SIMPLE AND COMPLEX BEHAVIOR LEARNING USING BEHAVIOR HIDDEN MARKOV MODEL AND COBART

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SEYİT SABRİ SEYHAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JANUARY 2013

Approval of the thesis:

SIMPLE AND COMPLEX BEHAVIOR LEARNING USING BEHAVIOR HIDDEN MARKOV MODEL AND COBART

submitted by **SEYİT SABRİ SEYHAN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Prof. Dr. Ferda Nur Alpaslan
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Varol Akman
Computer Engineering Department, Bilkent University _____

Prof. Dr. Ferda Nur Alpaslan
Computer Engineering Department, METU _____

Prof. Dr. Nihan Kesim Çiçekli
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Tolga Can
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Pınar Şenkul
Computer Engineering Department, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: SEYİT SABRİ SEYHAN

Signature :

ABSTRACT

SIMPLE AND COMPLEX BEHAVIOR LEARNING USING BEHAVIOR HIDDEN MARKOV MODEL AND COBART

Seyhan, Seyit Sabri
Ph.D., Department of Computer Engineering
Supervisor : Prof. Dr. Ferda Nur Alpaslan

January 2013, 85 pages

In this thesis, behavior learning and generation models are proposed for simple and complex behaviors of robots using unsupervised learning methods. Simple behaviors are modeled by simple-behavior learning model (SBLM) and complex behaviors are modeled by complex-behavior learning model (CBLM) which uses previously learned simple or complex behaviors. Both models have common phases named behavior categorization, behavior modeling, and behavior generation. Sensory data are categorized using correlation based adaptive resonance theory network that generates motion primitives corresponding to robot's base abilities in the categorization phase. In the modeling phase, Behavior-HMM, a modified version of hidden Markov model, is used to model the relationships among the motion primitives in a finite state stochastic network. In addition, a motion generator which is an artificial neural network is trained for each motion primitive to learn essential robot motor commands. In the generation phase, desired task is presented as a target observation and the model generates corresponding motion primitive sequence. Then, these motion primitives are executed successively by the motion generators which are specifically trained for the corresponding motion primitives.

The models are not proposed for one specific behavior, but are intended to be bases for all behaviors. CBLM enhances learning capabilities by integrating previously learned behaviors hierarchically. Hence, new behaviors can take advantage of already discovered behaviors. The proposed models are tested on a robot simulator and the experiments showed that simple and complex-behavior learning models can generate requested behaviors effectively.

Keywords: Robot behavior learning, hidden Markov model, artificial neural network, CobART

ÖZ

SAKLI MARKOV MODEL VE COBART KULLANIMI İLE BASİT VE KARMAŞIK DAVRANIŞLARI ÖĞRENME

Seyhan, Seyit Sabri
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Ferda Nur Alpaslan

Ocak 2013 , 85 sayfa

Bu tez kapsamında, basit ve karmaşık robot davranışlarını gözetmen ihtiyacı duymayan yöntemlerle öğrenen ve tekrarlayabilen davranış modelleri tasarlandı. Basit davranışlar basit-davranış öğrenme modeli (SBLM) ile karmaşık davranışlar ise daha önce öğrenilen basit ve karmaşık davranışları kullanabilen karmaşık-davranış öğrenme modeli (CBLM) ile modellendi. Her iki model de davranış sınıflandırma, davranış modelleme ve davranış üretme aşamalarından oluşmaktadır. Sınıflandırma aşamasında algılayıcılardan elde edilen veriler robotun temel yeteneklerine karşılık gelen temel hareketleri elde etmek amacıyla ilinti temelli uyarlanır rezonans kuramı kullanılarak sınıflandırılır. Modelleme aşamasında, saklı Markov modelinin değiştirilmiş bir şekli olan davranış-HMM kullanılarak temel hareketler arasındaki ilişki sonlu durumlu olasılıksal ağ biçiminde modellenir. Davranış modeline ilaveten robot motor komutlarını öğrenmek amacıyla her bir temel davranış için yapay sinir ağlarını kullanan bir hareket üretici eğitilir. Davranış üretme aşamasında, istenen görev amaç gözlem şeklinde sunulur ve model karşılık gelen temel hareket dizisini üretir. Sonra herbir temel hareket daha önce bu amaçla eğitilmiş hareket üreticiler aracılığıyla sırasıyla işletilir.

Modeller tek bir davranış için değil bütün davranışlar için bir dayanak oluşturması açısından tasarlandı. CBLM daha önce öğrenilen davranışları sıradüzensel olarak birleştirerek öğrenme yeteneğinin gelişmesini sağlar. Böylece yeni davranışlar daha önce öğrenilmiş davranışları kullanma yeteneğine sahip olur. Önerilen modeller robot benzeştiriciler üzerinde denendi ve deneme sonuçlarına göre basit ve karmaşık davranış modelleri istenen davranışları etkin bir biçimde tekrar üretmeyi başardı.

Anahtar Kelimeler: Robot davranışını öğrenme, saklı Markov model, yapay nöron ağları, CobART

To all members of my family

ACKNOWLEDGMENTS

I express sincere appreciations to my thesis advisor Prof. Dr. Ferda Nur Alpaslan for her guidance and supportive academic advisory throughout the thesis. I would like to thank the other members of my thesis committee: Prof. Dr. Varol Akman and Assoc. Prof. Tolga Can, for their valuable supports and feedbacks that helped to finalize my dissertation.

I owe a special thank to my dear friend Mustafa Yavaş for his academic and personal support through my undergraduate, graduate and Ph.D years. This thesis also built on his previous work. Also, thank to my friend Zafer Ataser for his support and encouragements to finish this thesis.

Many thanks to Environmental Tectonics Corporation for their tolerance and patience on my working hours.

I would like to give my very special thanks to my family. My deepest gratitude to my mother and father for being wonderful parents and their support through my education life. My sweet children: Gülşah and Akif, I would like to thank you for always making me smile and for your patience when I was studying instead of playing games with you. Finally, I want to thank to my deepest love, my wife Hatice for her never-ending encouragement and constant love. She took the whole responsibilities of our family during my study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Although there is no common definition of a robot, in dictionary terms, a robot is defined as *"a machine capable of carrying out a complex series of actions automatically"*. Robots can be autonomous, semi-autonomous, or remotely controlled. In this study, we focused on autonomous robots which can perform desired tasks without explicit human control. Today, autonomous robots can perform more complex tasks using advanced sensory-motor functions. In order to improve their capabilities, we need better learning methods to model a robot behavior.

It is necessary to define and inspect the meaning of a behavior before developing a robot behavior learning model. There are various definitions and types of behavior in the literature. Behavior is defined in the dictionary as *"actions or reactions of a person, animal, or artificial entity in response to external or internal stimuli"*. As a reaction to a stimulus, behavior is a process changing the environment or the agent itself from one state to another. Robot behaviors are defined as a time-extended action sequences achieving specified tasks [1] in the scope of this study. Behaviors which have one basic and specific goal such as *turn, approach, and grasp an object* are considered simple behaviors. On the other hand, behaviors having a complex goal that can be segmented into several basic goals are considered complex behaviors. Thus, a complex behavior may consist of several simple or complex behaviors. For example, in order to approach an object having a rotational angle difference, robot needs to perform turn and approach behaviors.

In robotics, there are two main approaches to control robots. The first one is programmer's perspective which controls the robots by preprogramming the behaviors and the second one is robot's perspective in which robots learn behaviors with or without the help of a human tutor. In this study, the second approach is followed. The motivation of this thesis arises from the question of how to learn and perform robot behaviors using observation data without the help of a human tutor. Unsupervised learning is selected to let robots to discover behaviors based on their capabilities without the supervisor's perception of the world. The major contribution of this thesis is to develop an extendible, generic and self-organizing behavior learning model. The model should reuse the already learned behaviors to learn more complex behaviors and new behaviors could be integrated with the model easily.

This thesis consists of two behavior learning models developed for simple and complex behaviors. Simple-behavior learning model (SBLM) is designed to learn simple behaviors and complex-behavior learning model (CBLM) is designed for complex behaviors. CBLM integrates already learned models hierarchically. *Behavior categorization, behavior modeling*, and *behavior generation* are the common phases of both models.

A behavior learning model should extract the unique and repeated parts of a behavior. These parts are considered as meaningful components of a behavior. In the previous studies, Yavas and Alpaslan [2][3]

proposed correlation based adaptive resonance theory (CobART) for the categorization of behaviors using observation data. This study uses CobART for the categorization of meaningful components of a behavior. The components generated by CobART are called *motion primitives*. There are some advantages to learn a behavior based on the motion primitives. The model can learn basic components more easily and correctly. Furthermore, the model can use previously learned components to avoid retraining [4].

Once the categorization of motion primitives completed, relations among them should be constructed. In the behavior modeling phase, relations and transition probabilities among the motion primitives are learned in a higher level model. As mentioned before, a behavior is a collection of action sequences. Thus, it is assumed that action sequences (i.e. motion primitives) can be modeled by a finite state stochastic network. Hidden Markov model (HMM) [5] is a tool to model generative sequences where the modeled system is a Markov process with unobserved states. The regular HMM is changed to model the relations between the motion primitives. This adopted version of HMM is named as Behavior-HMM throughout the study [4].

Motion primitive transition model constructed as Behavior-HMM helps robots to generate motion primitive sequences in order to achieve a given task. On the other hand, how to execute each motion primitive during runtime is not clear yet. For this purpose, a motion generator is used to learn generation of motor commands for each motion primitive. Motion generators use artificial neural networks (ANN) to learn the generation of motor commands. ANNs can learn the complex relations between the real-valued inputs and outputs.

Behavior generation phase generates motion primitive sequence to achieve the desired goal. It uses most likely path and Viterbi [6] algorithms to generate the most likely sequence of motion primitives. Then, the motion primitives in the sequence is executed one by one until the desired goal is achieved.

Modeling of complex behaviors need a different approach than simple behaviors. Applying simple-behavior learning model on complex behaviors causes a very complex behavior model that has repeated components. Different complex behaviors may contain common components and they can be modeled hierarchically to enhance the learning capabilities. CBLM is developed to manage the complexity of hierarchically integrated behavior models. It exploits the previously learned simple or complex behaviors as the parts of the new complex behavior [4].

Robot behavior learning is an active area of research as explained in Chapter 2. Many different frameworks using different types of neural networks and HMMs are proposed for behavior learning. This study introduces a new approach to learn simple and the complex behaviors in an extendible, unsupervised and self-organizing manner. Hierarchical structure of the model lets the use of already learned behaviors to generate more complex behaviors. The use of already learned behaviors reduces the training time and improves the quality of the learned behavior.

The model capabilities are tested on *Khepera* [7] robot using *Webots* [8] simulator. Simple-behavior model is experimented on *turn* and *approach* behaviors. Complex-behavior model is tested on the combination of previously learned *turn* and *approach* behaviors.

1.1 Contributions

The contributions of this thesis are listed below:

- This study contributes to other studies by the ability of learning of simple and complex behaviors effectively in an incremental manner.
- Robots learn behaviors without the help of a supervisor. Thus, robots are not restricted with the supervisor's perception of the world and they can discover behaviors based on their capabilities.
- New behaviors can be added to the model hierarchically. Integration of already learned behaviors improves the behavior generation performances and reduces the computational times of training.
- The model is extendible and self-organizing. Adding new behaviors to the model does not disrupt already learned behaviors. Furthermore, there is no need to retrain the old behaviors.
- The model can run on the continuous domain. It is not restricted with the range of the values inside the training data.

Followings are the differences from the previous studies of Yavas and Alpaslan [9][2][3]:

- Previous studies focused on behavior recognition and categorization process. This study extends the problem scope from behavior categorization to behavior learning.
- In the previous studies, CobART is used to recognize the different behaviors. This study uses CobART to extract motion primitives as meaningful parts of a single behavior.
- Simple and complex behavior models, Behavior-HMM, motion generators using ANNs and behavior generation algorithms (most likely path and Viterbi) are used only in this study.
- In [3], hierarchically integrated CobART networks are used to develop a better behavior categorization. In this study, already learned behavior models are integrated hierarchically to model the complex behaviors.

1.2 Outline

Outline of the remaining sections are organized as follows:

- **Chapter 2: Related Work** presents a review of the relevant previous work in this area.
- **Chapter 3: Background** gives background information about the topics discussed in the thesis.
- **Chapter 4: Simple Behavior Learning Model** explains behavior learning model for simple behaviors in detail.
- **Chapter 5: Complex Behavior Learning Model** explains behavior learning model for complex behaviors in detail.
- **Chapter 6: Experimental Evaluation** reports experiments and results obtained during the models evaluation.
- **Chapter 7: Discussion & Conclusion** discusses the experiment results and presents a summary of the work described in this thesis and gives information about the future work.

- **Appendix A: More Hidden Layer/Unit Experiments on Neural Network Training** reports more experiments about the hidden layer/unit effect on neural network training for two-input networks.
- **Appendix B: More Behavior Generation Experiments** presents more simple and complex-behavior experiments.
- **Appendix C: Video Recordings of Training and Generated Behaviors** presents samples of training and generated behavior video recordings.

CHAPTER 2

RELATED WORK

A review of related work in the areas of behavior recognition, behavior learning and behavior generation are presented in this chapter. Studies in behavior recognition generally categorizes the observed human or robot behaviors and predicts the next behavior. Behavior learning studies propose modeling and learning methodologies including different types of neural networks, genetic algorithms, hidden Markov models or stochastic models. Behavior generation studies are closely related with the learning algorithms and they are focused on the motion generation capabilities of the robots. The related works are grouped based on the main methodologies they use.

2.1 CobART

In their study, Yavas and Alpaslan [2] introduced a new categorization method of robot behaviors. It is intended to be a base model for behavior recognition by using correlation based adaptive resonance theory (CobART). They used derivation based correspondence and Euclidean distance as correlation analysis method for category matching. CobART is a competitive, unsupervised, self-organizing category generation network where patterns are formed by consecutive real-valued inputs. As a base model, it uses ART 2 (Adaptive Resonance Theory) network, but has a more simple architecture adoptable to different problems. CobART behavior categorization results were compared against ART 2 and ART 2 with SOM networks. In their study, they tested the model on approaching, pushing, and striking behaviors. Test results show that CobART produces more reasonable categorizations than ART 2 and ART 2 with SOM networks.

In [3], Yavas and Alpaslan proposed an improved behavior categorization model which uses hierarchically integrated CobART networks. In this study, three CobART networks exist in the first layer of the model, and they categorize robot self behavior data, object self behavior data, and the distance between them. Categorization results of first layer networks are combined at the second layer, and categorized second time by the CobART network located at that layer. Thus, the model generates behavior categories according to robot motion and its effect over the object. The model generates alternative categories for the different forms of the same behaviors and specifies the close correlation between them. According to test results, hierarchical model categorizations are better than the single CobART network.

2.2 Recurrent Neural Networks (RNN) and Continuous-time Recurrent Neural Networks (CTRNN)

Tani and Yamamoto [10] studied the modeling problem of a workspace through exploration. A combination of reinforcement learning with novelty rewarding, consolidation learning and recurrent neural networks (RNN) are used for this purpose. In the model, prediction of the rewards and next sensory data is learned by the RNN model. Since modeling the workspace requires new directions to be tested, novel behaviors are resulted in having more rewards. Consolidation process is applied by rehearsing based on the previous learning, because the RNNs have some difficulties in incremental learning. Test results show that the robot is successful in learning a model of the simple workspaces.

Similarly, Tani [11] proposed a method for behavior generation by using multi level RNNs for sensory-motor learning. They used two level RNNs operating on different time scales. There are parametric interactions in top-down and bottom-up directions between two levels of networks. The model is tested on a robot arm by using a visual system. The bottom-up process corresponds to recalling the past and the top-down process corresponds to predicting the future. The parametric interaction between the bottom-up and top-down processes makes the system more flexible and robust against real time noise and changes in the environment. The networks utilize the working memory storage which stores the sequences of the parametric biases and the sensory-motor inputs/outputs where the recall as well as planning process takes place. The network in the lower level learns motor command generation using sensory data and motor commands. The relations among the motion primitives are learned by the higher level network to predict future plans. Future plans as a motion primitives sequence are generated by the higher level network, and motor commands for each motion primitive are generated by the lower level network. The model performance on the defined motion primitives are satisfactory. However, when new behavior is added to the model, the model should be retrained.

Nishimoto and Tani [12] showed learning capabilities of RNNs by generating specific sensory-action sequences. The proposed model learns the alternative paths of a finite state machine (FSM). For this purpose, the model uses context units to encode the active state of FSM. As a result, it is concluded that correct action sequences of FSMs can be generated when respective initial state is set to the context units.

Paine and Tani [13] studied on the complex navigation task. In their study, a robot learns to reach multiple targets in a maze starting from a home position. They used two different continuous-time recurrent neural networks (CTRNN). In the first network, neurons are fully connected to each other. Second network has two layer fully connected networks which are connected to each other by bottle-neck neurons. Switching among the targets are accomplished by using task neurons. The parameters of the network are learned by genetic algorithms without crossover. Tests show that hierarchically connected network performs better than the fully connected network.

In addition, Tani et al. [14] worked on a codevelopmental learning between human and humanoid robot by using hierarchically integrated CTRNNs. Arm joint positions and visual inputs are connected to the input layer of the network. Output layer produces prediction for the next visual inputs and joint positions. Motor commands are applied by using closed loop control to achieve the predicted joint positions. Robot learns the tasks with the physical assistance of the human tutor. When robot performs the task, human tutor interfere to the robot physically in order to improve task generation. Single CTRNN is not qualified for the complex tasks. Therefore, hierarchically integrated CTRNNs are used for complex tasks. CTRNNs in the first layer learn generation of outputs with minimum error rate. The second layer network learns to predict a lower layer network which will generate minimum error for the next step. Hence, second layer network is responsible from selecting the proper network in the

lower layer. Experimental results present that the complex tasks can be learned by the hierarchically integrated CTRNNs.

Reinhart and Steil [15] proposed a model which can learn reaching behavior for a humanoid robot by using RNN. The model does not generate trajectory maps for this purpose, but uses transient task space inputs with the reservoir computing approach. Reservoir computing is a framework designed for recurrent neural networks. It is used to generate a nonlinear transformation from the inputs to a multi dimensional network state space. The model has input, output and reservoir neurons for different purposes. Input neurons are connected to task space inputs and output neurons generates joint angles. Thus, the network maps the joint angles to task space bidirectionally. The reservoir neurons have connections to each other and also they are connected to the input and the output neurons. In the experiments, reaching to the target positions are achieved successfully, even when the robot arm is forced to deviate from the target (simulates a strong hit) in the middle of reaching.

2.3 Hidden Markov Model (HMM)

Fox et al. [16] proposed a behavioral model of a robot as a finite state stochastic system. They used unsupervised learning techniques on raw sensory data. Their study does not focus on the planning and control aspects, but propose a learning and evaluation methodology. In order to predict and explain the behavior of a robot during execution of the task, an HMM is estimated. Because behavioral states are hidden to the robot and it is equipped with noisy sensors from which it can obtain only an estimate of its state, they used HMM. In the first step of HMM construction, observations are clustered into evidences. Multi dimensional feature vectors are obtained by processing the raw features of consecutive observations. Then, feature vectors are fed into Kohonen self-organizing network to produce clusters of evidence items. While the robot is performing a task, a human observer labels the observations. The labeling indicates the relation between an observation and a behavioral state, as perceived by human observer. The set of labels are used as an initial set of states. The state splitting algorithm is used to enhance the initial set of states. Then HMM parameters are re-estimated by using expectation maximization algorithm[1]. Most likely sequence of generated states for a given trajectory is calculated by using Viterbi algorithm. Then, the generated state sequence are compared to the state sequence observed by human. HMM performance is investigated on randomly initialized, different size models by comparing their clustering stability. Their behavior model performance is experimented on an indoor navigation task.

Morisset and Ghallab [18] studied on a robot controller model that can perform complex tasks. They use Markov decision processes (MDPs) and hierarchical task networks (HTNs) to model the controller. The designer specifies the set of sensory-motor functions as skills into HTN plans offline. Alternative ways of performing a skill is represented by a HTN using different combinations of sensory-motor functions. The right skill to achieve the high level task is selected at each moment depending on robot state and environmental parameters. The relations among the skills and the control states are represented as an MDP and they learned through experience. Available set of skills for the task corresponds to the action space and robot control space is the state space of MDP. Control space is the task-dependent properties describing the current context. The capabilities of the model is tested by using different skills for navigation task in different types of indoor environments.

[1] Expectation maximization algorithm iteratively computes most likely parameters of a statistical model with incomplete data [17].

Infantes et al. [19] introduced a probabilistic model based on dynamic Bayesian network[2] (DBN) to learn behavior model of an autonomous robot using sensor data. Their behavior model generally cannot be composed into a global functional model, because robots are designed by multiple behavioral components. They presented a methodology for modeling the behavior of a robot as DBN, structured into observable, controllable, and hidden components, by constructing causal relations between the components. After learning the behavioral model, performance of the robot is improved by optimizing control parameters using dynamic decision network[3] (DDN) approach. DDN formalism adds some cost and reward transitions to the DBN model and it provides some variables which are controllable by decision making algorithms. Capabilities of the model is presented on an autonomous navigation task within various environments.

Han and Veloso [20] introduced a stochastic model to recognize the high-level robotic behaviors from low-level sensor inputs. The model uses a pre-defined set of states and it is based on augmented hidden Markov model. When an observation does not match to the definition, reject states are inserted to the model to stop the processing of HMM. The model is used for narrating a robotic soccer game by recognizing opponent's next behavior. Each behavior has modeled by a specific HMM.

Osentoski et al. [21] proposed a method for modeling hierarchical activities by using abstract hidden Markov models (AHMMs). The AHMM is defined as a statistical model for representing behaviors in stochastic and noisy environments. The parameters of the model is learned from the raw sensory data using expectation maximization algorithm. In the lowest level, model has states, observations and mixture components. Behaviors and termination flags are defined at the upper levels. The study compares the performances of 1 and 2-level AHMMs on indoor navigation behavior. Experiments show that second (2-level) model performed better in the environments where the classification is difficult and the training data have similar entries.

Vazquez et al. [22] presented a growing hidden Markov model (GHMM) which is used to recognize human and vehicle motions. The key feature of a GHMM is the ability of on-line and incremental learning of parameters and structure of a model. When a new observation pattern is presented to the model, probability parameters, structure of the model, and number of states are updated. A topological map is constructed to implement their structure learning approach. Continuous observation space is represented as a graph where nodes corresponds to space regions, and contiguous nodes are connected by edges. Instantaneous topological map algorithms are used to update the model's structure for every observation. Then, the structure of GHMM is updated according to changes in the topological map. An incremental version of Baum-Welch algorithm is used to estimate the parameters of GHMM. The prediction accuracy of model is compared with the other two models: first one is based on HMM and uses expectation maximization algorithm for learning, while the second one uses hierarchical fuzzy K-means (HFKM) clustering. In the experiments, both the simulator data and real data from a visual tracking system is used. The results show that, the proposed model perform better than others.

Kulic et al. [23] proposed an incremental and autonomous model to learn patterns of motion from observation data. Motion patterns are organized in a hierarchical tree structure by using HMM and factorial hidden Markov model (FHMM). An FHMM is a type of HMM in which independent multiple dynamic chains interact to generate a single output. The outputs of each independent chains are summed and output is generated using an expectation function. Motion patterns are stored in a hier-

[2] A Bayesian network is defined as a directed acyclic graph that represents dependencies between variables in a probabilistic model. Dynamic Bayesian network is a version of Bayesian network which attempts to model events that include temporal and ambient aspects. HMMs are just a special type of dynamic Bayesian networks.

[3] Dynamic decision network is also a directed acyclic graph with chance and decision nodes and it extends the single stage decision network to allow for sequential decisions.

archical tree structure for easy retrieval and organization. Detailed motion patterns near the leaves are modeled by FHMMs and general motion patterns are modeled in the nodes near to roots by HMMs. The model is experimented on a set of different human movement observation sequence obtained from a motion capture system. The performance of traditional HMM and FHMM models are compared in the experiments. FHMM demonstrated better generalization capability and performed a higher performance when recognizing same type of motion patterns.

In addition, Kulic and Nakamura [24] worked on an incremental model that can learn recognition of human behaviors by using observed data. In their approach, motion primitives are segmented by a modified version of Kohlmorgen and Lehm algorithm. Motion primitives are modeled as hidden Markov models incrementally during on-line observation. Following that, motion primitive sequences are modeled by a higher layer HMM where each hidden state corresponds to a motion primitive. The model is tested on a whole body motion data set and generated reasonable results by properly constructing a probabilistic transition model between the motion primitives.

Okada and Nishida [25] worked on incremental clustering of human gesture patterns. HMM and self-organizing incremental neural network (SOINN) are used to learn observed gestures incrementally. The SOINN is an unsupervised incremental learning method, based on self-organizing map (SOM) and growing neural gas (GNG). In GNG, dimensions of the topological structures are determined on-line during the training. The proposed model is defined as an extension to SOINN to handle variable length sequence patterns. HMM is used to map the variable length gesture patterns into fixed-length patterns and HMM parameters are used as input to SOINN. They compared the proposed model with the adaptive factorial HMM (FHMM) and hierarchical agglomerative clustering using dynamic time warping (DTW) for measuring the distance. The performed experiments show that the proposed method improved the clustering performance over the other two approaches.

Kelley et al. [26] proposed a model that can predict agents' intent based on the experiences acquired through sensor data. They used a specifically designed version of HMM to model each experiences of robot. The robot acts as an observer of agent activities and infers the intents before the actions are finalized. They validated the model on a real robot having a vision system by predicting human intents in different activities.

2.4 Imitation Learning

Okuzawa et al. [27] introduced an imitation based motion generation model for humanoid robots. Recognition of instructed movement primitives is accomplished by using HMMs. Motion generation is learned by the robot using dynamical movement primitives. Motion symbol for the recognized motion is constructed and it is kept with the learned motion primitives within the motion knowledge database. When a new motion is presented to the robot, corresponding motion symbol is compared with the motion symbols in the motion knowledge database. When a similar one is found, motion primitives of the similar motion symbol from the motion knowledge database are used and modifying process is applied to learn generation of new motion. Experiments show that, modifying process decreases the cost of the learning, but increases the error.

In another study, Kubota [28] worked on an interactive imitation learning between an instructor and a partner robot. Steady-state genetic algorithms[4] (SSGA) are used for human hand detection. The spatial

[4] In steady-state genetic algorithm, only a small number of individuals are replaced in each genetic iteration [28].

and temporal patterns of the human hand motion is extracted by spiking neural networks[5] (SNN). Then, hand motion patterns are clustered into gestures by self-organizing maps (SOM). At the end, action patterns are generated based on the human gestures by using SSGAs. The model is tested on imitative learning experiments using the partner robot MOBiMac. Experimental results showed that the robot learned the action patterns of human gestures as a result of imitative learning process.

Borenstein and Ruppin [29] introduced an imitation based learning model using genetic algorithms and neural network. Motor command generation learned by neural network using inputs of human tutor's actions and environment data. Genetic algorithms are used to determine the parameters of neural network. The fitness function of individuals are evaluated by using the correctness of the performed actions. The evolved agents performs correct actions after 2000 generation.

Hajimirsadeghi et al. [30] proposed an imitation based incremental concept learning model which uses HMM and reinforcement learning methods. Spatio-temporal demonstrations are modeled into perceptual prototypes using HMM to represent relational concepts. Generated HMMs are stored in long term or working memories, based on their contents. Motor babbling approach for hand-eye coordination is used to regenerate the learned concepts. In this approach, robot starts with an initial joint positions and makes small perturbations in its joint variables to reach some temporary goals. Then, the robot maps the sensory space to motor space by combining visuomotor information at temporary goals. The performance of their study is experimented by imitating the signs of a human hand movement. The results show that the proposed algorithm is successful in learning the concepts and regenerating the conceptual behaviors.

[5] Spiking neural networks are designed to have an increased similarity to the biological neurons. They incorporates "time of spike firing" concept into the neural network models.

CHAPTER 3

BACKGROUND

In this chapter, a brief background information is presented about the models, architectures, and algorithms used in this study. In behavior categorization phase, CobART is used to extract the motion primitives. Hence CobART and its base pattern classification architecture ART networks are explained first. Hidden Markov model (HMM) is used to model the relationships between the motion primitives and Viterbi algorithm is used to generate the most likely motion primitive sequence in behavior modeling and generation phases. Then, artificial neural networks and backpropagation algorithm are used to learn the relations between the sensor data and motor commands. Thus, details about HMM, Viterbi algorithm, ANN, and backpropagation algorithm are presented in this chapter. Then, information about Webots simulator, Khepera robot, and s-curve motion control algorithm is given, because they are used in experimental evaluation of the proposed models.

3.1 Adaptive Resonance Theory (ART) Networks

The adaptive resonance theory (ART) is a family of neural networks that are capable of stable categorization of a sequence of input patterns in real time. These architectures propose solutions of the stability-plasticity dilemma. A learning system should keep its ability to learn new patterns (plasticity) while preserving its past knowledge in a stable way (stability). This phenomenon is called stability-plasticity dilemma. ART networks were introduced by Carpenter and Grossberg as an extension of the competitive learning scheme. In competitive systems, category nodes compete with others to classify the input patterns and the winner category is selected based on some correlation function. ART categorization algorithm checks the input pattern whether it corresponds to one of the already discovered categories, otherwise a new category is created. ART architectures self-organize recognition categories for arbitrary sequences of input patterns in unsupervised manner.

