

Type-2 Fuzzy Logic Induced Gesture Recognition for Robot Task Planning

Thesis

*submitted for the partial fulfilment of the requirement for the
degree of Masters of Engineering
in Electronics & Telecommunication Engineering
with the specialization Control Systems
at Jadavpur University*

by

Kaustuv Mukherji

Registration Number: 140694 of 2017-18

Examination Roll No: M4ETC19010

Under the Guidance of

Prof. Amit Konar

Department of Electronics & Telecommunication Engineering

Jadavpur University, Kolkata-700032

India

May, 2019

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

CERTIFICATE

This to certify that the thesis entitled “**Type-2 Fuzzy Logic Induced Gesture Recognition for Robot Task Planning**” has been carried out by **KAUSTUV MUKHERJI** (University Registration No. **140694** of 2017-18, Examination Roll No. **M4ETC19010**) under my guidance and supervision and be accepted in partial fulfillment of the requirement for the degree of Masters of Engineering in Electronics & Telecommunication Engineering with the specialization Control Systems. The research results presented in the thesis have not been included in any other paper submitted for the award of any degree to any other university or institute.

Supervisor
Prof. Amit Konar
Professor
Department of Electronics &
Telecommunication Engineering
Jadavpur University
Kolkata-700032

Prof. Sheli Sinha Chaudhuri
Head of the Department
Department of Electronics &
Telecommunication Engineering
Jadavpur University
Kolkata-700032

Prof. Chiranjib Bhattacharjee
Dean, Faculty Council of Engineering and Technology
Jadavpur University
Kolkata - 700032

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

CERTIFICATE OF APPROVAL*

The foregoing thesis is hereby approved as a creditable study of an engineering subject and presented in a manner satisfactory to warrant its acceptance as prerequisite to obtain the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn there in but approve the thesis only for which it is submitted.

**Committee on Final Examination
for the Evaluation of the Thesis**

Signature of the Examiner

Signature of the Supervisor

* Only in the case the thesis is approved

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

DECLARATION OF ORIGINALITY AND COMPLIANCE OF
ACADEMIC THESIS

I hereby declare that this thesis entitled “**Type-2 Fuzzy Logic Induced Gesture Recognition for Robot Task Planning**” contains literature survey and original research work by the undersigned candidate, as part of his degree of Masters of Engineering in Electronics & Telecommunication Engineering with the specialization Control Systems.

All information have 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 materials and results that are not original to this work.

Name: Kaustuv Mukherji

Examination Roll No.: M4ETC19010

Signature of the Candidate

Abstract

With the advancement of robotics in industry and healthcare, a key component of research is to develop sophisticated design automations to offer precise and accurate response of the system with the freedom to access the system easily. This thesis proposes a novel arm-gesture-driven position control of a robot arm, which can be operated by any person even without specialized domain knowledge. The uncertainty in gestures is modeled by type-2 fuzzy membership functions. Type-2 fuzzy rules are employed to recognize gestures. The robot learns different tasks from human demonstrations, and finally principles of non-monotonic reinforcement learning are used for mapping the gestures to the tasks.

Over 20,000 gestures obtained from 10 subjects are used to validate the effectiveness of the gesture recognition scheme. One interesting parameter is employed in the learning dynamics to control the unlearning rate of old knowledge and learning rate of the new knowledge about gesture-to-action mapping in non-monotonic circumstances. The choice of this parameter is left to the user for his specific application. Experiments have been undertaken to confirm that the proposed system works successfully with negligible errors in angular displacements of all the joints of the robot arm.

The proposed system is expected to have applications in robotic surgery and task-planning in hazardous environment.

Acknowledgement

This thesis marks the culmination of my two years as a Masters' student at Jadavpur University. Coming from a family of alumni of Jadavpur University, when the opportunity to study here came up in 2017, I fulfilled a childhood ambition. However, it was just the beginning. Over the past two years as a student and almost fifteen months as a young researcher in the Artificial Intelligence Laboratory, I have learnt a lot as a student and grown even more as a person. I would like to take this opportunity to express my sincere gratitude to just a few of the many people who have made these last two years so wonderful.

Firstly, I am deeply indebted to my research guide Prof. Amit Konar, who despite his innumerable commitments and extremely busy schedule always made time for me. Academically, he has shown me new ways to think creatively and with novelty. My research work so far, and in all likelihood for the foreseeable future will be massively influenced by his wonderful teaching and groundbreaking research topics. He taught me the importance to always keep learning new things and explore things that have never been done before. Beyond academics, he has taught me the virtue of working hard, discipline and never giving up. It has been an enormous pleasure to be his student, and I hope to keep learning from him for years to come.

I would like to acknowledge Prof. Sheli Sinha Chaudhuri, Head of the department, Electronics & Telecommunication Engineering, who was kind enough to provide me with all the necessary facilities to carry out this project. I would also like to acknowledge Prof. P. Venkateswaran, former Head of the department, who helped me settle in seamlessly when I first joined the University. I would like to thank Prof. Ananda Shankar Chowdhury for his continuous encouragement and belief in me. Lastly, I'd like to express my gratitude to Prof. Pratyusha Rakshit, my professor who has looked after me like an elder sister and helped me a lot in this journey.

I am especially thankful to Arindam Ray, who co-worked and co-authored my first paper to be published. It would not have been possible without him.

I have also been extremely fortunate with regards to my seniors whom I have had working with me in the laboratory. Arnab Rakshit, Mousumi Laha, Lidia Ghosh, Susenjit Ghosh and Sayantani Ghosh welcomed me with open arms and have helped me with everything I ever required. My lab mates Zeeshan Qadir, Eashita Chowdhury, Abir Chowdhury, Dipayan Dewan, Tabassum Hossain, Priti Hazra, Biswadeep Chakraborty, Saptak Ghoshal contributed to creating an atmosphere encouraging the most ambitious of ideas and constructive discussions which opened up many new areas of interest for research.

Finally I would like to thank my mother Swati Mukherji, my father Shilbhadra Mukherji, my grandparents and my extended family of close friends and relatives for keeping me motivated through futile days of work and encouraging me to express myself and follow my dreams. I could not have done this without them having my back at all times.

Table of Contents

Abstract	iv
Acknowledgement	v-vi
Table of Contents	vii-ix
List of figures	x-xi
List of Tables	xii
1 Introduction	
1.1 Overview of Automation and Gesture Analysis systems	2
1.2 Thesis Objective	3
1.3 Overview of the work	3
1.4 Thesis Outline	4
1.5 References	5
2 Fuzzy Set Theory	
2.1 Crispness, Vagueness, Fuzziness, Uncertainty	8
2.2 Type-1 Fuzzy Set Theory	9
2.3 Membership Functions	10
2.4 Fuzzy Rules	13
2.5 Type-2 Fuzzy Sets	16
2.6 Summary	19
2.7 References	19
3 Data Clustering	
3.1 Introduction to Clustering	21
3.2 Basic steps of a Clustering task	22
3.3 Definitions and Notations	24
3.4 Clustering Techniques	25
3.5 K-Medoids Clustering	30
3.6 Summary	31

