_pos;temp_pos];
            end
         end
      end
   end
    %finding  best next step
   max = -10000;
  for i=1:size(optional_next_pos,1)
     if
Q(optional_next_pos(i,1),optional_next_pos(i,2),optional_next_pos(i,
3))>=max
        max =
Q(optional_next_pos(i,1),optional_next_pos(i,2),optional_next_pos(i,
3));
        ind = i;
     end
  end
  best_next_pos = optional_next_pos(ind,:);
  %Choosing next step
  epsilon = 1  / iteration;
  if rand>epsilon
     next_pos = best_next_pos;
   else
     pos_ind = ceil(rand * size(optional_next_pos,1));
     next_pos = optional_next_pos(pos_ind,:);
   end
   delta = next_pos - toast_pos;
   next_toast = find(delta ~=0);
   % freeing toaster and butterer after toast moved from them
   if (state == 4) || (state == 5)
      toaster = 0;
   elseif state == 6
      butterer=0;
   end
 end % if event ==2
if event ==3
   toasts_free(cur_event(4))=0;
end
if event == 4
  toasts_free(cur_event(4))=0;
end
if (robot==0) && (toast_taken)
  toast=next_toast;
  next_state = toast_pos(toast);
  next_event = [1,next_state,
tnow+oper_times(state,next_state),toast];
  event_stack = [event_stack;next_event];
  toast_taken=false;
  robot=1;
end
% checking toast next station
if (robot==0) && (toasts_free(toast)==0)
   if ((state~=2) && (state~=4) && (state ~=6))
      next_state = toast_pos(toast)+1;
   else
      next_state = state;
   end
   % toaster is free - no need to go to queue
   if (next_state == 2) && (toaster==0)
      next_state = 3;
      toaster = 1;
      toasts_free(toast)=1;
      next_event = [2,next_state,
tnow+oper_times(toast_pos(toast),next_state),toast];
      event_stack = [event_stack;next_event];
      % next_event =
[3,0,tnow+oper_times(toast_pos(toast),next_state)+toating_time,toast
];
      next_event =
[3,0,tnow+oper_times(toast_pos(toast),next_state)+normrnd(toating_t
ime,toating_time/10),toast];
      event_stack = [event_stack;next_event];
      robot=1;
      % butterer is free - no need to go to queue
   elseif (next_state == 4) && (butterer==0)
      next_state = 5;
      butterer=1;
      toasts_free(toast)=1;
```

```
        next_event = [2,next_state,
tnow+oper_times(toast_pos(toast),next_state),toast];
        event_stack = [event_stack;next_event];
        % next_event =
[4,0,tnow+oper_times(toast_pos(toast),next_state)+buttering_time,toa
st];
        next_event =
[4,0,tnow+oper_times(toast_pos(toast),next_state)+normrnd(buttering
_time,buttering_time/10),toast];
        event_stack = [event_stack;next_event];
        robot=1;
    else
        next_event = [2,next_state,
tnow+oper_times(toast_pos(toast),next_state),toast];
        event_stack = [event_stack;next_event];
        robot=1;
end
end
[sorted_stack, ind] = sort(event_stack(:,3));
for i = 1:size(ind)
if i==1
    temp = event_stack(ind(i),:);
else
    temp = [temp;event_stack(ind(i),:)];
end
end
event_stack=temp(2:1:end,:);
end %while
%updating min_tnow
if tnow<min_tnow
    min_tnow = tnow;
end
factor  = 1/tnow;
for i=1:size(seq,1)-1
   avgR(seq(i,1),seq(i,2),seq(i,3)) =  (     visits(seq(i,1),seq(i,2),seq(i,3))
* avgR(seq(i,1),seq(i,2),seq(i,3)) + factor  )  /
(visits(seq(i,1),seq(i,2),seq(i,3)) + 1);
   visits(seq(i,1),seq(i,2),seq(i,3)) = visits(seq(i,1),seq(i,2),seq(i,3))+1;
   Q(seq(i,1),seq(i,2),seq(i,3)) = avgR(seq(i,1),seq(i,2),seq(i,3));
end
if iteration == 1
    tnow_array = tnow;
else
% check when convergence is achieved
if tnow_array(size(tnow_array)) ~=tnow
    convergence_iteration = iteration;
end
% update tnow_array
    tnow_array = [tnow_array;tnow];
end

end%case
```

Appendix VII. 3D Path Planning Task – Source Code

MATLAB Code

CCRL.m

```
function CCRL(arg)

%------------------------------------------------------------------------

%Variable Declarations
global block
global goal
global start
global nx ny nz ns na
global robot
global R
global Q
global alpha tau gamma delta
global human_tau
global move loss
global num_of_steps
global episode_number
global reached_goal
global enable_graphics
global enable_result_graphics
global enable_human_collaboration
global human_collaboration
global episodes
global human_next_state
global HumanQ
global convergence_episode
global collaboration_requests
global enable_rejection
global human_misleads
global reject_human_assistance
global check_conv
global stop_requests
global counter
global max_steps
global helping_steps

%----------------------------------------------------------------------

switch( arg)

%----------------------------------------------------------------------

% variable Assignment
case 'init'

% obstacles
block = [ 10,5,1; 10,6,1; 10,7,1;
                10,5,2; 10,6,2; 10,7,2;
                10,5,3; 10,6,3; 10,7,3;
                8,5,1; 8,6,1; 8,7,1;
                8,5,2; 8,6,2; 8,7,2;
                8,5,3; 8,6,3; 8,7,3;
                9,5,1; 9,5,2; 9,5,3;
                9,7,1; 9,7,2; 9,7,3;
                1,10,10; 2,10,10;3,10,10; 4,10,10; 5,10,10; 6,10,10;
7,10,10; 8,10,10; 9, 10, 10; 10,10,10;
                1,10,9; 2,10,9;3,10,9; 4,10,9; 5,10,9; 6,10,9; 7,10,9;
8,10,9; 9, 10, 9; 10,10,9;
                1,1,1; 1,2,1; 2,1,1; 2,2,1;
                10,1,10;
                3,10,2; 4,10,2; 5,10,2; 6,10,2;
                5,6,1; 5,7,1;
                5,6,2; 5,7,2;
                5,6,3; 5,7,3;
                ];
%goal point
goal = [9,6,4];
% starting point
start = [1,8,2];
% world size
nx = 10;    ny = 10;    nz = 10;    ns = nx*ny*nz ;
% actions
na = 6;     % Right,Left,Up,Down, Forward, Backward
move = [0,-1,0; 0,1,0; 0,0,1; 0,0,-1; 1,0,0; -1,0,0];
loss = -0.1*[ 1; 1; 1; 1; 1; 1];
% reward field
R = zeros( nx, ny, nz);
for i = 1:size(block,1)
   R(block(i,1),block(i,2),block(i,3)) = -1;
end
R(goal(1),goal(2),goal(3)) = 1.5;
% value field
Q = zeros( nx, ny, nz);
% learning parameters
alpha = 0.95; gamma = 0.99;
% maximum steps for learning episode
max_steps = 200;
% Number of times agent is asking for human collaboration
collaboration_requests = 0;
% for stoppingassistance requests if converged
stop_requests = false;
counter = 0;
check_conv=0;
% numner of times human gave bad suggetions
human_misleads = 0;
% Helping steps
helping_steps = 0;
% rejecting human assistance
reject_human_assistance = false;
% simulated human value matrix
load HumanQ
%graphics
enable_graphics = false;
enable_result_graphics  = false; % just for results.
% human collaboration
enable_human_collaboration = true;
human_collaboration = false; % updated during session.
% rejection of human assitance
enable_rejection = true;

% ----------------------------------------------------------------------

% Learning Session (made out of N episodes)
case 'run'

CCRL('init')
episodes = 200;
for  episode_number=1:episodes
reached_goal = false; % for printing data to excel - know if reached
target or block
K = 5; % parameter for averages and convergence calculation
% rejection of human assitance
if (enable_rejection) && (~reject_human_assistance) % after
rejection no need to go in the if again
   if human_collaboration == true % meaning human helped at
previous episode
       if num_of_steps(size(num_of_steps,1)) >=
mean(num_of_steps(size(num_of_steps,1)-K:size(num_of_steps,1)-
1))
           human_misleads = human_misleads +1;
       end
       if human_misleads > 5
           reject_human_assistance = true;
           human_collaboration = false;
           enable_graphics = false;
       end
   end
end % rejection
X = 30; %number of episodes from which to start checking
if (enable_human_collaboration) && (episode_number >= X)  &&
(~reject_human_assistance) % don't get in if rejection.
```

```
   % evalute learning rate to deside whether to request human
intervention - averages of K episdods
   prev_avg = mean(num_of_steps(size(num_of_steps,1)-(2*K-
1):size(num_of_steps,1)-K));
   curr_avg = mean(num_of_steps(size(num_of_steps,1)-(K-
1):size(num_of_steps,1)));
   % checking if converged for at least 2 human assisttances - if so,
stop asking for help
   if (check_conv == num_of_steps(size(num_of_steps,1))) &&
(check_conv < max_steps)
       counter = counter + 1;
       if counter > (2 * K)
          stop_requests = true;
       end
   else
       check_conv = num_of_steps(size(num_of_steps,1));
       counter = 0;
   end
   % asking for assistance only if performance is not good enough .
   if (curr_avg/prev_avg > 0.95)  &&  (~stop_requests) &&
(episode_number < episodes)
       human_collaboration = true;
       collaboration_requests = collaboration_requests+1;
   else
       human_collaboration = false;
       enable_graphics = false;
   end
end % enable_human_collaboration
CCRL('episode')
end % for

% ------------------------------------------------------------------------

% Display Resulting Path (achieved by the learning process)
case 'result'

enable_graphics = true;
enable_result_graphics = true;
episode_number = 200;
CCRL('world'), pause(0.0005)
CCRL('episode')
enable_result_graphics  = false;

% --------------------------------------------------------------------

% Display Graphics
case 'world'

clf
axis([1 11 1 11 1 11]); % grid world size
grid on;
for i = 1:size(block,1)
   voxel(block(i,:),[1 1 1],'r',0.7); % obstacles
end
voxel(goal,[1 1 1],'g',0.7); % goal

% --------------------------------------------------------------------

% Learning Episode
case 'episode'

robot = start;
for step = 1:max_steps % episode steps
% display graphics (pause for display)
if enable_graphics
   if ~enable_result_graphics
       CCRL('world'), pause(0.0005)
   end
   voxel(robot,[1 1 1],'m',0.7);pause(0.000005); % dislpay robot
moves
end
if R(robot(1), robot(2), robot(3)) == 1.5 % reached goal
   Q(robot(1), robot(2), robot(3)) = R(robot(1), robot(2), robot(3));
   reached_goal = true;
   break;
end
if R(robot(1), robot(2), robot(3)) == -1 % hit an obstacle
   Q(robot(1), robot(2), robot(3)) = R(robot(1), robot(2), robot(3));
   break;
end
%Choosing next action - autonomus or semi-autonomus...
if (human_collaboration)  % human can help everywhere
%if (human_collaboration) && (robot(1)>5) && ( (robot(2) >2) &&
(robot(2)<10) ) && ( (robot(3)>2) && (robot(3)<9) ) % human can
help just at a certain region
   % automatic human collaboration
   CCRL('human')
   next_state  = human_next_state;
   helping_steps = helping_steps+1;
else % choose action autonomously
   % predict next possible states: each row for an action
   pstate = repmat( robot, na, 1) + move;
   pstate = min( max( pstate,1), repmat([nx,ny,nz],na,1)); % set of
possible states to move to (inside grid only)
   % linear index
   istate = sub2ind( [nx,ny,nz], pstate(:,1), pstate(:,2), pstate(:,3));
    tau = 1 / episode_number^1.3;
   pq = loss + gamma*Q(istate);    % each row for an action
   prob = exp(pq/tau);
   prob = prob./(sum(prob));   % selection probablity
   act = find( cumsum(prob) > rand(1));
   softmax_move = act(1);
   next_state =  pstate(softmax_move,:);
end % choosing next action
% update Q value
delta =  - 0.1 + gamma*Q(next_state(1),next_state(2),next_state(3)) -
Q(robot(1),robot(2),robot(3));
Q(robot(1),robot(2),robot(3)) = Q(robot(1),robot(2),robot(3)) +
alpha*delta;
robot = next_state; % moving to next state
end % episode steps
% update number of steps array
if episode_number == 1 % first value in the array
   if reached_goal
      num_of_steps = step; % reached goal
   else
      num_of_steps  = max_steps + 1; % hit block - assign max_steps
+  1 to note it and penalise  for average calculation
   end
else % rest of array
   % check when convergence is achieved
   if num_of_steps(size(num_of_steps,1)) ~=step
      convergence_episode = episode_number;
   end
   if reached_goal
      num_of_steps = [num_of_steps;step];
   else
   num_of_steps = [num_of_steps; max_steps + 1];
   end
end

%------------------------------------------------------------------------

% simulated human collaboration
case 'human'