Feedback mechanism is used between the competitive and input layers. It is an important feature to solve the stability-plasticity problem by automatically switching between stable and plastic modes. Moreover, feedbacks provide the learning of new patterns while preserving past knowledge. A resonant state in an ART network occurs in one of two ways. The network shall enter a resonant state if a previously learned input pattern is presented again. During the resonance state, the memory of the stored pattern will be refreshed by the adaptation process. If the input pattern is a novel pattern which does not match to active category, then the network shall search all stored patterns. If there is no match in stored patterns, a new resonant state occurs in which the new pattern is stored as a new category. Hence the network has the ability to retrieve previously learned data while keeping learning capabilities when novel input patterns presented [31].

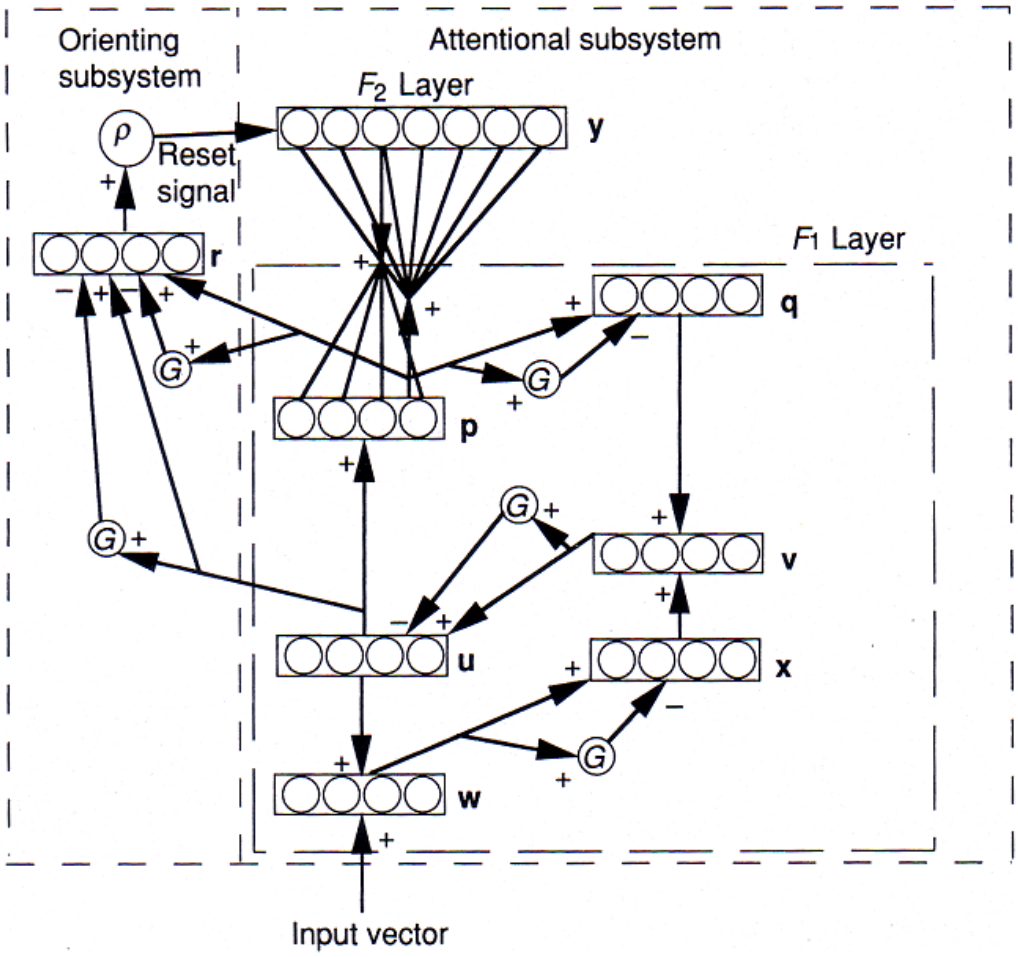


Figure 3.1: ART 2 architecture [31]

Carpenter and Grossberg first introduced ART 1 [32] for binary input networks. They developed ART 2 [33] for both analog and binary input patterns. ART 2-A [34] is a more efficient form of ART 2 network with faster runtime and better noise tolerance. Fuzzy ART [35] uses fuzzy logic for rapid learning of recognition categories for arbitrary sequence of analog or binary input patterns. ARTMAP [36] and Fuzzy ARTMAP [37] have a supervised learning mechanism for binary and analog data.

3.1.1 ART 2

ART 2 is a class of unsupervised competitive neural network which self-organizes pattern categories of analog or binary inputs. ART 2 includes the base components of all ART networks, attentional and orienting subsytems. As shown in Figure 3.1, attentional subsystem consists of an input representation field F_1, and a category representation field F_2. Orienting subsystems interacts with the attentional subsystem to perform a search process. There are top-down ($F_2 \rightarrow F_1$) and bottom-up ($F_1 \rightarrow F_2$) adaptive filters between the fields [34].

F_1 layer has six sublayers w, u, x, p, v, and q. Bottom-up input patterns and top-down signals (from F_2) are processed at different sublayers in F_1. G nodes are the gain controls which send inhibitory signals to the each unit on the layer it connected. Because pattern of activity over the nodes in the two

layers of the attentional subsystem exists only during single processing of an input pattern, they are called short-term memory (STM). Adaptive filters between the top-down and bottom-up connections are called long-term memory (LTM), because connection weights defining adaptive filters remains as a part of the network for long times [31].

When an input is presented to the F_1 layer as a pattern of activity, a pattern matching cycle is started until a match is found in F_2. If there is no match with the active category, orienting subsystem becomes active and sends a reset event to the active category at F_2. Reset event initiates a parallel search in attentional subsystem in which alternative categories are checked until a match is found or a new category is created. Bottom-up and top-down adaptive filters are updated when a search is finished and a matched F_1 pattern is found. The learning is a non-stop process which continues even while the pattern matching process is active. When a match occurs, no reset signal is generated and the network enters a resonant state which the weight are strengthened [33].

3.2 Correlation Based Adaptive Resonance Theory (CobART)

CobART is an unsupervised, competitive category generation network on real-valued inputs. It is based on ART 2 (adaptive resonance theory) [33] network, but has a simplified structure that can be adoptable to different problems. Figure 3.2 represents the architecture of the CobART network. It has orienting and attentional subsystems as its base model ART 2.

Input processing layer F_1 and category processing layer F_2 are the components of attentional subsystem. There are top-down and bottom-up weights among F_1 and F_2 layers. Input patterns are filtered to reduce the effect of noise and then presented to the network through F_1 layer. In this layer, there is one neuron for each dimension of the input pattern. The last active category has a small effect on the new input at p nodes to provide a continuous categorization process. The processed input data corresponding to p nodes are calculated by

$$p_{ij} = a \times w_{ij} + (1 - a) \times z_{kij}$$

where p_{ij} is the processed input data for the dimension of input vector i and input value j. w_{ij} is the filtered input value and z_{kij} is the k^{th} category data. Parameter a is the effect of new data on processed input vector [3].

There is a neuron in F_2 layer for each category. Winner category data for each input is updated according to the deviation from the input pattern and the learning rate. Learning rate decreases over time to preserve the previously learned categories.

After the input pattern is processed at F_1 layer, a matching score between the input pattern and the learned categories are calculated at r nodes in the orienting subsystem using correlation methods. Euclidean distance method (EDM) and derivation correspondence method (DCM) are used to calculate the correlation between the two vectors. DCM calculates the similarity of two vectors based on their derivations. On the other hand, EDM calculates the similarity of two vectors based on the distance between them. Matching score r_k is calculated by

$$r_k = \frac{1}{I} \times \sum_{i=1}^{I} (b \times DCM(p_i, z_{ki}) + (1 - b) \times EDM(p_i, z_{ki}))$$

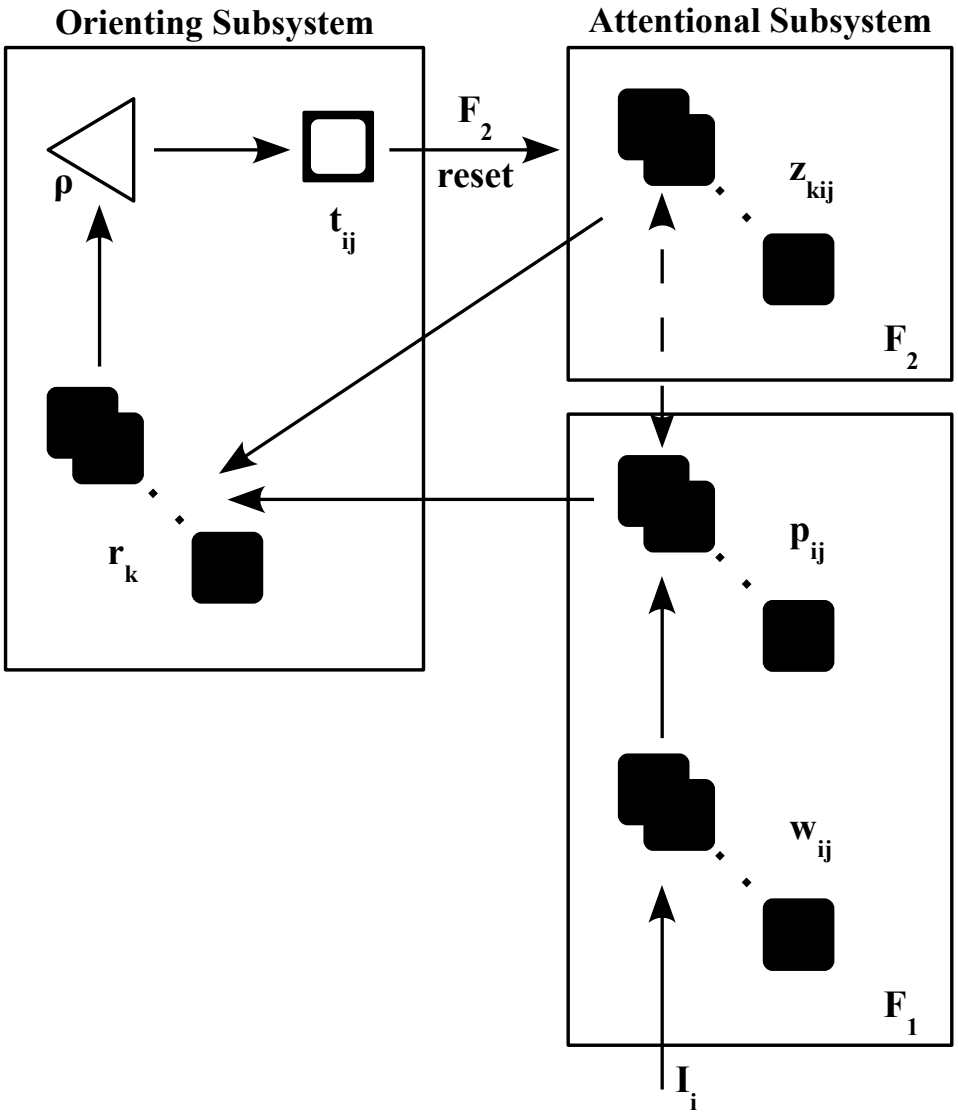


Figure 3.2: CobART network model [3]

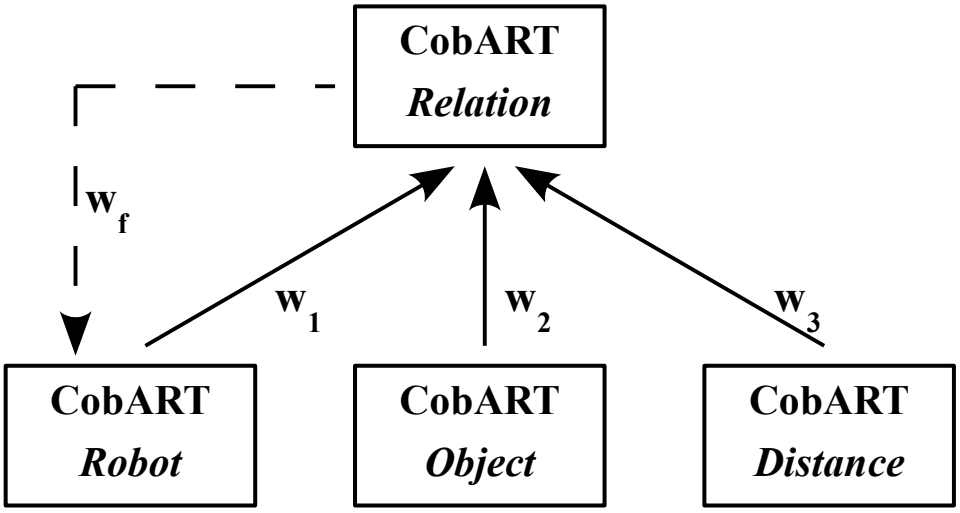


Figure 3.3: Hierarchically integrated CobART networks [3]

where p_i is the processed input data and z_{ki} is the k^{th} category data. I is the input vector length and b is the weight of DCM over EDM [3].

Vigilance parameter adjusts the coarseness of the generated categories and balances stability-plasticity features. If matching score of a category is greater than the vigilance parameter, that category is selected as the winner. If there is no match, a new category is created by using input data.

3.2.1 Hierarchically Integrated CobART Networks

In behavior recognition, sometimes it is necessary to classify the different forms of the same behavior. For example, we may need to determine that slow and fast *approach* behaviors are the different forms of the same behavior. Moreover, a robot can perform more than one behavior in parallel. When it is approaching an object, it can be moving away from another object. In order to learn relations among these behaviors, a second layer CobART network is added and CobART networks are integrated hierarchically as shown in Figure 3.3. CobART Robot categorizes robot behaviors using robot's motion data, CobART Object categorizes object's behaviors, and CobART Distance categorizes distance between robot and object. In the second layer, CobART Relation categorizes robot behaviors according to its effect on an object. w_1, w_2, and w_3 are the weight parameters connecting the first layer to the second layer. They determine the effect of the first layer CobART networks on the second layer. The association from second layer to first layer (clipped line) delivers matching information to the first layer and the effect of this feedback association is adjusted by weight w_f [3].

Matching information is used to detect the correlation between the input patterns and category data as in single CobART network, but input pattern of the second layer corresponds to active category outputs of the first layer networks. Matching score r_k in the second layer is calculated by

$$r_k = \sum_{m=1}^{M} \left(\frac{w_m}{I_m} \times \sum_{i=1}^{I_m} \left(DCM(p_{mi}, z_{mki}) \times b + EDM(p_{mi}, z_{mki}) \times (1-b) \right) \right)$$

where M is the number of first layer CobART networks while the weight of CobART network m is

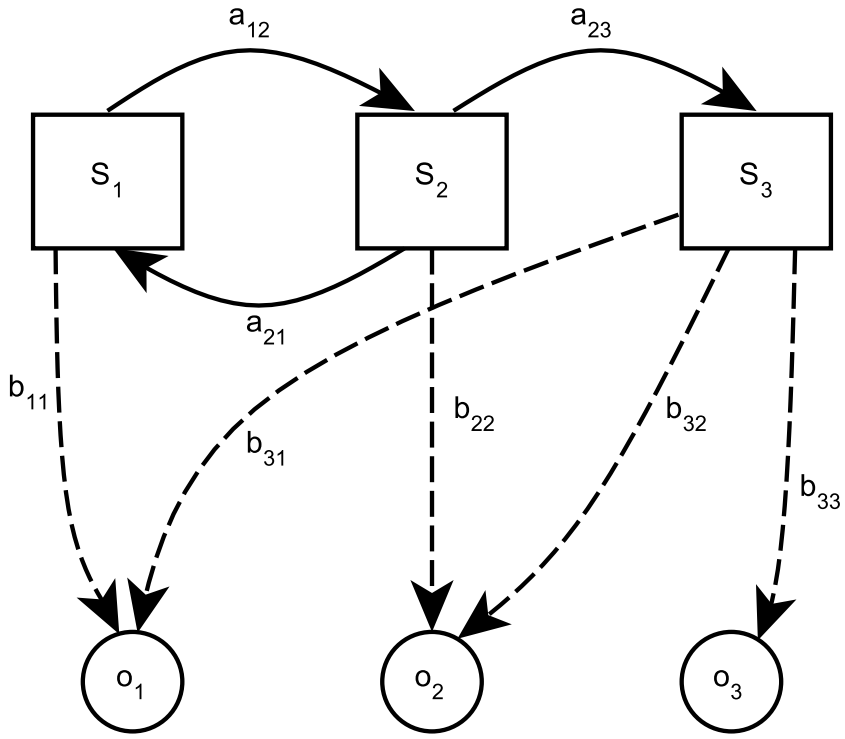


Figure 3.4: A hidden Markov model (HMM)

represented by w_m, and I_m is the input vector length for the corresponding CobART network. p_{mi} is the processed input data and z_{mki} represents the k^{th} category data for m^{th} CobART network. Experiments in [3] presents that hierarchically integrated CobART network outperforms to single CobART network by classifying input data into more correct and generic categories [3].

3.3 Hidden Markov Model (HMM)

A *Markov process* is a stochastic process where the future state distribution is determined only by the current state, not the sequence of previous events. It is a state machine in which the system stays in the active state and then transits to a different state probabilistically. A *hidden Markov model* (HMM) is a mathematical and statistical tool to model generative sequences where the modeled system is a Markov process with unobserved states as shown in Figure 3.4. States of the model are directly visible in a regular Markov model. However, the states are not visible to the observer, but the observations in the state are visible in an HMM.

An HMM can be described as a 5-tuple [5]:

- $S = \{S_1, \ldots, S_N\}$ is the set of states and the state at time t is q_t.
- $O = \{o_1, \ldots, o_M\}$ is the set of observations and the observation at time t is o_t.
- $A = \{a_{ij}\}$ is the probability distribution of state transitions where $a_{ij} = P[q_{t+1} = S_j \mid q_t = S_i]$, $1 \leq i, j \leq N$.

- $B = \{b_j(k)\}$ is the probability distribution of observations in state j where $b_j(k) = P[o_k \, at \, t \mid q_t = S_j]$, $1 \leq j \leq N, 1 \leq k \leq M$.

- $\pi = \{\pi_i\}$ is the probability distribution of initial states where $\pi_i = P[q_1 = S_i]$, $1 \leq i \leq N$.

After an HMM is properly defined by the parameters S, O, A, B and π, it can be used as a generator for a given observation sequence like $V = [V_1 V_2 \dots V_T]$ as follows [5]:

1. Using the initial state distribution π, choose an initial state $q_1 = S_i$.

2. Reset the time ($t = 1$.)

3. Choose $V_t = o_k$ according to $b_i(k)$ in state S_i.

4. Go to new state $q_{t+1} = S_j$ according to a_{ij} for state S_i.

5. Increase time $t = t + 1$; terminate if $t \geq T$; otherwise go to step 3.

The process described above can be used as a model explaining steps of a given observation sequence generation by a corresponding HMM.

3.3.1 Viterbi Algorithm

The problem given below is one of the main problems that should be solved to prove that HMM can be used in real applications.

Problem: Given the observation sequence $V = [V_1 V_2 \dots V_T]$ and an HMM, find the corresponding most likely state sequence $S = [S_1 S_2 \dots S_T]$.

The Viterbi algorithm [6] is a kind of dynamic programming algorithm to solve the above problem. As a first step, highest probability term is defined below as

$$\delta_t(i) = \max_{S_1, S_2, \dots S_{t-1}} P[S_1, S_2 \dots S_t = i, \;\; V_1 V_2 \dots V_t \mid \lambda]$$

for a single path including the first t observations. λ is the definition of the given HMM. It can be stated that

$$\delta_{t+1}(j) = [\max_i \delta_t(i) \, a_{ij}] \cdot b_j(V_{t+1})$$

by induction. In order to get the state sequence, the argument which maximizes the above equation should be tracked for each t and j. $\psi_t(j)$ is defined for this purpose. The procedure to find most likely state sequence is presented below [5]:

- Initialization

$$\delta_1(i) = \pi_i b_i(V_1), \quad 1 \leq i \leq N$$
$$\psi(i) = 0$$

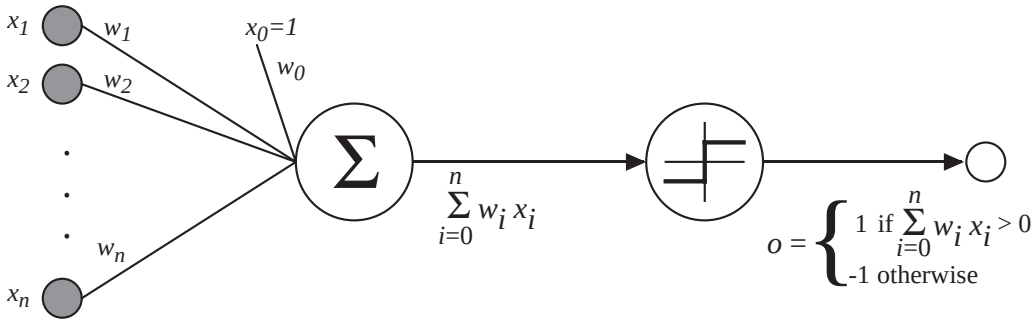


Figure 3.5: A perceptron [39]

- Recursion

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i)\, a_{ij}] \cdot b_j(V_t), \quad 2 \le t \le T \text{ and } 1 \le j \le N$$
$$\psi_t(j) = \operatorname*{argmax}_{1 \le i \le N} [\delta_{t-1}(i)\, a_{ij}], \quad 2 \le t \le T \text{ and } 1 \le j \le N$$

- Termination

$$P^* = \max_{1 \le i \le N} [\delta_T(i)]$$
$$S_T^* = \operatorname*{argmax}_{1 \le i \le N} [\delta_T(i)]$$

- State sequence tracking

$$S_t^* = \psi_{t+1}(S_{t+1}^*), \quad t = T-1, T-2, \ldots, 1$$

3.4 Artificial Neural Networks (ANN)

An artificial neural network, commonly referred as neural network, is a mathematical model motivated by biological neural networks which are build of very complex groups of interconnected neurons. Processing in these systems is parallel and distributed across layers of neurons. Each neuron processes its inputs simultaneously and independently [38]. Artificial neural networks try to simulate highly effective processing power of biological neural networks. Neural network methods provide a powerful approach to learn analog, binary, and vector based target functions [39]. They are very effective in learning the certain types of complex problems such as mapping of complex input data to actuator commands.

3.4.1 Perceptron

The perceptron is the first algorithmically described neural unit invented by Rosenblatt [40]. As shown in Figure 3.5, it contains a single neuron with weighted inputs and adjustable threshold to classify the linearly separable input patterns (i.e., patterns that can be separated by a single line). Rosenblatt proposed a perceptron learning algorithm for perceptron networks to solve pattern recognition problems. If the patterns are from two linearly separable classes, then the learning algorithm converges to the correct network weights [41].

A perceptron takes a set of analog inputs and determines the activation level by summing the weighted input values, then generates -1 if the result is less than some threshold and 1 otherwise. More precisely, given input values $x_1, \ldots, x_n$, weights $w_1, \ldots, w_n$ and a threshold $(-w_0)$, the output $o(x_1, \ldots, x_n)$ computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

where each w_i is the weight of inputs which determines the effect of input x_i over the output. Note that the value $(-w_0)$ is used as a threshold that the sum of weighted inputs must be greater than the threshold in order to output 1.

The perceptron presents a decision surface in the multi-dimensional instance space. It outputs 1 for the patterns on the one side and outputs -1 for the other patterns [39].

A perceptron learning is a process for choosing the correct values of weight vector $w_1, \ldots, w_n$ that fits to training data. One approach to learn an acceptable weight vector is the perceptron training rule. This approach initializes weights randomly, then applies each training example to the perceptron iteratively. The weight vector of the perceptron is updated when a training example is not classified correctly. This process is repeated until all training examples are classified correctly. For each training example in the training set, each weight w_i associated with input x_i is updated according to the rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Here t is the target output, o is the perceptron output ($t - o$ is the error in output), and η is a positive constant called learning rate. Learning rate determines the degree of change in weights in each step [39].

Perceptrons can represent most of the boolean functions (such as AND, OR, NAND and NOR). However, some boolean functions cannot be represented by a single perceptron, such as the XOR function which is not linearly separable.

3.4.2 Multilayer Networks

The perceptron network is inherently limited to the linearly separable patterns. Multilayer networks (multilayer perceptron) are constructed in order to overcome limitations of perceptron. Followings are the major features of multilayer networks [41]:

- Each neuron in the network includes a *differentiable* nonlinear activation function.
- The network includes one or more *hidden* layers.
- The network has a high degree of connectivity between the neurons.

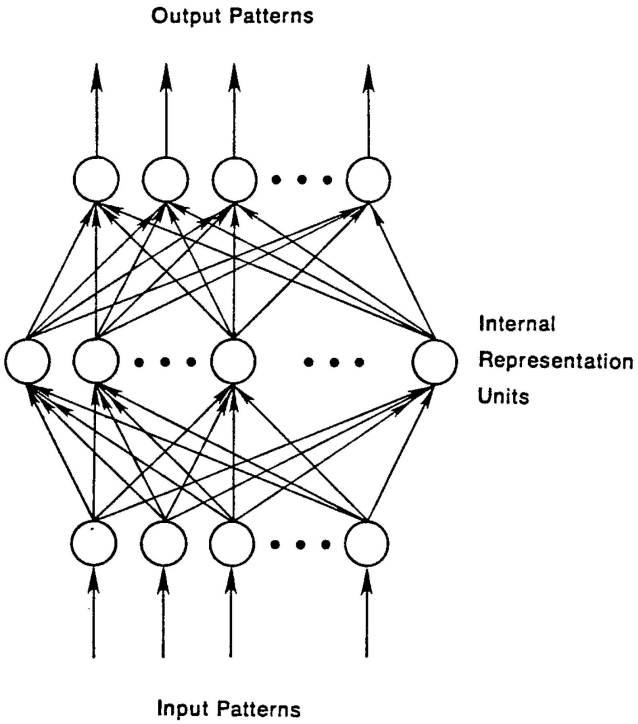


Figure 3.6: A multilayer network [42]

Figure 3.6 shows the architectural graph of a multilayer network with one input layer, one hidden (internal representation) layer and one output layer.

In order to extend our solutions to non-linearly separable patterns, we need units that has differentiable nonlinear activation function. One solution is the *sigmoid unit* which is like perceptron but has a smoothed, differentiable threshold function.

The sigmoid unit first calculates the activation level by summing the weighted input values, then applies a threshold which is a continuous function of its inputs as shown in Figure 3.7. The output of a sigmoid unit o is calculated as

$$o = \sigma(\vec{w} \cdot \vec{x})$$

where

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

σ is called the logistic function or the sigmoid function. Other differentiable sigmoid functions can be used in place of σ like hyperbolic tangent function (*tanh*) [39].

3.4.2.1 Backpropagation Algorithm

Backpropagation learning algorithm is a simple and general method to train multilayer neural networks. It applies *gradient descent* to find the minimum error rate between the target values and network outputs. *Delta rule* is developed to fix the convergence problem of perceptron training rule. Delta rule searches all possible weight vectors to find the best weight vector that fit the training examples by

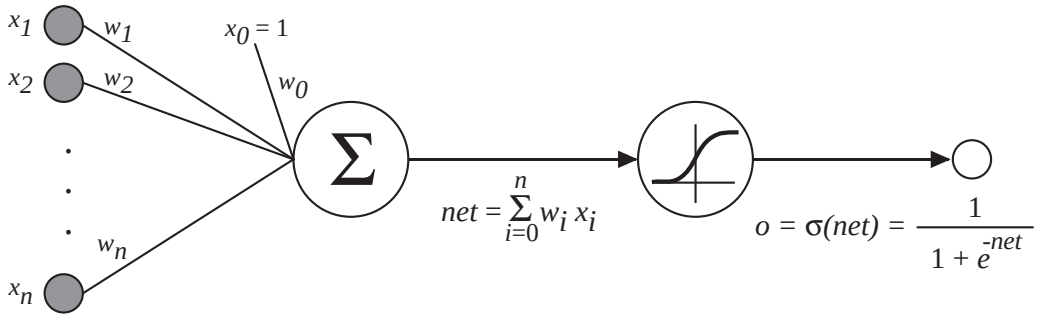


Figure 3.7: The sigmoid threshold unit [39]

using gradient descent. In multilayer backpropagation algorithm, delta rule is modified to adopt the hidden and output layer weight update rules. The modified delta rule is called generalized delta rule [42].

Different kinds of error functions can be used in gradient descent. One of the error measures on input/output pattern p is

$$E^p = \frac{1}{2}\sum_j (t_j^p - o_j^p)^2$$

where o_j^p and t_j^p are the actual and the target outputs associated with jth output unit for pattern p. The overall error measure is $E = \sum E^p$. Generalized delta rule uses gradient descent in E. The rule for changing weights for input/output pattern p is

$$w_{ji}(n) \leftarrow w_{ji}(n-1) + \Delta^p w_{ji}(n-1)$$

where

$$\Delta^p w_{ji} = \eta \delta_j^p o_i^p$$

n is the iteration number, w_{ji} is the weight value from the ith unit to the jth unit, η is the learning rate, δ_j^p is the error signal in jth node, and o_i^p denotes the input from node i to unit j. $\Delta^p w_{ji}$ is the amount of change to be made to the weight w_{ji} for the pattern p. The net total output of a given unit is

$$net_j^p = \sum_i w_{ji} o_i^p$$

and the output of a unit is calculated by the following formula:

$$o_j^p = f_j(net_j^p)$$

f is the differentiable activation function (e.g. sigmoid, tanh). The error signal, δ, differs for output units and hidden units. δ is calculated for output units as

$$\delta_j^p = (t_j^p - o_j^p) f_j'(net_j^p)$$

where f' is the derivation of activation function. δ is defined below for the hidden units:

$$\delta_j^p = f_j'(net_j^p) \sum_k (\delta_k^p w_{kj})$$

The application of the generalized delta rule includes two phases. In the first phase, the output of each unit o_j^p is calculated by propagating the input values through the network. Then the outputs of output units are compared with the targets and an error signal δ_j^p is calculated for each output unit. During the second phase, error signal calculated at the output units are propagated backward to the hidden units and the necessary weight changes are done [42].