3.7	References	31
4	Reinforcement Learning	
4.1	Learning	34
4.2	Machine Learning	34
4.3	Types of Machine Learning	36
4.4	Reinforcement Learning	37
4.5	Fundamentals of Reinforcement Learning	39
4.6	Q-Learning Algorithm for single agent systems	45
4.7	Summary	50
4.8	References	50
5	Non Monotonic Learning	
5.1	Introduction	53
5.2	Non-monotonic formalisms	54
5.3	Summary	56
5.4	References	56
6	Microsoft Kinect Sensor	
6.1	Microphone Array	59
6.2	RGB Camera	59
6.3	Depth Image Camera	60
6.4	Kinect SDK	64
6.5	Skeletal Tracking using Kinect	64
6.6	Summary	66
6.7	References	67
7	Jaco Robot Arm	
7.1	Introduction	69
7.2	Precautions	70
7.3	Movement of Jaco Arm	70

7.4	Different Parts Of Jaco Arm	72
7.5	Actuator Limitation	73
7.6	External Connection	73
7.7	Classic DH parameters frame position	74
7.8	Joystick	74
7.9	Communication with PC	77
7.10	Interface using API	78
7.11	Programming	79
7.12	Control Types	83
7.13	Summary	84
7.14	References	85
8	Experiment and Analysis	
8.1	System Framework	87
8.2	Experiment	89
8.3	Results	98
8.4	References	104
9	Conclusions	
	Appendix	106

List of figures

2.1	Triangular MF	11
2.2	Gaussian MF	12
2.3	Sigmoidal MF	13
2.4	Different Gaussian type-2 fuzzy sets formed from multiple type-1 fuzzy sets	17
2.5	Footprint of Uncertainty	18
3.1	Data Clustering	21
3.2	Stages of Clustering	23
3.3	A taxonomy of clustering approaches	25
3.4	Monothetic partitional clustering	26
3.5	k-means: Sensitive to initial partition	28
4.1	Reinforcement Learning Framework	41
4.2	Flowchart for Q-Learning	47
6.1	Front view of Microsoft Kinect	59
6.2	Visualisation of the depth image camera principle	62
6.3	Depth image of Microsoft Kinect	63
7.1	Different Parts Of Jaco Arm	72
7.2	Jaco Arm External Connections	73
7.3	Frame Assignments	74
7.4	Joystick	75
7.5	Angular Control via Programming	83
7.6	Cartesian Control via Programming	84
8.1	System Overview	87
8.2	Variations for a given hand position	89
8.3	Representation of the parts of the upper limb as vectors	90
8.4	Clustering the input data from multiple subjects	92
8.5	Construct ion of interval type-2 fuzzy sets from clusters	93

8.6	Interval Type-2 fuzzy set construction	94
8.7	Procedure for performing gesture driven tasks	97
8.8	Task Learning Process	98
8.9	An isolated frame during Robot Task Learning	101
8.10	Hand positions for a Gesture	102
8.11	Gesture inference using fuzzy rule sets	102
8.12	Variation of β against unlearning time	103

List of Tables

6.1	RGB Camera Specifications	60
6.2	Depth image camera specification	62
7.1	Jaco Arm Specifications	69
7.2	Minimum and Maximum Position of Joint Angles	73
7.3	Description of different parts of external connection	74
7.4	Description of joystick buttons	75
7.5	Blue LED condition	76
7.6	Green LED condition	77
7.7	Red LED condition	77
7.8	Folder description	78
8.1	Arm Positions	99
8.2	Observations for Shoulder Angle	99
8.3	Observations for Elbow Angle	100
8.4	Deviation of robot position from teacher	101

1

Introduction

1.1 Overview of Automation and Gesture Analysis systems

The role of human operators in several industries is declining rapidly, owing to the several drawbacks encountered in carrying out certain specific tasks by them. These drawbacks include imprecise execution of tasks, variance in task execution between different operators and carelessness among operators, to name a few. Moreover, in certain industrial scenarios, these drawbacks are accentuated due to the presence of a non-ideal work environment (like mining industries or in boiler plants). This creates a serious bottleneck in the industrial work flow and vastly decreases the productivity and efficacy of the items under production.

To overcome the impasse, various forms of automation have been realized over the last two decades to serve the needs. Robotic surgery is one important field with much scope for intelligent automation [1],[2]. However, despite technological advancements there continues to be reports of adverse events[3] and one of the reasons alluded to is the requirement of doctors highly skilled in surgeries. In addition, for robotic arms whose movements are controlled by using joysticks or similar controllers, accuracy is limited by the precision of the operators.

Over the years many works have been carried out on gesture recognition [4]-[11]. In one of the earlier works, Murakami and Taguchi [4] analyzed time series data using recurrent neural networks to recognize different words in Japanese sign language. Chen, Fu and Huang [9] proposed a Hidden Markov Model (HMM) based method which uses both temporal and spatial features to identify a continuous moving hand gesture. Su [6] employed a fuzzy rule based approach, in addition to use of spatio-temporal data and neural networks. Fang *et al.* [8] proposed an appearance model based method for real time tracking of the hand followed by gesture recognition using the finger and palm configurations. Rautaray and Agrawal [5], Mitra and Acharya [10] present surveys of the works on Human-computer interaction involving gesture recognition using vision systems using various techniques and devices.

The Kinect device, though initially released for Microsoft gaming console XBOX, has found varied use in computer vision systems [12]. Wang *et al.* [11] proposed an approach using HMM where the palm is taken as a Kinect node. In [7], Wang, Chen and Chen used the

Kinect skeleton tracking system for identification of two hand gestures with the aid of decoupled HMM.

1.2 Thesis Objective

In this thesis, we propose a system where the subject will be able to control a robot arm capable of executing complex tasks through simple gestures or a sequence of gestures. The subject need not be skilled in the particular field of application, and would only require knowledge about (a) the different tasks the robot is capable of executing and their functionality, and (b) the gestures which trigger each such task. The novelty of the work is that the system designed allows the detection of gestures that are imprecise. Moreover, the subject can operate the system from a place isolated from the actual robot application, thereby preventing any health hazards that may arise from working in an environment that is dangerous for the subject.

1.3 Overview of the work

In human-to-human communication, gesture is often used to describe the subjective intention without verbal communication. Normally, people do not take much trouble to accurately and precisely enact a gesture. However, due to natural intelligence of humans, they can easily recognize approximate gestures. Unfortunately, robots cannot recognize approximate gestures. One approach to make robot understand human natural gestures is to quantize gestural parameters, so that a gestural parameter falls between two distinct quantized levels. The logic of fuzzy sets provides us the benefits of quantizing gestural parameters. In fact, type-2 fuzzy sets can be used to model a gesture or its parameters, where the footprint of uncertainty [13] represents quite a large (theoretically infinite) number of gestures. Naturally, any unknown gesture can be described by a set of fuzzy attributes (parameters), which jointly coexist in the antecedent of a rule with the gesture level at the consequent. Given an unknown gesture, we compute the firing strength of all the fired rules and identify the rule with the highest firing strength. The gesture indicated in the consequent of the rule is used to derive the

fuzzy inference about the inferred gesture. Besides gesture recognition the other important part of the work is to learn to encode the gesture to action planning by the robot. The robot learns the gesture-to-action plan by using Reinforcement learning [15]. An interesting parameter is used in the learning dynamics to control the un-learning of noisy knowledge and relearn new knowledge about gesture-to-action planning. The acquired knowledge is used later in the planning phase of the robot to take the right action based on the gestural input provided by the subject.

In this work, we use the Kinect skeleton tracker to obtain different joint angles of the subject, which are then clustered to form gesture components during training. At the test phase, fuzzy rule based approach is used to recognize an unknown gesture. For training a robot to perform different tasks, we have used the principle of robot learning through demonstration [14] such that after training the robot is able to imitate the teacher. Finally, for matching a gesture to a task we use a network, such that a weight is attached to each gesture-task pair. User feedback is obtained after each execution, and based on reward-penalty approach of reinforcement learning, the weights are updated. As an input gesture is mapped to a task proportional to the weights, the weight updating principle ensures the network learns over time.