% predict next possible states: each row for an action
pstate = repmat( robot, na, 1) + move;
pstate = min( max( pstate,1), repmat([nx,ny,nz],na,1)); % set of
possible states to move to (inside grid only)
% linear index
istate = sub2ind( [nx,ny,nz], pstate(:,1), pstate(:,2), pstate(:,3));
human_tau = 0.01; % human expertise
% suggest next action using softmax
pq = loss + gamma*HumanQ(istate);% each row for an action
prob = exp(pq/human_tau);
prob = prob./(sum(prob));   % selection probablity
act = find( cumsum(prob) > rand(1));
softmax_move = act(1);
human_next_state =  pstate(softmax_move,:);

end % case
```

IA.m

```
function IA(arg)

%--------------------------------------------------------------------------

%Variable Declarations
global block
global goal
global start
global nx ny nz ns na
global robot
global R
global Q
global alpha tau gamma delta
global human_tau
global move loss
global num_of_steps
global episode_number
global reached_goal
global enable_graphics
global enable_result_graphics
global enable_human_collaboration
global human_collaboration
global episodes
global human_next_state
global HumanQ
global convergence_episode
global max_steps
global helping_steps
%-----------------------------------------------------------------------

switch( arg)

%-----------------------------------------------------------------------

% variable Assignment
case 'init'

% obstacles
block = [ 10,5,1; 10,6,1; 10,7,1;
          10,5,2; 10,6,2; 10,7,2;
          10,5,3; 10,6,3; 10,7,3;
          8,5,1; 8,6,1; 8,7,1;
          8,5,2; 8,6,2; 8,7,2;
          8,5,3; 8,6,3; 8,7,3;
          9,5,1; 9,5,2; 9,5,3;
          9,7,1; 9,7,2; 9,7,3;
          1,10,10; 2,10,10;3,10,10; 4,10,10; 5,10,10; 6,10,10;
7,10,10; 8,10,10; 9, 10, 10; 10,10,10;
          1,10,9; 2,10,9;3,10,9; 4,10,9; 5,10,9; 6,10,9; 7,10,9;
8,10,9; 9, 10, 9; 10,10,9;
          1,1,1; 1,2,1; 2,1,1; 2,2,1;
          10,1,10;
          3,10,2; 4,10,2; 5,10,2; 6,10,2;
          5,6,1; 5,7,1;
          5,6,2; 5,7,2;
          5,6,3; 5,7,3;
          ];
 %goal point
goal = [9,6,4];
% starting point
start = [1,8,2];
% world size
nx = 10;   ny = 10;   nz = 10;   ns = nx*ny*nz ;
% actions
na = 6;    % Right,Left,Up,Down, Forward, Backward
move = [0,-1,0; 0,1,0; 0,0,1; 0,0,-1; 1,0,0; -1,0,0];
loss = -0.1*[ 1; 1; 1; 1; 1; 1];
% reward field
R = zeros( nx, ny, nz);
for i = 1:size(block,1)
   R(block(i,1),block(i,2),block(i,3)) = -1;
end
R(goal(1),goal(2),goal(3)) = 1.5;
% value field
Q = zeros( nx, ny, nz);
% learning parameters
alpha = 0.95; gamma = 0.99; lambda = 0.5;
% maximum steps for learning episode
max_steps = 200;
% Helping steps
helping_steps = 0;
% simulated human value matrix
load HumanQ
%graphics
enable_graphics = false;
enable_result_graphics  = false; % just for results.

% -----------------------------------------------------------------------

% Learning Session (made out of N episodes)
case 'run'

IA('init')
episodes = 200;
for  episode_number=1:episodes
reached_goal = false;
IA('episode')
end % for

% -------------------------------------------------------------------------

% Display Resulting Path (achieved by the learning process)
case 'result'

enable_graphics = true;
enable_result_graphics = true;% show trail of moves just for results
episode_number = 200;
IA('world'), pause(0.0005)
IA('episode')
enable_result_graphics  = false;

% ---------------------------------------------------------------------

% Display Graphics
case 'world'

clf
axis([1 11 1 11 1 11]); % grid world size
grid on;
for i = 1:size(block,1)
   voxel(block(i,:),[1 1 1],'r',0.7); % undesired areas
end
voxel(goal,[1 1 1],'g',0.7); % goal

% ---------------------------------------------------------------------

% Learning Episode
case 'episode'

robot = start;
for step = 1:max_steps % episode steps
% display graphics (pause for display)
if enable_graphics
   if ~enable_result_graphics
      IA('world'), pause(0.0005)
   end
   voxel(robot,[1 1 1],'m',0.7);pause(0.000005); % dislpay robot
moves
end

if R(robot(1), robot(2), robot(3)) == 1.5 % reached goal
   Q(robot(1), robot(2), robot(3)) = R(robot(1), robot(2), robot(3));
   reached_goal = true;
   break;
end
if R(robot(1), robot(2), robot(3)) == -1 % hit a block
   Q(robot(1), robot(2), robot(3)) = R(robot(1), robot(2), robot(3));
   break;
end

  %Choosing next action - autonomus or semi-autonomus...
% for introspection
 width_parameter = 1.3;
% for introspection and choosing steps autonomously later...
 pstate = repmat( robot, na, 1) + move;
pstate = min( max( pstate,1), repmat([nx,ny,nz],na,1)); % set of
possible states to move to (inside grid only)
% linear index
istate = sub2ind( [nx,ny,nz], pstate(:,1), pstate(:,2), pstate(:,3));
minimum = min(Q(istate));
maximum = max(Q(istate));
X = 30;
if (maximum-minimum<=width_parameter) &&
(episode_number>=X)  % human can help everywhere
%if (maximum-minimum<=width_parameter) &&
(episode_number>=K) && (robot(1)>5) && ( (robot(2) >2) &&
(robot(2)<10) ) && ( (robot(3)>2) && (robot(3)<9) ) % human can
help just at a certain region
   % automatic human collaboration
   IA('human')
   next_state  = human_next_state;
   helping_steps = helping_steps+1;
else % choose action autonomously
   tau = 1 / episode_number^1.3;
   pq = loss + gamma*Q(istate);    % each row for an action
   prob = exp(pq/tau);
   prob = prob./(sum(prob));   % selection probablity
   act = find( cumsum(prob) > rand(1));
   softmax_move = act(1);
   next_state =  pstate(softmax_move,:);
end % choosing next action
% update Q value
delta =  - 0.1 + gamma*Q(next_state(1),next_state(2),next_state(3)) -
Q(robot(1),robot(2),robot(3));
Q(robot(1),robot(2),robot(3)) = Q(robot(1),robot(2),robot(3)) +
alpha*delta;
robot = next_state; % moving to next state
end % episode steps
% update number of steps array
if episode_number == 1 % first value in the array
   if reached_goal
      num_of_steps = step; % reached goal
   else
      num_of_steps  = max_steps + 1; % hit block
   end
else % rest of array
   % check when convergence is achieved
   if num_of_steps(size(num_of_steps,1)) ~=step
      convergence_episode = episode_number;
   end
   if reached_goal
      num_of_steps = [num_of_steps;step];
   else
   num_of_steps = [num_of_steps; max_steps + 1];
   end
end

%-------------------------------------------------------------------------

% simulated human collaboration
case 'human'

% predict next possible states: each row for an action
pstate = repmat( robot, na, 1) + move;
pstate = min( max( pstate,1), repmat([nx,ny,nz],na,1)); % set of
possible states to move to (inside grid only)
% linear index
istate = sub2ind( [nx,ny,nz], pstate(:,1), pstate(:,2), pstate(:,3));
% suggest next action using softmax
human_tau = 0.01; % human expertise
pq = loss + gamma*HumanQ(istate);% each row for an action
prob = exp(pq/human_tau);
prob = prob./(sum(prob));  % selection probablity
act = find( cumsum(prob) > rand(1));
softmax_move = act(1);
human_next_state =  pstate(softmax_move,:);

end % case
```

Combined.m

```
function Combined(arg)

%-------------------------------------------------------------------------

%Variable Declarations
global block
global goal
global start
global nx ny nz ns na
global robot
global R
global Q
global alpha tau gamma delta
global human_tau
global move loss
global num_of_steps
global episode_number
global reached_goal
global enable_graphics
global enable_result_graphics
global enable_human_collaboration
global human_collaboration
global episodes
global human_next_state
global HumanQ
global convergence_episode
global collaboration_requests
global enable_rejection
global human_misleads
global reject_human_assistance
global check_conv
global stop_requests
global counter
global max_steps
global helping_steps

%-----------------------------------------------------------------------

switch( arg)

%-----------------------------------------------------------------------

% variable Assignment
case 'init'

% obstacles
block = [ 10,5,1; 10,6,1; 10,7,1;
          10,5,2; 10,6,2; 10,7,2;
          10,5,3; 10,6,3; 10,7,3;
          8,5,1; 8,6,1; 8,7,1;
          8,5,2; 8,6,2; 8,7,2;
          8,5,3; 8,6,3; 8,7,3;
          9,5,1; 9,5,2; 9,5,3;
          9,7,1; 9,7,2; 9,7,3;
          1,10,10; 2,10,10;3,10,10; 4,10,10; 5,10,10; 6,10,10;
7,10,10; 8,10,10; 9, 10, 10; 10,10,10;
          1,10,9; 2,10,9;3,10,9; 4,10,9; 5,10,9; 6,10,9; 7,10,9;
8,10,9; 9, 10, 9; 10,10,9;
          1,1,1; 1,2,1; 2,1,1; 2,2,1;
          10,1,10;
          3,10,2; 4,10,2; 5,10,2; 6,10,2;
          5,6,1; 5,7,1;
          5,6,2; 5,7,2;
          5,6,3; 5,7,3;
          ];

%goal point
goal = [9,6,4];
% starting point
start = [1,8,2];
% world size
nx = 10;    ny = 10;    nz = 10;    ns = nx*ny*nz ;
% actions
na = 6;     % Right,Left,Up,Down, Forward, Backward
move = [0,-1,0; 0,1,0; 0,0,1; 0,0,-1; 1,0,0; -1,0,0];
loss = -0.1*[ 1; 1; 1; 1; 1; 1];
% reward field
R = zeros( nx, ny, nz);
for i = 1:size(block,1)
   R(block(i,1),block(i,2),block(i,3)) = -1;
```

```
end
R(goal(1),goal(2),goal(3)) = 1.5;
% value field
Q = zeros( nx, ny, nz);
% learning parameters
alpha = 0.95; gamma = 0.99; lambda = 0.5;
% maximum steps for learning episode
max_steps = 200;
% Number of times agent is asking for human collaboration
collaboration_requests = 0;
% for stoppingassistance requests if converged
stop_requests = false;
counter = 0;
check_conv=0;
% numner of times human gave bad suggetions
human_misleads = 0;
% Helping steps
helping_steps = 0;
% rejecting human assistance
reject_human_assistance = false;
% simulated human value matrix
load HumanQ
%graphics
enable_graphics = false;
enable_result_graphics  = false; % just for results
% human collaboration
enable_human_collaboration = true;
human_collaboration = false; % updated during session
% rejection human assitance
enable_rejection = true;

% ----------------------------------------------------------------------

% Learning Session (made out of N episodes)
case 'run'

Combined('init')
episodes = 200;
for  episode_number=1:episodes
reached_goal = false;
K = 5; % parameter for averages and convergence calculation
% rejection of human assitance
if (enable_rejection) && (~reject_human_assistance) % after
rejection no need to go in the if again
   if human_collaboration == true % meaning human helped at
previous episode
      if num_of_steps(size(num_of_steps,1)) >=
mean(num_of_steps(size(num_of_steps,1)-K:size(num_of_steps,1)-
1));
         human_misleads = human_misleads +1;
      end
      if human_misleads > 1
         reject_human_assistance = true;
         human_collaboration = false;
          enable_graphics = false;
        end
   end
end % rejection
X = 30; %number of episodes from which to start checking (checking
2 K backwards....)
if (enable_human_collaboration) && (episode_number >= X)  &&
(~reject_human_assistance) % don't get in if rejection.
   % evalute learning rate to deside whether to request human
intervention - averages of K episdods
   prev_avg = mean(num_of_steps(size(num_of_steps,1)-(2*K-
1):size(num_of_steps,1)-K));
   curr_avg = mean(num_of_steps(size(num_of_steps,1)-(K-
1):size(num_of_steps,1)));
   % checking if converged for at least 2 human assisttances - if so,
stop asking for help
   if (check_conv == num_of_steps(size(num_of_steps,1))) &&
(check_conv < max_steps)
      counter = counter + 1;
      if counter > (2 * K)
         stop_requests = true;
      end
   else
      check_conv = num_of_steps(size(num_of_steps,1));
      counter = 0;
   end
   % asking for assistance only if performance is not good enough .
   if (curr_avg/prev_avg > 0.95)  &&  (~stop_requests) &&
(episode_number < episodes)
       human_collaboration = true;
       collaboration_requests = collaboration_requests+1;
   else
      human_collaboration = false;
      enable_graphics = false;
   end
end % enable_human_collaboration
Combined('episode')
end % for