Many variations developed for backpropagation algorithm. One of the variations is using momentum in the weight update rule. In order to speed up the learning process, larger learning rates needed, but large learning rates may lead oscillations during gradient descent. The use of momentum term in neural network training lets the use of larger learning rates without oscillation. Following is the weight update rule with momentum:

$$\Delta^p w_{ji}(n) = \eta \delta_j^p o_i^p + \alpha \Delta^p w_{ji}(n-1)$$

$\Delta^p w_{ji}(n)$ is the weight update in the iteration n, and $0 \leq \alpha < 1$ is the momentum constant. The momentum allows fast convergence by using bigger weight steps.

3.4.3 Recurrent Neural Networks

A recurrent neural network (RNN) is a kind of neural network with feedback connections. It uses outputs of network at time t as the input to other units at time $t + 1$. This creates an internal memory which allows processing of temporal sequences of inputs. The feedback connections improve the neural network learning capabilities that can be used in many problems including behavior learning and sequence processing tasks.

There are many types of recurrent neural networks developed for different domains. Figure 3.8 shows a simple RNN which have one feedback connection. Assume that we have a task of forecasting the next day's weather $y(t + 1)$ based on the current day's weather $x(t)$. One solution to this task is to use feedforward network as shown in Figure 3.8(a). This solution is limited to the today's weather. It might be necessary to forecast the tomorrow's weather not only based on today's weather but based on the arbitrary window of time in the past. The recurrent network shown in Figure 3.8(b) provides a proper solution to this problem. New units b and $c(t)$ are added for feedback connections. The value of $c(t)$ is defined as the value of b at time $t - 1$. In this recurrent relation, b corresponds to information about the network inputs history. Because b depends both on the $x(t)$ and $c(t)$, it is the summary of information from previous values of x in time. Figure 3.8(c) shows the recurrent neural network status which is unfolded in time [39].

In practice, training of RNN is more difficult than the networks with no feedback loops. RNNs cannot be easily trained for larger networks having large number of input units. In spite of the training difficulties, they remain important because of their representational power [39].

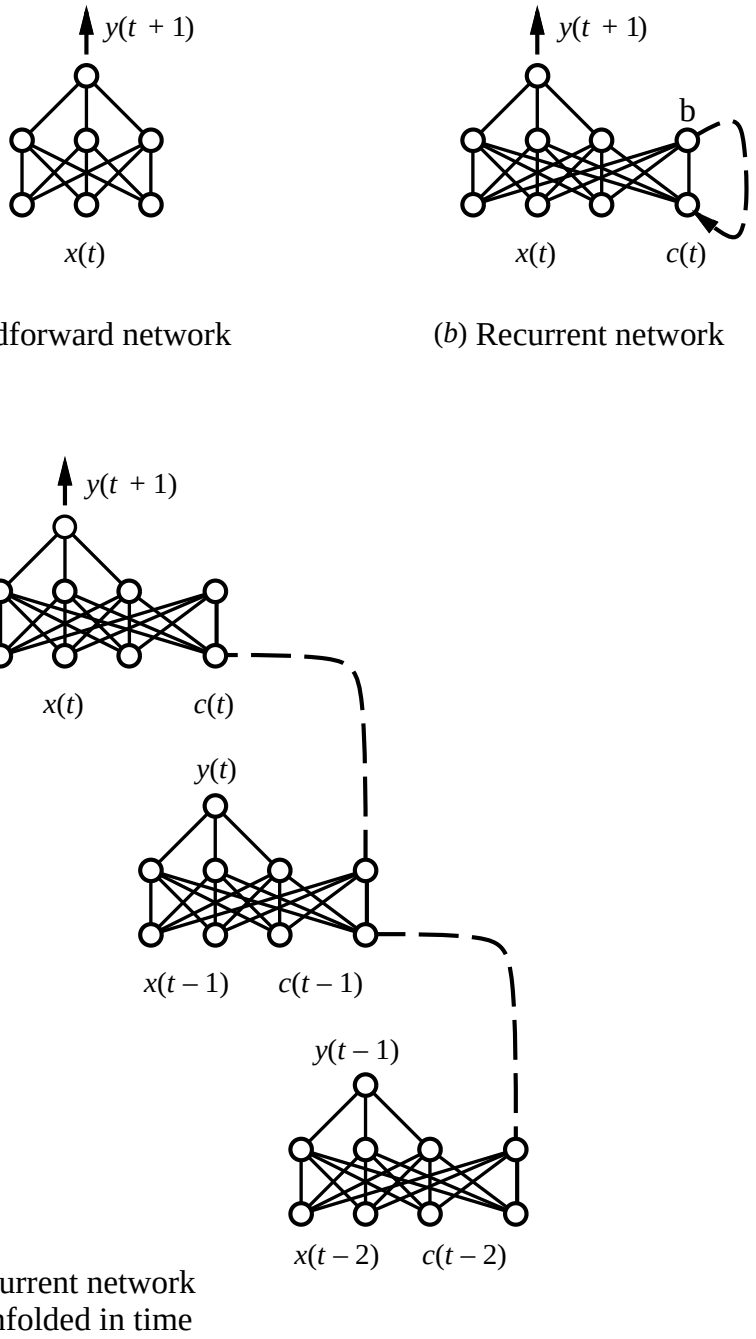


Figure 3.8: Recurrent neural networks [39]

3.5 Webots Simulator

Webots [8] is a mobile robot simulation software that provides a prototyping environment for modeling, programming and simulating mobile robots. A simulated robot environment is less expensive than real robots. Application development and testing in simulated environments are more flexible and safer than the real environments. Webots lets the user to add passive objects (e.g. walls, obstacles) and active objects (e.g. mobile robots). These robots can have a sensor and actuator devices, such as distance sensors, wheels, video cameras, and touch sensors. In this simulated environment, there are various models of robots and program examples for robot controllers. The user can control each robot separately to achieve the desired behavior by using the example programs. Webots provides a couple of interfaces to real robots, hence the user can transfer control program to the real robot after the robot behaves as intended in the simulated environment.

Webots have been used in many research and educational projects in the following areas [43]:

- Prototyping mobile robots
- Robot locomotion studies
- Multi-agent robot studies (collaborative mobile robots groups, swarm intelligence, etc.)
- Adaptive behavior studies (neural networks, genetic algorithm, AI, etc.).
- Teaching robotics
- Robot contests

A simple Webots simulation is consists of a *world* file and one or more controller programs for the robots. A world describes the environment and properties of robots. Each object in a world can contain other objects hierarchically. For example, a robot can have two wheels, two encoders and a servo which itself contains a camera. Figure 3.9 shows a sample Webots environment.

A controller program should be implemented to control each robot specified in world file. Implementation languages supported by Webots are C, C++, Java, Matlab, and Python. As soon as the simulation starts, specified controllers are launched for each robot as a separate process.

Webots have a privileged type of robot called supervisor. Each supervisor controller is associated with a controller program. It can access to privileged operations such as positioning robots to specific positions and making video capture of the simulation [43].

3.5.1 Programming Fundementals

A robot controller program should start with the initialization function *robot_live()*. Then the program should enter an infinite loop to run the controller continuously until the simulator exits. The loop must contain at least one call to the *robot_step(SIMULATOR_TIME_STEP)* function which advances the simulator time. *robot_step()* function is used to synchronize the actuator and sensor data with the simulator. Time duration given in *robot_step()* specifies an amount of simulated time, so user can run the simulator faster than the real world.

Following is the code snippet describing a typical robot controller application which reads the sensors, actuates the motors and runs the simulator by the time given in the *robot_step()* function.

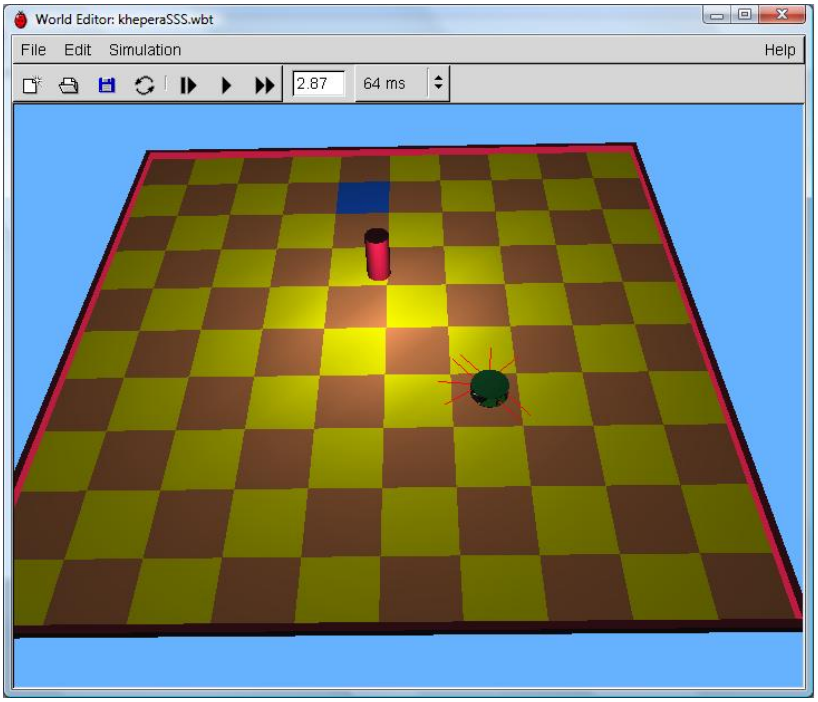


Figure 3.9: Webots environment

```
while (true)
{
    readSensors();
    actuateMotors();
    robot_step(SIMULATOR_TIME_STEP);
}
```

Following code shows a complete example of a robot controller application. The robot in this example has left and right motors for differential steering. It has left and right proximity sensors to sense obstacles [43].

```
#define SIMULATOR_TIME_STEP 32

int main()
{
    robot_live();
    DeviceTag leftSensor  = robot_get_device("leftSensor");
    DeviceTag rightSensor = robot_get_device("rightSensor");
    distance_sensor_enable(leftSensor, SIMULATOR_TIME_STEP);
    distance_sensor_enable(rightSensor, SIMULATOR_TIME_STEP);

    while (true)
    {
        // read sensors
        double leftDistance  = distance_sensor_get_value(leftSensor);
        double rightDistance = distance_sensor_get_value(rightSensor);
```

```
        // compute motor commands
        double leftCommand  = compute_left_speed(leftDistance, rightDistance);
        double rightCommand = compute_right_speed(leftDistance, rightDistance);

        // apply motor commands to wheel motors
        differential_wheels_set_speed(leftCommand, rightCommand);

        robot_step(SIMULATOR_TIME_STEP);
    }

    return 0;
}
```

3.5.2 Supervisor Controller

The supervisor is a special type of super robot that is able to perform everything a robot can perform. It is useful for communicating with the other robots by using the *Receiver* and *Emitter* nodes. In addition, a supervisor can move or rotate any object and can track the position of robots in order to record the trajectories. It can also take a snapshot or a video of the simulation. When describing a simulation *world*, a *Supervisor* node can be inserted as a basis node to the model of a robot. The *Supervisor* node offers the *supervisor_*()* functions in addition to the *robot_*()* functions. All this functions can be used from a controller program associated with a *Supervisor* node. Following is the supervisor code example showing how to keep track of a single robot [43].

```
#define SIMULATOR_TIME_STEP 32

int main()
{
    // Coordinates in X, Y, Z. Rotation in X, Y, Z, Rotation angle
    float robot_coordinates[7];

    robot_live();

    // do this once only
    NodeRef robot_node = supervisor_node_get_from_def("MY_ROBOT");
    supervisor_field_get(robot_node,
                         SUPERVISOR_FIELD_TRANSLATION_AND_ROTATION,
                         robot_coordinates,
                         SIMULATOR_TIME_STEP);

    while (true)
    {
        printf("Robot position is: %g %g %g\n",
                robot_coordinates[0],
                robot_coordinates[1],
                robot_coordinates[2]);
```

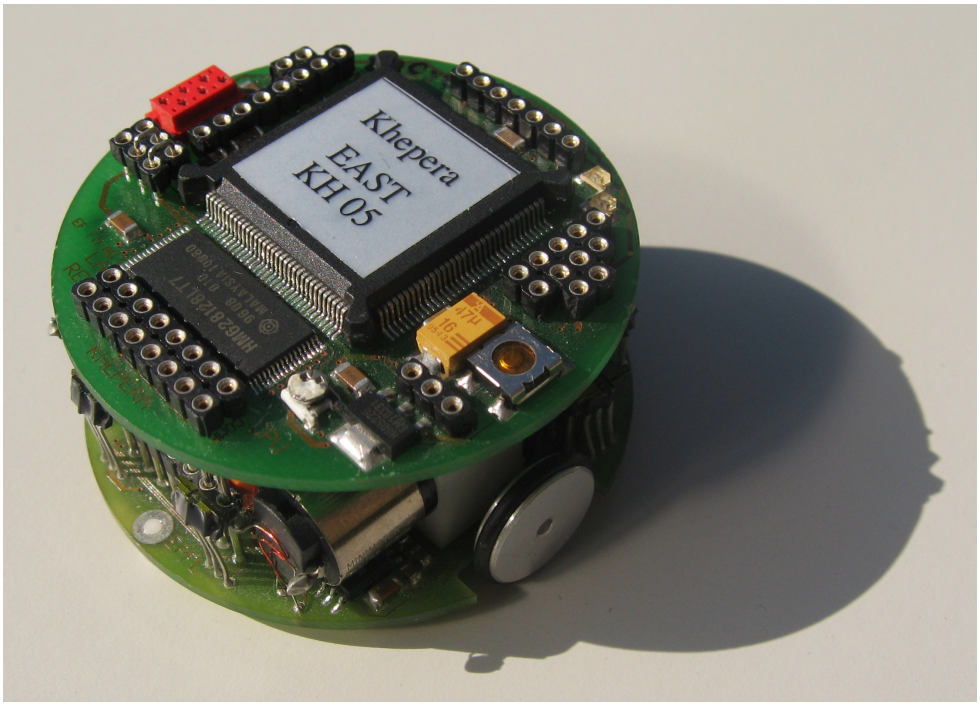


Figure 3.10: Khepera robot

```
        robot_step(SIMULATOR_TIME_STEP);
    }

    return 0;
}
```

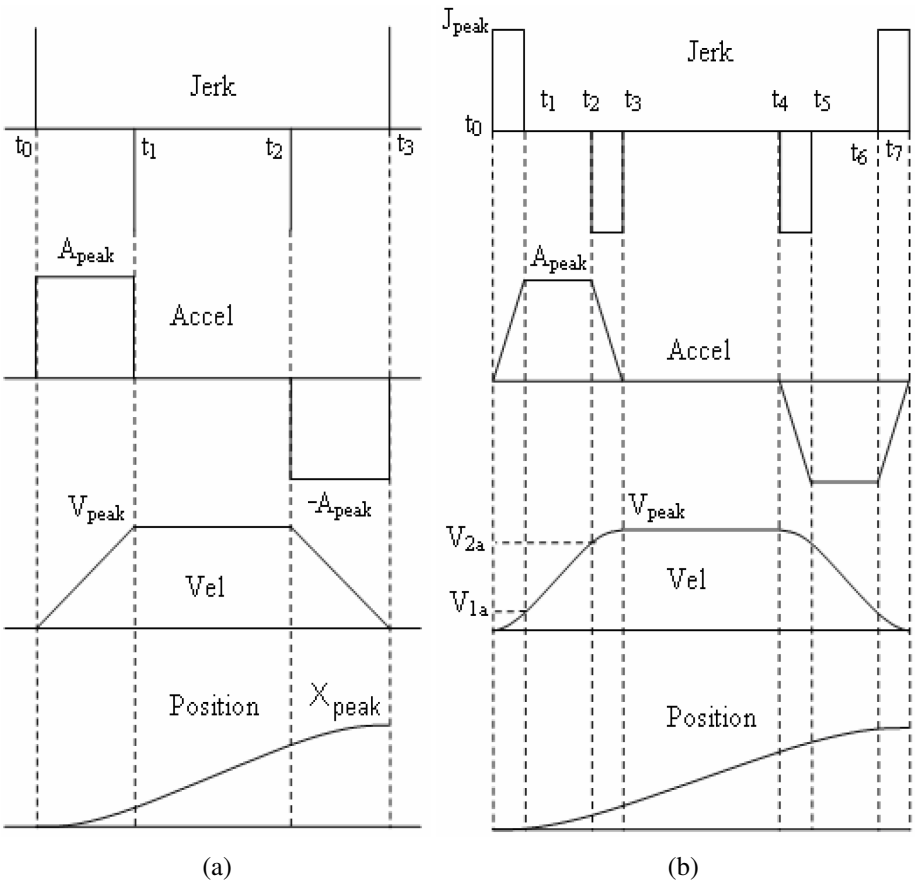
3.6 Khepera Robot

The Khepera is a differential wheeled mobile research robot developed by LAMI (Microprocessor and Interface Laboratory) at the Swiss Federal Institute of Technology Lausanne (EPFL) in 1992. The initial version of Khepera is shown on Figure 3.10. It is used in many academic and research studies around the world for many robotics experiments.

The Khepera robot has 8 infrared and light sensors allowing to detect proximity of objects in front, behind, right and left. Light sensors measure the level of lights to detect light sources. The robot has two independently controlled motors which enabling forward and backward movements. Due to its small size and wide range of possible movements, it can be used in various experiments including obstacle avoidance, target search and collective behavior [7].

3.7 S-Curve Algorithm

One of the main challenges in motion control systems is to generate the desired motion with minimum oscillation and overshoot in position and velocity. In order to solve these problems, trajectory planning topic has been researched extensively. The goal is to generate feasible and smooth motion with the



(a) (b)

Figure 3.11: (a) Trapezoidal s-curve model and (b) third order polynomial s-curve model [44]

minimized overshoot [44].

S-curve algorithms rely on the information of the possible limits of acceleration and deceleration of the controlled system. Simple trapezoidal models, sometimes called second order s-curve models, can perform fast motions, but as shown in Figure 3.11a there are acceleration jumps at some points in time like t_0, t_1, t_2, and t_3 when the orientation of velocity changes. This jumps force the jerk to have infinite values. This may cause problems in some systems. Third or n^{th} order s-curve models are proposed to overcome the jumps in acceleration and infinite jerk problems. S-curve model with finite jerk and infinite snap (derivation of jerk) segments the desired motion into multiple pieces as shown in Figure 3.11b. The algorithm uses limited jerk, depicted by J_{peak} in the figure, to make the motion smoother [44].

The model definition of the trapezoidal system (M_2) is presented below where the velocity is not smooth. Its acceleration is calculated by second order polynomials [44].

$$a = M_2 = \begin{cases} A_{peak}, & t_0 \leq t \leq t_1 \\ -A_{peak}, & t_2 \leq t \leq t_3 \end{cases}$$

In polynomial models higher than two, the jerk can be finite and makes the velocity profiles smoother. Third order polynomial s-curve model (M_3) is presented below [44].

$$j = M_3 = \begin{cases} M_2, & t_0 \le t \le t_3 \\ -M_2, & t_4 \le t \le t_7 \end{cases}$$

Similarly, n^{th} order s-curve models can be defined recursively in the following expression [44].

$$M_n = \begin{cases} M_{n-1}, & t_0 \le t \le t_{2^{n-1}-1} \\ -M_{n-1}, & t_{2^{n-1}} \le t \le t_{2^n-1} \end{cases}$$

CHAPTER 4

SIMPLE-BEHAVIOR LEARNING MODEL (SBLM)

SBLM is developed to model simple behaviors which have one specific and basic goal such as *turn, approach, or grasp an object*. Overall structure of this model is presented in Figure 4.1. The model is composed of *behavior categorization, behavior modeling, and behavior generation* phases. The categorization phase categorizes raw sensory data into motion primitives and observation categories. The modeling phase constructs a transition model between motion primitives and learns to generate motor commands for each motion primitive using sensory data. The last phase handles the generation of behaviors based on the current and the goal observations. Following sections give detailed information about the model phases [4].

4.1 Behavior Categorization

A behavior as a time-extended action sequences has some unique and repeated parts. A behavior learning model should recognize these parts as meaningful components of a behavior. Thus the purpose of behavior categorization is to determine hidden or visible capabilities of robots while performing a task. Components of behaviors are categorized as meaningful motion sequences by the model. Hence behaviors are decomposed into more easily and more accurately learned reusable components. These components are called *motion primitives*. There are some advantages to learn a behavior based on the motion primitives. The model can learn small components more easily and correctly. Furthermore, the model can use previously learned components without repeating the same training process. CobART [9] is used as a categorization method to determine motion primitives and observation categories [4].

CobART (see Section 3.2) is a competitive, self-organizing category generation network for real-valued inputs. It is based on ART 2 (adaptive resonance theory) [33] network but has a simpler architecture that can be adoptable to different problems.

The adaptation of CobART for different problems is performed through the adjustment of the parameters. Categorization capabilities of CobART is directly related with the parameters. The important CobART parameters are listed in Table 4.1. Parameter a adjusts the effect of new input over the last selected category. If it is higher, new input has more power on the next category. Parameter b determines the effect of correlation methods (EDM and DCM) on the calculation of correlation between two vectors. EDM (Euclidean distance method) calculates the correlation of two vectors based on the distance between them. DCM (Derivation correspondence method) uses derivations to calculate the similarity between the vectors. Learning rate decreases over time in order to protect previously learned categories from continuous updates. d is the initial learning rate which decreases over time according to decrease rate α. Vigilance parameter ρ determines coarseness of learned categories and balances

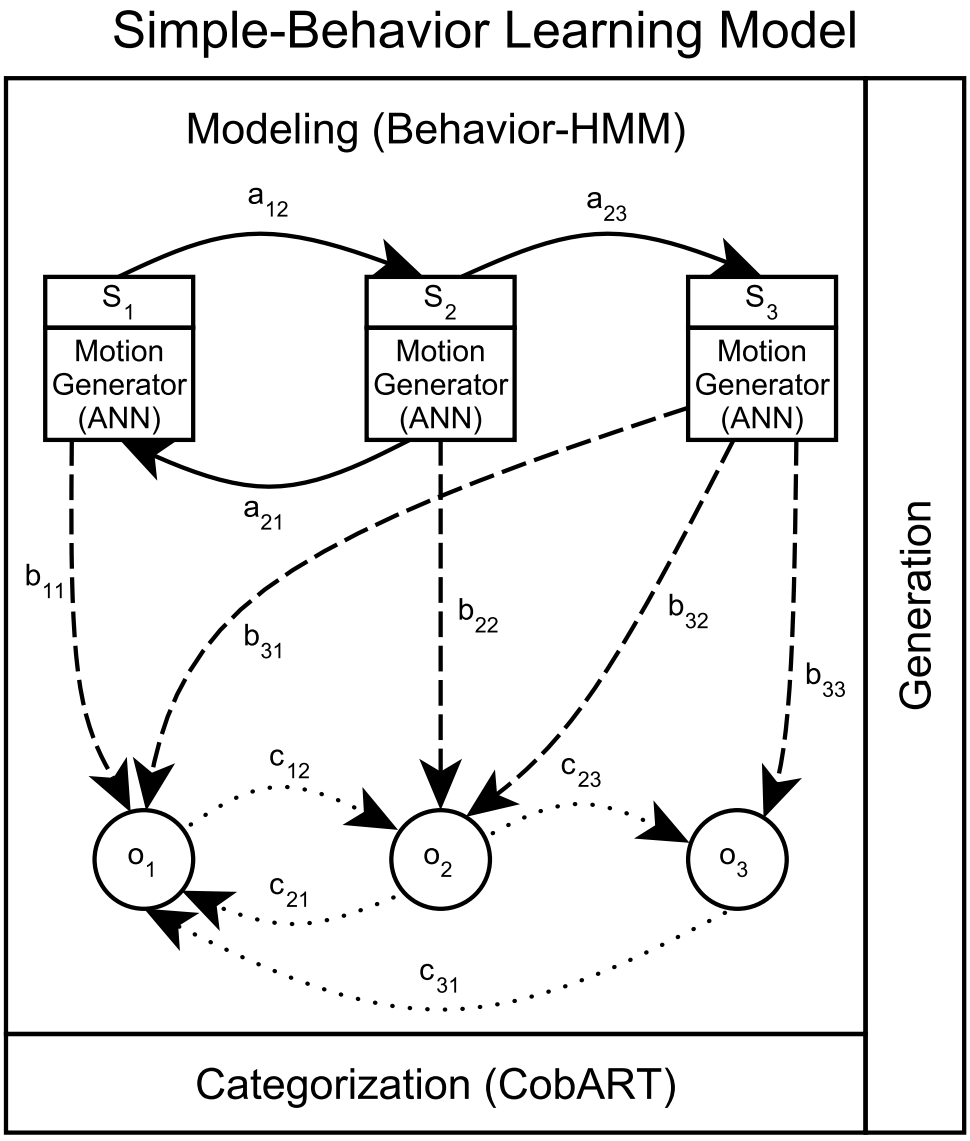


Figure 4.1: Components of simple-behavior learning model [4]

Table4.1: Definitions of CobART parameters [4]

Parameter	Explanation
a	The weight of new input on the last active category where $0 \leq a \leq 1$
b	Weight of DCM over EDM where $0 \leq b \leq 1$
d	Initial learning rate where $0 \leq d \leq 1$
α	Learning rate decrease amount where $-\infty < \alpha \leq 0$
ρ	Vigilance parameter where $0 \leq \rho \leq 1$
I	Length of input vector

Table4.2: CobART parameter values used in motion primitive and observation categorization

Parameter	Motion Primitives	Observations
a	0.8	0.95
b	0.4	0.1
d	0.2	0.2
α	-1.0	-1.0
ρ	0.85	0.95
I	10	3

stability-plasticity features. The bigger vigilance generates more categories with finer distinction. Decreasing the vigilance generates fewer categories. At last, parameter I is the number of consecutive inputs fed into the CobART [3].

An observation can be defined as a collection of sensor readings (mostly real-valued) at some point in time. Thus, observation space contains very large real-valued sensory inputs and they should be mapped into meaningful finite observation categories to be used by the model. For this purpose, in addition to motion primitive categorization, CobART is used with different inputs and parameters to categorize observation space. The CobART parameter values which are used in motion primitive and observation categorization are presented in Table 4.2.

CobART parameter values in Table 4.2 are selected based on the previous studies of Yavas and Alpaslan [2][3]. Parameter a is higher for observation categorization, because observation categories are less dependent on the previous data. Hence new inputs should have more impact on the observation categorization. Also, EDM method is more effective to compare the two observation vectors. Thus parameter b is smaller in observation categorization. Higher vigilance parameter is used in observation categorization, because more categories of observations increases behavior generation quality and resolution. In motion primitive categorization, 10 consecutive inputs ($I = 10$) are presented to the input layer of CobART. This value is the same as in [2]. Because observations are instantaneous values, input length is selected as 3 ($I = 3$) in observation categorization.

Followings are the formal definitions of the terms used in the categorization process [4]. An observation x_t at time t is defined as

$$x_t = \{p_1^t, \ldots, p_N^t\}$$

where p^t is a sensor value at time t and N is the number of sensor readings. Observation sequence X_T is a finite set of consecutive observations which is defined as

$$X_T = [x_{t=1} x_{t=2} \ldots x_{t=T}]$$

where T is the number of observations in the sequence. Observation category o is defined as a function

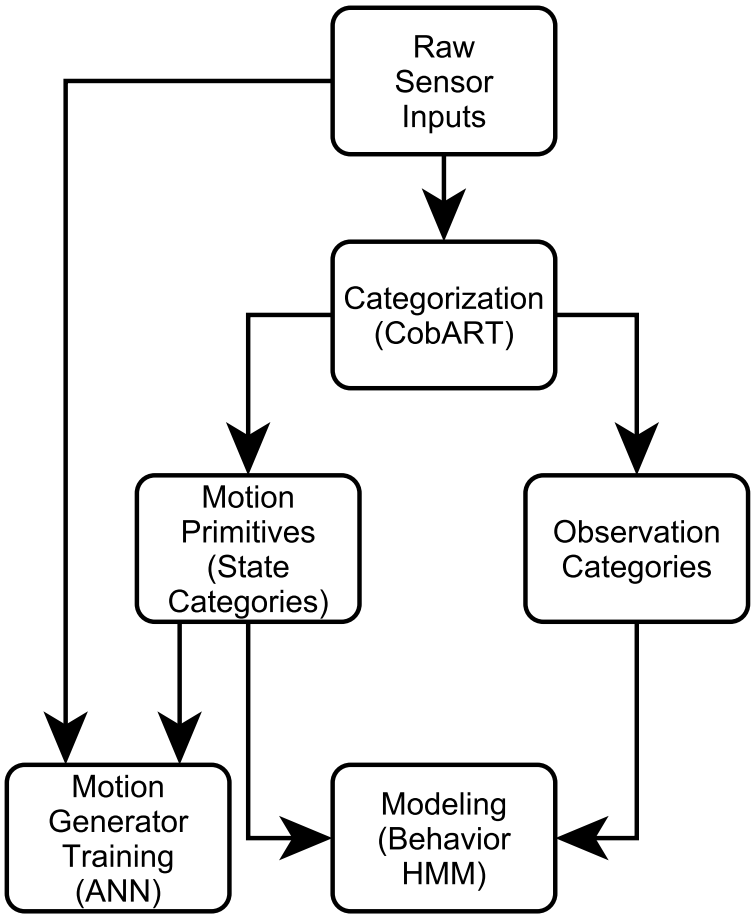


Figure 4.2: Behavior categorization and modeling process [4]

of CobART as

$$o = CobART(P_o, X_{T=I_o})$$

where P_o is the set of CobART parameters and I_o is the length of observation sequence used in observation categorization. In addition to observation category, motion primitive category S is also defined as a function of CobART as

$$S = CobART(P_s, X_{T=I_s})$$

where P_s is the set of CobART parameters and I_s is the length of observation sequence used in motion primitive categorization.