Furthermore, this approach is flexible towards changing of user feedback over time, and can unlearn a set of mappings, in order to learn a new set of mappings, following the principle of non-monotonic learning.

1.4 Thesis Outline

This chapter outlines the need for automation, but also the roadblocks for its implementation. It begins with an overview of the existing systems in use to highlight the need for our work. A broad overview of the proposed system is covered in the preceding sections. Chapters 2 to 5 covers the basic principles of the different concepts used in our work. Chapter 6 contains an overview of the Microsoft Kinect Device used in our experiment. In Chapter 7, the Jaco robot arm, used for our study, is described in detail. Chapter 8 discusses the experimental work carried out and the obtained results. In Chapter 9, some conclusive remarks and comments regarding prospective future work has been presented.

1.5 References

- [1] Lanfranco A.R., Castellanos A.E., Desai J.P. and Meyers W.C., “Robotic Surgery: A Current Perspective,” *Annals of Surgery*, 239(1), pp. 14-21, 2004.
- [2] Abboudi H., Khan M.S., Aboumarzouk O., Guru K.A., Challacombe B., Dasgupta P. and Ahmed K., “Current status of validation for robotic surgery simulators,” *BJU International*, vol. 111, issue 2, pp. 194-205, 2013.
- [3] Alemzadeh H., Raman J., Leveson N., Kalbarczyk Z., Iyer R.K., “Adverse Events in Robotic Surgery: A Retrospective Study of 14 Years of FDA Data,” *PLoS ONE*, vol. 11, issue 4, 2016.
- [4] Murakami K. and H. Taguchi, “Gesture Recognition using Recurrent Neural Networks,” *Proceedings, SIGCHI Conference on Human Factors in Computing Systems*, pp. 237-242, 1991.
- [5] S. S. Rautaray and A. Agrawal, “Vision based hand gesture recognition for human computer interaction: a survey,” *Artificial Intelligence Review*, vol. 43, issue 1, pp. 1-54, January 2015.
- [6] Mu-Chun Su, “A Fuzzy Rule-Based Approach to Spatio-Temporal Hand Gesture Recognition,” *IEEE Transactions on Systems, Man, and Cybernetics—part c: Applications and Reviews*, vol. 30, no. 2, May2000.
- [7] B. Wang, Z. Chen and J. Chen, “Gesture Recognition by Using Kinect Skeleton Tracking System,” *Intelligent Human-Machine Systems and Cybernetics*, 2013.
- [8] Y. Fang, K. Wang, J. Cheng and H. Lu, “A Real-Time Hand Gesture Recognition Method,” *Proceedings, IEEE International Conference Multimedia and Expo*, pp. 995–998, 2007.
- [9] F.-S. Chen, C.-M. Fu, and C.-L. Huang, “Hand Gesture RecognitionUsing a Real-Time Tracking Method and Hidden Markov Models,”*Image and Vision Computing*, vol. 21, no. 8, pp. 745-758, 2003.
- [10] S. Mitra and T. Acharya, "Gesture Recognition: A Survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 3, pp. 311-324, May 2007.

- [11] Wang Y., Yang C., Wu X., Xu S., and Li H., “Kinect Based Dynamic Hand Gesture Recognition Algorithm Research,” International Conference on Intelligent Human-Machine Systems and Cybernetics, pp.274–279, 2012.
- [12] Han J., Shao L., Xu D., &Shotton J., “Enhanced Computer Vision With Microsoft Kinect Sensor: A Review,” IEEE Transactions on Cybernetics, vol. 43, pp. 1318-1334, 2013.
- [13] Q. Liang and J. M. Mendel, “Interval Type-2 Fuzzy Logic Systems: Theory and Design,” IEEE Transactions On Fuzzy Systems, vol. 8, No.5, October 2000.
- [14] B. D. Argall, S. Chernova, M. Veloso and B. Browning, “A survey of robot learning from demonstration,” Robotics and Autonomous Systems, vol. 57, issue 5, pp. 469-483, May 2009.
- [15] A. Konar, Computational Intelligence: principles, techniques and applications. Springer Science & Business Media, 2006.

2

Fuzzy Set Theory

2.1 Crispness, Vagueness, Fuzziness, Uncertainty

Most of our traditional tools for formal modeling, reasoning, and computing are crisp, deterministic, and precise in character. By crisp we mean dichotomous, that is, yes-or-no-type rather than more-or-less type. In conventional dual logic, for instance, a statement can be true or false—and nothing in between. In set theory, an element can either belong to a set or not; and in optimization, a solution is either feasible or not. Precision assumes that the parameters of a model represent exactly either our perception of the phenomenon modeled or the features of the real system that has been modeled. Generally, precision also implies that the model is unequivocal, that is, that it contains no ambiguities [1].

However, Schwarz brings up another argument against the non-reflective use of mathematics when he states: "An argument, which is only convincing if it is precise loses all its force if the assumptions on which it is based are slightly changed, while an argument, which is convincing but imprecise may well be stable under small perturbations of its underlying axioms." For factual models or modeling languages, two major complications arise:

1. Real situations are very often not crisp and deterministic, and they cannot be described precisely.
2. The complete description of a real system often would require far more detailed data than a human being could ever recognize simultaneously, process, and understand.

Fuzziness can be found in many areas of daily life, such as in engineering, medicine, meteorology, manufacturing, and others. It is particularly frequent, however, in all areas in which human judgment, evaluation, and decisions are important. These are the areas of decision making, reasoning, learning, and so on. Some reasons for this fuzziness have already been mentioned. Others are that most of our daily communication uses "natural languages," and a good part of our thinking is done in it. In these natural languages, the meaning of words is very often vague. The meaning of a word might even be well defined, but when using the word as a label for a set, the boundaries within which objects do or do not belong to the set become fuzzy or vague. Examples are words such as "birds" (how about penguins, bats, etc.?) or "red roses," but also terms such as "tall men," "beautiful women," and "creditworthy customers." In this context we can probably distinguish two kinds of fuzziness with respect to

their origins: intrinsic fuzziness and informational fuzziness. The former is illustrated by "tall men." This term is fuzzy because the meaning of tall is fuzzy and dependent on the context (height of observer, culture, etc.). An example of the latter is the term "creditworthy customers": A creditworthy customer can possibly be described completely and crisply if we use a large number of descriptors. These descriptors are more, however, than a human being could handle simultaneously. Therefore the term, which in psychology is called a "subjective category," becomes fuzzy.

2.2 Type-1 Fuzzy Set Theory

Let X be a space of objects and x be a generic element of X . A classical set A , $A \subseteq X$, is defined by a collection of elements or objects $x \in X$, such that each x can either belong or not belong to the set A [2]. By defining a "characteristic function" for each element $x \in X$, we can represent a classical set A by a set of order pairs $(x,0)$ or $(x,1)$, which indicates $x \notin A$ or $x \in A$, respectively. Unlike the aforementioned conventional set, a fuzzy set [3] expresses the degree to which an element belongs to a set. Hence the characteristic function of a fuzzy set is allowed to have values between 0 and 1, which denotes the degree of membership of an element in a given set.