% ------------------------------------------------------------------------

% Display Resulting Path (achieved by the learning process)
case 'result'

enable_graphics = true;
enable_result_graphics = true;% show trail of moves just for results
episode_number = 200;
Combined('world'), pause(0.0005)
Combined('episode')
enable_result_graphics  = false;

% ---------------------------------------------------------------------

% Display Graphics
case 'world'

clf
axis([1 11 1 11 1 11]); % grid world size
grid on;
for i = 1:size(block,1)
   voxel(block(i,:),[1 1 1],'r',0.7); % obstacles
end
voxel(goal,[1 1 1],'g',0.7); % goal


% ---------------------------------------------------------------------

% Learning Episode
case 'episode'

robot = start;
for step = 1:max_steps % episode steps
% display graphics (pause for display)
if enable_graphics
   if ~enable_result_graphics
      Combined('world'), pause(0.0005)
   end
   voxel(robot,[1 1 1],'m',0.7);pause(0.000005); % dislpay robot
moves
end
if R(robot(1), robot(2), robot(3)) == 1.5 % reached goal
   Q(robot(1), robot(2), robot(3)) = R(robot(1), robot(2), robot(3));
   reached_goal = true;
   break;
end
if R(robot(1), robot(2), robot(3)) == -1 % hit an obstacle
      Q(robot(1), robot(2), robot(3)) = R(robot(1), robot(2), robot(3));
   break;
end
%Choosing next action - autonomus or semi-autonomus...
% for introspection
width_parameter = 0.7;
 % for introspection and choosing steps autonomously later...
 pstate = repmat( robot, na, 1) + move;
pstate = min( max( pstate,1), repmat([nx,ny,nz],na,1)); % set of
possible states to move to (inside grid only)
% linear index
istate = sub2ind( [nx,ny,nz], pstate(:,1), pstate(:,2), pstate(:,3));
minimum = min(Q(istate));
maximum = max(Q(istate));
if (maximum-minimum<=width_parameter)  &&
(human_collaboration)  % human can help everywhere
```

```
% if (maximum-minimum<=width_parameter)  &&
(human_collaboration) && (robot(1)>5) && ( (robot(2) >2) &&
(robot(2)<10) ) && ( (robot(3)>2) && (robot(3)<9) ) % human can
help just at a certain region
   % automatic human collaboration
   Combined('human')
   next_state  = human_next_state;
   helping_steps = helping_steps+1;
else % choose action autonomously
   tau = 1 / episode_number^1.3;
   pq = loss + gamma*Q(istate);    % each row for an action
   prob = exp(pq/tau);
   prob = prob./(sum(prob));   % selection probablity
   act = find( cumsum(prob) > rand(1));
   softmax_move = act(1);
   next_state =  pstate(softmax_move,:);
end % choosing next action
% update Q value

delta =  - 0.1 + gamma*Q(next_state(1),next_state(2),next_state(3)) -
Q(robot(1),robot(2),robot(3));
Q(robot(1),robot(2),robot(3)) = Q(robot(1),robot(2),robot(3)) +
alpha*delta;
robot = next_state; % moving to next state
end % episode steps


% update number of steps
if episode_number == 1 % first value in the array
   if reached_goal
      num_of_steps = step; % reached goal
   else
      num_of_steps  = max_steps + 1; % hit block - assign max_steps
+  1 to note it and penalise  for average calculation
   end
else % rest of array
   % check when convergence is achieved
   if num_of_steps(size(num_of_steps,1)) ~=step
      convergence_episode = episode_number;
   end
   if reached_goal
      num_of_steps = [num_of_steps;step];
   else
   num_of_steps = [num_of_steps; max_steps + 1];
   end
end

%------------------------------------------------------------------------

% simulated human collaboration
case 'human'

% predict next possible states: each row for an action
pstate = repmat( robot, na, 1) + move;
pstate = min( max( pstate,1), repmat([nx,ny,nz],na,1)); % set of
possible states to move to (inside grid only)
% linear index
istate = sub2ind( [nx,ny,nz], pstate(:,1), pstate(:,2), pstate(:,3));
% suggest next action using softmax
human_tau = 0.01; % human expertise
pq = loss + gamma*HumanQ(istate);% each row for an action
prob = exp(pq/human_tau);
prob = prob./(sum(prob));   % selection probablity
act = find( cumsum(prob) > rand(1));
softmax_move = act(1);
human_next_state =  pstate(softmax_move,:);

end % case
```

Appendix VIII. Task Sequencing for a FMS – Source Code

MATLAB Code

SRL.m

```
function SRL(arg,arg2,arg3,arg4)

global Q
global tnow_array
global iteration
global min_tnow
global seq
global robot_seq
global best_seq
global best_robot_seq

% ------------------------------------------------------------------------

switch( arg)

% ------------------------------------------------------------------------

case 0

Q = ones(3,3,3,3,3,3,3,3,3);
min_tnow = 100000000;
iterations = 200;
for iteration=1:iterations
   SRL(1,0,0,arg4);
end

% ------------------------------------------------------------------------

case 1

global enable_graphics
global tnow
global sys_state
global proc_time
global order
global processed_parts
global part_waiting
global station
global target_station
global next_part
global vacuum_time
global Gamma % algorithm parameter
global Alpha % algorithm parameter
global trans_time
global event_stack
global SIGMA %for normal arrival times
global MU %for exponential inter arrival times

vacuum_time = 240;
trans_time= zeros(11);
trans_time(1,2) = 65; %
trans_time(1,3) = 15; %
trans_time(1,4) = 45; %
trans_time(1,5) = 45; %
trans_time(1,6) = 45; %
trans_time(1,7) = 25; %
trans_time(1,8) = 25; %
trans_time(1,9) = 25; %
trans_time(1,10) = 40; %
% trans_time(1,11) = 50; % N/A
trans_time(2,1) = 25; %
trans_time(2,3) = 20; %
trans_time(2,4) = 45; %
trans_time(2,5) = 45; %
trans_time(2,6) = 45; %
trans_time(2,7) = 25; %
trans_time(2,8) = 25; %
trans_time(2,9) = 25; %
trans_time(2,10) = 40; %
% trans_time(2,11) = 50; % N/A
trans_time(3,1) = 25; %
trans_time(3,2) = 45; %
trans_time(3,4) = 45; %
trans_time(3,5) = 45; %
trans_time(3,6) = 45; %
trans_time(3,7) = 25; %
trans_time(3,8) = 25; %
trans_time(3,9) = 25; %
trans_time(3,10) = 40; %
trans_time(3,11) = 40; %
trans_time(4,1) = 45; %
trans_time(4,2) = 45; %
trans_time(4,3) = 45; %
trans_time(4,5) = 65; %
trans_time(4,6) = 15; %
trans_time(4,7) = 50; %
trans_time(4,8) = 50; %
trans_time(4,9) = 50; %
trans_time(4,10) = 60; %
% trans_time(4,11) = 50; % N/A
trans_time(5,1) = 45; %
trans_time(5,2) = 45; %
trans_time(5,3) = 45; %
trans_time(5,4) = 25; %
trans_time(5,6) = 20; %
trans_time(5,7) = 50; %
trans_time(5,8) = 50; %
trans_time(5,9) = 50; %
trans_time(5,10) = 60; %
% trans_time(5,11) = 50; % N/A
trans_time(6,1) = 45; %
trans_time(6,2) = 45; %
trans_time(6,3) = 45; %
trans_time(6,4) = 25; %
trans_time(6,5) = 45; %
trans_time(6,7) = 50; %
trans_time(6,8) = 50; %
trans_time(6,9) = 50; %
trans_time(6,10) = 60; %
trans_time(6,11) = 60; %
trans_time(7,1) = 25; %
trans_time(7,2) = 25; %
trans_time(7,3) = 25; %
trans_time(7,4) = 50; %
trans_time(7,5) = 50; %
trans_time(7,6) = 50; %
trans_time(7,8) = 50; %
trans_time(7,9) = 15; %
trans_time(7,10) = 30; %
% % trans_time(7,11) = 50; %N/A
trans_time(8,1) = 25; %
trans_time(8,2) = 25; %
trans_time(8,3) = 25; %
trans_time(8,4) = 50; %
trans_time(8,5) = 50; %
trans_time(8,6) = 50; %
trans_time(8,7) = 25; %
trans_time(8,9) = 25; %
trans_time(8,10) = 30; %
% trans_time(8,11) = 50; %N/A
trans_time(9,1) = 25; %
trans_time(9,2) = 25; %
trans_time(9,3) = 25; %
trans_time(9,4) = 50; %
trans_time(9,5) = 50; %
trans_time(9,6) = 50; %
trans_time(9,7) = 25; %
trans_time(9,8) = 25; %
trans_time(9,10) = 30; %
trans_time(9,11) = 30; %
trans_time(10,1) = 40; %
trans_time(10,4) = 60; %
trans_time(10,7) = 30; %
trans_time(11,1) = 40; %
trans_time(11,2) = 40; %
trans_time(11,3) = 40; %
trans_time(11,4) = 60; %
trans_time(11,5) = 60; %
trans_time(11,6) = 60; %
trans_time(11,7) = 30; %
trans_time(11,8) = 30; %
trans_time(11,9) = 30; %
trans_time(11,10) = 20; %
% M1 - Mill, M2 - Lathe 1,  M3 - LAthe 2
proc_time = zeros(1,3);
proc_time(1) = 185; %M2
proc_time(2) = 185; %M3
proc_time(3) = 600; %M1
enable_graphics = arg2;
Gamma = 0.9;
Alpha = 0.05;
% order = [1;1;2;1;2]; % 1 - ROOK, 2 - SIGN
order = arg4;
order = [order;-1]; %mark order is finished with -1 (end of list)
part_number = 1; % starting with part in the head of the batch
next_part = order(part_number);
sys_state = zeros(1,9);
processed_parts = 0;
tnow = 0;
seq = [sys_state,tnow];
robot_seq = [10,tnow];
MU = 200;
SIGMA = MU/10;

%------------------------------------------------------------------------

%Stations:
% 1 - M2_IB
% 2 - M2
% 3 - M2_OB
% 4 - M3_IB
% 5 - M3
% 6 - M3_OB
% 7 - M1_IB
% 8 - M1
% 9 - M1_OB
% 10 - AGV_raw
% 11 - AGV_proccessed
% event structure:
% [event,station,time, target_station]
%events:
% 1: robot arrive empty
% 2: robot arrive with template
% 3: robot arrive with part
% 4: robot arrive with template +part
% 5:M2 finish proccessing
% 6:M3 finish proccessing
% 7:M1 finish proccessing
% 8: part arriveal
% system states:
% [M2_IB, M2, M2_OB, M3_IB, M3, M3_OB, M1_IB, M1,
M1_OB]
%Machine: 0-free, 1-Working, 2-full after process
%Buffer:0-free, 1-template, 2-template+part
% arrival from store:
% next_part = 1 %rook
% next_part = 2 %sign
% next_part = 0 %no part arrived
% next_part = -1 %end of order
%station = robot state

%------------------------------------------------------------------------

%init

% inter_arrival_time = round(normrnd(MU,SIGMA));
% inter_arrival_time = exprnd(MU);
inter_arrival_time = MU;
event_stack = [8,-1, tnow+inter_arrival_time,-1];
station = 10; % robot starts at AGV _raw
if next_part == 1
   target_station = 1; %go to M2
else
   target_station = 7; % go to M1
end

%------------------------------------------------------------------------

while 0<1 % endless loop. will end with break
prev_sys_state = sys_state;
event_arr = event_stack(1,:);
event = event_arr(1);
tnow = event_arr(3);
if ((event==1) || (event==2) || (event ==3) || (event==4)) % only robot
moving events change the station
   station = event_arr(2);
   target_station = event_arr(4);
   %update robot_seq
   if event == 2
      if station == 3
         robot_seq = [1,tnow-trans_time(1,3);robot_seq];
      elseif station == 6
         robot_seq = [4,tnow-trans_time(4,6);robot_seq];
      elseif station == 9
         robot_seq = [7,tnow-trans_time(7,9);robot_seq];
      end
   end
   if event == 3
      if station == 2
         robot_seq = [1,tnow-trans_time(1,2);robot_seq];
      elseif station == 5
         robot_seq = [4,tnow-trans_time(4,5);robot_seq];
      elseif station == 8
         robot_seq = [7,tnow-trans_time(7,8);robot_seq];
      elseif station == 3
         robot_seq = [2,tnow-trans_time(2,3);robot_seq];
      elseif station == 6
         robot_seq = [5,tnow-trans_time(5,6);robot_seq];
      elseif station == 9
         robot_seq = [8,tnow-trans_time(8,9);robot_seq];
      end
   end
  if (event == 4) && (station == 11)
      if target_station == 3
         robot_seq = [3,tnow-trans_time(3,11);robot_seq];
      elseif target_station == 6
         robot_seq = [6,tnow-trans_time(6,11);robot_seq];
     elseif target_station == 9
         robot_seq = [9,tnow-trans_time(9,11);robot_seq];
      end
   end
   robot_seq = [station,tnow;robot_seq];
end
if (event == 8) % part arrives
   if (station == 10) %robot waiting for it
      new_event = [4,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to IB
      event_stack = [event_stack;new_event];
      part_waiting = 0;
      part_number = part_number +1;
      next_part = order(part_number);
      if next_part ~= -1
         % inter_arrival_time = round(normrnd(MU,SIGMA));
         % inter_arrival_time = exprnd(MU);
        inter_arrival_time = MU;
        new_event = [8,-1, tnow+inter_arrival_time,-1];% new part
arrival
        event_stack = [event_stack;new_event];
      end
   else
      part_waiting = 1; % part arrived and waiting
   end
end % event == 8
if(event == 1) % robot arrives empty
   if (station == 10) %came to take part
      if (part_waiting == 1) %part waiting
```