4.2 Behavior Modeling

Once the categorization of motion primitives and observations completed, relations among the motion primitives should be constructed. In the behavior modeling phase, relations and transition conditions among the motion primitives are learned in a higher level model. Furthermore, a motion generator for each motion primitive is trained to associate sensor inputs to motor commands. In Figure 4.2, behavior categorization and behavior modeling process is presented.

A behavior is considered as a collection of action sequences. Thus, it is assumed that motion primitives

can be modeled as a Markov process which is a state machine in which the system stays in the active state and then transits to a different state probabilistically. Markov process does not answer the question of *what is the exit condition of a state?* As an answer to this question, observations are used as exit conditions of the states in this study. If the exit condition (i.e. exit observation) is encountered during the execution of a state, the model goes to the next state in the sequence. Thus, the use of hidden Markov model (HMM) instead of Markov processes is a more proper solution to behavior modeling problem. HMM (see Section 3.3) is a statistical tool to model generative sequences where the modeled system is a Markov process with hidden states. States of the model are directly visible to the observer in a regular Markov model. However, states are not directly visible, but the observations dependent on the state are visible in a hidden Markov model [4].

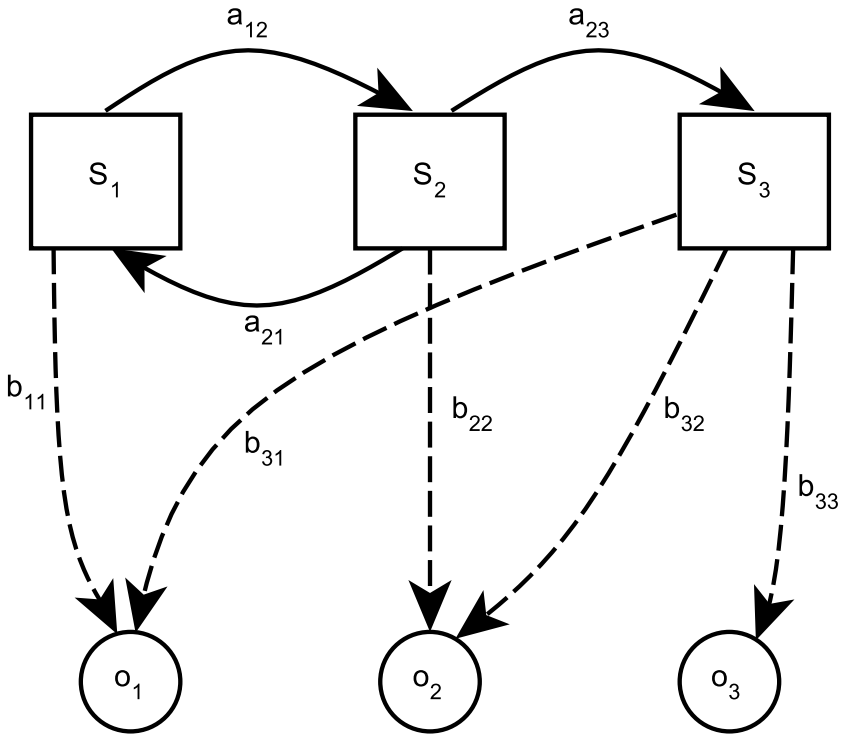
In this study, a modified version of HMM, called as Behavior-HMM, is used to model the relations between the motion primitives. Observation-to-observation transitions are added to Behavior-HMM to be used in the behavior generation phase. Also, motion generators are added for each motion primitive in a Behavior-HMM. Figure 4.3 show samples of HMM and Behavior-HMM.

The Behavior-HMM is defined as a 7-tuple denoted by $\{S, G, O, A, B, C, \pi\}$ [4]:

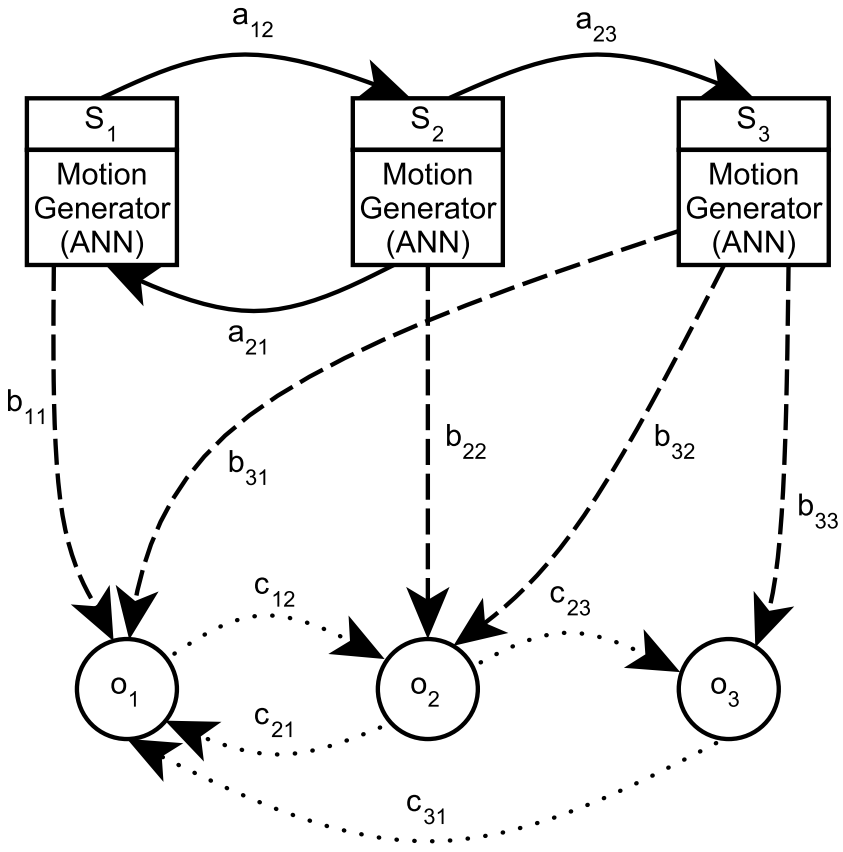
- $S = \{S_1, \ldots, S_N\}$ is the set of motion primitives where the motion primitive at time t is q_t.
- $G = \{G_1, \ldots, G_N\}$ is the set of motion generators defined for each motion primitive.
- $O = \{o_1, \ldots, o_M\}$ is the set of observation categories where the observation category at time t is o_t.
- $A = \{a_{ij}\}$ is the motion primitive transition probability distribution where $a_{ij} = P[q_{t+1} = S_j \mid q_t = S_i]$, $1 \leq i, j \leq N$.
- $B = \{b_j(k)\}$ is the observation category probability distribution in motion primitive j where $b_j(k) = P[o_k \text{ at } t \mid q_t = S_j]$, $1 \leq j \leq N, 1 \leq k \leq M$.
- $C = \{c_{ij}\}$ is the observation category transition probability distribution where $c_{ij} = P[o_{t+1} = O_j \mid o_t = O_i]$, $1 \leq i, j \leq M$.
- $\pi = \{\pi_i\}$ is the initial motion primitive distribution where $\pi_i = P[q_1 = S_i]$, $1 \leq i \leq N$.

Set of motion primitives S, motion generators G, and observation categories O are determined in the categorization phase. The training data are processed to determine all transition probabilities (A, B, C, and π) in a Behavior-HMM. A transition is added from each state to itself for self-transition. During the Behavior-HMM construction, when a new transition occurs from the current state to another, count of this transition is increased by one. After processing training data, each transition probability from the current state is calculated by dividing the number of specific transition to the total number of all transitions from the current state. As an example, Table 4.3 shows a transition probability calculation for state S_1. Consider that state S_1 has transitions to the states S_2 and S_3, and o_1 and o_2 are the observations in S_1. Transition probabilities are calculated by using the number of transitions in the training data as shown in the table. All state-to-state (A), state-to-observation (B), and observation-to-observation (C) transition probabilities are calculated by using the same process. Initial motion primitive distribution (π) is also calculated according to same process using training data. Density of a motion primitive in the training data determines its initial probability [4].

Motion primitive transition model constructed as Behavior-HMM helps robots to generate motion primitive sequences in order to achieve a given task. On the other hand, correlations of sensory data



(a)



(b)

Figure 4.3: (a) Hidden Markov model (HMM) and (b) Behavior hidden Markov model (Behavior-HMM) [4]

Table4.3: Sample of transition probabilities calculation [4]

	S_1	S_2	S_3	o_1	o_2
Number of transitions from S_1 to	1	4	3	2	5
Transition probabilities	$a_{11} = 1/8$	$a_{12} = 4/8$	$a_{13} = 3/8$	$b_{11} = 2/7$	$b_{12} = 5/7$

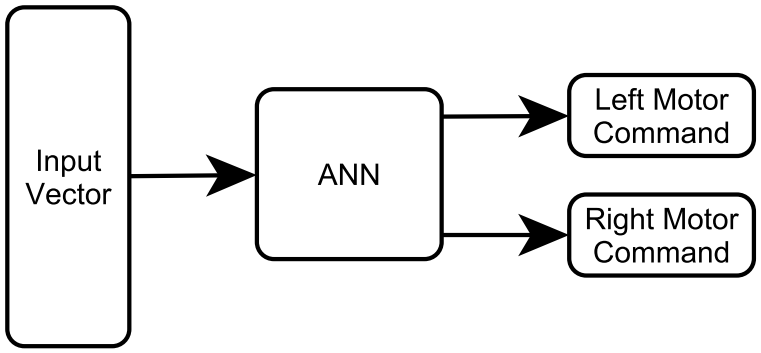


Figure 4.4: ANN inputs and outputs

and motor commands should be learned in order for the robot to perform those motion primitives. For this purpose, a motion generator is used to learn generation of motor commands for each motion primitive. Motion generators use artificial neural networks (ANNs) (see Section 3.4) to learn the generation of continuous motor commands. ANNs can learn the complex relations between the real-valued inputs and outputs. Error backpropagation [42] (see Section 3.4.2.1) learning algorithm is used to train the neural networks. It is a simple and general method to train multilayer neural networks. It uses *gradient descent* to find the minimum error rate between the target values and network outputs. Error backpropagation is a kind of supervised learning method, but the way it is used in this study does not need a supervisor. Robot's sensory data is used as inputs of network and motor commands are expected as target values. Thus, the training of neural network is performed without the help of a supervisor. As shown in Figure 4.4, the inputs of network is the raw sensory data and outputs are left and right motor commands. Input vector containing consecutive sensor readings are fed into the input layer of neural network. In each iteration, input vector is shifted by one and shifted vector is presented as new input to network for the next iteration. By using shifted inputs, network learns sequence of sensor readings as a pattern instead of instantaneous data. This improves the learning capabilities of motion generators [4].

4.3 Behavior Generation

In behavior generation phase, the aim is to generate the most likely motion primitive sequence to perform the requested behavior. Requested behavior is given to the model as goal observation. First, most likely observation sequence is generated from the current observation to goal observation by most likely path algorithm. This observation sequence is generated by the help of Behavior-HMM's observation-to-observation transition probabilities as explained below. Then, Viterbi algorithm generates the most likely motion primitive sequence by using the previously generated observation sequence [4].

Behavior generation phase can be stated as a two-step problem. The problems and proposed solutions are given below [4]:

Problem 1: Given the goal observation category o_g and Behavior-HMM, find the most likely observation category sequence $O_{goal} = [o_1 o_2 \dots o_g]$ from the current observation category o_1 to the goal observation category o_g.

Problem 2: Given the observation category sequence O_{goal} and Behavior-HMM, find the corresponding most likely motion primitive sequence $S_{goal} = [S_1 S_2 \dots S_g]$.

Solution to Problem 1: The first problem is solved by developing a most likely path algorithm which is a modified version of Dijkstra's shortest path algorithm [45]. Shortest path logic is changed to most likely path by reversing the comparators in the algorithm. Logarithms of transition probabilities are used as edge costs to prevent floating-point underflow problems. The algorithm constructs a tree from starting node to every other nodes in the network to find the most likely path. At the end, it generates the most likely path from the current observation category to the goal observation category.

Solution to Problem 2: Viterbi algorithm [6] (see Section 3.3.1) is used to solve the second problem by generating the most likely motion primitive sequence using the observation sequence generated in the previous step. This algorithm is a kind of dynamic programming algorithm and finds the most likely sequence of hidden states for the given observation sequence.

The behavior generation algorithm is explained in Algorithm 1 as a pseudo code. The process of behavior generation is presented in Figure 4.5. After generating observation category sequence O_{goal} and motion primitive sequence S_{goal}, behavior generation algorithm starts to execute the first motion primitive in the sequence. If the exit condition (i.e. exit observation) is encountered during the execution of a motion primitive, the model goes to the next motion primitive in the sequence. Figure 4.6 shows the motion primitive transitions using observations as transition conditions. This process is repeated until the goal observation category o_g is reached. The algorithm checks if a new state sequence is needed in each cycle. This check is used to detect any problems during the behavior generation. The robot may generate a wrong command or it may lose its direction because of environmental changes. This check helps to improve the quality of behavior generation.

Algorithm 1 Behavior Generation

{P_o: CobART parameters used in observation categorization}
{I_o: Observation sequence length in observation categorization}
{$X^c_{T=I_o}$: Current observation sequence}
{$X^g_{T=I_o}$: Goal observation sequence}
{Find current and goal observation categories}
$o_c \leftarrow CobART(P_o, X^c_{T=I_o})$
$o_g \leftarrow CobART(P_o, X^g_{T=I_o})$
{Generate observation and motion primitive sequences}
$O_{goal} \leftarrow MostLikelyPath(BehaviorHMM, o_c, o_g)$
$S_{goal} \leftarrow Viterbi(BehaviorHMM, O_{goal})$
{Start from first primitive behavior}
$S_c \leftarrow S_{goal}[0]$
while $o_c \neq o_g$ **do**
  {Generate motor command and send to robot}
  $motorCommand \leftarrow ANNHandler(S_c, X^c_{T=I_o})$
  $robot \leftarrow motorCommand$
  {If the exit observation of the current state is encountered, go to next state}
  if $O_{goal}[S_c] = o_c$ **then**
    $S_c \leftarrow$ next state in S_{goal}
  end if
  {Find current observation category}
  $o_c \leftarrow CobART(P_o, X^c_{T=I_o})$
  {Generate observation and motion primitive sequences}
  $NewO_{goal} \leftarrow MostLikelyPath(BehaviorHMM, o_c, o_g)$
  $NewS_{goal} \leftarrow Viterbi(BehaviorHMM, NewO_{goal})$
  {Check whether a new state sequence is needed}
  if $S_c \neq$ first state of $NewS_{goal}$ **then**
    $S_{goal} \leftarrow NewS_{goal}$
    $S_c \leftarrow$ first state of S_{goal}
    $O_{goal} \leftarrow NewO_{goal}$
  end if
end while

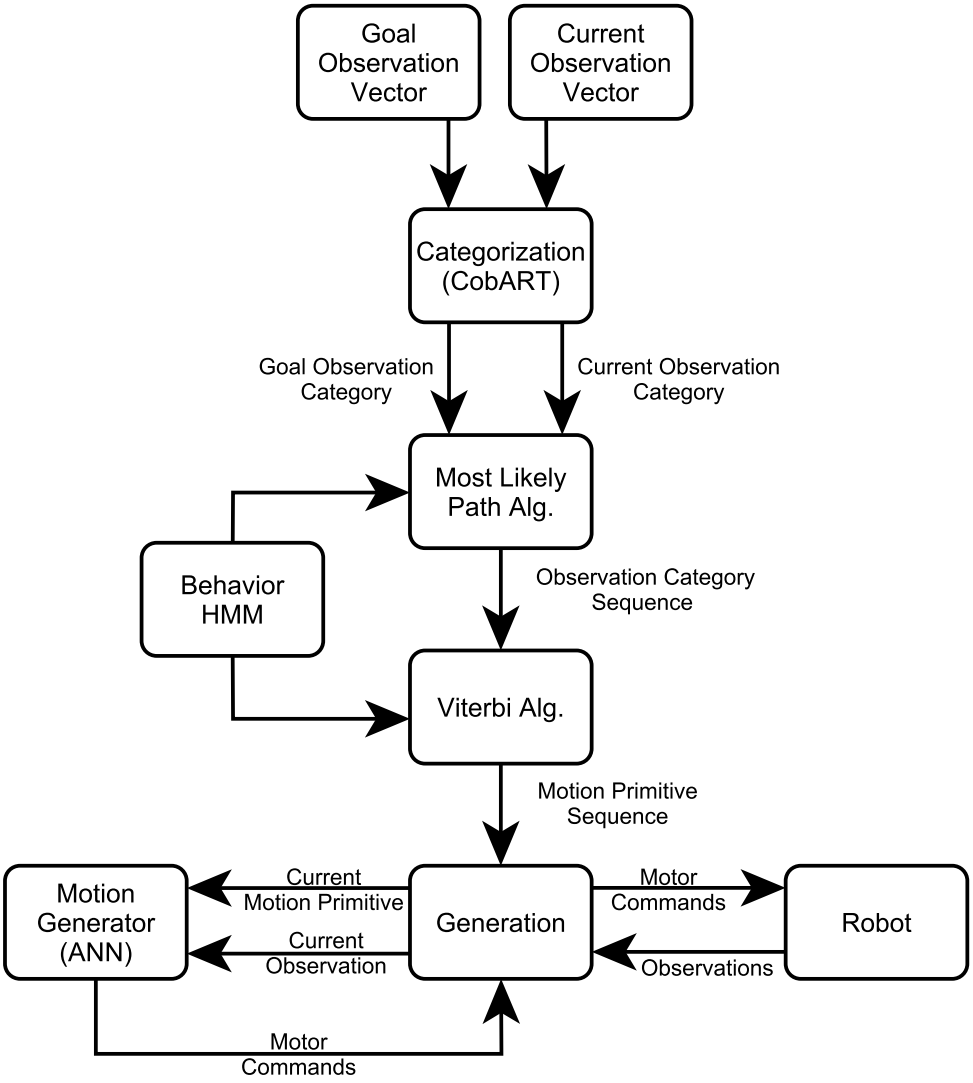


Figure 4.5: Behavior generation process [4]

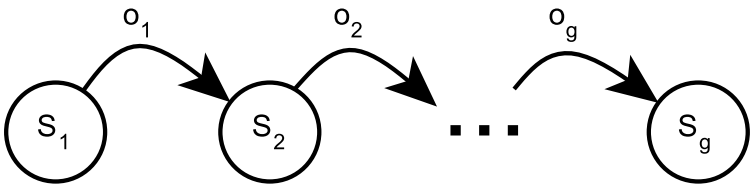


Figure 4.6: State transitions during behavior generation [4]

CHAPTER 5

COMPLEX-BEHAVIOR LEARNING MODEL (CBLM)

Behaviors having a complex goal that can be decomposed into several basic goals are considered complex behaviors. Thus, a complex behavior may consists of several simple or complex behaviors. For example, grasping an object which is located at the backside of a robot may need turn, approach, and grasp behaviors to be performed respectively. Grasp, turn, and approach behaviors are independent behaviors, but they can be modeled by CBLM as a single complex behavior. Different complex behaviors may contain common components. Applying simple-behavior learning model on complex behaviors results in a very complex behavior model that has repeated components in it. Moreover, this complex model may have performance problems and longer training times. Thus, modeling of complex behaviors need a different approach than simple behaviors. Driven by these ideas, CBLM which hierarchically integrates common components to enhance the learning capabilities is developed for complex behaviors [4].

The structure of the complex-behavior learning model is presented in Figure 5.1. As shown in the figure, its structure is similar to SBLM and it has the same phases of *behavior categorization, behavior modeling, and behavior generation*.

The model needs some information about the already learned behaviors in order to construct the Behavior-HMM. This information is used to calculate behavior-to-behavior, behavior-to-observation, and observation-to-observation transition conditions. Thus, CBLM needs a new training data to construct the relationships between the behaviors. The training data includes sensory inputs as in SBLM. But in CBLM training, sensory data are tagged by corresponding behaviors to indicate the relation between sensory data and the behaviors. This data is used in behavior categorization and behavior modeling phases.

Behavior categorization phase uses the tagged sensory data to extract the components of new complex behavior. These components are the already learned simple or complex behaviors. Hence, there is no motion primitive categorization in this phase. But, the observation categorization is still needed in order to determine observation sequence from current observation to goal observation in behavior generation phase. Specifying the target tasks as goal observation is the key feature of the proposed models. Thus, in order to generate complex-behaviors, observation-to-observation transitions must be constructed by the help of observation categorization. The process of behavior categorization and modeling is presented in Figure 5.2.

In complex-behavior modeling phase, first already learned components are used as states of Behavior-HMM. In Figure 5.1, these behaviors are sampled by the $SBLM_1$, $CBLM_2$ and $SBLM_3$. Then, probabilities of behavior-to-behavior, behavior-to-observation, and observation-to-observation transitions are determined as in simple-behavior learning model. In CBLM, motor command generation task is

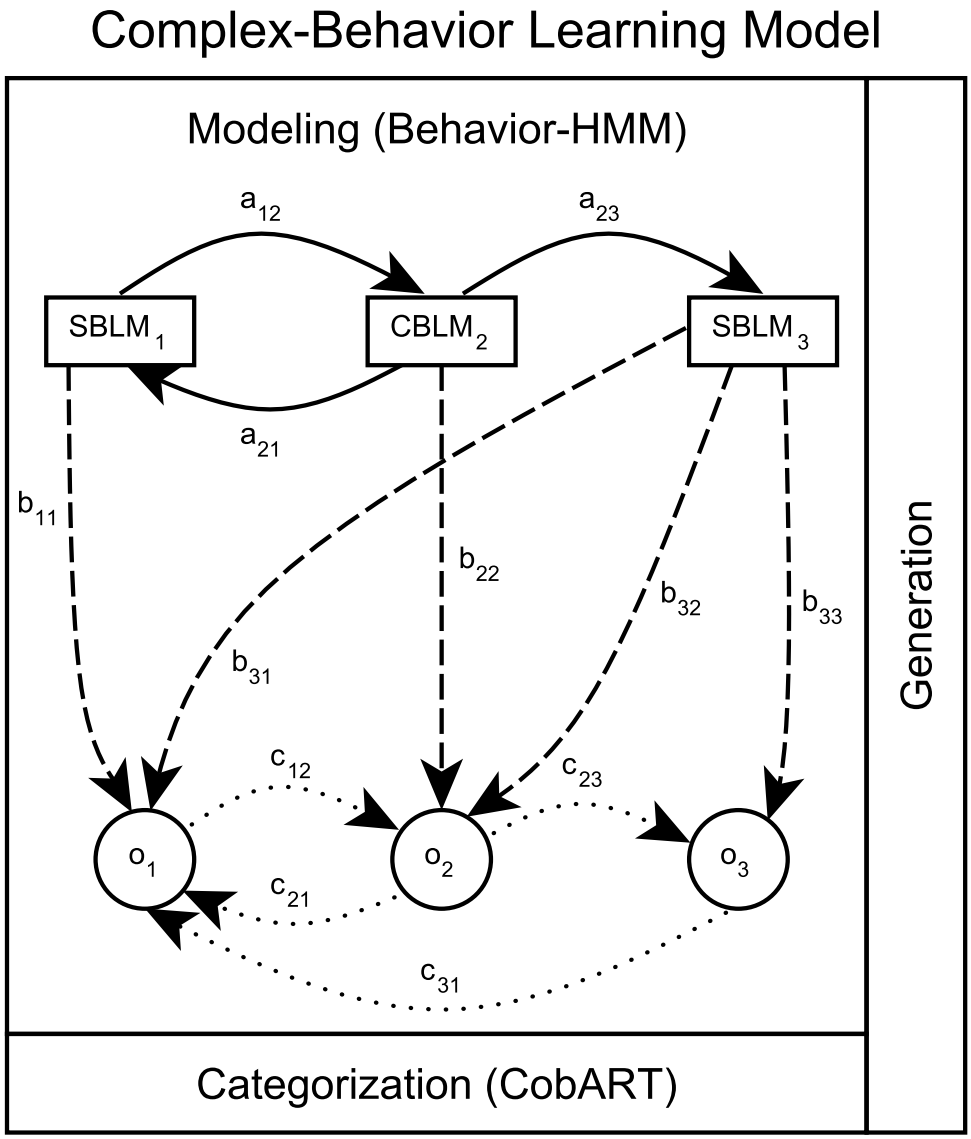


Figure 5.1: Components of complex-behavior learning model. Previously learned simple or complex behaviors correspond to states in a Behavior-HMM [4].

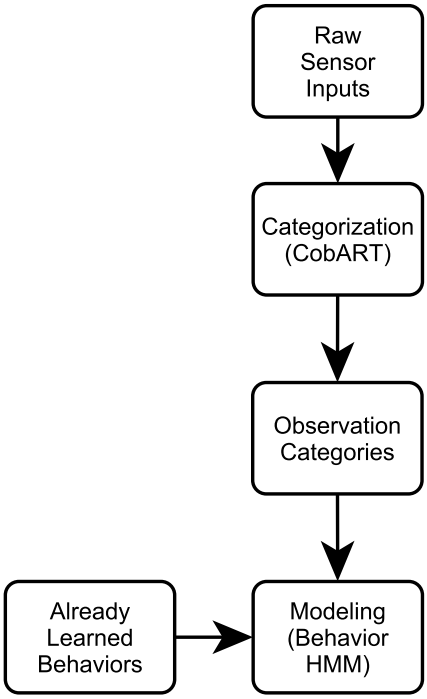


Figure 5.2: CBLM behavior categorization and modeling process

left to the already learned behaviors. Thus, complex-behavior model does not have motion generator training phase. During complex behavior generation, active behavior is responsible to generate the appropriate motor commands. If it is a complex behavior, it also delegates the command generation to its sub-behaviors [4].

Complex behavior generation phase is similar to simple behavior generation phase. First, observation sequence is generated by most likely path algorithm based on the current and the goal observations. Then, Viterbi algorithm generates the sequence of simple or complex behavior components in the highest probability to achieve the given task. After that, command generation starts from the first behavior in the sequence. Note that corresponding intermediate observation in the observation sequence is given to the low level behavior as goal observation. Each low level behavior executes its behavior generation algorithm according to given intermediate goal observation. When execution of a behavior is completed, the system proceeds to the next behavior in the sequence. This process is repeated until all behaviors are executed and the goal observation category is reached [4].

The process of behavior generation can be explained with an example. Suppose that a CBLM has three low level behaviors like *turn*, *approach*, and *grasp*. The robot is located according to an object with 70° rotational angle and 0.8 m distance. The current observation o_c and the goal observation o_g is defined as below:

$$o_c : \{angle = 70°, distance = 0.8\,m, grasp\ status = no\ grasp\}$$
$$o_g : \{angle = 0°, distance = 0\,m, grasp\ status = grasp\}$$

The model, first, generates observation sequence from current observation to goal observation like

$$O_{goal} = [o_1 o_2 o_g]$$

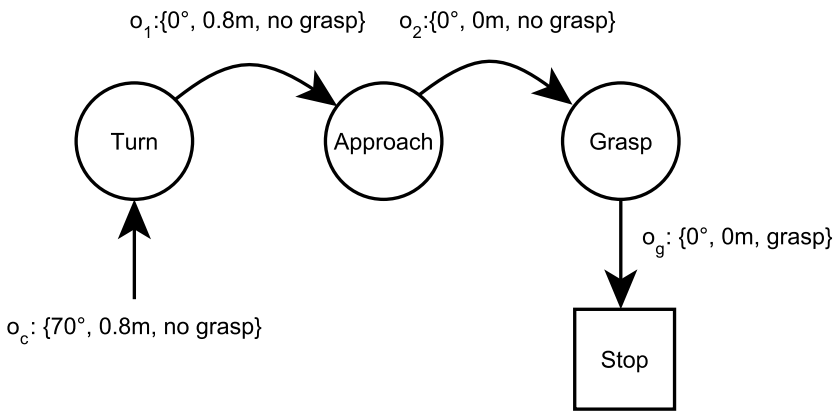


Figure 5.3: Example behavior transitions using observation sequence

where o_1, o_2, and o_g is determined by the model as follows:

$$o_1 : \{angle = 0^\circ, distance = 0.8\,m, grasp\ status = no\ grasp\}$$
$$o_2 : \{angle = 0^\circ, distance = 0\,m, grasp\ status = no\ grasp\}$$
$$o_g : \{angle = 0^\circ, distance = 0\,m, grasp\ status = grasp\}$$

Behavior sequence is generated according to observation sequence O_{goal} as below:

$$S_{goal} = [Turn, Approach, Grasp]$$

Behavior generation starts with *turn* behavior until the observation o_1 is encountered. Then, *approach* behavior is performed to see the observation o_2. Finally, *grasp* behavior is applied to reach the goal observation o_g. Figure 5.3 shows the behavior transitions using observation sequence for this example. Note that if the initial position of the robot is 0° rotational angle and 0.8 m distance, then the behavior sequence shall be generated as $S_{goal} = [Approach, Grasp]$. Similarly, the behavior sequence shall be generated as $S_{goal} = [Turn, Grasp]$, if the robot is located as 70° rotational angle and 0 m distance.