2.2.1 Fuzzy sets and membership functions

If X is a collection of objects denoted generically by x , then a "fuzzy set" A in X is defined as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) \mid x \in X\}$$

where $\mu_A(x)$ is called "membership function" (or MF for short) for the fuzzy set A .

The MF maps each element of X to a membership grade (or membership value) between 0 and 1. Obviously, the definition of a fuzzy set is a simple extension of the definition of a classical set in which the characteristic function is permitted to have any values between 0 and 1. If the values of the membership function $\mu_A(x)$ is restricted to either 0 or 1, then A is reduced to a classical set and $\mu_A(x)$ is the characteristic function of A .

2.2.2 Union

The "union" of two fuzzy sets A and B is a fuzzy set C, written as $C = A \cup B$ or $C = A \text{ OR } B$, whose MF is related to those of A and B by:

$$\mu_C(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$$

2.2.3 Intersection

The "intersection" of two fuzzy sets A and B is a fuzzy set C, written as $C = A \cap B$ or $C = A \text{ AND } B$, whose MF is related to those of A and B by

$$\mu_C(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$$

2.2.4 Complement or Negation

The "complement" of a fuzzy set A, denoted by $\neg A$ (NOT A), is defined as

$$\mu_{\neg A}(x) = 1 - \mu_A(x)$$

2.3 Membership Functions

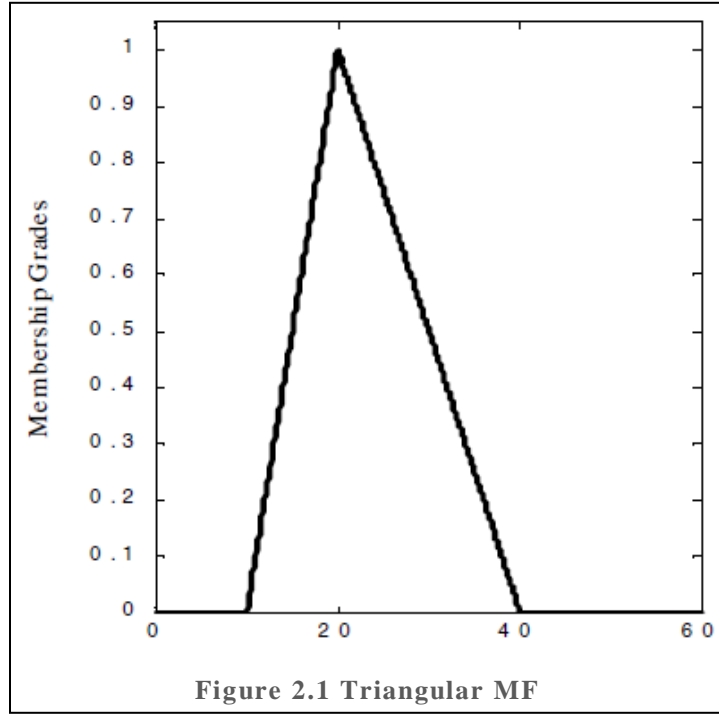
A fuzzy set is completely characterized by its MF. Since most fuzzy sets in use have a universe of discourse X consisting of the real line R, it would be impractical to list all the pairs defining a membership function. A more convenient and concise way to define a MF is to express it as a mathematical formula. Here we define several classes of parameterized MFs of one dimension.

2.3.1 Triangular MFs

A "triangular MF" is specified by three parameters $\{a, b, c\}$ as follows:

$$y = \text{triangle}(x; a, b, c) = \begin{cases} = 0, & x \leq a \\ = (x - a) / (b - a), & a \leq x \leq b \\ = (c - x) / (c - b), & b \leq x \leq c \\ = 0, & c \leq x \end{cases}$$

The parameters $\{a, b, c\}$ (with $a < b < c$) determine the x coordinates of the three corners of the underlying triangular MF. Figure 2.1 illustrates a triangular MF defined by $\text{triangle}(x; 10, 20, 40)$.



2.3.2 Gaussian MFs

A "Gaussian MF" is specified by two parameters $\{c, \sigma\}$:

$$\text{gaussian}(x; c, \sigma) = \exp\left(-\frac{(x - c)^2}{2\sigma^2}\right)$$

A "Gaussian" MF is determined completely by c and σ ; c represents the MFs center and σ determines the MFs width. Figure 2.2 plots a Gaussian MF defined by $\text{gaussian}(x; 50, 20)$.

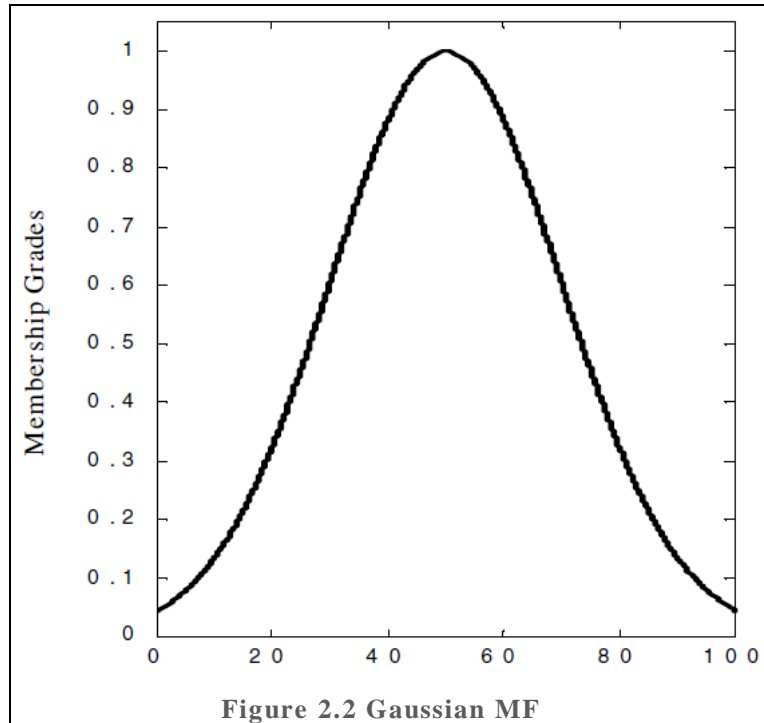


Figure 2.2 Gaussian MF

2.3.3 Sigmoidal MFs

A "Sigmoidal MF" is defined by the following equation:

$$\text{sig}(x; a, c) = \frac{1}{1 + \exp[-a(x - c)]}$$

where a controls the slope at the crossover point $x = c$.

Depending on the sign of the parameter " a ", a sigmoidal MF is inherently open right or left and thus is appropriate for representing concepts such as "very large" or "very negative".

Figure 2.3 shows two sigmoidal functions $y_1 = \text{sig}(x; 1, -5)$ and $y_2 = \text{sig}(x; -2, 5)$.