```
          new_event = [4,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to IB
          event_stack = [event_stack;new_event];
          part_waiting = 0;
          part_number = part_number +1;
         next_part = order(part_number);
         if next_part ~= -1
            % inter_arrival_time = round(normrnd(MU,SIGMA));
            % inter_arrival_time = exprnd(MU);
            inter_arrival_time = MU;
            new_event = [8,-1, tnow+inter_arrival_time,-1];% new part
arrival
            event_stack = [event_stack;new_event];
         end
      end
   end % station == 10
   if (station == 2) % M2
      if (sys_state(2) == 2) % M2 finished process
          new_event = [3,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to OB
          event_stack = [event_stack;new_event];
          end
   end % station == 2
   if (station == 5)
      if (sys_state(5) == 2) % M3 finished process
          new_event = [3,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to OB
          event_stack = [event_stack;new_event];
      end
   end % station == 5
   if (station == 8)
      if (sys_state(8) == 2) % M1 finished process
          new_event = [3,target_station,
tnow+vacuum_time+trans_time(station,target_station),target_station];
% vacuum + taking part to OB
          event_stack = [event_stack;new_event];
      end
   end % station == 8
end    % event == 1
if (event ==2 ) % robot arrives with empty template to OB
   if (station == 3)
       sys_state(3) = 1; % update sys_state - OB  with template
       sys_state(1) = 0; % update sys_state - OB  with template
   end
   if (station == 6)
       sys_state(6) = 1; % update sys_state - OB  with template
      sys_state(4) = 0; % update sys_state - OB  with template
   end
   if (station == 9)
       sys_state(9) = 1; % update sys_state - OB  with template
       sys_state(7) = 0; % update sys_state - OB  with template
    end
 end % event == 2
if (event ==3 ) % robot arrives with part to OB  or to machine
   if (station == 2) % inserting to M2
      sys_state(1) = 1; % update sys_state - IB  with template alone
      sys_state(2) = 1; % update sys_state - M2 proccessing
      new_event = [5, -1, tnow+proc_time(1),-1];% M2 finish
proccessing
      event_stack = [event_stack;new_event];
   end
   if (station == 3)  % arrive to M2_OB with part
      sys_state(3) = 2; % update sys_state - OB  with template+part
      sys_state(2) = 0; % M2 is free
   end
   if (station == 5) % inserting to M3
      sys_state(4) = 1; % update sys_state - IB  with template alone
      sys_state(5) = 1; % update sys_state - M3 proccessing
      new_event = [6, -1, tnow+proc_time(2),-1];% M3 finish
proccessing
      event_stack = [event_stack;new_event];
    end
    if (station == 6) % arrive to M3_OB with part
      sys_state(6) = 2; % update sys_state - OB  with template+part
      sys_state(5) = 0; % M3 is free
      end
      if (station == 8) % inserting to M1
        sys_state(7) = 1; % update sys_state - IB  with template alone
        sys_state(8) = 1; % update sys_state - M3 proccessing
        new_event = [7, -1, tnow+proc_time(3),-1];% M3 finish
proccessing
        event_stack = [event_stack;new_event];
      end
      if (station == 9) % arrive to M1_OB with part
         sys_state(9) = 2; % update sys_state - OB  with template+part
         sys_state(8) = 0; % M1 is free
      end
  end % event == 3
 if (event == 4 ) % robot arrives with tempale + part to IB or to
AGV_proccessed
     if (station ==  1)
        sys_state(1) = 2; % update sys_state - OB  with template+part
     end
     if (station == 4)
        sys_state(4) = 2; % update sys_state - OB  with template+part
     end
     if (station == 7)
        sys_state(7) = 2; % update sys_state - OB  with template+part
     end
        if (station == 11)
        processed_parts = processed_parts +1; % finished proccessing
one more part
        sys_state(target_station) = 0;
       if processed_parts == size(order,1)-1
          break
       end
    end
 end % event == 4
 if (event == 5 ) % M2 finished processing
    sys_state(2) = 2; % M2 after process
    if (station ==  2)  % robot waiting for part
       new_event = [3,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to OB
       event_stack = [event_stack;new_event];
    end
  end % event == 5
  if (event == 6 ) % M3 finished processing
      sys_state(5) = 2; % M3 after process is free
     if (station ==  5)  % robot waiting for part
          new_event = [3,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to OB
          event_stack = [event_stack;new_event];
     end
   end % event == 6
  if (event == 7 ) % M1 finished processing
   sys_state(8) = 2; % M1 after process
     if (station ==  8)  % robot waiting for part
          new_event = [3,target_station,
tnow+vacuum_time+trans_time(station,target_station),target_station];
% vacuum + taking part to OB
          event_stack = [event_stack;new_event];
    end
end % event == 7

% ------------------------------------------------------------------------

 if ( (event == 2)  || (event==3)  || (event==4) )  % choose next state
only at there events
% not choosing next event if robot is waiting for part to finish
processing....
optional_next_state = [0,0,0,0,0,0,0,0,0,0]; %initiate array with fictive
state
optional_next_event = [0,0,0,0]; %initiate array with fictive event
% M1:
if (sys_state(7) == 0) && (next_part == 2)  % M1_IB free + next part
is sign
    optional_next_state = [optional_next_state;
sys_state+[0,0,0,0,0,0,2,0,0]];
    optional_next_event = [optional_next_event;
1,10,tnow+trans_time(station,10),7]; % go and take part or wait for it
and then take to M1_IB
```

```
end
if (sys_state(7) == 2) && (sys_state(8) == 0)  % M1_IB
template+part + M1 free
    optional_next_state = [optional_next_state; sys_state+[0,0,0,0,0,0,-
1,1,0]];
    optional_next_event = [optional_next_event;
3,8,tnow+trans_time(station,7)+trans_time(7,8),8]; % go to template.
take part from template and insert to M1
end
if (sys_state(7) == 1) && (sys_state(9) == 0)  % M1_IB template +
M1_OB free
    optional_next_state = [optional_next_state; sys_state+[0,0,0,0,0,0,-
1,0,1]];
    optional_next_event = [optional_next_event;
2,9,tnow+trans_time(station,7)+trans_time(7,9),9]; % go to template.
take template from IB to OB
end
if (sys_state(8) == 1 ) && (sys_state(9) ==1)  % M1 during process +
M1_OB tempale
    optional_next_state = [optional_next_state;
sys_state+[0,0,0,0,0,0,0,-1,1]];
    optional_next_event = [optional_next_event;
1,8,tnow+trans_time(station,8),9]; % go to M1. take part to template
or wait for it and then take to template
end
if (sys_state(8) == 2) && (sys_state(9) ==1)  % M1 after process +
M1_OB tempale
    optional_next_state = [optional_next_state;
sys_state+[0,0,0,0,0,0,0,-2,1]];
    optional_next_event = [optional_next_event;
1,8,tnow+trans_time(station,8),9]; % go to M1. take part to template
or wait for it and then take to template
end
if (sys_state(9) == 2)  % M1_OB tempale+part (move tempalate +
part to AGV)
    optional_next_state = [optional_next_state;
sys_state+[0,0,0,0,0,0,0,0,-2]];
    optional_next_event = [optional_next_event;
4,11,tnow+trans_time(station,9)+trans_time(9,11),9]; % go to
M1_OB. take template + part to AGV_proccessed
        %here target_station in the origin station
end
% M2:
if (sys_state(1) == 0) && (next_part == 1)  % M2_IB free + next part
is rook
    optional_next_state = [optional_next_state;
sys_state+[2,0,0,0,0,0,0,0,0]];
    optional_next_event = [optional_next_event;
1,10,tnow+trans_time(station,10),1]; % go and take part or wait for it
and then take to M2_IB
end
if (sys_state(1) == 2) && (sys_state(2) == 0)  % M2_IB
template+part + M2 free
    optional_next_state = [optional_next_state; sys_state+[-
1,1,0,0,0,0,0,0,0]];
    optional_next_event = [optional_next_event;
3,2,tnow+trans_time(station,1)+trans_time(1,2),2]; % go to template.
take part from template and insert to M2
end
if (sys_state(1) == 1) && (sys_state(3) == 0)  % M2_IB template +
M2_OB free
    optional_next_state = [optional_next_state; sys_state+[-
1,0,1,0,0,0,0,0,0]];
    optional_next_event = [optional_next_event;
2,3,tnow+trans_time(station,1)+trans_time(1,3),3]; % go to template.
take template from IB to OB
end
if (sys_state(2) == 1)  && (sys_state(3) == 1)  % M2 during process +
M2_OB tempale
    optional_next_state = [optional_next_state; sys_state+[0,-
1,1,0,0,0,0,0,0]];
    optional_next_event = [optional_next_event;
1,2,tnow+trans_time(station,2),3]; % go to M2. take part to template
of wait for it and than take to template
end
if (sys_state(2) == 2) && (sys_state(3) == 1)  % M2 after process +
M2_OB tempale
    optional_next_state = [optional_next_state; sys_state+[0,-
2,1,0,0,0,0,0,0]];
    optional_next_event = [optional_next_event;
1,2,tnow+trans_time(station,2),3]; % go to M2. take part to template
of wait for it and than take to template
end
if (sys_state(3) ==2)  % M2_OB tempale+part (move tempalate + part
to AGV)
    optional_next_state = [optional_next_state; sys_state+[0,0,-
2,0,0,0,0,0,0]];
    optional_next_event = [optional_next_event;
4,11,tnow+trans_time(station,3)+trans_time(3,11),3]; % go to
M2_OB. take template + part to AGV_proccessed
    %here target_station in the origin station
end
% M3:
if (sys_state(4) == 0) && (next_part == 1)  % M3_IB free + next part
is rook
    optional_next_state = [optional_next_state;
sys_state+[0,0,0,2,0,0,0,0,0]];
    optional_next_event = [optional_next_event;
1,10,tnow+trans_time(station,10),4]; % go and take part or wait for it
and then take to M3_IB
end
if (sys_state(4) == 2) && (sys_state(5) == 0)  % M3_IB
template+part + M3 free
    optional_next_state = [optional_next_state; sys_state+[0,0,0,-
1,1,0,0,0,0]];
    optional_next_event = [optional_next_event;
3,5,tnow+trans_time(station,4)+trans_time(4,5),5]; % go to template.
take part from template and insert to M3
end
if (sys_state(4) == 1) && (sys_state(6) == 0)  % M3_IB template +
M3_OB free
    optional_next_state = [optional_next_state; sys_state+[0,0,0,-
1,0,1,0,0,0]];
    optional_next_event = [optional_next_event;
2,6,tnow+trans_time(station,4)+trans_time(4,6),6]; % go to template.
take template from IB to OB
end
if (sys_state(5) == 1) && (sys_state(6) == 1)  % M3 during process +
M3_OB tempale
    optional_next_state = [optional_next_state; sys_state+[0,0,0,0,-
1,1,0,0,0]];
    optional_next_event = [optional_next_event;
1,5,tnow+trans_time(station,5),6]; % go to M3. take part to template
or wait for it and then take to template
end
if (sys_state(5) == 2) && (sys_state(6) == 1)  % M3 after process +
M3_OB tempale
    optional_next_state = [optional_next_state; sys_state+[0,0,0,0,-
2,1,0,0,0]];
    optional_next_event = [optional_next_event;
1,5,tnow+trans_time(station,5),6]; % go to M3. take part to template
or wait for it and then take to template
end
if (sys_state(6) == 2)  % M3_OB tempale+part (move tempalate +
part to AGV)
    optional_next_state = [optional_next_state; sys_state+[0,0,0,0,0,-
2,0,0,0]];
    optional_next_event = [optional_next_event;
4,11,tnow+trans_time(station,6)+trans_time(6,11),6]; % go to
M2_OB. take template + part to AGV_proccessed
        %here target_station in the origin station
end
optional_next_state=optional_next_state(2:1:end,:); % erase fictive
state
optional_next_event=optional_next_event(2:1:end,:); % erase fictive
event