CHAPTER 6

EXPERIMENTAL EVALUATION

Webots [8] (see Section 3.5) simulator is used as a test platform for the proposed learning models. As a test robot *Khepera* [7] (see Section 3.6) is selected. It has 8 light sensors, 8 distance sensors, 2 motor outputs and 2 encoder inputs for left and right differential motors.

Turn to the object and *approach the object* behaviors are selected as test behaviors for SBLM. Furthermore, CBLM is experimented on the combination of the previously learned *turn to the object* and *approach the object* behaviors. Test and training data for the experimentation of selected behaviors are collected by the help of a test data collector application. Smooth motion commands for *turn* and *approach* behaviors are generated using third order polynomial s-curve algorithm [44] (see Section 3.7). This algorithm segments the requested motion into multiple pieces and uses different equations and limited jerk[1] for each segment to generate a smooth motion.

The test data collector application records the following data into a log file at 16ms intervals: *time, distance, linear velocity, angle, and angular velocity*. These variables are selected by considering the test behaviors (*turn* and *approach*). But the proposed models are capable of learning different behaviors using different set of sensor readings on alternative robot platforms [4].

The performance of a machine learning system is dependent on the distribution of the input data. Thus the input data should be normalized to be used in a learning process. In this study, all input data are normalized into $[-1 : +1]$ interval before fed into a network. Also, inputs of ANN and CobART are filtered using low-pass second-order filters[2] [46] in order to decrease the effect of noise [4].

Success rate of the experiments are measured by Euclidean distance method (EDM). It calculates the similarity of two vectors by comparing the distances between them. In this study, EDM is used to compare the generated and the training motor commands. Output of EDM is a relative value which depends on the properties of compared vectors. Thus, EDM results of the similar types of vectors are comparable. Following equation is used to compare the training and generated motor command vectors [4].

$$EDM(a,b) = \sqrt{\frac{\sum_{i=1}^{N}(a_i - b_i)^2}{N}}$$

In the equation, a and b are the input vectors and N is the length of the input vectors.

[1] Jerk is the derivative of acceleration with respect to time. In order words, it is the rate of change of acceleration.

[2] A low-pass filter lets low frequency signals and cuts high frequency signals. A second-order filter reduces the amplitude of higher frequencies more steeply.

Table6.1: Sample motion primitive sequences for *turn* behaviors [4]

Angle	Sequence of motion primitives
-115	**3, 4, 1, 2**
-168	7, 13, **3, 4, 1, 2**
86	5, **6, 0**
123	10, 12, **6, 0**

6.1 Behavior Categorization and Modeling Experiments

Categorization and modeling capabilities of the proposed models are tested in some preliminary experiments. These experiments are performed on *turn* behavior. The data for *turn* behavior from different rotational angles are categorized by CobART into motion primitives. Then, the corresponding transition diagram for the motion primitives is constructed.

In the first experiment, a higher vigilance parameter (0.85) is selected to produce finer categories in CobART categorization. Figure 6.1 shows the categorization results of the *turn* behavior with higher vigilance parameter. As shown in the figure, separate categories are generated for different angles. In the experiment, constant velocity (slope of angle vectors) is used during the turn behavior. By this way, we can observe the category similarities for the similar turn behaviors. Figure 6.2 shows the transition diagram of generated motion primitives for this experiment.

In the second experiment, a lower vigilance parameter (0.75) in CobART categorization is used to produce a simple transition diagram. In this simple version, results can be analyzed clearly. Figure 6.3 shows the categorization results of the *turn* behavior with lower vigilance parameter. Figure 6.4 shows the transition diagram of generated motion primitives.

Some sample motion primitive sequences for *turn* behaviors are shown in Table 6.1. The first two rows present the sequences of motion primitives from negative angles and last two rows presents the sequences of motion primitives from positive angles. Note that motion primitives in the beginning of motion sequences are different, because they start from different rotational angles. On the other hand, while performing *turn* behavior, each turn behavior starts to repeat the common motion primitives as soon as the rotational angles approach to each other. As presented in Table 6.1, motion primitives {6, 0} and {3, 4, 1, 2} are common in *turn* behaviors from the same directions.

CobART produced proper categories for *turn* behavior in preliminary categorization experiments. The experiments also showed that the relations and transition probabilities among motion primitives are constructed properly.

6.2 ANN Learning Experiments

The learning performance of artificial neural network (ANN) is critical in behavior generation experiments. In order to find the best neural network parameters for behavior generation, some experiments are conducted. The effect of momentum, learning rate, and hidden layer organization in neural network training is investigated in this preliminary experiments. The learning rate determines the speed of learning. If it is larger, converge will be fast. But larger learning rates may cause oscillations in the learning curve. The use of momentum term in network training lets the use of larger learning rates

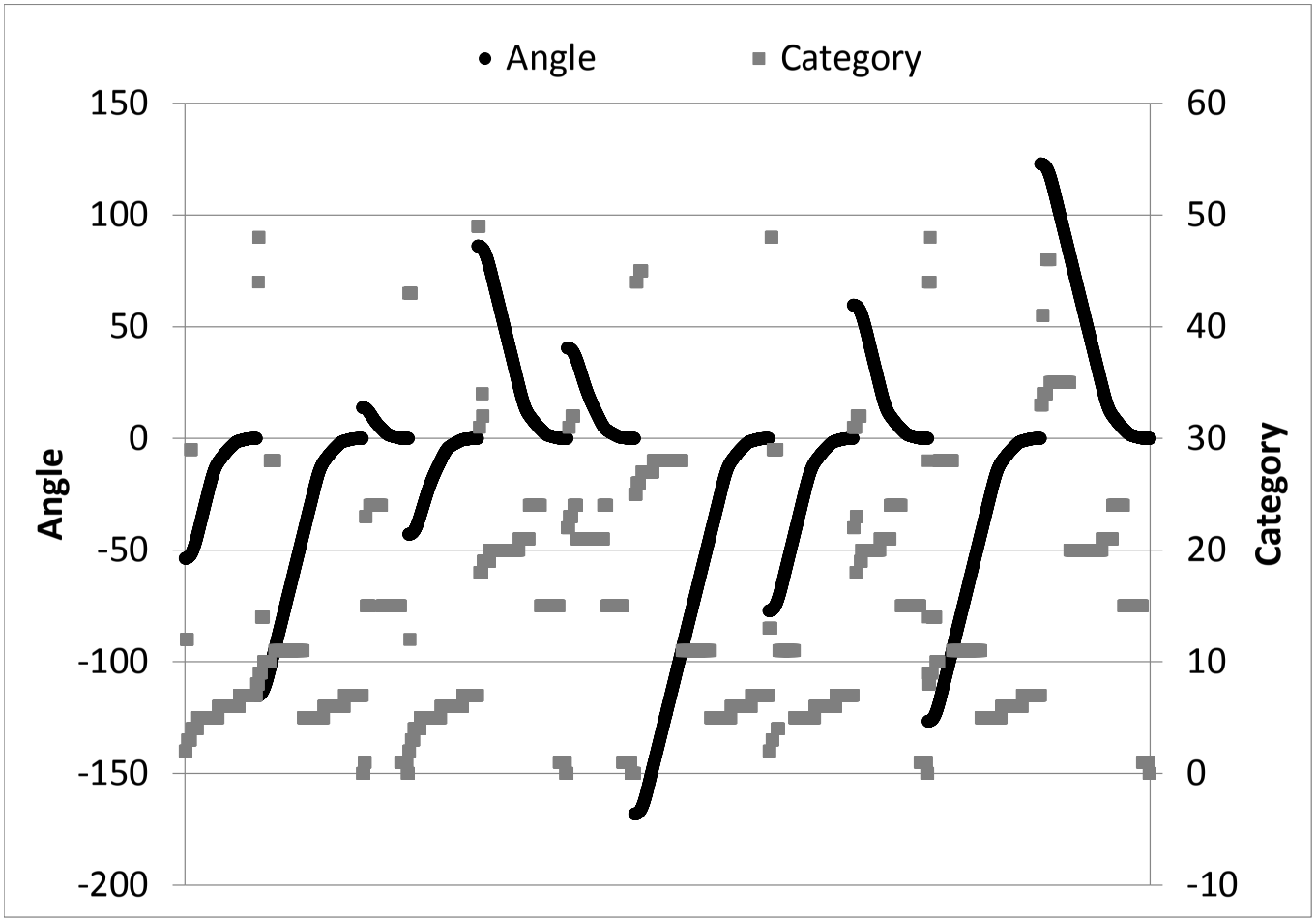


Figure 6.1: Higher vigilance (0.85) *Turn* behavior categorizations in a rotational angles versus category identifiers chart.

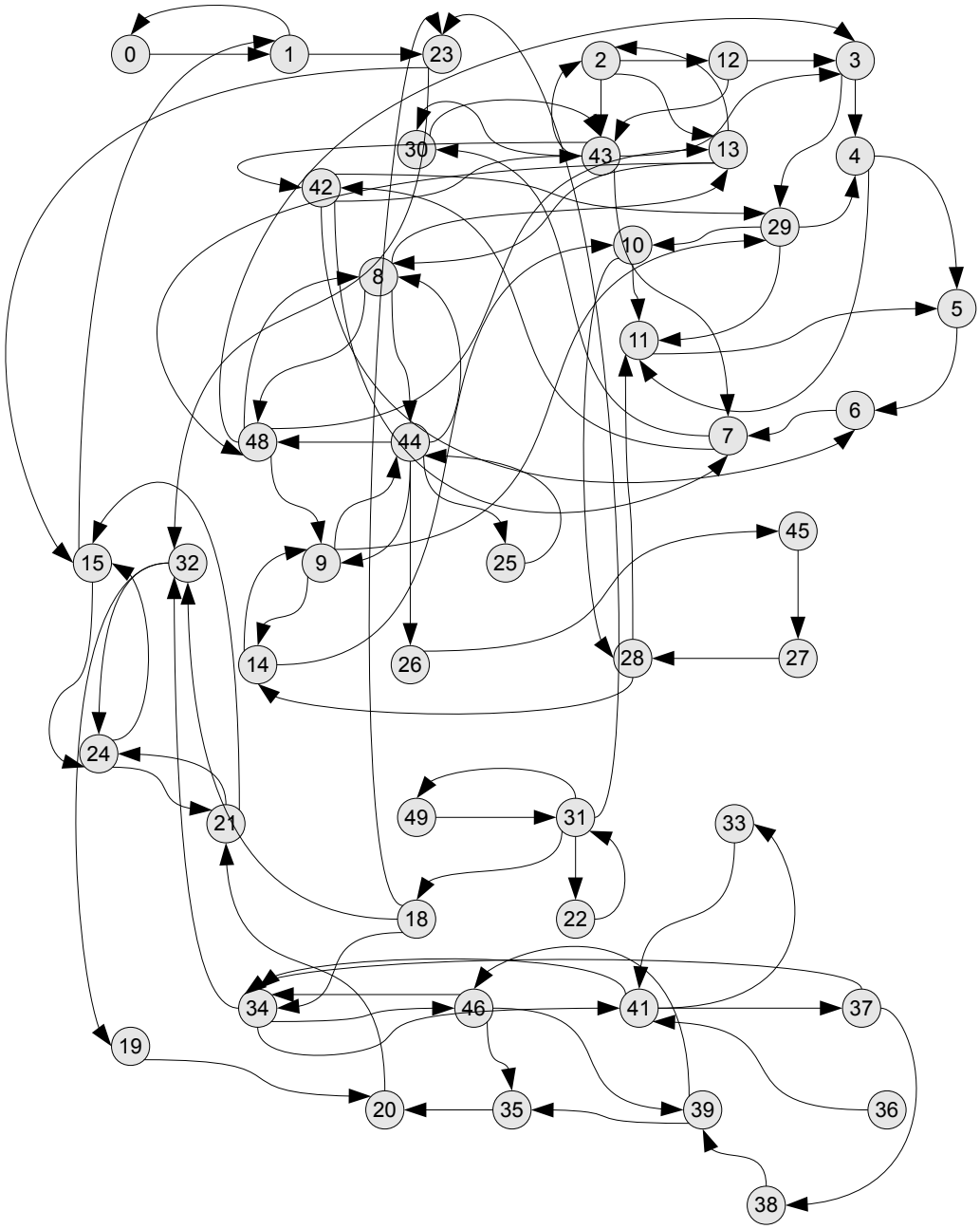


Figure 6.2: Higher vigilance (0.85) *Turn* behavior transition diagram.

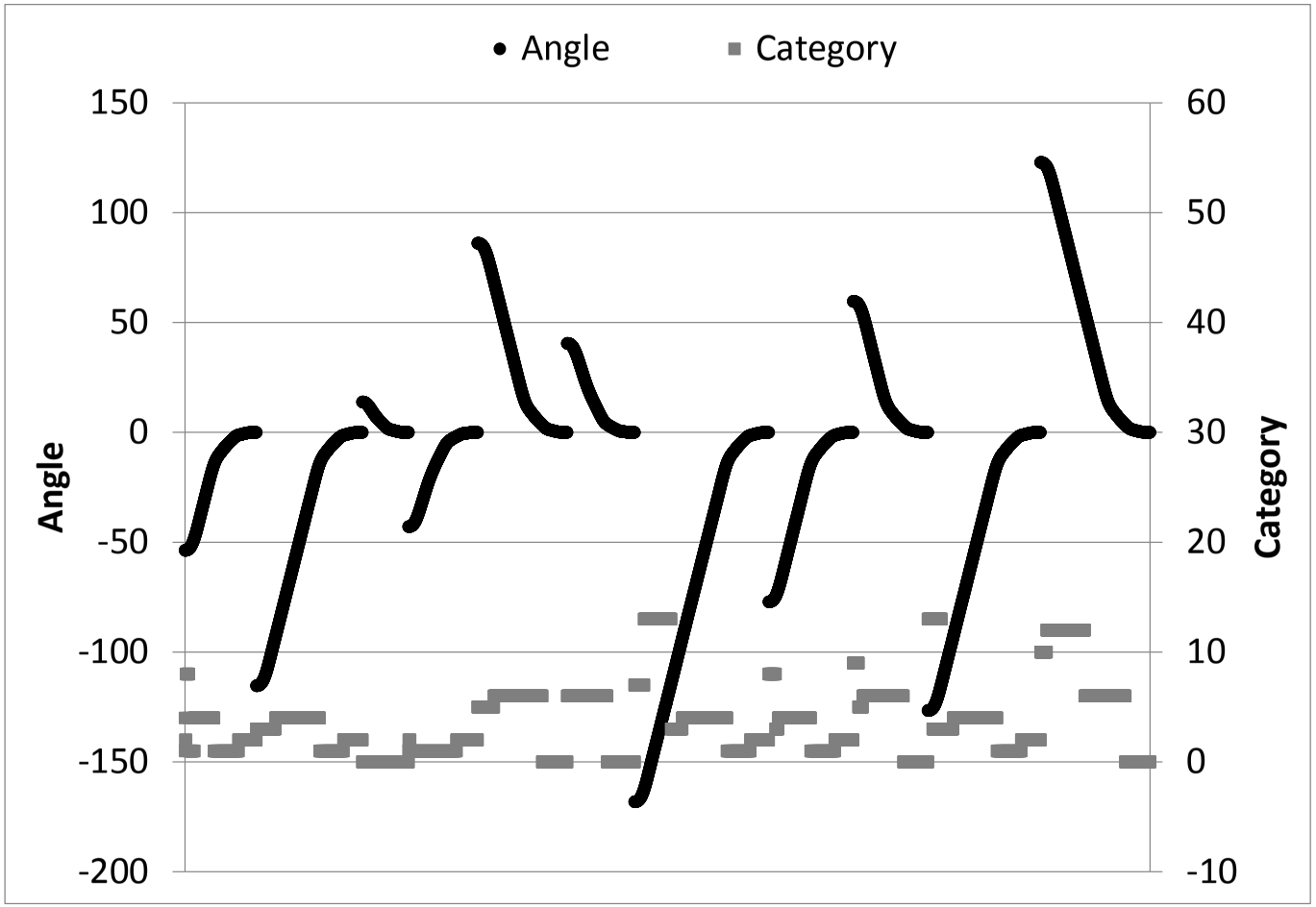


Figure 6.3: Lower vigilance (0.75) *Turn* behavior categorizations in a rotational angles versus category identifiers chart.

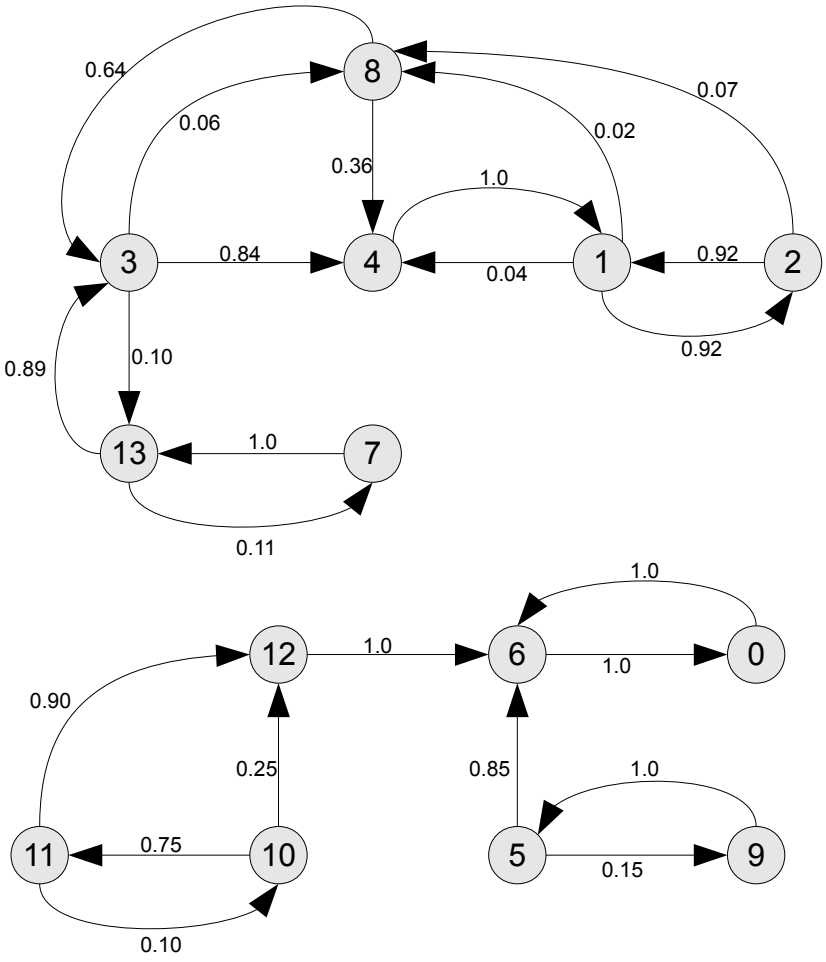


Figure 6.4: Lower vigilance (0.75) *Turn* behavior transition diagram. The upper part shows the transitions for the negative rotation angles, and the lower part shows the transitions for the positive rotation angles [4].

without oscillation. Number of hidden units also effect the learning capabilities of a neural network. A network having too much hidden units cannot generalize input signal properly, but memorize all data even the noise (known as overfitting). On the other hand, if a network has less than enough hidden units, it cannot learn the input signal (known as underfitting).

Experiments are performed on *turning* behavior to analyze the network performance clearly. The same input vector length as in CobART categorization is used in ANN training. Input vector containing 10 consecutive data is fed into the input layer of neural network. Left and right motor commands are expected as the target values in the backpropagation phase. In each iteration, input vector is shifted by one and shifted vector is fed into the network as new input for the next training cycle.

In the implementation of *backpropagation* algorithm, *tanh*() is used as the activation function. *tanh*() is a nonlinear and differentiable function which the outputs are between $[-1 : +1]$. The network inputs and expected outputs are normalized between $[-1 : +1]$ to be processed properly by *tanh*() function. The network is also initialized with the random weights between $[-1 : +1]$. In the *backpropagation* implementation, incremental training scheme is used in which the weights are updated in each iteration.

One of the motion primitives category data is selected as a test data. 20% of the sensory data is used as test data and 80% is used as training data. Test data is not used in training process.

6.2.1 Learning Rate Experiments

In the learning rate experiments, effect of learning rate in neural network training is investigated. Different learning rates are tested and the results are plotted in a test error versus iteration number chart as shown in the Figure 6.5. Fastest convergence is obtained when the learning rate is set to 0.01 and 0.1. Lower learning rates (< 0.01) produced equivalent success, but they needed longer training times. When the learning rate is set too high (0.5), convergence is very slow as shown in the Figure 6.5.

6.2.2 Momentum Experiments

The momentum stabilizes the weight changes in the backpropagation algorithm. It speeds up the converge and helps to avoid the local minimas. Momentum is strictly related with the learning rate and they affect each other. A larger momentum requires a smaller learning rate.

In the experiments, effect of momentum in neural network training is investigated. Different momentums are tested and the results are plotted in a test error versus iteration number chart as shown in the Figure 6.6. All momentum values produced similar success rates. The lowest test error obtained when the momentum is set to 0.8 as shown in the figure.

6.2.3 Hidden Layer/Unit Experiments

A neural network should use sufficient number of hidden units for a good training. It should not be too much (overfitting) or less than enough (underfitting).

In the experiments, different configurations are tested and the results are plotted in a test error versus iteration number chart as shown in the Figure 6.7. All network configurations produced similar success rates. Best result is obtained when 1 hidden layer with 10 units is used in the training as shown in the

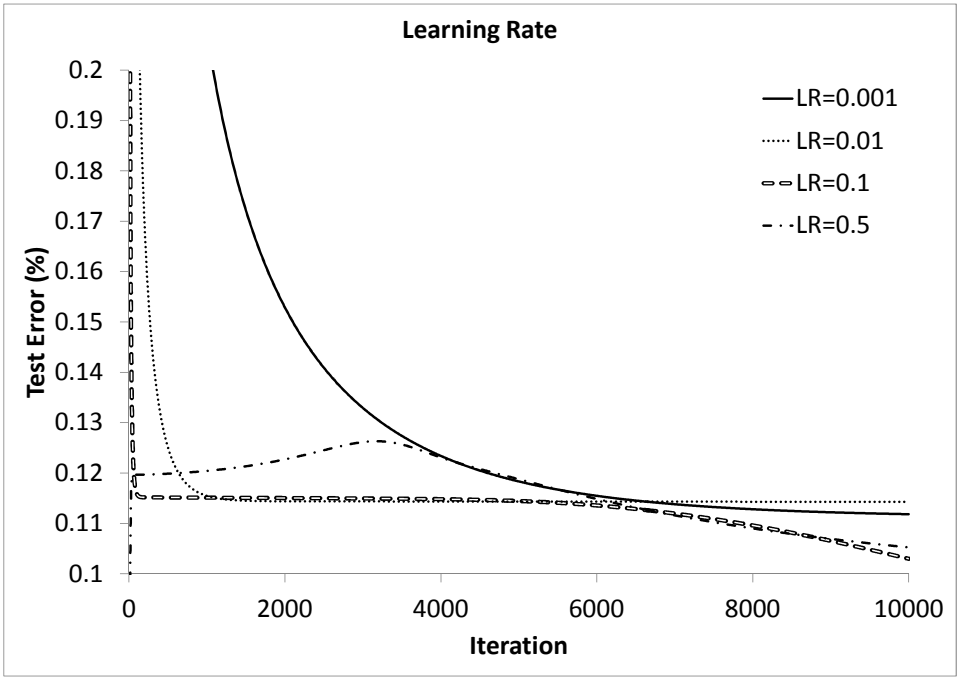


Figure 6.5: Learning rate experiments (Momentum=0.4, Hidden units=1x5). Shows different learning rate performances in a test error versus iteration number chart.

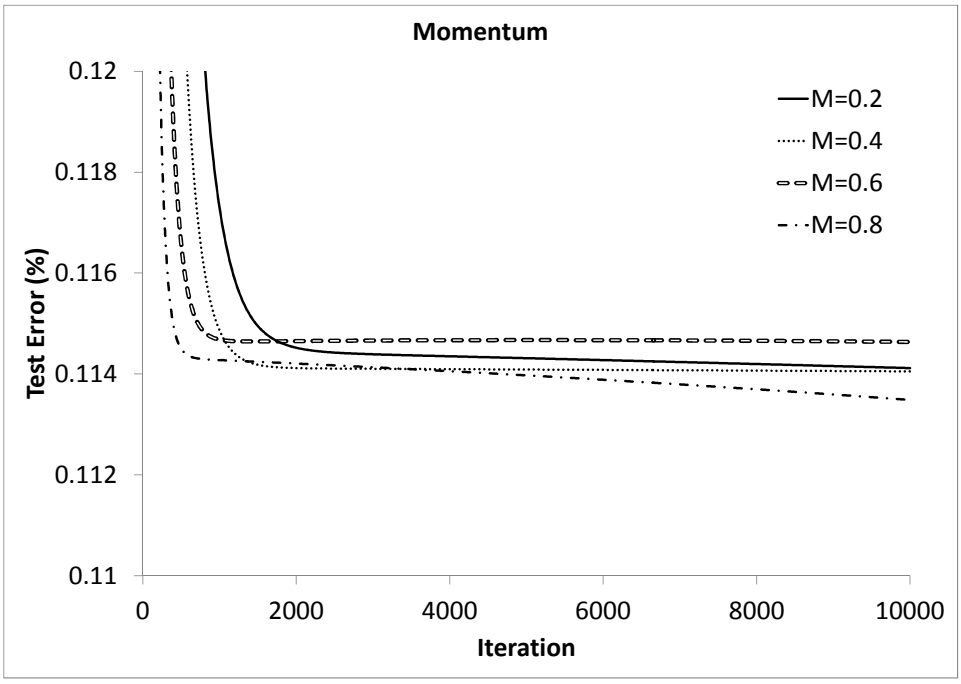


Figure 6.6: Momentum experiments (Learning rate=0.01, Hidden units=1x5). Shows different momentum performances in a test error versus iteration number chart.

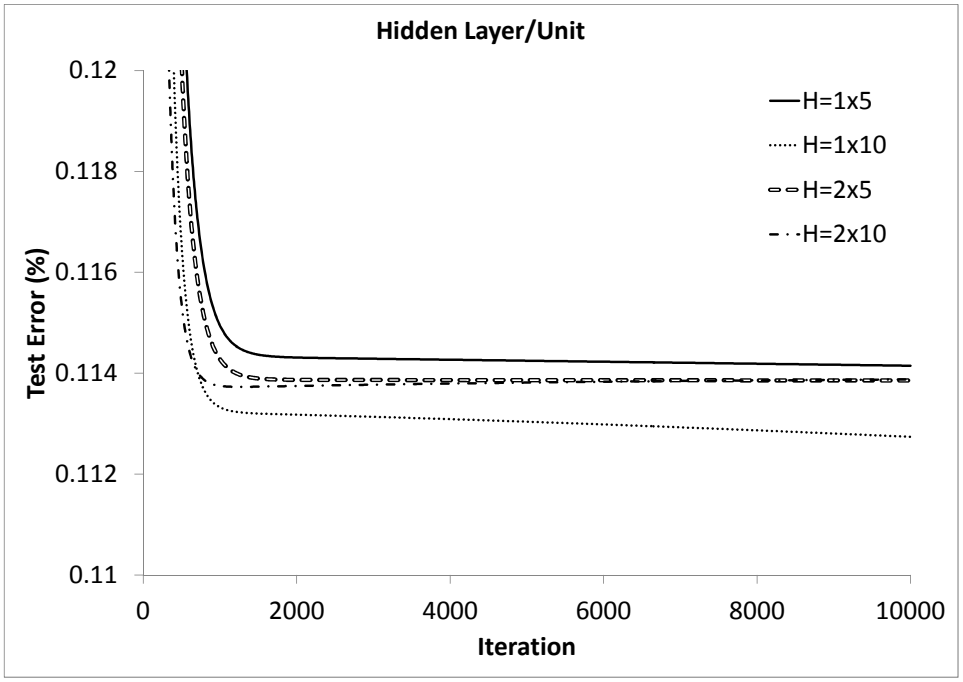


Figure 6.7: Hidden layer/unit experiments (Learning rate=0.01, Momentum=0.4). Shows different hidden layer organization performances in a test error versus iteration number chart.

figure.

6.2.4 ANN Experiments Conclusion

The best ANN performance is obtained when learning rate is set to 0.1 or 0.01, momentum is set to 0.8 and 1 hidden layer with 10 units configuration is used in the network training. But these results are dependent to training data set, number of network inputs and outputs.

In behavior generation tests, optimum neural network parameters should be selected which are supposed to generate optimum performance for all motion primitives. Aforementioned experiments are tested on one of the motion primitives data. So, it is necessary to consider all motion primitives during training parameter selection. Although, setting momentum to 0.8 gives the best result, bigger momentums may lead to oscillation. In the behavior generation experiments, momentum is set to 0.4. Best result is obtained when 1 hidden layer with 10 units configuration is used. But training of large networks takes longer times. By considering the training times, it is used 1 hidden layer with 5 units for one-input networks. The parameters shown in Table 6.2 are used for one, two, and three-input networks.