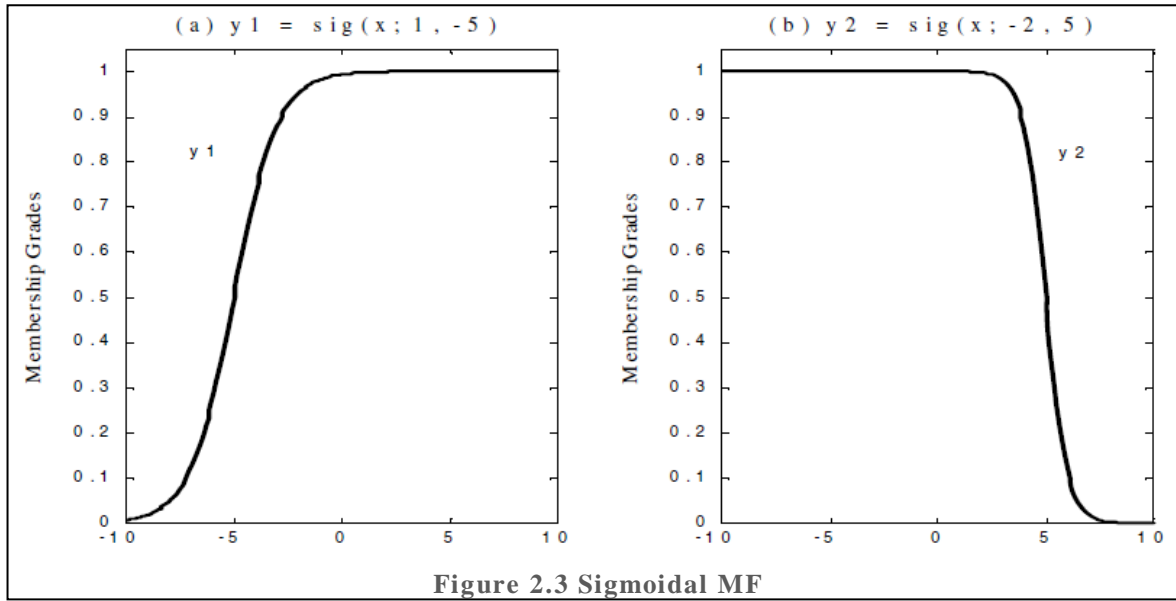


Figure 2.3 Sigmoidal MF

2.4 Fuzzy Rules

As was pointed out by Zadeh in his work on this area [6], conventional techniques for system analysis are intrinsically unsuited for dealing with humanistic systems, whose behavior is strongly influenced by human judgment, perception, and emotions. This is a manifestation of what might be called the "principle of incompatibility": "As the complexity of a system increases, our ability to make precise and yet significant statements about its behavior diminishes until a threshold is reached beyond which precision and significance become almost mutually exclusive characteristics" [6]. It was because of this belief that Zadeh proposed the concept of linguistic variables [7] as an alternative approach to modeling human thinking.

2.4.1 Linguistic Variables

A "Linguistic variable" is characterized by a quintuple $(x, T(x), X, G, M)$ in which x is the name of the variable; $T(x)$ is the "term set" of x -that is, the set of its "linguistic values" or "linguistic terms"; X is the universe of discourse, G is a "syntactic rule" which generates the

terms in $T(x)$; and M is a "semantic rule" which associates with each linguistic value A its meaning $M(A)$, where $M(A)$ denotes a fuzzy set in X .

2.4.2 Fuzzy If-Then Rules

A "fuzzy if-then rule" (also known as "fuzzy rule", "fuzzy implication", or "fuzzy conditional statement") assumes the form

if x is A then y is B ,

where A and B are linguistic values defined by fuzzy sets on universes of discourse X and Y , respectively. Often " x is A " is called "antecedent" or "premise", while " y is B " is called the "consequence" or "conclusion".

Examples of fuzzy if-then rules are widespread in our daily linguistic expressions, such as the following:

- If pressure is high, then volume is small.
- If the road is slippery, then driving is dangerous.
- If the speed is high, then apply the brake a little.

Before we can employ fuzzy if-then rules to model and analyze a system, first we have to formalize what is meant by the expression "if x is A then y is B ", which is sometimes abbreviated as $A \rightarrow B$. In essence, the expression describes a relation between two variables x and y ; this suggests that a fuzzy if-then rule is defined as a binary fuzzy relation R on the product space $X \times Y$. Generally speaking, there are two ways to interpret the fuzzy rule $A \rightarrow B$. If we interpret $A \rightarrow B$ as A "coupled with" B then

$$R = A \rightarrow B = A \times B = \int_{x \times y} \mu_A(x) * \mu_B(y) / (x, y)$$

where $*$ is an operator for intersection [8]. On the other hand, if $A \rightarrow B$ is interpreted as A "entails" B , then it can be written as one of two different formulas:

- Material implication:

$$R = A \rightarrow B = \neg A \cup B$$

- Propositional Calculus:

$$R = A \rightarrow B = \neg A \cup (A \cap B)$$

Fuzzy reasoning, also known as approximate reasoning, is an inference procedure that derives conclusions from a set of fuzzy if-then rules and known facts. The basic rule of inference in traditional two-valued logic is "modus ponens", according to which we can infer the truth of a proposition B from the truth of A and the implication $A \rightarrow B$. This concept is illustrated as follows:

premise 1 (fact): x is A ,
 premise 2 (rule): if x is A then y is B ,

 consequence (conclusion): y is B .

However, in much of human reasoning, modus ponens is employed in an approximate manner. This is written as

premise 1 (fact): x is A'
 premise 2 (rule): if x is A then y is B ,

 consequence (conclusion): y is B'

where A' is close to A and B' is close to B . When A , B , A' and B' are fuzzy sets of appropriate universes, the foregoing inference procedure is called "approximate reasoning" or "fuzzy reasoning"; it is also called "generalized modus ponens" (GMP for short), since it has modus ponens as a special case.

2.4.3 Fuzzy Reasoning

Let A , A' , and B be fuzzy sets of X , X , and Y respectively. Assume that the fuzzy implication $A \rightarrow B$ is expressed as a fuzzy relation R on $X \times Y$. Then the fuzzy set B induced by " x is A " and the fuzzy rule "if x is A then y is B " is defined by

$$\begin{aligned}\mu_{B'}(y) &= \max_x \min[\mu_{A'}(x), \mu_R(x, y)] \\ &= \vee_x [\mu_{A'}(x) \wedge \mu_R(x, y)]\end{aligned}$$

Now we can use the inference procedure of fuzzy reasoning to derive conclusions provided that the fuzzy implication $A \rightarrow B$ is defined as an appropriate binary fuzzy relation.

2.5 Type-2 Fuzzy Sets

The concept of a type-2 fuzzy set, was introduced by Zadeh[4] as an extension of the concept of an ordinary fuzzy set (henceforth called a “type-1 fuzzy set”). A type-2 fuzzy set is characterized by a fuzzy membership function, i.e., the membership grade for each element of this set is a fuzzy set in $[0,1]$, unlike a type-1 set where the membership grade is a crisp number in $[0,1]$. Such sets can be used in situations where there is uncertainty about the membership grades themselves, e.g., an uncertainty in the shape of the membership function or in some of its parameters. Consider the transition from ordinary sets to fuzzy sets. When we cannot determine the membership of an element in a set as 0 or 1, we use fuzzy sets of type-1. Similarly, when the situation is so fuzzy that we have trouble determining the membership grade even as a crisp number in $[0,1]$, we use fuzzy sets of type-2. This does not mean that we need to have extremely fuzzy situations to use type-2 fuzzy sets. There are many real-world problems where we cannot determine the exact form of the membership functions, e.g., in time series prediction because of noise in the data. Another way of viewing this is to consider type-1 fuzzy sets as a first order approximation to the uncertainty in the real-world. Then type-2 fuzzy sets can be considered as a second order approximation. Of course, it is possible to consider fuzzy sets of higher types but the complexity of the fuzzy system increases very rapidly. For this reason, we will only consider very briefly type-2 fuzzy sets. Let us consider some simple examples of type-2 fuzzy sets.