%-------------------------------------------------------------------------

%finding  best next step
% +1 to work in array from 1 to 3 insteed of 0 to 2
max = -10000;
    for i=1:size(optional_next_state,1)
        if
Q(optional_next_state(i,1)+1,optional_next_state(i,2)+1,optional_nex
t_state(i,3)+1,optional_next_state(i,4)+1,optional_next_state(i,5)+1,o
ptional_next_state(i,6)+1,optional_next_state(i,7)+1,optional_next_st
ate(i,8)+1,optional_next_state(i,9)+1)>=max
          max =
Q(optional_next_state(i,1)+1,optional_next_state(i,2)+1,optional_nex
t_state(i,3)+1,optional_next_state(i,4)+1,optional_next_state(i,5)+1,o
ptional_next_state(i,6)+1,optional_next_state(i,7)+1,optional_next_st
ate(i,8)+1,optional_next_state(i,9)+1);
          ind = i;
       end
   end
best_next_state = optional_next_state(ind,:);
best_next_event =  optional_next_event(ind,:);
%Choosing next step
% epsilon = 1;
epsilon = 1  / iteration;
if rand>epsilon
   next_state = best_next_state;
   new_event = best_next_event;
else
    state_ind = ceil(rand * size(optional_next_state,1));
    next_state = optional_next_state(state_ind,:);
    new_event = optional_next_event(state_ind,:);
end
event_stack = [event_stack;new_event];
end %if ( (event == 2) || (event==3)  || (event==4) )  % choose next
state

% ---------------------------------------------------------------------------------
--------

if   sum(prev_sys_state==sys_state) ~=9 % only if changed state
   % updating position sequence
   seq = [sys_state,tnow; seq];
   % updating Q of previous step
   delta = Gamma * Q
(sys_state(1)+1,sys_state(2)+1,sys_state(3)+1,sys_state(4)+1,sys_stat
e(5)+1,sys_state(6)+1,sys_state(7)+1,sys_state(8)+1,sys_state(9)+1)  -
Q
(prev_sys_state(1)+1,prev_sys_state(2)+1,prev_sys_state(3)+1,prev_s
ys_state(4)+1,prev_sys_state(5)+1,prev_sys_state(6)+1,prev_sys_stat
e(7)+1,prev_sys_state(8)+1,prev_sys_state(9)+1);
   Q
(prev_sys_state(1)+1,prev_sys_state(2)+1,prev_sys_state(3)+1,prev_s
ys_state(4)+1,prev_sys_state(5)+1,prev_sys_state(6)+1,prev_sys_stat
e(7)+1,prev_sys_state(8)+1,prev_sys_state(9)+1) = Q
(prev_sys_state(1)+1,prev_sys_state(2)+1,prev_sys_state(3)+1,prev_s
ys_state(4)+1,prev_sys_state(5)+1,prev_sys_state(6)+1,prev_sys_stat
e(7)+1,prev_sys_state(8)+1,prev_sys_state(9)+1) + Alpha * delta;
 end
% removing event that was performed and sorting event_stack
event_stack=event_stack(2:1:end,:);
event_stack=sortrows(event_stack,3);
end %while

% -------------------------------------------------------------------------

% after "break" update seq with last move to AVG_processed
tnow = tnow+trans_time(station,11);
sys_state = zeros(1,9);
seq = [sys_state,tnow; seq];
% update Q with last move
Q
(sys_state(1)+1,sys_state(2)+1,sys_state(3)+1,sys_state(4)+1,sys_stat
e(5)+1,sys_state(6)+1,sys_state(7)+1,sys_state(8)+1,sys_state(9)+1) =
1.5 + Gamma * Q
(sys_state(1)+1,sys_state(2)+1,sys_state(3)+1,sys_state(4)+1,sys_stat
e(5)+1,sys_state(6)+1,sys_state(7)+1,sys_state(8)+1,sys_state(9)+1);
% reward of 1.5
% update best_seq, best_robot_seq
 if tnow<min_tnow
   min_tnow = tnow;
   best_seq = seq;
   best_robot_seq = robot_seq;
 end
%updating Q according to tnow, episode results
factor = 1/tnow;
for i=1:size(seq,1)-1
Q(seq(i,1)+1,seq(i,2)+1,seq(i,3)+1,seq(i,4)+1,seq(i,5)+1,seq(i,6)+1,se
q(i,7)+1,seq(i,8)+1,seq(i,9)+1) =
Q(seq(i,1)+1,seq(i,2)+1,seq(i,3)+1,seq(i,4)+1,seq(i,5)+1,seq(i,6)+1,se
q(i,7)+1,seq(i,8)+1,seq(i,9)+1)*factor;
end
% update tnow_array (for chart)
if iteration == 1
   tnow_array = tnow;
else
   tnow_array = [tnow_array;tnow];
end

end%case
```

FIFO.m

```
function FIFO(arg,arg2,arg3,arg4)

global tnow_array
global iteration
global min_tnow
global seq
global best_seq
global best_robot_seq

switch( arg)

% -------------------------------------------------------------------------

case 0

min_tnow = 100000000; %just a big number
iterations = 1;
for iteration=1:iterations
   FIFO(1,0,0,arg4);
end

% -------------------------------------------------------------------------

case 1

global enable_graphics
global tnow
global sys_state
global proc_time % machine processing times
global order % parts batch
global processed_parts
global part_waiting
global station
global target_station
global next_part % next part arriveing to system
global vacuum_time % vacuuming time
global prev_tasks % previous tasks isnerted to task_queue for the
various machines [M1,M2,M3]
global trans_time
global event_stack
global MU %for exponential and normal inter arrival times
global SIGMA % for normal inter arrival times
global next_lathe % for taking parts to M2 and M3 alternately
global task_queue % fifo queue [task]
global next_M1_task
global next_M2_task
global next_M3_task
global last_lathe
global M1_part_taken

%-------------------------------------------------------------------------

vacuum_time = 240;
trans_time= zeros(11);
trans_time(1,2) = 65; %
trans_time(1,3) = 15; %
trans_time(1,4) = 45; %
trans_time(1,5) = 45; %
trans_time(1,6) = 45; %
trans_time(1,7) = 25; %
trans_time(1,8) = 25; %
trans_time(1,9) = 25; %
trans_time(1,10) = 40; %
% trans_time(1,11) = 50; % N/A
trans_time(2,1) = 25; %
trans_time(2,3) = 20; %
trans_time(2,4) = 45; %
trans_time(2,5) = 45; %
trans_time(2,6) = 45; %
trans_time(2,7) = 25; %
trans_time(2,8) = 25; %
trans_time(2,9) = 25; %
trans_time(2,10) = 40; %
% trans_time(2,11) = 50; % N/A
 trans_time(3,1) = 25; %
trans_time(3,2) = 45; %
trans_time(3,4) = 45; %
trans_time(3,5) = 45; %
trans_time(3,6) = 45; %
trans_time(3,7) = 25; %
trans_time(3,8) = 25; %
trans_time(3,9) = 25; %
trans_time(3,10) = 40; %
trans_time(3,11) = 40; %
 trans_time(4,1) = 45; %
trans_time(4,2) = 45; %
trans_time(4,3) = 45; %
trans_time(4,5) = 65; %
trans_time(4,6) = 15; %
trans_time(4,7) = 50; %
trans_time(4,8) = 50; %
trans_time(4,9) = 50; %
trans_time(4,10) = 60; %
% trans_time(4,11) = 50; % N/A
trans_time(5,1) = 45; %
trans_time(5,2) = 45; %
trans_time(5,3) = 45; %
trans_time(5,4) = 25; %
trans_time(5,6) = 20; %
trans_time(5,7) = 50; %
trans_time(5,8) = 50; %
trans_time(5,9) = 50; %
trans_time(5,10) = 60; %
% trans_time(5,11) = 50; % N/A
 trans_time(6,1) = 45; %
trans_time(6,2) = 45; %
trans_time(6,3) = 45; %
trans_time(6,4) = 25; %
trans_time(6,5) = 45; %
trans_time(6,7) = 50; %
trans_time(6,8) = 50; %
trans_time(6,9) = 50; %
trans_time(6,10) = 60; %
trans_time(6,11) = 60; %
trans_time(7,1) = 25; %
trans_time(7,2) = 25; %
trans_time(7,3) = 25; %
trans_time(7,4) = 50; %
trans_time(7,5) = 50; %
trans_time(7,6) = 50; %
trans_time(7,8) = 50; %
trans_time(7,9) = 15; %
trans_time(7,10) = 30; %
% % trans_time(7,11) = 50; %N/A
trans_time(8,1) = 25; %
trans_time(8,2) = 25; %
trans_time(8,3) = 25; %
trans_time(8,4) = 50; %
trans_time(8,5) = 50; %
trans_time(8,6) = 50; %
trans_time(8,7) = 25; %
trans_time(8,9) = 25; %
trans_time(8,10) = 30; %
% trans_time(8,11) = 50; %N/A
 trans_time(9,1) = 25; %
trans_time(9,2) = 25; %
trans_time(9,3) = 25; %
trans_time(9,4) = 50; %
trans_time(9,5) = 50; %
trans_time(9,6) = 50; %
trans_time(9,7) = 25; %
trans_time(9,8) = 25; %
trans_time(9,10) = 30; %
trans_time(9,11) = 30; %
trans_time(10,1) = 40; %
trans_time(10,4) = 60; %
trans_time(10,7) = 30; %
trans_time(11,1) = 40; %
trans_time(11,2) = 40; %
trans_time(11,3) = 40; %
trans_time(11,4) = 60; %
trans_time(11,5) = 60; %
trans_time(11,6) = 60; %
trans_time(11,7) = 30; %
trans_time(11,8) = 30; %
trans_time(11,9) = 30; %
trans_time(11,10) = 20; %
enable_graphics = arg2;
prev_tasks = [-1,-1,-1];
% M1 - Mill, M2 - Lathe 1, M3 - LAthe 2
proc_time = zeros(1,3);
proc_time(1) = 185; %M2
proc_time(2) = 185; %M3
proc_time(3) = 600; %M1
% order = [1;1;2;1;2]; % 1 - ROOK, 2 - SIGN
order = arg4;
order = [order;-1]; %mark order is finished with -1 (end of list)
part_number = 1; % starting with part in the head of the batch
next_part = order(part_number);
sys_state = zeros(1,9);
processed_parts = 0;
tnow = 0;
seq = [sys_state,tnow];
robot_seq = [10,tnow];
MU = 100;
SIGMA = MU/10;
task_queue = [];

%--------------------------------------------------------------------------

% Tasks:
% 300 - load part from M2_IB to M2
% 301 - move template from M2_ID to M2_OB
% 302 - move part out of M2 to M2_OB
% 400 - load part from M3_IB to M3
% 401 - move template from M3_ID to M3_OB
% 402 - move part out of M3 to M3_OB
% 500 - load part from M1_IB to M1
% 501 - move template from M1_ID to M1_OB
% 502 - move part out of M1 to M1_OB
% 6 - taking template from M1_OB to AGV_processed
% 7 - taking template from M2_OB to AGV_processed
% 8 - taking template from M3_OB to AGV_processed
% 9 - taking template from AGV_raw to M1_IB
% 10 - taking template from AGV_raw to M2_IB
% 11 - taking template from AGV_raw to M3_IB
%Stations:
 % 1 - M2_IB
% 2 - M2
% 3 - M2_OB
% 4 - M3_IB
% 5 - M3
% 6 - M3_OB
% 7 - M1_IB
% 8 - M1
% 9 - M1_OB
% 10 - AGV_raw
% 11 - AGV_proccessed
% event structure:
% [event,station,time, target_station]
%events:
% 1: robot arrive empty
% 2: robot arrive with template
% 3: robot arrive with part
% 4: robot arrive with template +part
% 5:M2 finish proccessing
% 6:M3 finish proccessing
% 7:M1 finish proccessing
% 8: part arriveal
% system states
% [M2_IB, M2, M2_OB, M3_IB, M3, M3_OB, M1_IB, M1,
M1_OB]
%Machine: 0-free, 1-Working, 2-full after process
%Buffer:0-free, 1-template, 2-template+part, (-1)-part+template
headed towards it
% arrival from store
% next_part = 1 %rook
% next_part = 2 %sign
% next_part = 0 %no part arrived
% next_part = -1 %end of order
%station = robot state

%--------------------------------------------------------------------------

%init:

% inter_arrival_time = round(normrnd(MU,SIGMA));
% inter_arrival_time = exprnd(MU);
inter_arrival_time = MU;
event_stack = [8,-1, tnow+inter_arrival_time,-1];
station = 10; % robot starts at AGV _raw
if next_part == 1
   target_station = 1; % go to M2
else
   target_station = 7; % go to M1
end
next_lathe = 2;
last_lathe = -1;
M1_part_taken = 0;