6.3 Simple-Behavior Generation Experiments

Simple behaviors have one specific and basic goal. In simple-behavior generation experiments, performance of whole simple-behavior learning model is tested including behavior categorization, behavior

Table6.2: Neural network parameters used in behavior generation tests [4]

Parameter	One input network (E.g. distance)	Two inputs network (E.g. distance, linear velocity)	Three inputs network (E.g. distance, angle, linear velocity)
Learning rate	0.01	0.01	0.01
Momentum	0.4	0.4	0.4
Input units	10	20	30
Hidden layers/units	1 layer x 5 units	1 layer x 10 units	1 layer x 15 units

Table6.3: Average performances of 15 different *turn* behaviors using different inputs [4]

Turn behavior	Input: angle	Input: angle, angular velocity
Success rate (%)	90.9	84.2

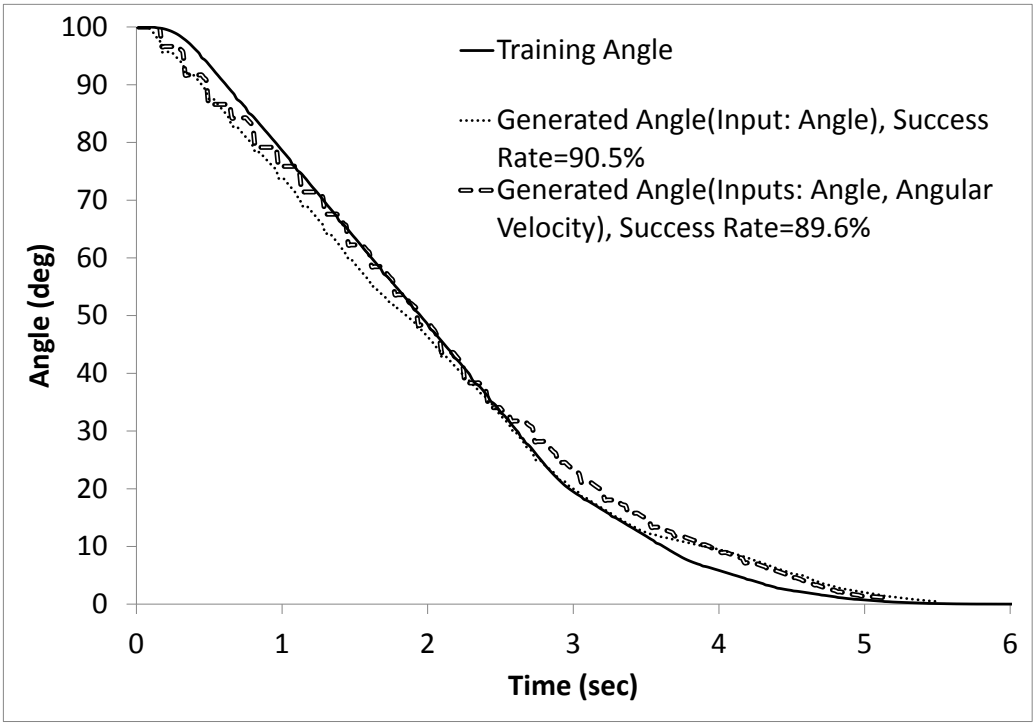
modeling and behavior generation phases. The performance of SBLM is experimented on *turn* and *approach* behaviors. Success rate of each experiment is calculated using Euclidean distance method by comparing the training and the generated motor command vectors throughout the behaviors.

6.3.1 *Turn* Experiments

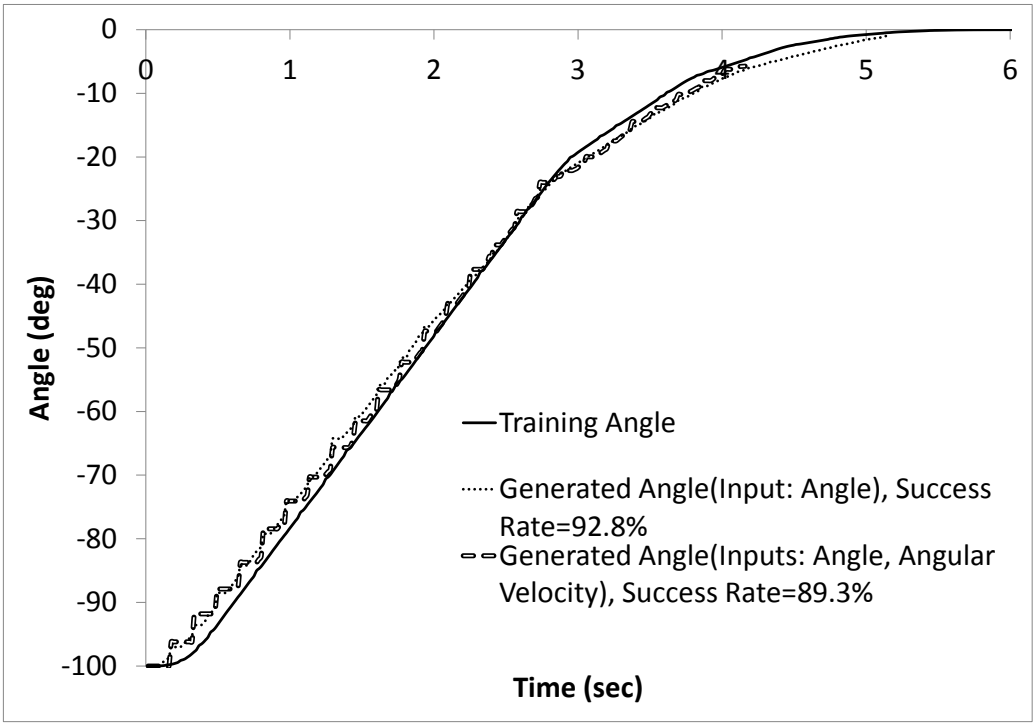
In this section, different *turn* behaviors from -180° to $+180^\circ$ rotational angles are experimented to present capabilities of simple-behavior learning model. Results of the experiments are presented in Figure 6.8 and Table 6.3. Behavior generation performances of the experiments using different set of inputs in categorization and neural network training are compared. The first experiment uses only *angle* vector as input to generate *turn* behavior. The second experiment uses *angle* and *angular velocity* as inputs to the model.

Figure 6.8 shows the charts of training and generated angles for sample *turn* behaviors (see Appendix B.1.1 for more *turn* experiments). *Turn* behavior starts with an acceleration up to an angular velocity level (see slope of angle vector for angular velocity), then keeps this constant velocity for a while, and decelerates and stops at the target angle. Plotted curves of training and generated angles in the figure show that the model can generate this pattern successfully. Average performances in Table 6.3 confirm that SBLM learns and performs *turn* behavior effectively.

Performance of the second experiment (uses *angle* and *angular velocity* as inputs) is lower than the first experiment (uses only *angle* as input). This implies that angular velocity has a negative effect on the performance of *turn* behavior. Angular velocity is the derivation of rotational angle and it is calculated using the rate of rotational angle change in time. Thus, noise in angle data may have more impact on the velocity calculation. Moreover, range of velocity is narrower than the range of rotational angle in these experiments. This makes velocity data more sensitive to the noise effect. Therefore, angular velocity does not provide significant information to the model, because it is derived from the angle. CobART network uses DCM which uses inputs' derivations in score calculation, hence the model can already deduce velocity information implicitly during the categorization phase [4].



(a)



(b)

Figure 6.8: *Turn* behaviors from different angles using different inputs. Shows *turn* behaviors from (a) 100° and (b) -100° rotational angles in a time versus angle chart [4].

Table6.4: Average performances of 15 different *approach* behaviors using different inputs [4]

Approach behavior	Input: distance	Input: distance, angle	Input: distance, angle, linear velocity
Success rate (%)	73.9	82.6	73.7

6.3.2 *Approach* Experiments

In this section, different *approach* behaviors from 0 to 1 m distances are experimented to present capabilities of simple-behavior learning model. Results of the experiments are presented in Figure 6.9 and Table 6.4. Behavior generation performances of the experiments using different set of inputs in categorization and neural network training are compared. The first experiment uses only *distance* vector as input to generate *approach* behavior. The second experiment uses *distance* and *angle* variables as inputs. In the third experiment, *distance, angle,* and *linear velocity* are used as inputs to the model.

Figure 6.9 shows training and generated distance graphs for sample *approach* behaviors (see Appendix B.1.2 for more *approach* experiments). *Approach* behavior starts with an acceleration up to a linear velocity level (see slope of distance vector for linear velocity), then keeps this constant velocity for a while, and decelerates and stops at the target position. Plotted curves of training and generated distances in the figure show that the model can generate this pattern successfully. Average performances in Table 6.4 confirm that SBLM learns and performs *approach* behavior effectively.

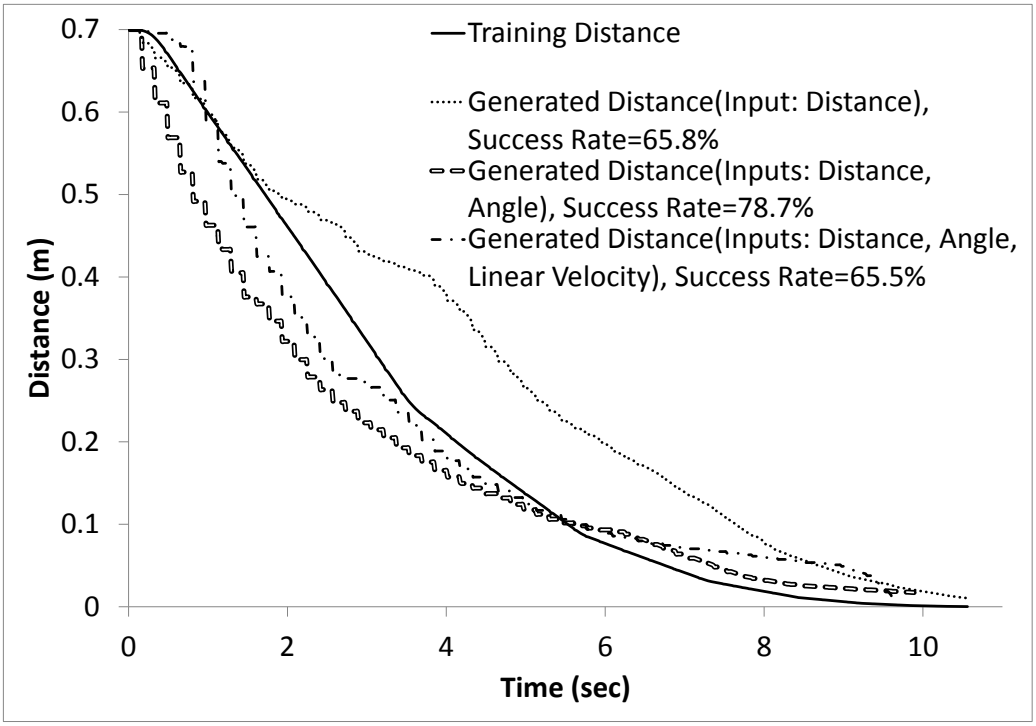
Performance of the third experiment (uses *distance, angle,* and *linear velocity* as inputs) is lower than the second experiment (uses *distance* and *angle* as inputs). This implies that the use of linear velocity has a negative effect on the performance of *approach* behavior as in *turn* behavior experiments. Highest performance is obtained in the second experiment. This shows that use of angle information in *approach* behavior improves the performance. Small variations in left and right motor commands generates a rotational shift during *approach* behavior. In our opinion, use of angle information in addition to distance helps to correct those rotational shifts [4].

Note that *approach* behaviors from closer distances have better success rates as shown in Figure 6.9. Approaching from long distances causes greater accelerations and decelerations and results big differences in consecutive motor commands. Thus, big changes in motor commands in a short time cause differences in left and right motor responses. It leads rotational shifts to the object during approach and causes performance problems in *approach* behaviors from longer distances [4].

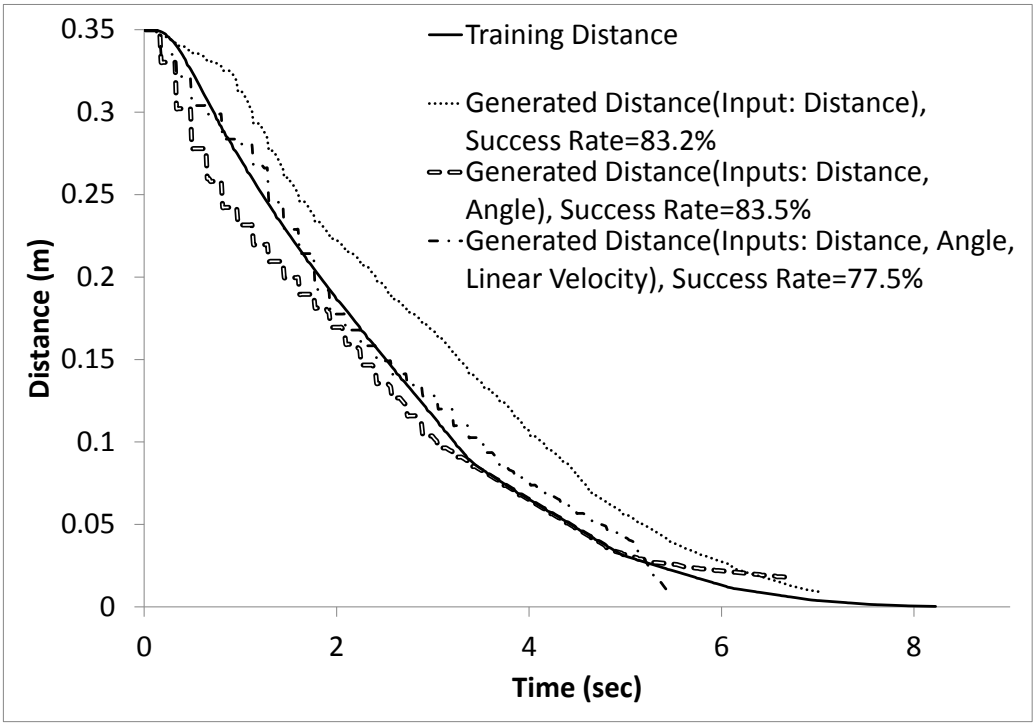
6.4 Complex-Behavior Generation Experiments

In CBLM, complex behaviors are decomposed into components corresponding to the already learned behaviors such as *turn*, *approach*, and *grasp*. The execution sequence of behaviors are determined by the model according to requested task. After the sequence of behaviors are determined, each behavior is executed one by one until the goal task is accomplished [4].

In this section, performance of complex-behavior learning model is tested. It is experimented on the combination of the already learned *turn* and *approach* behaviors. This new complex behavior is called *turn&approach* behavior. Figure 6.10 shows the charts of training and generated angle/distance for sample *turn&approach* behaviors (see Appendix B.2 for more *turn&approach* experiments).



(a)



(b)

Figure 6.9: *Approach* behaviors from different distances using different inputs. Shows *approach* behaviors from (a) 0.7 m and (b) 0.35 m distances in a time versus distance chart [4].

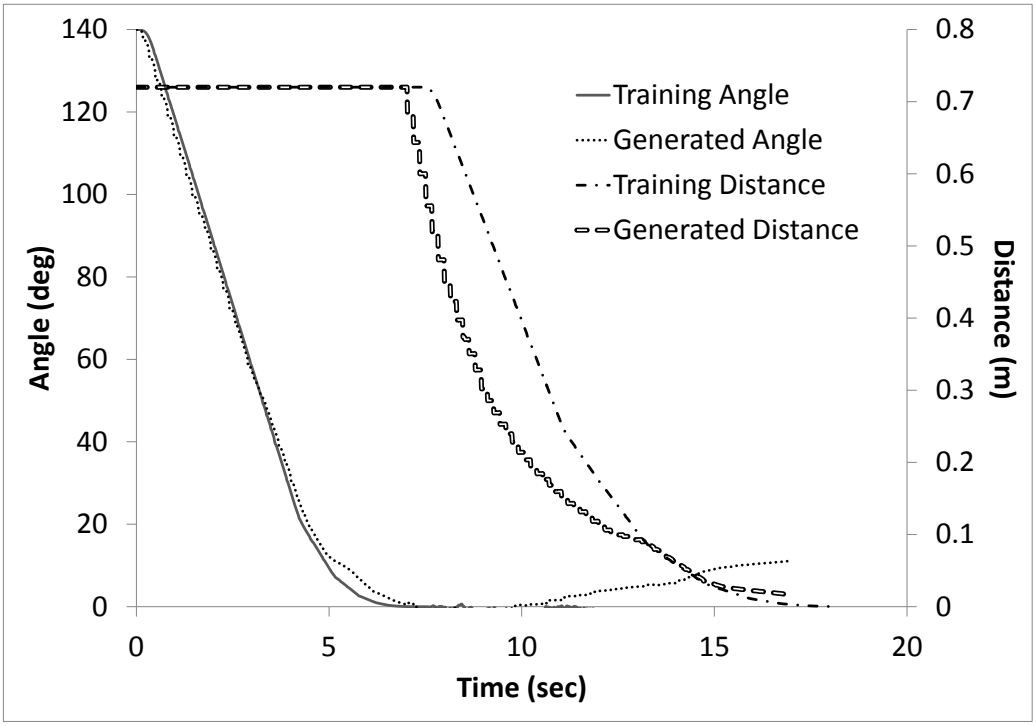
Table6.5: Average performance of 15 different *turn&approach* behaviors [4]

Turn&approach behavior	Input: distance, angle
Success rate (%)	79.8

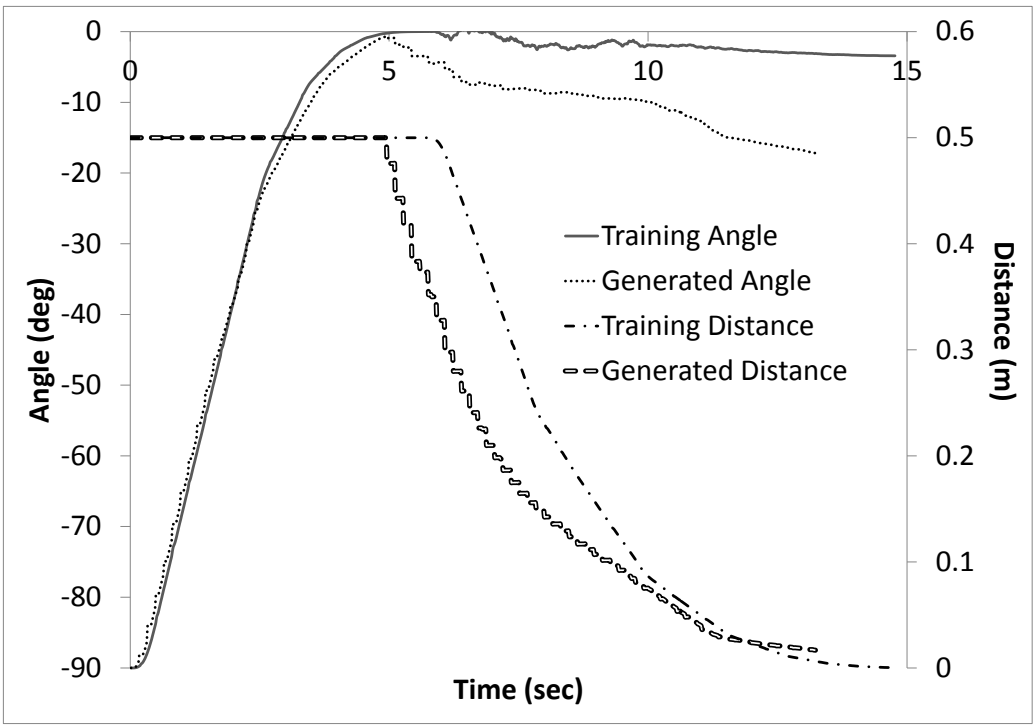
Turn&approach behavior starts with a *turn* behavior (if necessary), and then performs an *approach* behavior (if necessary). Rotational angle of the robot approaches to zero during the turn behavior as shown in the Figure 6.10. After the rotational angle is aligned with the object (around 7^{th} second for the first experiment and 5^{th} second for the second experiment in the figure), the model switches to *approach* behavior. Then, the distance to the object decreases over time until the robot reaches to the object (around 17^{th} second for the first experiment and 14^{th} second for the second experiment in the figure). Plotted curves of training and generated distances/angles in the figure show that the model can generate this pattern successfully. Average performances in Table 6.5 confirm that CBLM learns and performs *turn&approach* behavior effectively.

As the components of *turn&approach*, behaviors which have better performances in simple-behavior experiments are selected. In other words, approach behavior using *distance* and *angle* as inputs and *turn* behavior using only *angle* as input are used in the complex behavior experiments. In the previous experiments, it is observed that the use of velocity inputs did not improve the performance. Thus, only *distance* and *angle* informations are used as inputs to CobART for observation categorization in these experiments [4].

There is a general rotational shift problem during *approach* behaviors both in training and generated data as shown in Figure 6.10. Rotational angle to the object is slightly shifted because of left and right motor responses even if the same commands are applied. This rotational shift increases more in approaches from longer distances because of high accelerations and decelerations. Small differences in left and right motor commands in generated behaviors also cause more rotational shifts from the target object. We consider that the problem may be minimized by using more responsive motors. In the experiments, it is also observed that the use of limited accelerations and decelerations reduce the rotational shifts during *approach* behavior [4].



(a)



(b)

Figure 6.10: *Turn&approach* behaviors from different angles/distances in a time versus angle/distance chart. (a) rotational angle is 140° and distance is 0.72 m. (b) rotational angle is −90° and distance is 0.50 m [4].

CHAPTER 7

DISCUSSION & CONCLUSION

7.1 Discussion

Proposed models for simple and complex behaviors showed reasonable performances in learning and performing the experimented behaviors. CobART performed well in categorization of behaviors into motion primitives. Behavior-HMM performance was satisfactory as a modeling tool for motion primitives. ANNs as motion generators could learned sensory motor functions successfully. Furthermore, robot capabilities are enhanced incrementally by using hierarchically integrated behaviors for complex behaviors.

This study is not a complete robot controller application within its current capabilities. Self-organizing and extendible properties can make it a base model that can learn different behaviors effectively when new capabilities are built into the model. The proposed models can be classified as a deliberative architecture from the point of robot control perspective. Mainly, following robot control architectures exist:

- Deliberative/hierarchical models perform behaviors according to sense-plan-act logic [47]. They are built on representation, reasoning and planning concepts. These models make abstractions on the world model and behavior internal states. They are effective for complex behaviors that require abstract knowledge of the environments.
- Reactive models control the robot according to sense-act logic [47]. They are fast responsive and effective in dynamically changing environments.
- Hybrid paradigm aims to combine advantages of reactive and deliberative approaches and controls robots according to plan-sense-act logic [47]. High level planning is done before performing a task. Lower level reactive system is activated when needed in real-time.
- Behavior-based architecture combines several basic behaviors each implementing a specific goal [48][49]. They receive sensor data and activate their commands depending on the control mechanism selection without constructing world models. Depending on the architecture, multiple behaviors can activate their commands in parallel with different strengths. Subsumption is a well-known behavior-based architecture [48]. It consists of a collection of behaviors build in layers.

In order to achieve the desired goal, our model first extracts observation sequence and then determines required actions before starting motion generation which is similar to deliberative models. However,

the extendible structure of the study can be enhanced by implementing parallel behavior generation, and by determining the strengths of the behaviors depending on the observations. Besides, the deviations from the target because of potential barriers can be recovered by integrating with reactive behaviors. The combination of our models with reactive behaviors can be done either using behavior-based or hybrid architectures.

The proposed simple and complex behavior models can be criticized in the following aspects. They need lots of training data to learn a behavior with high quality and precision. For example, turn behavior needs as much as possible training data ranging from -180° to $+180^{\circ}$ rotational angles. When a new turn behavior, which is not trained before, is requested, the model approximates to the nearest rotational angle and generates the behavior accordingly. Thus the quality of behavior learning and generation is related with the training data set.

Another issue is related with the extendibility of CBLM. In the current implementation, off-line learning scheme is used. Assume that a CBLM was constructed for *turn* and *approach* behaviors and a new *grasping object* behavior is desired to be integrated to the model. In order to integrate the new behavior, a new CBLM should be constructed including the old CBLM (*turn* and *approach*) and the new SBLM (*grasp*). The new CBLM needs a new training data to construct the relationships between the behaviors. Training data is used to generate observation categories and transition conditions between the already learned behaviors, in order to generate observation and state sequences for a given goal. This issue may be addressed by using an on-line learning scheme which adds the ability to learn new behaviors incrementally. When adding a new behavior, both the parameters and the structure of Behavior-HMM can be updated in this scheme. An example of incremental HMM based model is presented in [22].

The proposed models are not applicable to all types of behaviors. Some behavior goals cannot be stated as target observation. Complex robot behaviors like floor cleaning, exploration of a workspace or obstacle avoidance need a detailed planning and goal of the behavior cannot be specified easily as a target observation. These complex behaviors can be modeled in a CBLM by decomposing them into many simple behaviors and subsumption architecture can be used to execute each simple behavior. Furthermore, the model can be extended to learn multiple target observations [4].

Some behaviors have a very large observation space. Modeling of these kind of behaviors may cause some scalability problems. For instance, categorization of a behavior using cartesian coordinates as inputs may exhibit some resolution and scalability problems, because the model shall have very large set of observation categories. For these behaviors, categorization of observation space can be handled differently by preprocessing the input space and using only the most relevant inputs [4].

The measurement error analysis on performances of experiments are not performed in this study, because we used a simulated environment. But, the models use low-pass second-order filters on the inputs of ANN and CobART to reduce the possible effects of random measurement errors in real robot platforms. Furthermore, robot executes behaviors in a closed-loop approach by making a continuous checks on observations and possible state changes during behavior generation. Thus, it reduces the random measurement effect on the generated behaviors. On the other hand, systematic measurement errors (biases in measurement) on the sensors are considered as part of the environment. We consider that systematic errors to be learned by the model as a part of the environment and the model shall perform the learned behaviors accordingly [4].

If a behavior model is trained by using different forms of a same behavior (e.g. fast or slow approach behaviors), different motion primitives may be generated by CobART depending on the selected categorization parameters. This alternative motion primitives as states of Behavior-HMM may result

different branches in the model. Within its current capabilities, proposed models do not aware of those alternative branches. Hence, when a desired behavior is requested, the model generates the most likely motion primitive sequence without considering alternative paths. The model can be enhanced to handle those alternative forms or alternative forms of a behavior can be modeled as different behaviors [4].

Our main focus was on the modeling and learning capabilities of the system throughout the study. Thus, having the maximum performances were not our main goal. But, performances of the models can be enhanced by looking for the optimum parameters experimentally. Categorization parameters of CobART and learning parameters of ANN have a direct effect on the performances. For instance, using a higher vigilance in categorization results more observation categories and more motion primitives and hence increases the resolution of the system and improves the behavior generation performances. On the other hand, it may cause scalability problems and increase computational times of training procedure [4].

7.2 Conclusion and Future Work

In the scope of this study, we looked behavior learning problem from the robot's perspective not from the programmer's. Robot learns behaviors without the help of a supervisor. For this purpose, two robot behavior learning models are introduced for simple and complex behaviors. Followings are the summary of methodologies used in this study:

- CobART was used to categorize sensory data into observation categories and motion primitives.
- Behavior-HMM modeled relationships among the motion primitives.
- ANNs as motion generators learned to generate continuous motor commands.
- Most likely motion primitive sequences for a requested task were generated by the help of most likely path and Viterbi algorithms.
- Complex behaviors were modeled in a CBLM by hierarchically integrating the already learned behaviors.

The capabilities of the models are investigated by some experiments tested on a robot simulator using *turn* and *approach* behaviors. Test results presented that the models can learn and perform the experimented behavior effectively [4].

This thesis contributes to the previous behavior learning studies by proposing generic, unsupervised, and extendible behavior learning models for simple and complex behaviors. The behavior models can be a base study for future research. The use methodologies such as CobART, Behavior-HMM, and ANN eliminate the requirement of a supervisor [4].

The capabilities of this study can be improved in many aspects. This improvements as future works can be summarized as below [4]:

- The models can be experimented on different and more complex behaviors.
- The actual performance of the models can be tested on a real robot.

- Behavior generation performance can be improved by using a different neural network architecture specifically designed for motion generation (such as recurrent neural networks).
- The model can be enhanced by adding a parallel behavior execution mechanism in a CBLM. For instance, while a robot is turning to an object, it can also approach to object in parallel.
- Feature selection mechanisms can be integrated to the model to learn relevant and redundant input data. Applying feature selection to the models needs a detailed study, but it will improve the self-organizing capabilities.

REFERENCES

[1] N. Koenig and M. Mataric. Demonstration-based behavior and task learning. In *AAAI Spring Symposium To Boldy Go Where No Human-Robot Team Has Gone Before, Working notes*, 2006.

[2] M. Yavas and F. N. Alpaslan. Behavior categorization using correlation based adaptive resonance theory. In *17th Mediterranean Conference on Control & Automation*, pages 724–729, 2009.

[3] M. Yavas and F. N. Alpaslan. Hierarchical behavior categorization using correlation based adaptive resonance theory. *Neurocomputing*, 77:71–81, 2012.

[4] S. S. Seyhan, F. N. Alpaslan, and M. Yavas. Simple and complex behavior learning using Behavior Hidden Markov Model and CobART. *Neurocomputing*, 103:121–131, 2013.