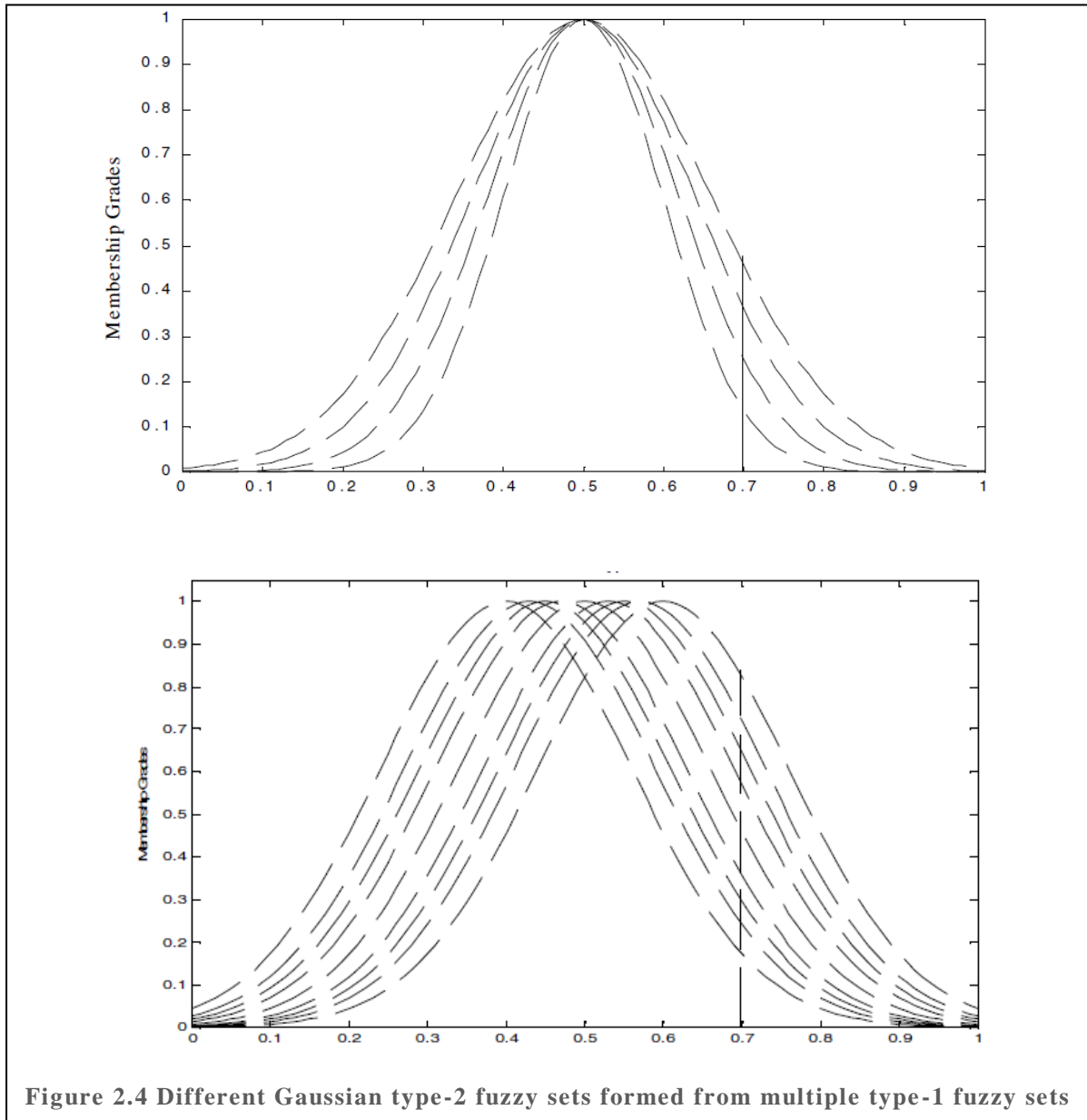


Figure 2.4 Different Gaussian type-2 fuzzy sets formed from multiple type-1 fuzzy sets

2.5.1 Gaussian Type-2 Fuzzy Set

A Gaussian type-2 fuzzy set is one in which the membership grade of every domain point is a Gaussian type-1 set contained in $[0,1]$.

2.5.2 Interval Type-2 Fuzzy Set

An interval type-2 fuzzy set is one in which the membership grade of every domain point is a crisp set whose domain is some interval contained in $[0,1]$.

2.5.3 Footprint of Uncertainty

Uncertainty in the primary memberships of a type-2 fuzzy set, \tilde{A} , consists of a bounded region that we call the “footprint of uncertainty” (FOU). Mathematically, it is the union of all primary membership functions [5]. We show as an illustration in Figure 2.5 the footprint of uncertainty for a type-2 Gaussian membership function.

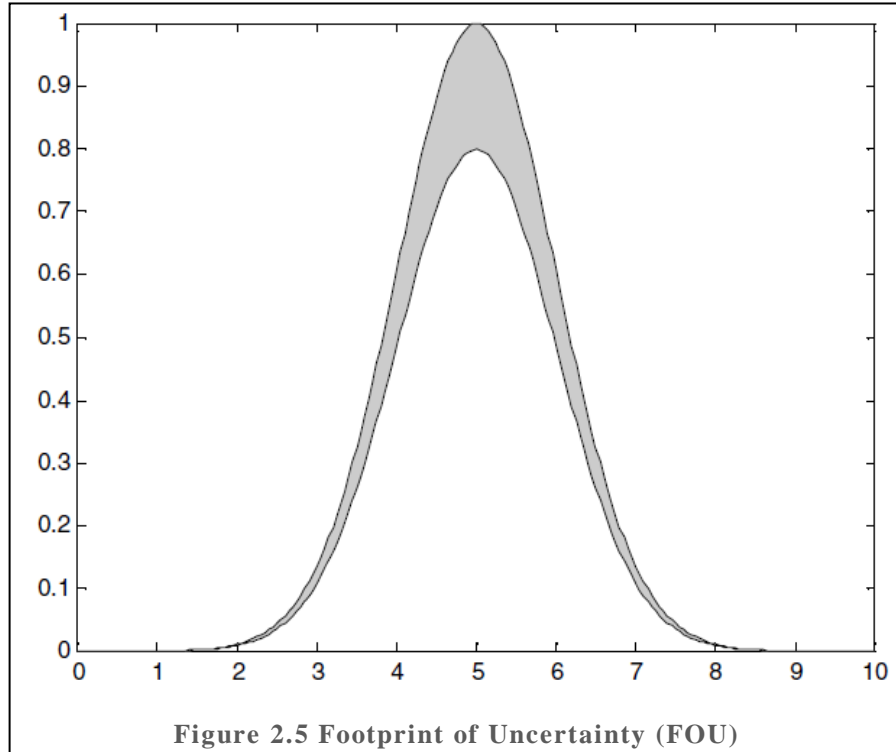


Figure 2.5 Footprint of Uncertainty (FOU)

2.5.4 Upper and Lower Membership Functions

An “upper membership function” and a “lower membership function” are two type-1 membership functions that are bounds for the FOU of a type-2 fuzzy set \tilde{A} . The upper

membership function is associated with the upper bound of $\text{FOU}(\tilde{A})$. The lower membership function is associated with the lower bound of $\text{FOU}(\tilde{A})$.

2.6 Summary

In this chapter, a basic overview of fuzzy set theory is provided. First, the motivation and need for fuzzy theory is discussed. Then, some introductory principles and definitions regarding type-1 fuzzy sets is included. In the subsequent sections rudiments of fuzzy reasoning is considered. Finally, an elementary foundation of type-2 fuzzy sets is provided. In our proposed work, many of these principles have been utilized to design the system.