%--------------------------------------------------------------------------

while 0<1 % endless loop.  ends with  "break"
prev_sys_state = sys_state;
event_arr = event_stack(1,:);
event = event_arr(1);
tnow = event_arr(3);
if ((event==1) || (event==2) || (event ==3) || (event==4)) % only robot
moving events change the station
   station = event_arr(2);
   target_station = event_arr(4);
    %update robot_seq
   if event == 2
      if station == 3
         robot_seq = [1,tnow-trans_time(1,3);robot_seq];
      elseif station == 6
         robot_seq = [4,tnow-trans_time(4,6);robot_seq];
      elseif station == 9
         robot_seq = [7,tnow-trans_time(7,9);robot_seq];
      end
   end
   if event == 3
      if station == 2
         robot_seq = [1,tnow-trans_time(1,2);robot_seq];
      elseif station == 5
         robot_seq = [4,tnow-trans_time(4,5);robot_seq];
      elseif station == 8
         robot_seq = [7,tnow-trans_time(7,8);robot_seq];
      elseif station == 3
         robot_seq = [2,tnow-trans_time(2,3);robot_seq];
      elseif station == 6
         robot_seq = [5,tnow-trans_time(5,6);robot_seq];
      elseif station == 9
         robot_seq = [8,tnow-trans_time(8,9);robot_seq];
      end
   end
   if (event == 4) && (station == 11)
      if target_station == 3
         robot_seq = [3,tnow-trans_time(3,11);robot_seq];
      elseif target_station == 6
         robot_seq = [6,tnow-trans_time(6,11);robot_seq];
     elseif target_station == 9
         robot_seq = [9,tnow-trans_time(9,11);robot_seq];
      end
   end
  robot_seq = [station,tnow;robot_seq];
end
if (event == 8) % part arrives
       part_waiting = 1; % part arrived and waiting
end % event == 8
if(event == 1) % robot arrives empty
   if (station == 10) %came to take part
      if (part_waiting == 1) %part waiting
         new_event = [4,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to IB
         event_stack = [event_stack;new_event];
     if target_station == 1
        next_lathe = 3;
     end
     if target_station == 4
        next_lathe = 2;
     end
     part_waiting = 0;
     part_number = part_number +1;
     next_part = order(part_number);
     if next_part ~= -1
         % inter_arrival_time = round(normrnd(MU,SIGMA));
         % inter_arrival_time = exprnd(MU);
         inter_arrival_time = MU;
         new_event = [8,-1, tnow+inter_arrival_time,-1];% new part
arrival
         event_stack = [event_stack;new_event];
     end
     end
   end % station == 10
   if (station == 2) % M2
      if (sys_state(2) == 2) % M2 finished process
         new_event = [3,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to OB
         event_stack = [event_stack;new_event];
      end
   end % station == 2
   if (station == 5)
      if (sys_state(5) == 2) % M3 finished process
         new_event = [3,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to OB
         event_stack = [event_stack;new_event];
      end
   end % station ==5
   if (station == 8)
      if (sys_state(8) == 2) % M1 finished process
         new_event = [3,target_station,
tnow+vacuum_time+trans_time(station,target_station),target_station];
% vauucm + taking part to OB
         event_stack = [event_stack;new_event];
      end
   end % station == 8
end     % event == 1
if (event ==2 ) % robot arrives with empty template to OB
    if (station == 3)
       sys_state(3) = 1; % update sys_state - OB  with template
       sys_state(1) = 0; % update sys_state - OB  with template
    end
    if (station == 6)
       sys_state(6) = 1; % update sys_state - OB  with template
      sys_state(4) = 0; % update sys_state - OB  with template
    end
    if (station == 9)
       sys_state(9) = 1; % update sys_state - OB  with template
       sys_state(7) = 0; % update sys_state - OB  with template
    end
 end % event == 2
 if (event ==3 ) % robot arrives with part to OB  or to machine
     if (station == 2) % inserting to M2
       sys_state(1) = 1; % update sys_state - IB  with template alone
       sys_state(2) = 1; % update sys_state - M2 proccessing
       new_event = [5, -1, tnow+proc_time(1),-1];% M2 finish
proccessing
       event_stack = [event_stack;new_event];
    end
    if (station == 3)  % arrive to M2_OB with part
       sys_state(3) = 2; % update sys_state - OB  with template+part
       sys_state(2) = 0; % M2 is free
     end
     if (station == 5) % inserting to M3
       sys_state(4) = 1; % update sys_state - IB  with template alone
       sys_state(5) = 1; % update sys_state - M3 proccessing
       new_event = [6, -1, tnow+proc_time(2),-1];% M3 finish
proccessing
       event_stack = [event_stack;new_event];
    end
     if (station == 6) % arrive to M3_OB with part
       sys_state(6) = 2; % update sys_state - OB  with template+part
       sys_state(5) = 0; % M3 is free
     end
     if (station == 8) % inserting to M1
       sys_state(7) = 1; % update sys_state - IB  with template alone
       sys_state(8) = 1; % update sys_state - M3 proccessing
       new_event = [7, -1, tnow+proc_time(3),-1];% M3 finish
proccessing
       event_stack = [event_stack;new_event];
    end
     if (station == 9) % arrive to M1_OB with part
       sys_state(9) = 2; % update sys_state - OB  with template+part
       sys_state(8) = 0; % M1 is free
     end
  end % event == 3
 if (event == 4 ) % robot arrives with tempale + part to IB or to
AGV_proccessed
    if (station ==  1)
       sys_state(1) = 2; % update sys_state - OB  with template+part
    end
    if (station == 4)
       sys_state(4) = 2; % update sys_state - OB  with template+part
    end
    if (station == 7)
       sys_state(7) = 2; % update sys_state - OB  with template+part
    end
    if (station == 11)
       processed_parts = processed_parts +1; % finished proccessing
one more part
       sys_state(target_station) = 0;
       if target_station == 9 % note part taken from M1_OB
          M1_part_taken = 0;
       end
       if processed_parts == size(order,1)-1
          break
        end
     end
 end % event == 4
  if (event == 5 ) % M2 finished processing
     sys_state(2) = 2; % M2 after process
     if (station ==  2)  % robot waiting for part
         new_event = [3,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to OB
         event_stack = [event_stack;new_event];
     end
end % event == 5
if (event == 6 ) % M3 finished processing
     sys_state(5) = 2; % M3 after process is free
     if (station ==  5)  % robot waiting for part
         new_event = [3,target_station,
tnow+trans_time(station,target_station),target_station];% taking part
to OB
         event_stack = [event_stack;new_event];
     end
end % event == 6
if (event == 7 ) % M1 finished processing
   sys_state(8) = 2; % M1 after process
     if (station ==  8)  % robot waiting for part
         new_event = [3,target_station,
tnow+vacuum_time+trans_time(station,target_station),target_station];
% vacuum + taking part to OB
         event_stack = [event_stack;new_event];
     end
end % event == 7
```

```
%------------------------------------------------------------------------

if (event ~= 1) % choose next task only at there events. not choosing
next event if robot is coming empty to take part
% M1:
if (sys_state(7) == 0) && (next_part == 2) && (part_waiting == 1) %
M1_IB free + next part is sign + part has arrived + there is no part
already headed to M3
   if isempty(task_queue) ==1 && prev_tasks(1) ~=9
      task_queue = [task_queue;9];
      prev_tasks(1)  = 9;
   else
      if isempty(task_queue) ~=1
         if bsearch(task_queue,9) == -1 && prev_tasks(1) ~=9
            task_queue = [task_queue;9]; % go and take part or wait for
it and then take to M1_IB
            prev_tasks(1)  = 9;
         end
      end
   end
end
if (sys_state(7) == 2) && (sys_state(8) == 0)  % M1_IB
template+part + M1 free
   next_M1_task = [next_M1_task;500];% go to template. take part
from template and insert to M1
end
if (sys_state(7) == 1) && (sys_state(9) == 0)  % M1_IB template +
M1_OB free
   next_M1_task = [next_M1_task;501]; % go to template. take
template from IB to OB
end
if (sys_state(8) == 2) && (sys_state(9) ==1)  % M1 after process +
M1_OB tempale
   next_M1_task = [next_M1_task;502]; % go to M1. vacuum. take
part to template
end
if isempty(next_M1_task) ~=1
   for i = 1:length(next_M1_task)
      if next_M1_task(i) ~= prev_tasks(1)
         task_queue = [task_queue;next_M1_task(i)];
         prev_tasks(1) = next_M1_task(i);
      end
   end % for
end % isempty
next_M1_task = [];
if (sys_state(9) == 2)  % M1_OB tempale+part (move tempalate +
part to AGV)
   %next_M1_task = [next_M1_task;6]; % go to M1_OB. take
template + part to AGV_proccessed
   if isempty(task_queue) ==1
      if M1_part_taken == 0
      task_queue = [task_queue;6];
      M1_part_taken = 1;
      end
   else
      if bsearch(task_queue,6) == -1 && M1_part_taken == 0
         task_queue = [task_queue;6]; % go and take part or wait for it
and then take to M1_IB
         M1_part_taken = 1;
     end
   end
end
% M2:
if (sys_state(1) == 0) && (next_part == 1)  && (part_waiting == 1)
&& (next_lathe==2)  % M2_IB free + next part is rook + and part has
arrived + there is no part already headed to M2
   if  last_lathe == 3 || last_lathe == -1
      if isempty(task_queue) ==1
         task_queue = [task_queue;10];
        last_lathe = 2;
      else
         if bsearch(task_queue,10) == -1
            task_queue = [task_queue;10]; % go and take part or wait
for it and then take to M2_IB
            last_lathe = 2;
         end
      end
   end
end
if (sys_state(1) == 2) && (sys_state(2) == 0)  % M2_IB
template+part + M2 free
   next_M2_task = 300;% go to template. take part from template and
insert to M2
end
if (sys_state(1) == 1) && (sys_state(3) == 0)  % M2_IB template +
M2_OB free
   next_M2_task = 301; % go to template. take template from IB to
OB
end
if (sys_state(2) == 2) && (sys_state(3) == 1)  % M2 after process +
M2_OB tempale
   next_M2_task = 302;% go to M2. take part to template
end
if (sys_state(3) ==2)  % M2_OB tempale+part (move tempalate + part
to AGV)
   next_M2_task =  7;  % go to M2_OB. take template + part to
AGV_proccessed
end
 if isempty(next_M2_task) ~=1
  if next_M2_task ~= prev_tasks(2)
     task_queue = [task_queue;next_M2_task];
      prev_tasks(2) = next_M2_task;
  end
end
next_M2_task = [];
% M3:
if (sys_state(4) == 0) && (next_part == 1)  && (part_waiting == 1)
&& (next_lathe==3)  % M3_IB free + next part is rook + part has
arrived + there is no part already headed to M3
   if  last_lathe == 2 || last_lathe == -1
      if isempty(task_queue) ==1
         task_queue = [task_queue;11];
         last_lathe = 3;
      else
         if bsearch(task_queue,11) == -1
              task_queue = [task_queue;11]; % go and take part or wait
for it and then take to M3_IB
              last_lathe = 3;
         end
      end
   end
end
if (sys_state(4) == 2) && (sys_state(5) == 0)  % M3_IB
template+part + M3 free
   next_M3_task = 400; % go to template. take part from template and
insert to M3
end
if (sys_state(4) == 1) && (sys_state(6) == 0)  % M3_IB template +
M3_OB free
   next_M3_task = 401; % go to template. take template from IB to
OB
end
if (sys_state(5) == 2) && (sys_state(6) == 1)  % M3 after process +
M3_OB tempale
   next_M3_task = 402; % go to M3. take part to template or wait for
it and then take to template
end
if (sys_state(6) == 2)  % M3_OB tempale+part (move tempalate +
part to AGV)
   next_M3_task = 8; % go to M2_OB. take template + part to
AGV_proccessed
end
if isempty(next_M3_task) ~=1
  if next_M3_task ~= prev_tasks(3)
     task_queue = [task_queue;next_M3_task];
      prev_tasks(3) = next_M3_task;
  end
end
next_M3_task = [];
%end % if M3

%------------------------------------------------------------------------------
------------------------------------

% updating event_stack according to next task
if isempty(task_queue) ~=1
next_task = task_queue(1);
if ((event~=5) && (event ~=6) && (event~=7) && (event~=8)) ||
(size(event_stack,1) == 1)
   if next_task == 10;
      event_stack = [event_stack; 1,10,tnow+trans_time(station,10),1];
% go and take part and then take to M2_IB
   end
   if next_task == 300;
      event_stack = [event_stack;
3,2,tnow+trans_time(station,1)+trans_time(1,2),2]; % go to template.
take part from template and insert to M2
   end
   if next_task == 301;
      event_stack = [event_stack;
2,3,tnow+trans_time(station,1)+trans_time(1,3),3]; % go to template.
take template from IB to OB
   end
   if next_task == 302;
      event_stack = [event_stack; 1,2,tnow+trans_time(station,2),3]; %
go to M2. take part to template
   end
   if next_task == 7;
      event_stack = [event_stack;
4,11,tnow+trans_time(station,3)+trans_time(3,11),3];% go to
M2_OB. take template + part to AGV_proccessed
      % here target station is origin station for robot_seq
   end
   if next_task == 11;
      event_stack = [event_stack; 1,10,tnow+trans_time(station,10),4];
% go and take part or wait for it and then take to M3_IB
   end
   if next_task == 400;
     event_stack = [event_stack;
3,5,tnow+trans_time(station,4)+trans_time(4,5),5]; % go to template.
take part from template and insert to M3
   end
   if next_task == 401;
      event_stack = [event_stack;
2,6,tnow+trans_time(station,4)+trans_time(4,6),6]; % go to template.
take template from IB to OB
   end
   if next_task == 402;
      event_stack = [event_stack; 1,5,tnow+trans_time(station,5),6]; %
go to M3. take part to template or wait for it and then take to template
   end
   if next_task == 8;
      event_stack = [event_stack;
4,11,tnow+trans_time(station,6)+trans_time(6,11),6]; % go to
M2_OB. take template + part to AGV_proccessed
      % here target station is origin station for robot_seq
   end
   if next_task == 9;
      event_stack = [event_stack; 1,10,tnow+trans_time(station,10),7];
% go and take part or wait for it and then take to M1_IB
   end
   if next_task == 500;
    event_stack = [event_stack;
3,8,tnow+trans_time(station,7)+trans_time(7,8),8]; % go to template.
take part from template and insert to M1
   end
   if next_task == 501;
     event_stack = [event_stack;
2,9,tnow+trans_time(station,7)+trans_time(7,9),9]; % go to template.
take template from IB to OB
   end
   if next_task == 502;
      event_stack = [event_stack; 1,8,tnow+trans_time(station,8),9]; %
go to M1. vacuum. take part to template or wait for it and then take to
template
   end
   if next_task == 6;
      event_stack = [event_stack;
4,11,tnow+trans_time(station,9)+trans_time(9,11),9]; % go to
M1_OB. take template + part to AGV_proccessed
     % here target station is origin station for robot_seq
   end
```