[5] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, 1989.

[6] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269, 1967.

[7] *Khepera user manual, version 5.02*. K-Team. http://www.k-team.com, 1999.

[8] Webots commercial mobile robot simulation software. Cyberbotics Ltd. http://www.cyberbotics.com.

[9] M. Yavas and F. N. Alpaslan. CobART: Correlation based adaptive resonance theory. In *17th Mediterranean Conference on Control & Automation*, pages 742–747, 2009.

[10] J. Tani and J. Yamamoto. On the dynamics of robot exploration learning. *Cognitive Systems Research*, 3:459–470, 2002.

[11] J. Tani. Learning to generate articulated behavior through the bottom-up and the top-down interaction processes. *Neural Networks*, 16:11–23, 2003.

[12] R. Nishimoto and J. Tani. Learning to generate combinatorial action sequences utilizing the initial sensitivity of deterministic dynamical systems. *Neural Networks*, 17:925–933, 2004.

[13] R. W. Paine and J. Tani. How hierarchical control self-organizes in artificial adaptive systems. *Adaptive Behavior*, 13(3):211–225, 2005.

[14] J. Tani, R. Nishimoto, J. Namikawa, and M. Ito. Codevelopmental learning between human and humanoid robot using a dynamic neural network model. *IEEE Transaction on Systems, Man, and Cybernetics – Part B: Cybernetics*, 38:43–59, 2008.

[15] R. F. Reinhart and J. J. Steil. Reaching movement generation with a recurrent neural network based on learning inverse kinematics for the humanoid robot iCub. In *IEEE-RAS International Conference on Humanoid Robots*, pages 323–330, 2009.

[16] M. Fox, M. Ghallab, G. Infantes, and D. Long. Robot introspection through learned hidden Markov models. *Artificial Intelligence*, 170:59–113, 2006.

[17] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[18] B. Morisset and M. Ghallab. Learning how to combine sensory-motor functions into a robust behavior. *Artificial intelligence*, 172:392–412, 2008.

[19] G. Infantes, M. Ghallab, and F. Ingrand. Learning the behavior model of a robot. *Autonomous Robots*, 30(2):155–177, 2011.

[20] K. Han and M. Veloso. Automated robot behavior recognition applied to robotic soccer. In *Proceedings of the IJCAI-99 Workshop on Team Behaviors and Plan Recognition*, 1999.

[21] S. Osentoski, V. Manfredi, and S. Mahadevan. Learning hierarchical models of activity. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 891–896, 2004.

[22] D. Vazquez, T. Fraichard, and C. Laugier. Growing hidden Markov models: An incremental tool for learning and predicting human and vehicle motion. *The International Journal of Robotics Research*, 28(11–12):1486–1506, 2009.

[23] D. Kulic, W. Takano, and Y. Nakamura. Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden Markov chains. *The International Journal of Robotics Research*, 27(7):761–784, 2008.

[24] D. Kulic and Y. Nakamura. Incremental learning of human behaviors using hierarchical hidden Markov models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4649–4655, 2010.

[25] S. Okada and T. Nishida. Incremental clustering of gesture patterns based on a self organizing incremental neural network. In *Proceedings of International Joint Conference on Neural Networks*, pages 2316 – 2322, 2009.

[26] R. Kelley, A. Tavakkoli, C. King, M. Nicolescu, M. Nicolescu, and G. Bebis. Understanding human intentions via hidden Markov models in autonomous mobile robots. In *Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction*, pages 367–374, 2008.

[27] Y. Okuzawa, S. Kato, M. Kanoh, and H. Ito. Imitative motion generation for humanoid robots based on the motion knowledge learning and reuse. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 4031–4036, 2009.

[28] N. Kubota. Computational intelligence for structured learning of a partner robot based on imitation. *Information Sciences*, 171:403–429, 2005.

[29] E. Borenstein and E. Ruppin. The evolution of imitation and mirror neurons in adaptive agents. *Cognitive Systems Research*, 6:229–242, 2005.

[30] H. Hajimirsadeghi, M. N. Ahmadabadi, M. Ajallooeian, B. N. Araabi, and H. Moradi. Conceptual imitation learning: An application to human-robot interaction. In *JMLR: Workshop and Conference Proceedings*, volume 13, pages 331–346, 2010.

[31] J.A. Freeman and D.M. Skapura. *Neural networks algorithms, applications, and programming techniques*. Addison Wesley, Reading, MA, 1992.

[32] G.A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1987.

[33] G.A. Carpenter and S. Grossberg. ART 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26(23):4919–4930, 1987.

[34] G.A. Carpenter, S. Grossberg, and D.B. Rosen. ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition. *Neural Networks*, 4:493–504, 1991.

[35] G.A. Carpenter, S. Grossberg, and D.B. Rosen. Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, 4:759–771, 1991.

[36] G.A. Carpenter, S. Grossberg, and J.H. Reynolds. ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks*, 4:565–588, 1991.

[37] G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3(5):698–713, 1992.

[38] G. F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Pearson Addison-Wesley, fourth edition, 2002.

[39] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[40] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[41] S. Haykin. *Neural Networks and Learning Machines*. Pearson, third edition, 2009.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Cambrigde, MA:MIT Press*, 1:318–362, 1986.

[43] *Webots user guide, version 6.4.1*. Cyberbotics. http://www.cyberbotics.com, 2011.

[44] K. D. Nguyen, I.-M. Chen, and T.-C. Ng. Planning algorithms for s-curve trajectories. In *IEEE International Conference on Advanced Intelligent Mechatronics*, pages 1–6, 2007.

[45] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[46] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab. *Signals & Systems*. Prentice Hall, 1996.

[47] R. R. Murphy. *Introduction to AI Robotics*. MIT Press, 2000.

[48] R.C. Arkin. *Behavior-Based Robotics*. MIT Press, third edition, 1998.

[49] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence and Technology of Self-Organizing Machines*. MIT Press, 2000.

APPENDIX A

MORE HIDDEN LAYER/UNIT EXPERIMENTS ON NEURAL NETWORK TRAINING

In order to investigate the learning performance of neural networks for two-input networks, several hidden layer/unit configurations are tested. Experiments are performed on *turning* behavior to analyze the network performance easily. Input vector containing 10 consecutive data is presented to the input layer of the network for each input (e.g. 10 inputs for *angle* and 10 inputs for *angular velocity*). Left and right motor commands are expected as the target values in the backpropagation phase. In each iteration, input vector is shifted by one and shifted vector is presented as a new input to network for the next iteration.

One of the motion primitives category data is selected as a test data. 20% of the sensory data is used as test data and 80% is used as training data. Test data is not used in training process.

In the experiments, different hidden layer/unit configurations are tested and the results are plotted in a test error versus iteration number chart as shown in the Figure A.1. All network configurations produced similar success rates. Best result is obtained when 2 hidden layers with 20 units configuration is used in the training as shown in the figure. Training of larger networks requires longer times and a network having too much hidden units cannot generalize input signal properly, but memorize all data even the noise (known as overfitting). By considering the training times and overfitting, it is used 1 hidden layer with 10 units for two-input networks.

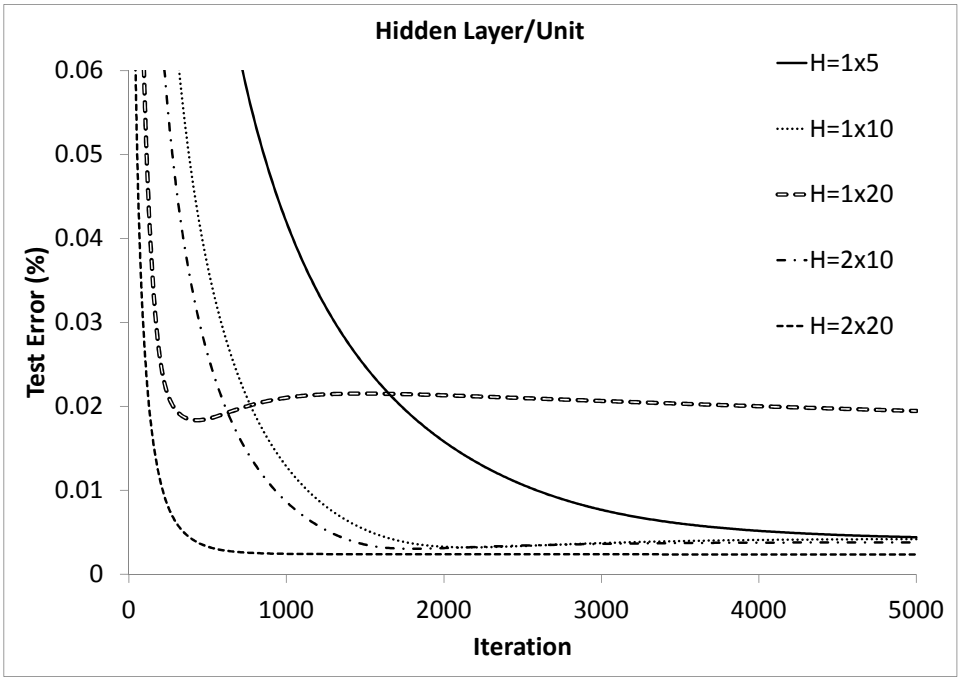


Figure A.1: Hidden layer/unit experiments for two-input networks (Learning rate=0.01, Momentum=0.4). Shows different hidden layer organization performances in a test error versus iteration number chart.

APPENDIX B

MORE BEHAVIOR GENERATION EXPERIMENTS

In Section 6.3 and Section 6.4 only limited number of experiment results are reported for demonstration and analysis purposes. In this section, more simple and complex-behavior experiments are presented.

SBLM is experimented on *turn to the object* and *approach the object* behaviors. Furthermore, performance of CBLM model is experimented on the combination of the already learned *turn* and *approach* behaviors.

B.1 Simple-Behavior Generation Experiments

B.1.1 *Turn* Experiments

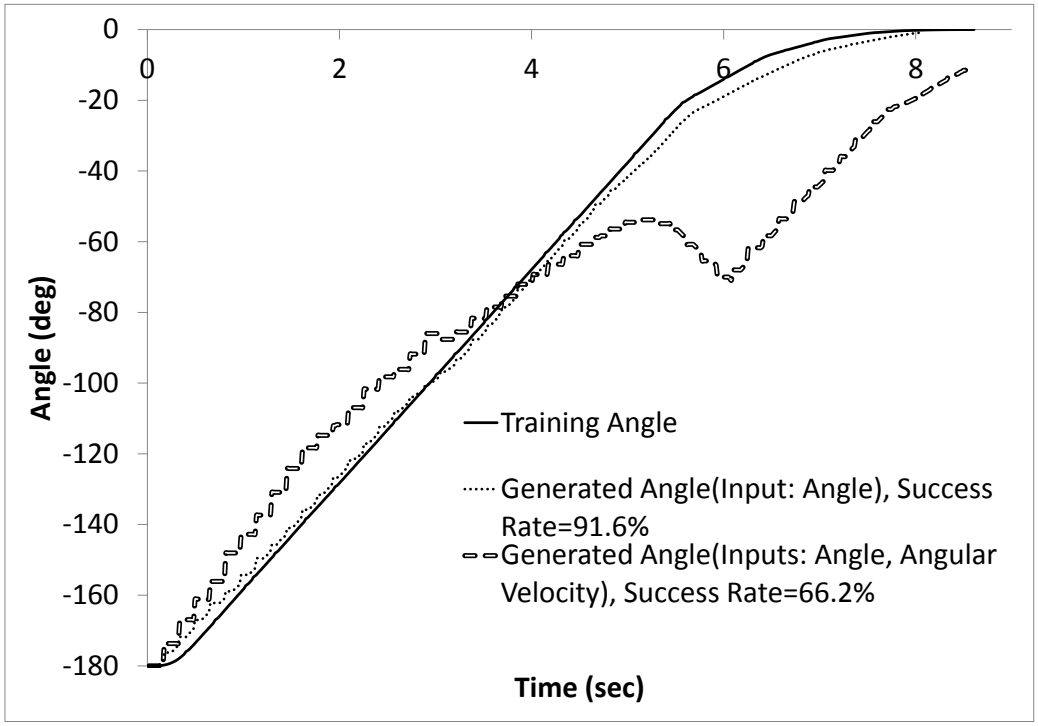
Different *turn* behaviors from -180° to $+180^{\circ}$ rotational angles are experimented in this section. The test results are shown in Figure B.1. Behavior generation performances of the experiments using different set of inputs in categorization and neural network training are compared. The first experiment uses only *angle* vector to generate *turn* behavior. The second experiment uses *angle* and *angular velocity* as inputs to the model.

As shown in Figure B.1, generated behaviors can follow the training behaviors successfully. According to success rates on the charts, the use of angular velocity in behavior learning has a negative effect on the performance of *turn* behavior as stated in Section 6.3.1.

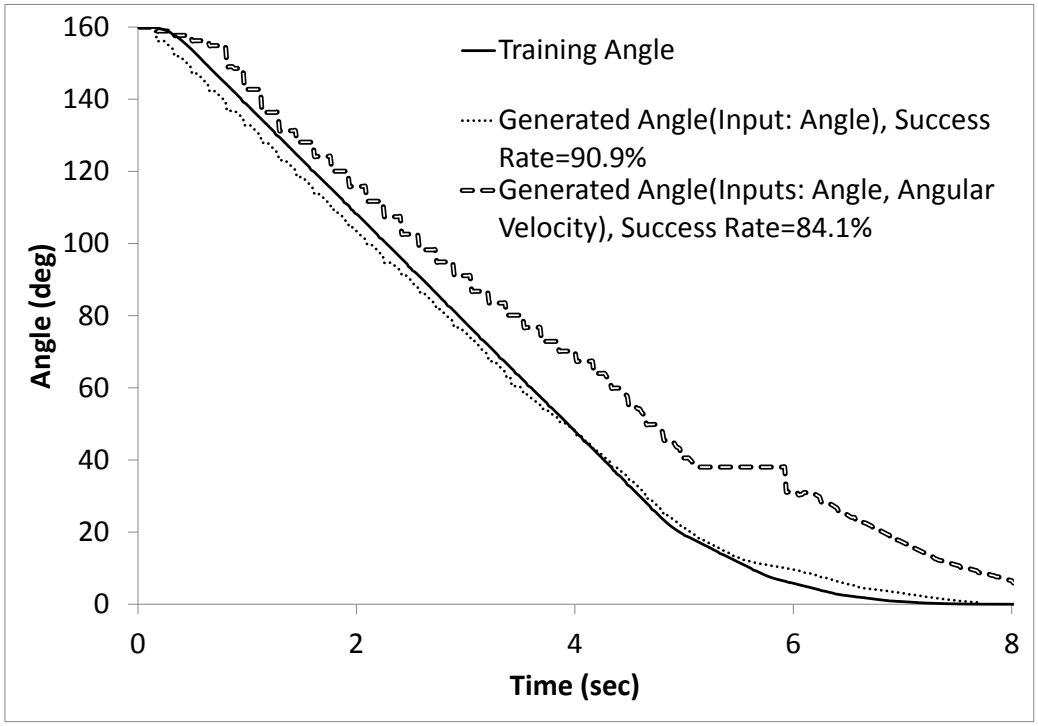
B.1.2 *Approach* Experiments

Different *approach* behaviors from 0 to 1 meter distances are experimented in this section. The test results are shown in Figure B.2. Behavior generation performances of the experiments using different set of inputs in categorization and neural network training are compared. The first experiment uses only *distance* vector to generate *approach* behavior. The second experiment uses *distance* and *angle* variables as inputs. In the third experiment, *distance, angle,* and *linear velocity* are used as inputs to the model.

As shown in Figure B.2, generated behaviors can follow the training behaviors successfully. According to success rates on the charts, the use of linear velocity in behavior learning has a negative effect on the performance of *approach* behavior, as in *turn* behavior experiments. However, the use of angle information improves the quality of *approach* behavior. Small variations in left and right motor commands generate a rotational shift during *approach* behavior. In our opinion, the use of angle

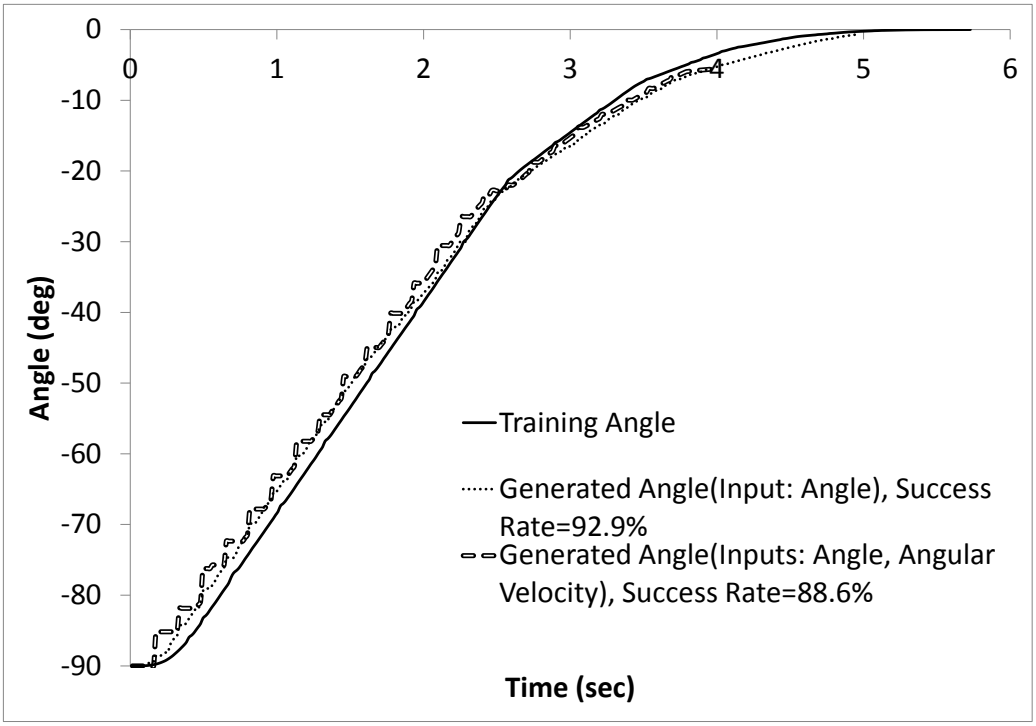


(a) Angle = $-180°$

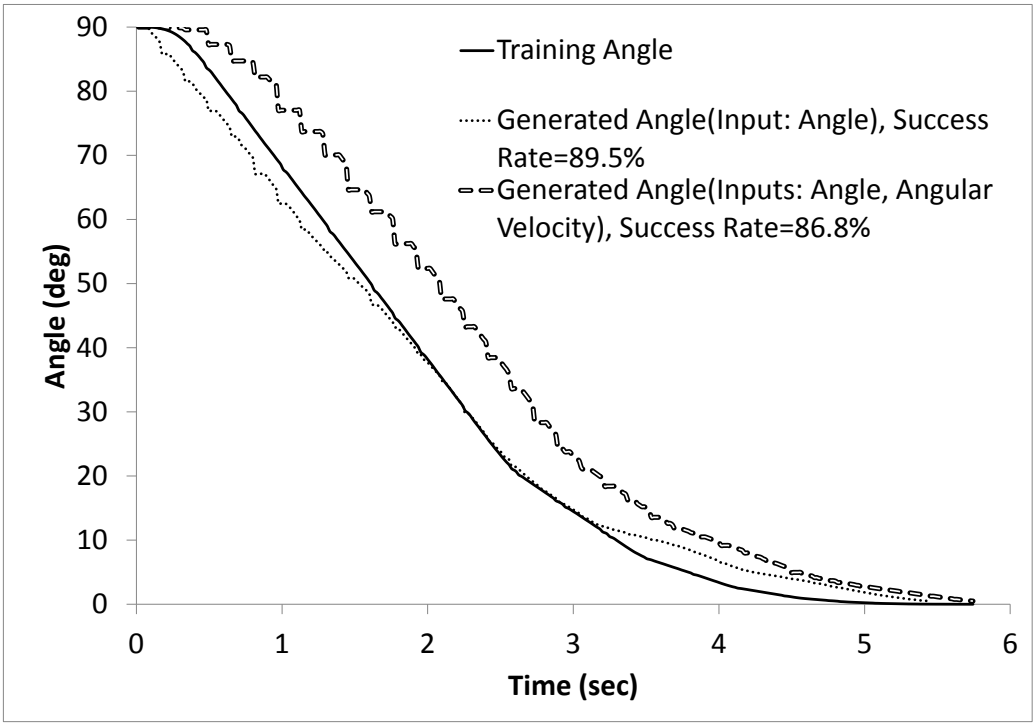


(b) Angle = $160°$

Figure B.1: (Part 1/3) More samples of *turn* behaviors from different rotational angles using different inputs.

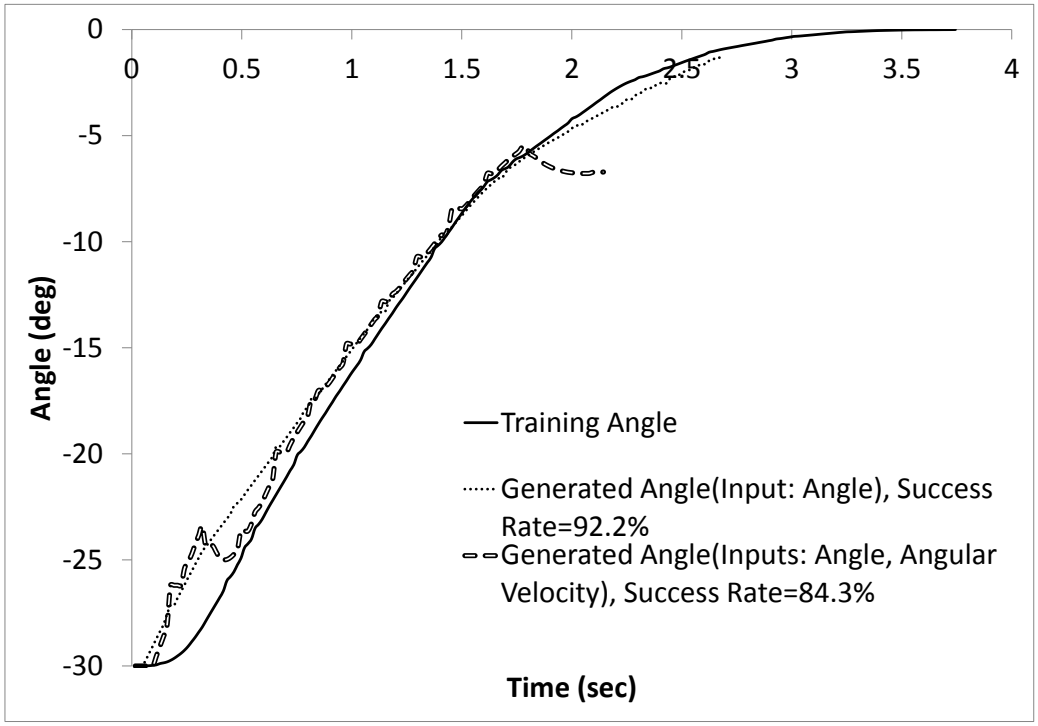


(c) Angle = −90°

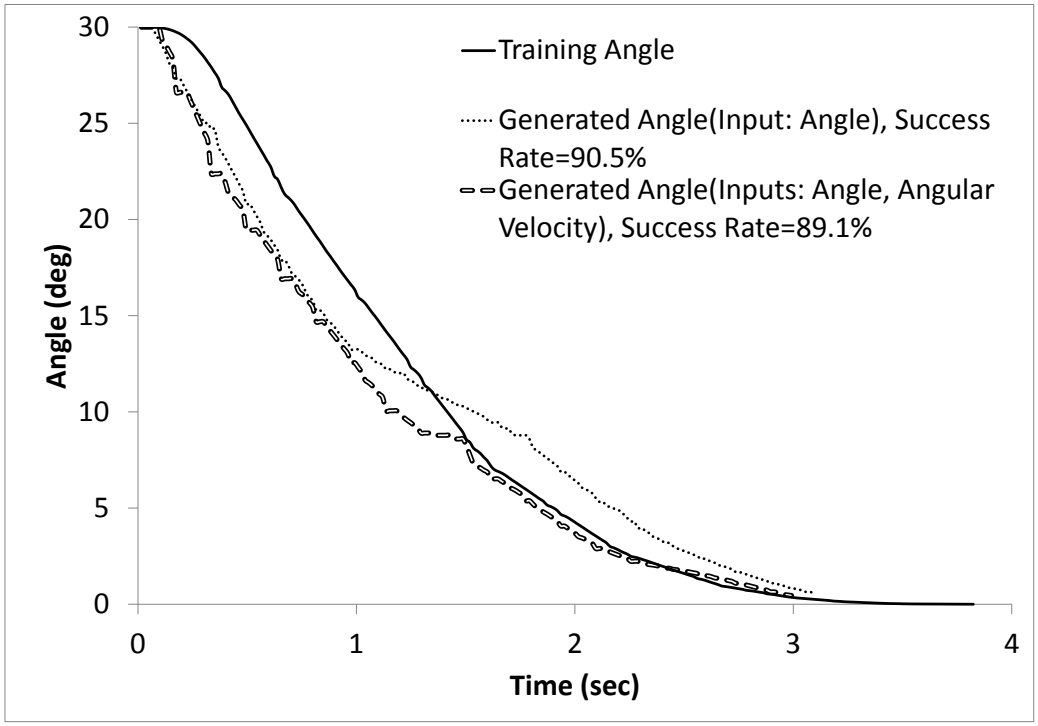


(d) Angle = 90°

Figure B.1: (Part 2/3) More samples of *turn* behaviors from different rotational angles using different inputs.



(e) Angle = −30°



(f) Angle = 30°

Figure B.1: (Part 3/3) More samples of *turn* behaviors from different rotational angles using different inputs.

information in addition to distance helps to correct those rotational shifts [4].

Note that *approach* behaviors from closer distances have better performances. Approaching from longer distances requires greater accelerations and decelerations and results big differences in motor commands. Big changes in motor commands in a short time cause differences in left and right motor responses. It leads rotational shifts to the object during approach and causes performance problems in *approach* behaviors from longer distances [4].

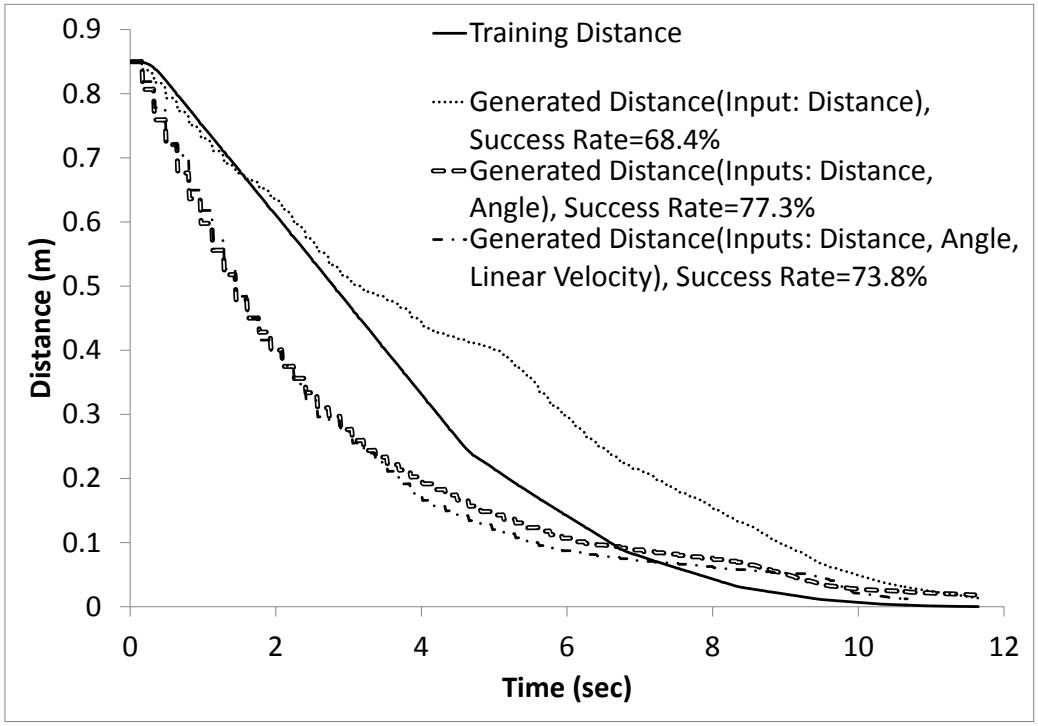
B.2 Complex-Behavior Generation Experiments

CBLM is experimented on the combination of the already learned *turn* and *approach* behaviors. This new complex behavior is called *turn&approach* behavior. *Turn&approach* behavior starts with a *turn* behavior (if necessary), and then performs an *approach* behavior (if necessary). The execution sequence of behaviors are determined by the model according to requested task. After the sequence of behaviors are determined, each behavior is executed one by one until the goal task is accomplished [4].