2.7 References

- [1] Zimmermann, Hans-Jürgen. *Fuzzy set theory—and its applications*. Springer Science & Business Media, 2011.
- [2] Castillo, Oscar, and Patricia Melin. *Type-2 Fuzzy Logic: Theory and Applications*. Vol. 223. Springer Science & Business Media, 2008.
- [3] Zadeh, Lotfi A. "Fuzzy sets." *Information and control* 8.3 (1965): 338-353.
- [4] Zadeh, Lotfi Asker. "The concept of a linguistic variable and its application to approximate reasoning—I." *Information sciences* 8.3 (1975): 199-249.
- [5] Karnik, Nilesh N., and Jerry M. Mendel. "Operations on type-2 fuzzy sets." *Fuzzy sets and systems* 122.2 (2001): 327-348.
- [6] Zadeh, Lotfi A. "Outline of a new approach to the analysis of complex systems and decision processes." *IEEE Transactions on systems, Man, and Cybernetics* 1 (1973): 28-44.
- [7] Zadeh, Lotfi A. "Quantitative fuzzy semantics." *Information sciences* 3.2 (1971): 159-176.
- [8] Mamdani, Ebrahim H., and SedrakAssilian. "An experiment in linguistic synthesis with a fuzzy logic controller." *International journal of man-machine studies* 7.1 (1975): 1-13.

3

Data Clustering

Clustering is the unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters). The clustering problem has been addressed in many contexts and by researchers in many disciplines; this reflects its broad appeal and usefulness as one of the steps in exploratory data analysis.

3.1 Introduction to Clustering

Data analysis underlies many computing applications, either in a design phase or as part of their on-line operations [1]. Data analysis procedures can be dichotomized as either exploratory or confirmatory, based on the availability of appropriate models for the data source, but a key element in both types of procedures (whether for hypothesis formation or decision-making) is the grouping, or classification of measurements based on either (i) goodness-of-fit to a postulated model, or (ii) natural groupings (clustering) revealed through analysis. Cluster analysis is the organization of a collection of patterns (usually represented as a vector of measurements, or a point in a multidimensional space) into clusters based on similarity. Intuitively, patterns within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster. An example of clustering is depicted in Figure 3.1. The input patterns are shown in Figure 3.1(a), and the desired clusters are shown in Figure 3.1(b).

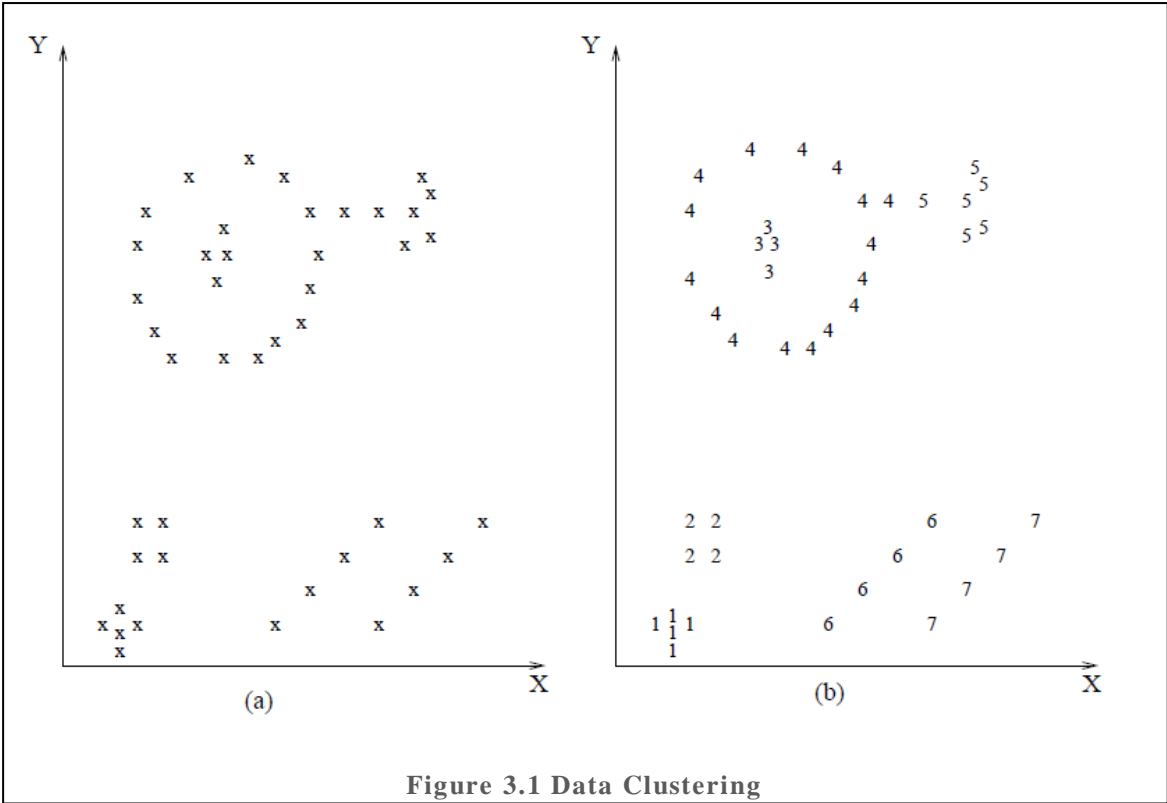


Figure 3.1 Data Clustering

Here, points belonging to the same cluster are given the same label.

The variety of techniques for representing data, measuring proximity (similarity) between data elements, and grouping data elements has produced a rich and often confusing assortment of clustering methods. It is important to understand the difference

between clustering (unsupervised classification) and discriminant analysis (supervised classification).

In supervised classification, we are provided with a collection of *labeled* (pre-classified) patterns; the problem is to label a newly encountered, yet un-labeled, pattern. Typically, the given labeled (*training*) patterns are used to learn the descriptions of classes which in turn are used to label a new pattern.

In the case of clustering, the problem is to group a given collection of unlabeled patterns into meaningful clusters. In a sense, labels are associated with clusters also, but these category labels are *data driven*; that is, they are obtained solely from the data. Clustering is useful in several exploratory pattern-analysis, grouping, decision-making, and machine-learning situations, including data mining, document retrieval, image segmentation, and pattern classification. However, in many such problems, there is little prior information (e.g., statistical models) available about the data, and the decision-maker must make as few assumptions about the data as possible. It is under these restrictions that clustering methodology is particularly appropriate for the exploration of inter-relationships among the data points to make an assessment (perhaps preliminary) of their structure.

3.2 Basics steps of a Clustering task

Typical pattern clustering activity involves the following steps [3]:

- I. Pattern representation (optionally including feature extraction and/or selection),
- II. Definition of a pattern proximity measure appropriate to the data domain,
- III. Clustering or grouping,
- IV. Data abstraction (if needed), and
- V. Assessment of output (if needed).

Figure 3.2 depicts a typical sequencing of the first three of these steps, including a feedback path where the grouping process output could affect subsequent feature extraction and similarity computations.