```
%   updating task_queue (deleting first task which have been
converted to event
   if size(task_queue,1) == 1
      task_queue = [];
   else
      task_queue = task_queue(2:end,:);
   end
end % if ((event~=5( && (event ~=6) && (event~=7))...
end %if isempty(task_queue) ~=1
end %if (event ~= 1)

%-------------------------------------------------------------------------

% updating system state  sequence
if   sum(prev_sys_state==sys_state) ~=9 % only if changed state
     seq = [sys_state,tnow; seq];
end
% removing event that was performed and sorting event_stack
event_stack=event_stack(2:1:end,:);
event_stack=sortrows(event_stack,3);
if size(event_stack,1) == 0
   event_stack = [-2,station, tnow, -2]; % fictive event if stack is
empty
end
end %while

%--------------------------------------------------------------------------------

% after "break" update seq with last move to AVG_processed
 sys_state = [0,0,0,0,0,0,0,0,0];
 seq = [sys_state,tnow; seq];
if tnow<min_tnow
   min_tnow = tnow;
   best_seq = seq;
   best_robot_seq = robot_seq;
end

%-------------------------------------------------------------------------

if iteration == 1
   tnow_array = tnow;
else
   tnow_array = [tnow_array;tnow];
end

end%case
```

FMS.m

```
function FMS(arg,arg2,arg3,arg4,arg5)

global Q
global tnow_array
global iteration
global min_tnow
global seq
global robot_seq
global best_seq
global best_robot_seq

% ------------------------------------------------------------------------

switch( arg)

% ------------------------------------------------------------------------

case 0

Q = ones(3,3,3,3,3,3,3,3,3);
min_tnow = 100000000; %just a big number
iterations = 1;
for iteration=1:iterations
   FMS(1,0,0,arg4,arg5);
end

% ------------------------------------------------------------------------

case 1

global enable_graphics
global tnow
global sys_state
global proc_time
global order
global processed_parts
global part_waiting
global station
global origin_station
global next_part
global vacuum_time
global Gamma % algorithm parameter
global Alpha % algorithm parameter
global trans_time
global event_stack
global SIGMA %for normal arrival times
global MU %for exponential inter arrival times
global seq_ind
global next_station

vacuum_time = 240; % the vacuum time will be regarded as part of
the trans time, to make sure the robot is not free for the whole time
trans_time= zeros(11);
trans_time(1,2) = 65; %
trans_time(1,3) = 15; %
trans_time(1,4) = 45; %
trans_time(1,5) = 45; %
trans_time(1,6) = 45; %
trans_time(1,7) = 25; %
trans_time(1,8) = 25; %
trans_time(1,9) = 25; %
trans_time(1,10) = 40; %
% trans_time(1,11) = 50; % N/A
trans_time(2,1) = 25; %
trans_time(2,3) = 20; %
trans_time(2,4) = 45; %
trans_time(2,5) = 45; %
trans_time(2,6) = 45; %
trans_time(2,7) = 25; %
trans_time(2,8) = 25; %
trans_time(2,9) = 25; %
trans_time(2,10) = 40; %
% trans_time(2,11) = 50; % N/A
trans_time(3,1) = 25; %
trans_time(3,2) = 45; %
trans_time(3,4) = 45; %
trans_time(3,5) = 45; %
trans_time(3,6) = 45; %
trans_time(3,7) = 25; %
trans_time(3,8) = 25; %
trans_time(3,9) = 25; %
trans_time(3,10) = 40; %
trans_time(3,11) = 40; %
 trans_time(4,1) = 45; %
trans_time(4,2) = 45; %
trans_time(4,3) = 45; %
trans_time(4,5) = 65; %
trans_time(4,6) = 15; %
trans_time(4,7) = 50; %
trans_time(4,8) = 50; %
trans_time(4,9) = 50; %
trans_time(4,10) = 60; %
% trans_time(4,11) = 50; % N/A
trans_time(5,1) = 45; %
trans_time(5,2) = 45; %
trans_time(5,3) = 45; %
trans_time(5,4) = 25; %
trans_time(5,6) = 20; %
trans_time(5,7) = 50; %
trans_time(5,8) = 50; %
trans_time(5,9) = 50; %
trans_time(5,10) = 60; %
% trans_time(5,11) = 50; % N/A
 trans_time(6,1) = 45; %
trans_time(6,2) = 45; %
trans_time(6,3) = 45; %
trans_time(6,4) = 25; %
trans_time(6,5) = 45; %
trans_time(6,7) = 50; %
trans_time(6,8) = 50; %
trans_time(6,9) = 50; %
trans_time(6,10) = 60; %
trans_time(6,11) = 60; %
trans_time(7,1) = 25; %
trans_time(7,2) = 25; %
trans_time(7,3) = 25; %
trans_time(7,4) = 50; %
trans_time(7,5) = 50; %
trans_time(7,6) = 50; %
trans_time(7,8) = 50; %
trans_time(7,9) = 15; %
trans_time(7,10) = 30; %
% % trans_time(7,11) = 50; %N/A
trans_time(8,1) = 25; %
trans_time(8,2) = 25; %
trans_time(8,3) = 25; %
trans_time(8,4) = 50; %
trans_time(8,5) = 50; %
trans_time(8,6) = 50; %
trans_time(8,7) = 25; %
trans_time(8,9) = 25; %
trans_time(8,10) = 30; %
% trans_time(8,11) = 50; %N/A
trans_time(9,1) = 25; %
trans_time(9,2) = 25; %
trans_time(9,3) = 25; %
trans_time(9,4) = 50; %
trans_time(9,5) = 50; %
trans_time(9,6) = 50; %
trans_time(9,7) = 25; %
trans_time(9,8) = 25; %
trans_time(9,10) = 30; %
trans_time(9,11) = 30; %
trans_time(10,1) = 40; %
trans_time(10,4) = 60; %
trans_time(10,7) = 30; %
trans_time(11,1) = 40; %
trans_time(11,2) = 40; %
trans_time(11,3) = 40; %
trans_time(11,4) = 60; %
trans_time(11,5) = 60; %
trans_time(11,6) = 60; %
trans_time(11,7) = 30; %
trans_time(11,8) = 30; %
trans_time(11,9) = 30; %
trans_time(11,10) = 20; %
% M1 - Mill, M2 - Lathe 1,  M3 - LAthe 2
proc_time = zeros(1,3);
proc_time(1) = 185; %M2
proc_time(2) = 185; %M3
proc_time(3) = 600; %M1
enable_graphics = arg2;
Gamma = 0.9;
Alpha = 0.05;
%order = [1;1;2;1;2]; % 1 - ROOK, 2 - SIGN
order = arg4;
order = [order;-1]; %mark order is finished with -1 (end of list)
part_number = 1; % starting with part in the head of the batch
next_part = order(part_number);
sys_state = zeros(1,9);
processed_parts = 0;
tnow = 0;
seq = [sys_state,tnow];
robot_seq = [10,tnow];
MU = 100;
SIGMA = MU/10;

%-------------------------------------------------------------------------

%Stations:
% 1 - M2_IB
% 2 - M2
% 3 - M2_OB
% 4 - M3_IB
% 5 - M3
% 6 - M3_OB
% 7 - M1_IB
% 8 - M1
% 9 - M1_OB
% 10 - AGV_raw
% 11 - AGV_proccessed
% event structure:
% [event,station,time, target_station]
%events:
% 1: robot arrive empty
% 2: robot arrive with template
% 3: robot arrive with part
% 4: robot arrive with template +part
% 5:M2 finish proccessing
% 6:M3 finish proccessing
% 7:M1 finish proccessing
% 8: part arriveal
% system states:
% [M2_IB, M2, M2_OB, M3_IB, M3, M3_OB, M1_IB, M1,
M1_OB]
%Machine: 0-free, 1-Working, 2-full after process
%Buffer:0-free, 1-template, 2-template+part
% arrival from store:
% next_part = 1 %rook
% next_part = 2 %sign
% next_part = 0 %no part arrived
% next_part = -1 %end of order
%station = robot state

%-------------------------------------------------------------------------

%init

% inter_arrival_time = round(normrnd(MU,SIGMA));
% inter_arrival_time = exprnd(MU);
inter_arrival_time = MU;
event_stack = [8,-1, tnow+inter_arrival_time,-1];
station = 10; % robot starts at AGV _raw
sequence = arg5;
seq_ind= size(sequence,1);
seq_ind = seq_ind-1;
next_station = sequence(seq_ind);

%-------------------------------------------------------------------------

while 0<1 % endless loop. will end with break
prev_sys_state = sys_state;

event_arr = event_stack(1,:);
event = event_arr(1);
tnow = event_arr(3);

% '============================='
% sys_state
% event_stack

if ((event==1) || (event==2) || (event ==3) || (event==4)) % only robot
moving events change the station

    station = event_arr(2);
    origin_station = event_arr(4);

    if seq_ind>1
    seq_ind = seq_ind-1;
    next_station = sequence(seq_ind);
    while next_station == station % remove redundancies in seq
        seq_ind = seq_ind-1;
        next_station = sequence(seq_ind);
    end
    end
% next_station
end
if (event == 8) % part arrives
    part_waiting = 1;
    if (station == 10) %robot waiting for it
```