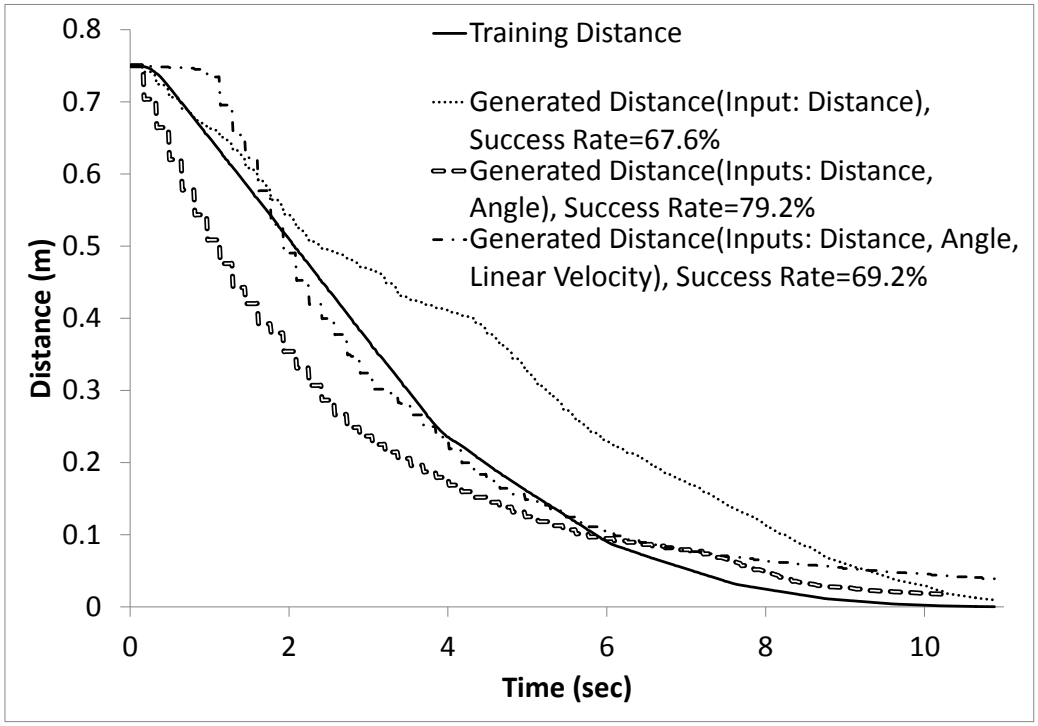
Plotted curves of training and generated distances/angles in Figure B.3 show that the model can learn and perform *turn&approach* behavior successfully. As the components of *turn&approach*, behaviors which have better performances in simple-behavior experiments are selected. In other words, approach behavior using *distance* and *angle* as inputs and *turn* behavior using only *angle* as input are used in the complex behavior experiments. In the previous experiments, it is observed that the use of velocity inputs did not improve the performance. Thus, *distance* and *angle* are used as inputs to CobART for observation categorization in this experiments [4].

There is a general rotational shift problem during *approach* behaviors both in training and generated data as shown in Figure B.3. Rotational angle to the object is slightly shifted because of left and right motor responses even if the same commands are applied. This rotational shift increases more in approaches from longer distances because of high accelerations and decelerations. Small differences in left and right motor commands in generated behaviors also cause more rotational shifts from the target object. We consider that the problem may be minimized by using more responsive motors. In the experiments, it is also observed that the use of limited accelerations and decelerations reduce the rotational shifts during *approach* behavior [4].

Also note that in Figure B.3 (f), rotational angle to the object is 0°. It means that the robot does not need to turn to the object. Thus CBLM infers this situation and it starts to execute *approach* behavior without performing *turn* behavior.

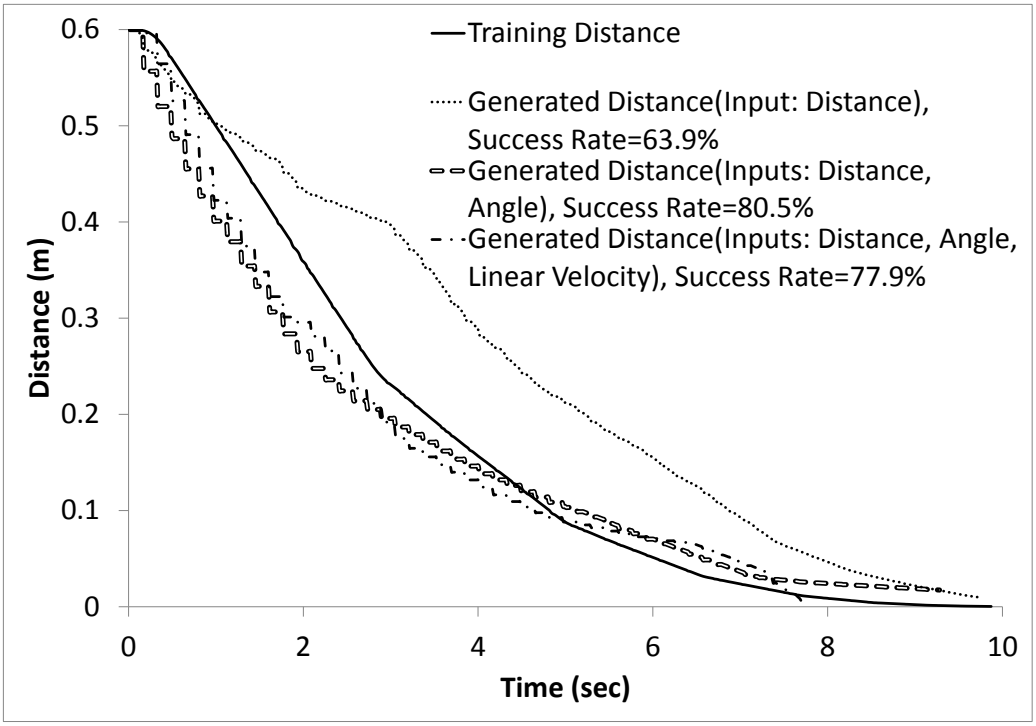


(a) Distance = 0.85 m

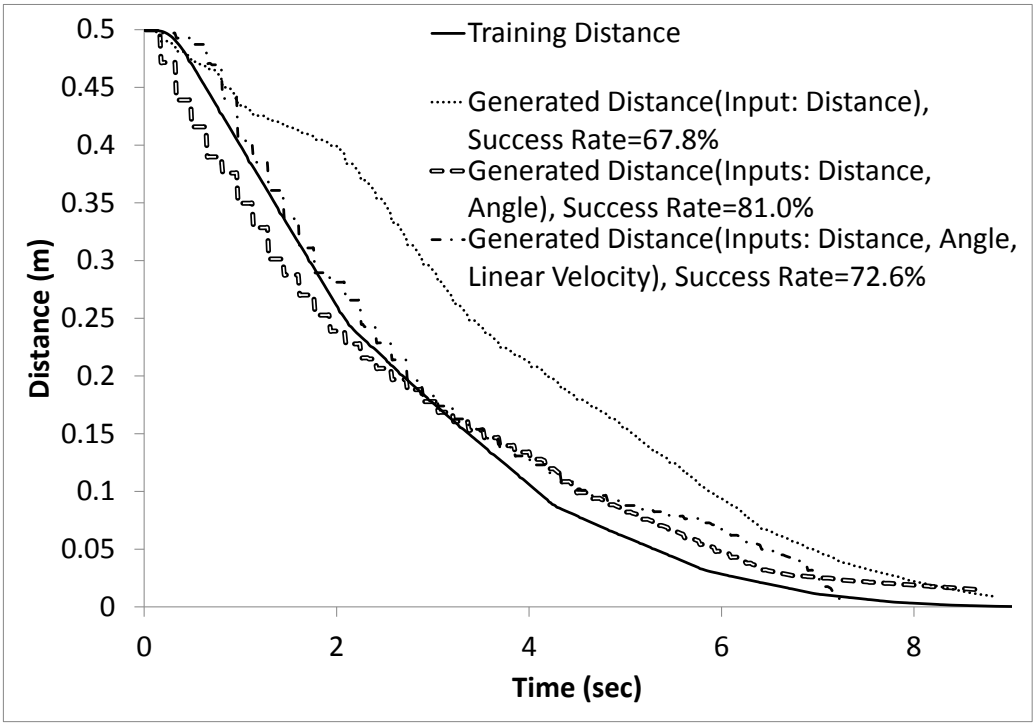


(b) Distance = 0.75 m

Figure B.2: (Part 1/3) More samples of *approach* behaviors from different distances using different inputs.

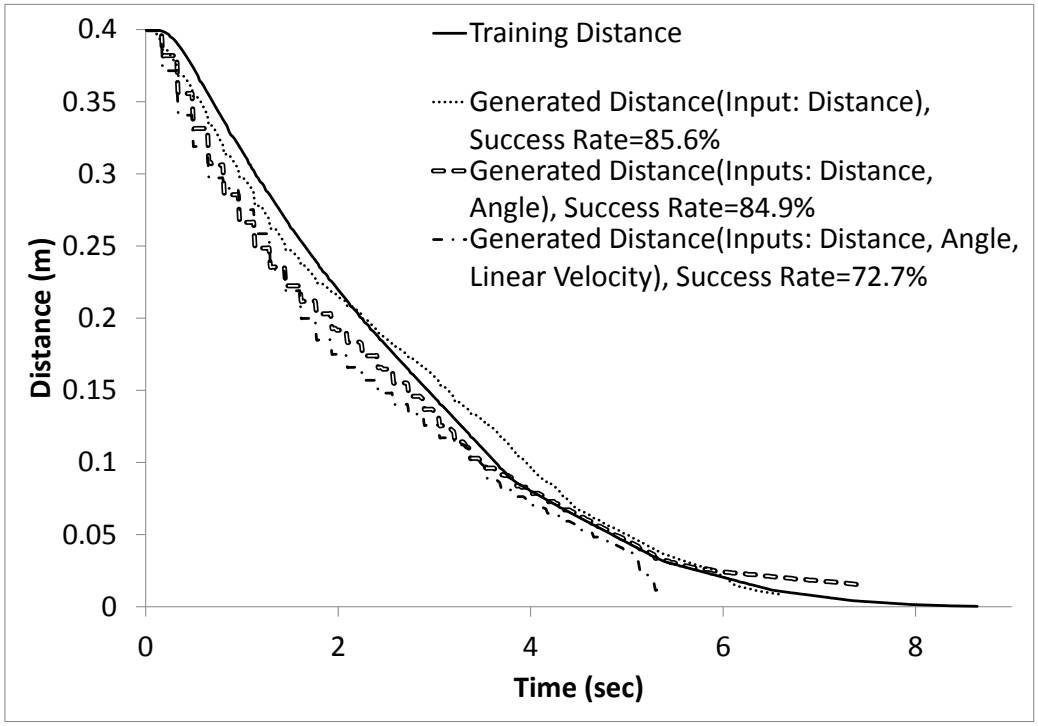


(c) Distance = 0.60 m

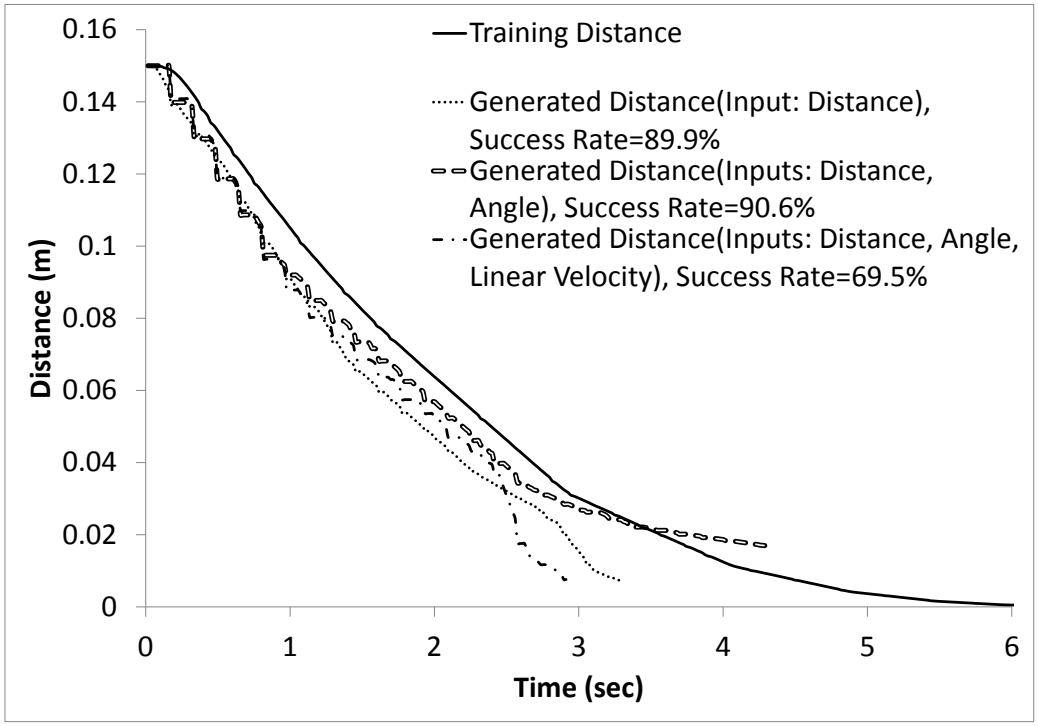


(d) Distance = 0.50 m

Figure B.2: (Part 2/3) More samples of *approach* behaviors from different distances using different inputs.

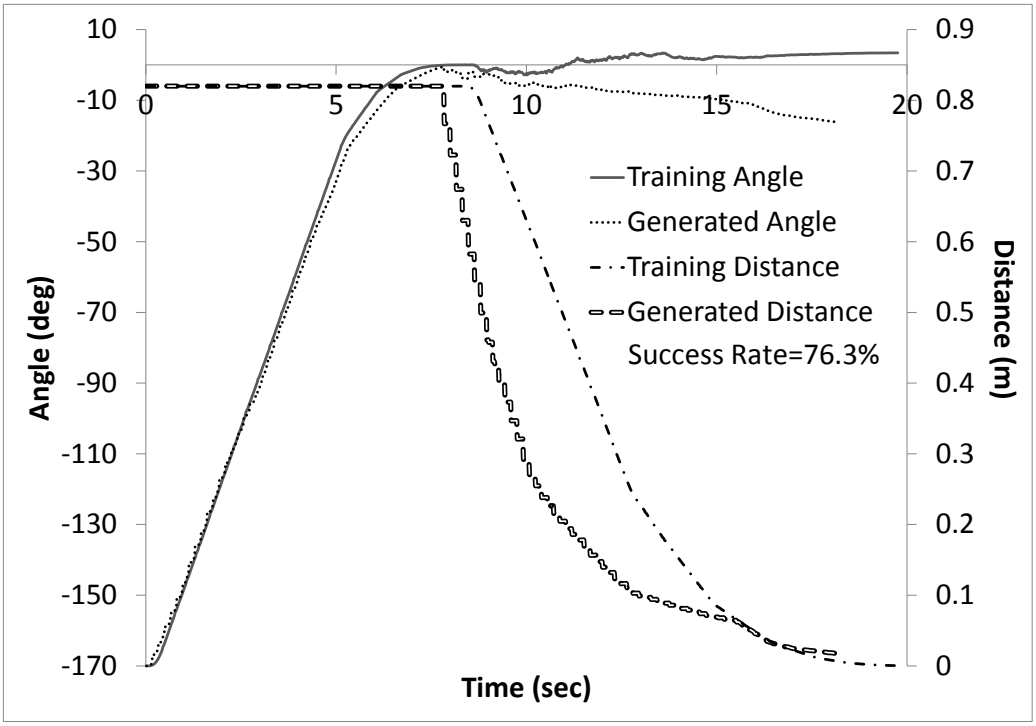


(e) Distance = 0.40 m

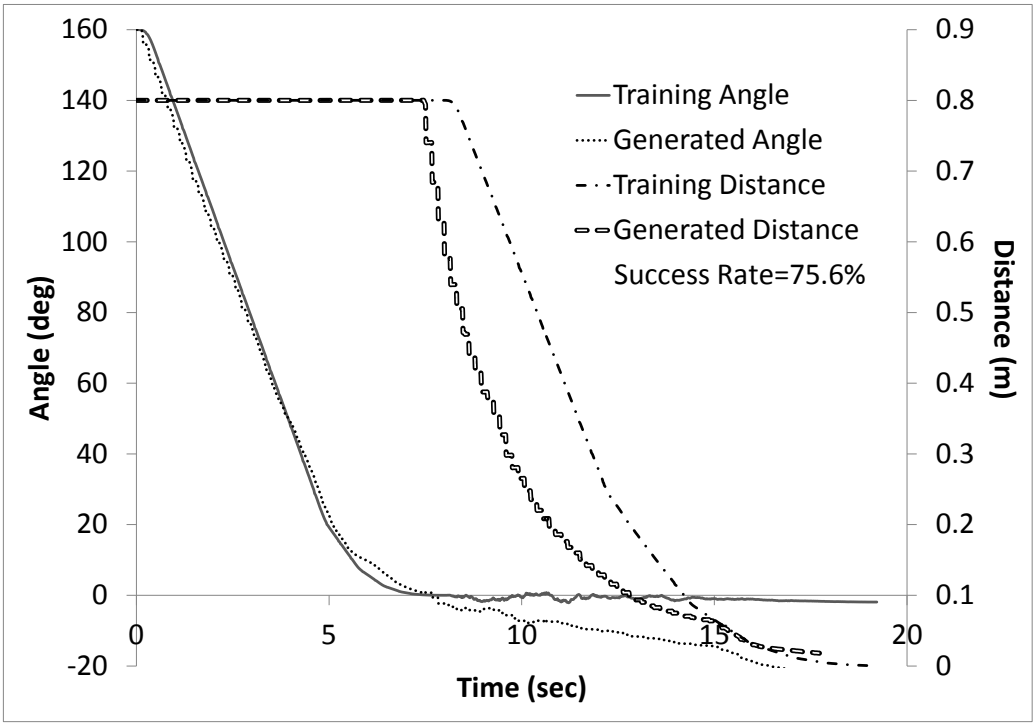


(f) Distance = 0.15 m

Figure B.2: (Part 3/3) More samples of *approach* behaviors from different distances using different inputs.

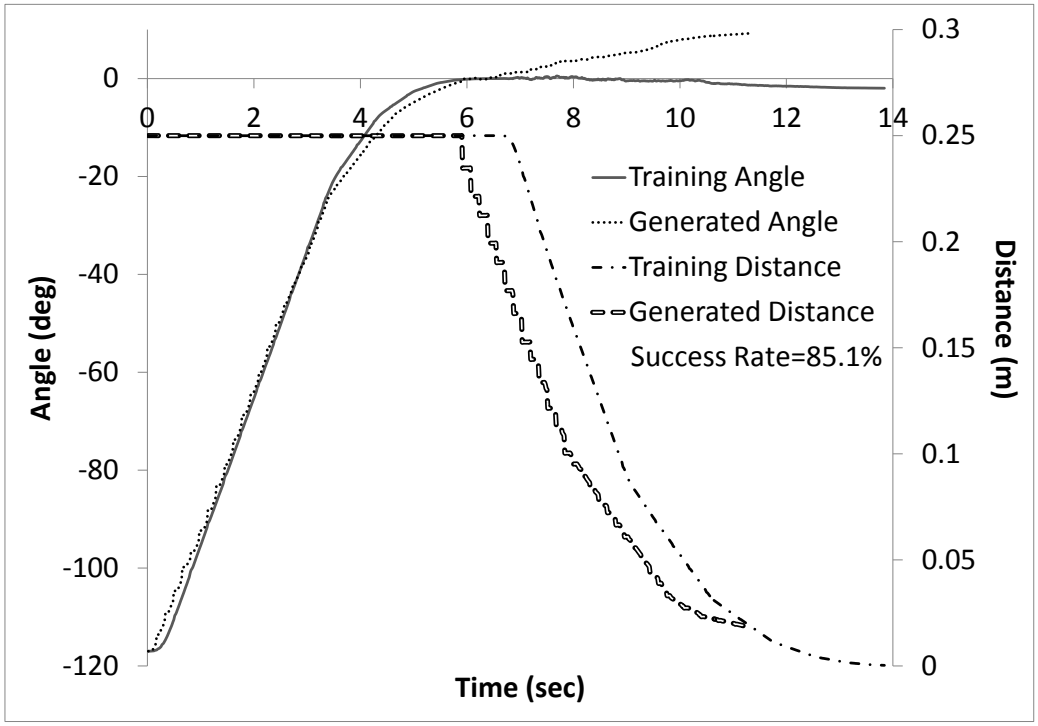


(a) Angle = −170°, distance = 0.82 m

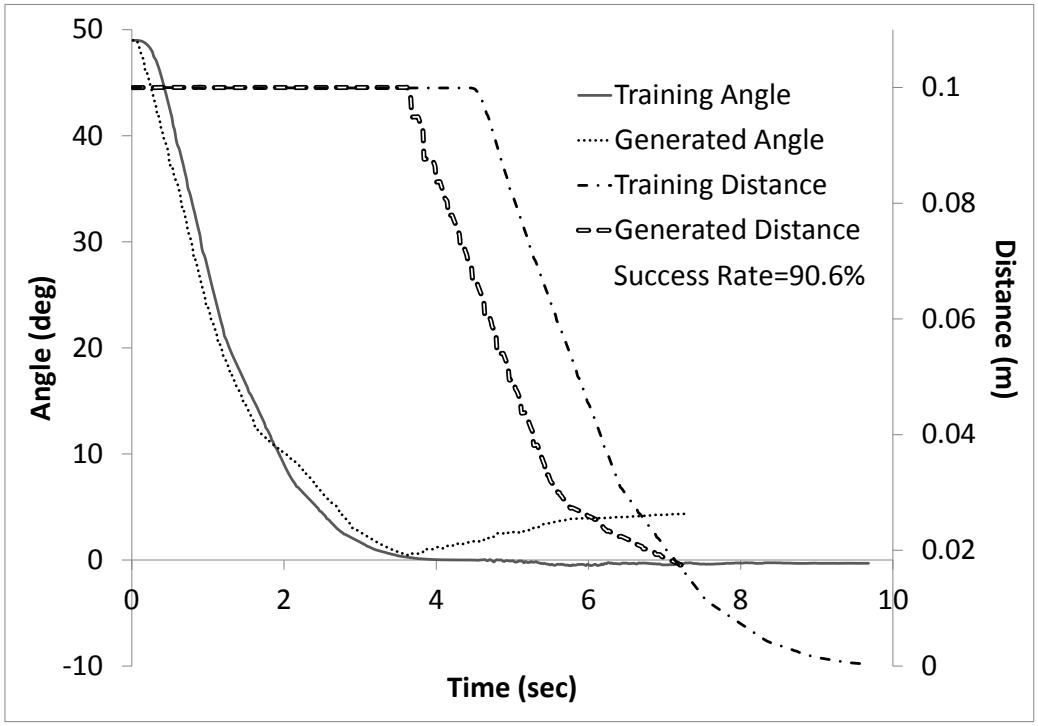


(b) Angle = 160°, distance = 0.80 m

Figure B.3: (Part 1/3) More samples of *turn&approach* behaviors from different angles/distances in a time versus angle/distance chart.

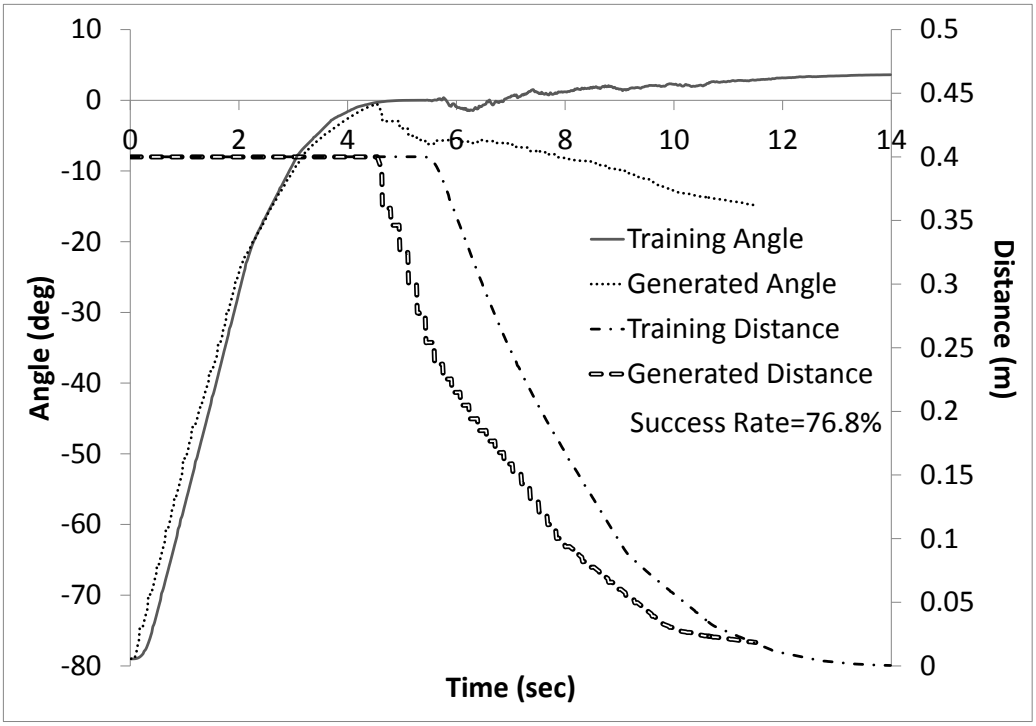


(c) Angle = $-117°$, distance = 0.25 m

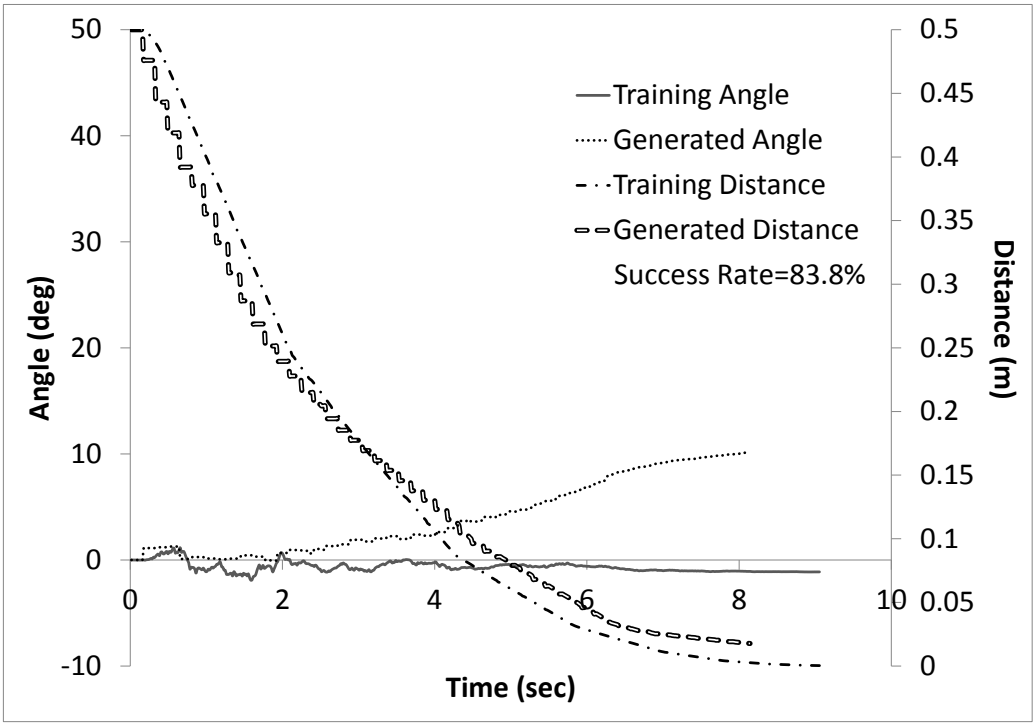


(d) Angle = $49°$, distance = 0.10 m

Figure B.3: (Part 2/3) More samples of *turn&approach* behaviors from different angles/distances in a time versus angle/distance chart.



(e) Angle = −79°, distance = 0.40 m



(f) Angle = 0°, distance = 0.50 m

Figure B.3: (Part 3/3) More samples of *turn&approach* behaviors from different angles/distances in a time versus angle/distance chart.

APPENDIX C

VIDEO RECORDINGS OF TRAINING AND GENERATED BEHAVIORS

In this section, samples of training and generated behavior video recordings are presented. Table C.1 shows the video recordings of *turn*, *approach*, and *turn&approach* behaviors. Video files can be found in the CD-ROM attached to the thesis.

TableC.1: Samples of training and generated behavior video recordings

File names	Explanation
Video_Training_Turn.avi	*Turn* training behavior
Video_Generated_Turn.avi	*Turn* generated behavior
Video_Training_Approach.avi	*Approach* training behavior
Video_Generated_Approach.avi	*Approach* generated behavior
Video_Training_Turn_Approach.avi	*Turn&Approach* training behavior
Video_Generated_Turn_Approach.avi	*Turn&Approach* generated behavior

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Seyhan, Seyit Sabri

Nationality: Turkish (TC)

Date and Place of Birth: 13 May 1975, Sivas

Marital Status: Married

email: seyits@gmail.com

EDUCATION

Degree	Institution	Year of Graduation
MS	METU Computer Engineering	2001
BS	Hacettepe Uni. Computer Science & Eng.	1998

WORK EXPERIENCE

Year	Place	Enrollment
1998-1999	Hacettepe Uni. Computer Science & Eng.	Research Assistant
1999-Present	ETC Turkey Office	Software Engineer

FOREIGN LANGUAGES

English

PUBLICATIONS

1. S.S. Seyhan, F.N. Alpaslan and M. Yavas. "Simple and Complex Behavior Learning using Behavior Hidden Markov Model and CobART". Neurocomputing, Volume 103, Pages 121-131, March 2013, DOI: 10.1016/j.neucom.2012.09.013.