```
        new_event = [4,next_station,
tnow+trans_time(station,next_station),next_station]; % % take
part+template to M3_IB
                event_stack = [event_stack;new_event];
        part_waiting = 0;
        part_number = part_number +1;
       next_part = order(part_number);
       if next_part ~= -1
          % inter_arrival_time = round(normrnd(MU,SIGMA));
          % inter_arrival_time = exprnd(MU);
          inter_arrival_time = MU;
         new_event = [8,-1, tnow+inter_arrival_time,-1];% new part
arrival
         event_stack = [event_stack;new_event];
       end
    end
end % event == 8
if(event == 1) % robot arrives empty
    if (station == 10) %came to take part
        if (part_waiting == 1) %part waiting
                new_event = [4,next_station,
tnow+trans_time(station,next_station),next_station]; % % take
part+template to M3_IB
                        event_stack = [event_stack;new_event];
        part_waiting = 0;
           part_number = part_number +1;
           next_part = order(part_number);
          if next_part ~= -1
           % inter_arrival_time = round(normrnd(MU,SIGMA));
           % inter_arrival_time = exprnd(MU);
         inter_arrival_time = MU;
            new_event = [8,-1, tnow+inter_arrival_time,-1];% new part
arrival
            event_stack = [event_stack;new_event];
         end
       end
    end % station == 10
    if (station == 2) % M2
        if (sys_state(2) == 2) % M2 finished process
           new_event = [3,next_station,
tnow+trans_time(station,next_station),next_station];% taking part to
OB
           event_stack = [event_stack;new_event];
           end
    end % station == 2
    if (station == 5)
        if (sys_state(5) == 2) % M3 finished process
           new_event = [3,next_station,
tnow+trans_time(station,next_station),next_station];% taking part to
OB
           event_stack = [event_stack;new_event];
        end
    end % station == 5
    if (station == 8)
        if (sys_state(8) == 2) % M1 finished process
           new_event = [3,next_station,
tnow+vacuum_time+trans_time(station,next_station),next_station];%
vacuum + taking part to OB
           event_stack = [event_stack;new_event];
        end
    end % station == 8
 end    % event == 1
 if (event ==2 ) % robot arrives with empty template to OB
    if (station == 3)
        sys_state(3) = 1; % update sys_state - OB  with template
        sys_state(1) = 0; % update sys_state - OB  with template
    end
    if (station == 6)
        sys_state(6) = 1; % update sys_state - OB  with template
       sys_state(4) = 0; % update sys_state - OB  with template
    end
    if (station == 9)
        sys_state(9) = 1; % update sys_state - OB  with template
        sys_state(7) = 0; % update sys_state - OB  with template
     end
 end % event == 2
 if (event ==3 ) % robot arrives with part to OB  or to machine
     if (station == 2) % inserting to M2
        sys_state(1) = 1; % update sys_state - IB  with template alone
        sys_state(2) = 1; % update sys_state - M2 proccessing
        new_event = [5, -1, tnow+proc_time(1),-1]; % M2 finish
proccessing
        event_stack = [event_stack;new_event];
     end
    if (station == 3)  % arrive to M2_OB with part
        sys_state(3) = 2; % update sys_state - OB  with template+part
        sys_state(2) = 0; % M2 is free
      end
      if (station == 5) % inserting to M3
        sys_state(4) = 1; % update sys_state - IB  with template alone
        sys_state(5) = 1; % update sys_state - M3 proccessing
        new_event = [6, -1, tnow+proc_time(2),-1];% M3 finish
proccessing
        event_stack = [event_stack;new_event];
      end
      if (station == 6) % arrive to M3_OB with part
        sys_state(6) = 2; % update sys_state - OB  with template+part
        sys_state(5) = 0; % M3 is free
       end
       if (station == 8) % inserting to M1
        sys_state(7) = 1; % update sys_state - IB  with template alone
        sys_state(8) = 1; % update sys_state - M3 proccessing
        new_event = [7, -1, tnow+proc_time(3),-1];% M3 finish
proccessing
        event_stack = [event_stack;new_event];
       end
        if (station == 9) % arrive to M1_OB with part
          sys_state(9) = 2; % update sys_state - OB  with template+part
          sys_state(8) = 0; % M1 is free
       end
  end % event == 3
 if (event == 4 ) % robot arrives with tempale + part to IB or to
AGV_proccessed
      if (station ==  1)
         sys_state(1) = 2; % update sys_state - OB  with template+part
      end
      if (station == 4)
         sys_state(4) = 2; % update sys_state - OB  with template+part
      end
      if (station == 7)
         sys_state(7) = 2; % update sys_state - OB  with template+part
      end
        if (station == 11)
         processed_parts = processed_parts +1; % finished proccessing
one more part
         sys_state(origin_station) = 0;
        if processed_parts == size(order,1)-1
           break
        end
     end
 end % event == 4
 if (event == 5 ) % M2 finished processing
        sys_state(2) = 2; % M2 after process
       if (station ==  2)  % robot waiting for part
           new_event = [3,next_station,
tnow+trans_time(station,next_station),next_station];% taking part to
OB
           event_stack = [event_stack;new_event];
      end
end % event == 5
if (event == 6 ) % M3 finished processing
            sys_state(5) = 2; % M3 after process is free
      if (station ==  5)  % robot waiting for part
           new_event = [3,next_station,
tnow+trans_time(station,next_station),next_station];% taking part to
OB
           event_stack = [event_stack;new_event];
      end
end % event == 6
if (event == 7 ) % M1 finished processing
   sys_state(8) = 2; % M1 after process
      if (station ==  8)  % robot waiting for part
           new_event = [3,next_station,
tnow+vacuum_time+trans_time(station,next_station),next_station];%
vacuum + taking part to OB
           event_stack = [event_stack;new_event];
    end
end % event == 7

% ------------------------------------------------------------------------

if ( (event == 1) || (event == 2)  || (event==3)  || (event==4) )  %
choose next state only at there events
% not choosing next event if robot is waiting for part to finish
processing....
new_event = [];
% M2 Stations
if next_station == 1
   if station == 10 %&& part_waiting == 1 % at AGV_raw and part is
waiting there
   else
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot goes
empty to M2_IB
   end
end
if next_station == 2
   if station == 1 && sys_state(1) == 2  && sys_state(2) ==0 %
template+part at M2_IB and M2 free
       new_event = [3,next_station,
tnow+trans_time(station,next_station),next_station]; % insert part to
M2
   else
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
empty to take part out from M2
   end
end
if next_station == 3
   if  station == 1 && sys_state(1) == 1  % at M2_IB and template
there
       new_event = [2,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
with template to M2_OB
   elseif station == 2 && sys_state(2) == 2 % at M2 and it finished
process
 %       new_event = [3,next_station,
tnow+trans_time(station,next_station),next_station];% take finished
part from M2 to M2_OB
   elseif station == 2 && sys_state(3) == 2 % going to M2_OB to
take part+template to finish
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
empty to take part+template from M2_OB
   elseif station ~=2
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
empty to take part+template from M2_OB
   end
end
% M3 Stations
if next_station == 4
   if station == 10 % && part_waiting == 1 % at AGV_raw and part
is waiting there
   else
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot goes
empty to M3_IB
   end
end
if next_station == 5
   if station == 4 && (sys_state(4) == 2)  && sys_state(5) ==0 %
template+part at M3_IB and M3 free
       new_event = [3,next_station,
tnow+trans_time(station,next_station),next_station]; % insert part to
M3
   else
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
empty to take part out from M2
   end
end
if next_station == 6
   if  station == 4 && (sys_state(4) == 1)  % at M3_IB and template
there
       new_event = [2,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
with template to M3_OB
   elseif station == 5 && sys_state(5) == 2 % at M2 and it finished
process
   elseif station == 5 && sys_state(6) == 2 % going to M3_OB to
take part+template to finish
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
empty to take part+template from M3_OB
   elseif station~=5
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
empty to take part+template from M3_OB
   end
end
% M1 Stations
if next_station == 7
   if station == 10 %&& part_waiting == 1 % at AGV_raw and part is
waiting there
   else
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot goes
empty to M1_IB
   end
end
if next_station == 8
   if station == 7 && (sys_state(7) == 2)  && sys_state(8) ==0 %
template+part at M1_IB and M1 free
       new_event = [3,next_station,
tnow+trans_time(station,next_station),next_station]; % insert part to
M1
   else
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
empty to take part out from M1
   end
end
if next_station == 9
   if  station == 7 && (sys_state(7) == 1)  % at M1_IB and template
there
       new_event = [2,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
with template to M1_OB
   elseif station == 8 && sys_state(8) == 2 % at M1 and it finished
process
   elseif station == 8 && sys_state(9) == 2 % going to M1_OB to
take part+template to finish
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
empty to take part+template from M1_OB
   elseif station~=8
       new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
empty to take part+template from M1_OB
   end
end
if next_station == 10 % going to bring new part
   new_event = [1,next_station,
tnow+trans_time(station,next_station),next_station]; % robot comes
to take part form AGV_raw
end
if next_station == 11 % taking template + finished part to
AGV_processed
   new_event = [4,next_station,
tnow+trans_time(station,next_station),station]; % robot comes to
AGV_processed with template+part
end
event_stack = [event_stack;new_event];
end %if ( (event == 2) || (event==3)  || (event==4) )  % choose next
state

% ------------------------------------------------------------------------------
--------

if   sum(prev_sys_state==sys_state) ~=9 % only if changed state
```

```
   % updating position sequence
   seq = [sys_state,tnow; seq];
end
% removing event that was performed and sorting event_stack
new_event = [];
event_stack=event_stack(2:1:end,:);
 event_stack=sortrows(event_stack,3);
end %while

% -------------------------------------------------------------------------

% after "break" update seq with last move to AVG_processed
tnow = tnow+trans_time(station,11);
sys_state = zeros(1,9);
seq = [sys_state,tnow; seq];
% update best_seq, best_robot_seq
 if tnow<min_tnow
    min_tnow = tnow;
    best_seq = seq;
    best_robot_seq = robot_seq;
 end
% update tnow_array (for chart)
if iteration == 1
   tnow_array = tnow;
else
   tnow_array = [tnow_array;tnow];
end

end%case
```

תקציר

על מנת שמערכות רובוטיות יוכלו להשתלב בסביבות עבודה משתנות בעולם האמיתי, קיים הכרח שיתוכננו בצורה שתאפשר להן להתמודד עם מגוון רחב של משימות ותנאי עבודה חדשים ומשתנים. משום שלא ניתן למדל כל סביבה או תנאי למידה, יש להעניק לרובוטים את היכולת ללמוד ולהסתגל בצורה עצמאית.

אחת הגישות ללמידה, אשר יכולה לתת מענה לחלק מהנושאים שתוארו, נקראת שיטת החיזוקים (Reinforcement Learning). בשיטה זו הרובוט מונחה על ידי קבלת חיזוקים מן הסביבה בה הוא מתפקד. חיזוקים אלו מספקים לרובוט אינדיקציה לגבי רמת התפקוד שלו בעת ביצוע משימה נתונה. שיטת החיזוקים מהווה אלטרנטיבה טובה לתכנות מערכות אוטונומיות, משום שהיא מאפשרת למידה על בסיס משובים מועטים מהסביבה. אולם, למרות שלשיטה יתרונות רבים, ונעשה בה שימוש נרחב עבור יישומים רובוטיים, היא כוללת גם מספר חסרונות המונעים ממנה לתת מענה ראוי לאתגרים המוצגים על-ידי יישומים מעשיים, כגון הצורך באינטראקציה נרחבת עם הסביבה או העובדה שהיא מתקשה להתמודד עם משימות מורכבות.

עבודה זו מהווה נדבך נוסף ב"מאבק" המתמשך להתגבר על חסרונות אלה. הגישה המוצעת, למידה היררכית משולבת אדם בשיטת החיזוקים, משלבת שתי שיטות מוכרות, למידת חיזוקים היררכית ושיתוף פעולה אדם-רובוט, לשם שדרוג הלמידה. הגישה מאפשרת ביצוע משימות מורכבות ושיפור של תהליך הלמידה, על-ידי פירוק המשימה לשתי רמות של היררכיה. רמה ראשונה בה מתבצעת בנייה של רצף הפעולות הנדרש לביצוע המשימה הכוללת, ורמה שנייה בה מתבצעת למידה של ביצוע הפעולות עצמן. החלוקה לשתי רמות ההיררכיה מקטינה את מרחב החיפוש ומאפשרת למידה אפקטיבית. בשתי הרמות מתאפשר שילוב אדם בתהליך הלמידה, בכדי להאיצו ולקדמו על-ידי שימוש ביכולות האדם ובנסיונו. בכדי להוכיח את מעשיותה, הגישה יושמה לתפעול מערכת רובוטית לייצור טוסטים, המציבה משימות למידה בשתי רמות ההיררכיה.

שני אלגוריתמים מבוססי שיטת החיזוקים פותחו בכדי לתמוך בגישה המוצעת: אלגוריתם זימון פותח בכדי לספק רצף פעולות אופטימלי לביצוע משימה מורכבת, כחלק מהרמה הראשונה של ההיררכיה. האלגוריתם נבחן על-ידי יישומו במערכת ייצור הטוסטים שהוזכרה ובמערכת ייצור גמישה נוספת, והציג תוצאות טובות.

אלגוריתם מבוסס מודל קוגניטיבי פותח בכדי לאפשר שילוב אדם בתהליך הלמידה. לרובוט מוענקות היכולת להחליט מתי לבקש עזרה, בהתבסס על מודעות לרמת הביצועים שלו, והיכולת להחליט לדחות את העזרה המוצעת, אם הוא מזהה כי אינה תורמת לתהליך הלמידה. גישה זו של אוטונומיה גמישה נבחנה על-ידי יישומה במערכת ייצור הטוסטים, וכן על-ידי משימת ניווט תלת-מימדית, עבורן דומו רמות שונות של יועצים. האלגוריתם שיפר והאיץ את תהליך הלמידה על-ידי שימוש בידע של יועצים טובים, ולמד להתעלם מעצות שהתקבלו מיועצים טובים פחות.

תרומתו העיקרית של מחקר זה הינה בהצגת גישת הלמידה ההיררכית משולבת האדם בשיטת החיזוקים, ובפיתוח האלגוריתם התומכים במימושה.

מילות מפתח: שיטת החיזוקים, למידה היררכית, למידה רובוטית, שיתוף פעולה אדם-רובוט, זימון.

העבודה בוצעה בהדרכת

פרופ׳ הלמן שטרן

פרופ׳ יעל אידן

במחלקה להנדסת תעשיה וניהול

הפקולטה למדעי ההנדסה

אוניברסיטת בן-גוריון בנגב

אוניברסיטת בן-גוריון בנגב
הפקולטה למדעי ההנדסה
המחלקה להנדסת תעשייה וניהול

מסגרת למידה היררכית משולבת אדם בשיטת החיזוקים

חיבור זה מהווה חלק מהדרישות לקבלת תואר מגיסטר בהנדסה

מאת: עמית גיל
מנחים: הלמן שטרן
יעל אידן

חתימת המחבר	___________	תאריך	_________
אישור מנחה	___________	תאריך	_________
אישור מנחה	___________	תאריך	_________
אישור יו״ר ועדת הוראה מחלקתית תואר שני	________	תאריך	_________

תשס״ח 2008

באר-שבע

אוניברסיטת בן-גוריון בנגב
הפקולטה למדעי ההנדסה
המחלקה להנדסת תעשייה וניהול

מסגרת למידה היררכית משולבת אדם בשיטת החיזוקים

חיבור זה מהווה חלק מהדרישות לקבלת תואר מגיסטר בהנדסה

מאת
עמית גיל

תשס״ח 2008

באר-שבע