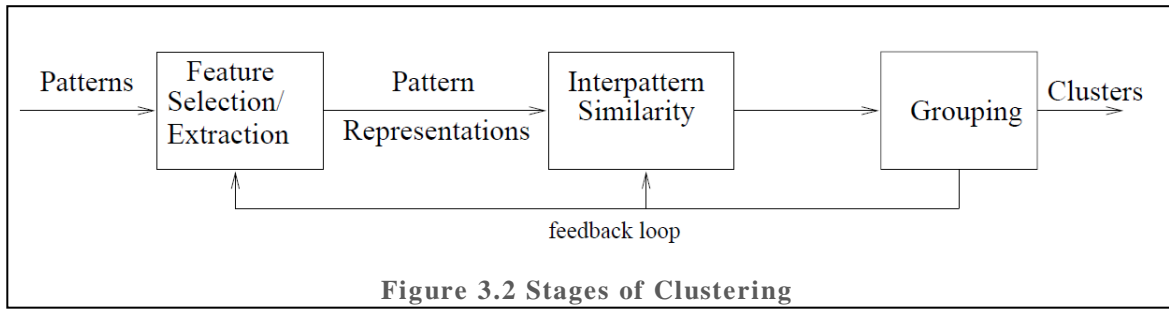


Figure 3.2 Stages of Clustering

Pattern representation refers to the number of classes, the number of available patterns, and the number, type, and scale of the features available to the clustering algorithm. Some of this information may not be controllable by the practitioner.

Feature selection is the process of identifying the most effective subset of the original features to use in clustering.

Feature extraction is the use of one or more transformations of the input features to produce new salient features. Either or both of these techniques can be used to obtain an appropriate set of features to use in clustering.

Pattern proximity is usually measured by a distance function defined on pairs of patterns. A variety of distance measures are in use in the various communities[3]-[5]. A simple distance measure like Euclidean distance can often be used to reflect dissimilarity between two patterns, whereas other similarity measures can be used to characterize the conceptual similarity between patterns [6].

The *grouping* step can be performed in a number of ways. The output clustering(or clusterings) can be hard (a partition of the data into groups) or fuzzy (where each pattern has a variable degree of membership in each of the output clusters). For example, Hierarchical clustering algorithms produce a nested series of partitions based on a criterion for merging or splitting clusters based on similarity. While partitional clustering algorithms identify the partition that optimizes(usually locally) a clustering criterion.

Data abstraction is the process of extracting a simple and compact representation of a data set. Here, simplicity is either from the perspective of automatic analysis (so that a machine can perform further processing efficiently) or it is human-oriented (so that the representation obtained is easy to comprehend and intuitively appealing). In the clustering context, a typical data abstraction is a compact description of each cluster, usually in terms of cluster prototypes or representative patterns such as the centroid[5].

How is the output of a clustering algorithm evaluated? What characterizes a ‘good’ clustering result and a ‘poor’ one?

All clustering algorithms will, when presented with data, produce clusters — regardless of whether the data contain clusters or not. If the data does contain clusters, some clustering algorithms may obtain ‘better’ clusters than others. The assessment of a clustering procedure’s output, then, has several facets. One is actually an assessment of the data domain rather than the clustering algorithm itself— data which do not contain clusters should not be processed by a clustering algorithm. *Cluster validity* analysis, by contrast, is the assessment of a clustering procedure’s output. Often this analysis uses a specific criterion of optimality; however, these criteria are usually arrived at subjectively. Hence, little in the way of ‘gold standards’ exist in clustering except in well-prescribed sub-domains. A clustering structure is valid if it cannot reasonably have occurred by chance or as an artifact of a clustering algorithm. When statistical approaches to clustering are used, validation is accomplished by carefully applying statistical methods and testing hypotheses.

3.3 Definitions and Notations

The following terms and notation are used throughout this chapter.

- A *pattern* (or *feature vector*, *observation*, or *datum*) \mathbf{x} is a single data item used by the clustering algorithm. It typically consists of a vector of d measurements: $\mathbf{x} = (x_1, \dots, x_d)$
- The individual scalar components x_i of a pattern \mathbf{x} are called *features* (or *attributes*).
- d is the *dimensionality* of the pattern or of the pattern space.
- A *pattern set* is denoted $\mathbf{x} = \{x_1, \dots, x_n\}$.

The i th pattern in \mathbf{x} is denoted $\mathbf{x}_i = (x_{i,1}, \dots, x_{i,d})$.

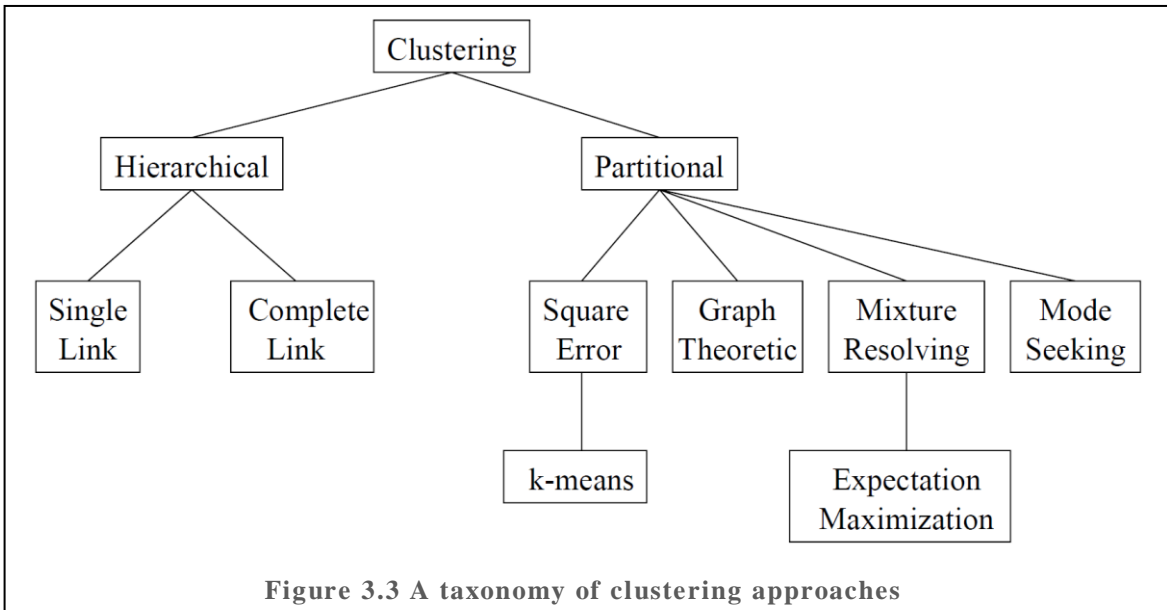
In many cases a pattern set to be clustered is viewed as an $n \times d$ *pattern matrix*.

- A *class*, in the abstract, refers to a state of nature that governs the pattern generation process in some cases. More concretely, a class can be viewed as a source of patterns whose distribution in feature space is governed by a probability density specific to the class. Clustering techniques attempt to group patterns so that the classes thereby obtained reflect the different pattern generation processes represented in the pattern set.
- Hard* clustering techniques assign a *class label* l_i to each patterns \mathbf{x}_i , identifying its class. The set of all labels for a pattern set \mathbf{x} is $\mathbf{l} = \{l_1, \dots, l_n\}$ with $l_i \in \{1, \dots, k\}$ where k is the number of clusters.

- Fuzzy* clustering procedures assign to each input pattern \mathbf{x}_i a fractional degree of membership f_{ij} in each output cluster j .
- A *distance measure* (a specialization of a proximity measure) is a metric (or quasi-metric) on the feature space used to quantify the similarity of patterns.

3.4 Clustering Techniques

Different approaches to clustering data can be described with the help of the hierarchy shown in Figure 3.3.

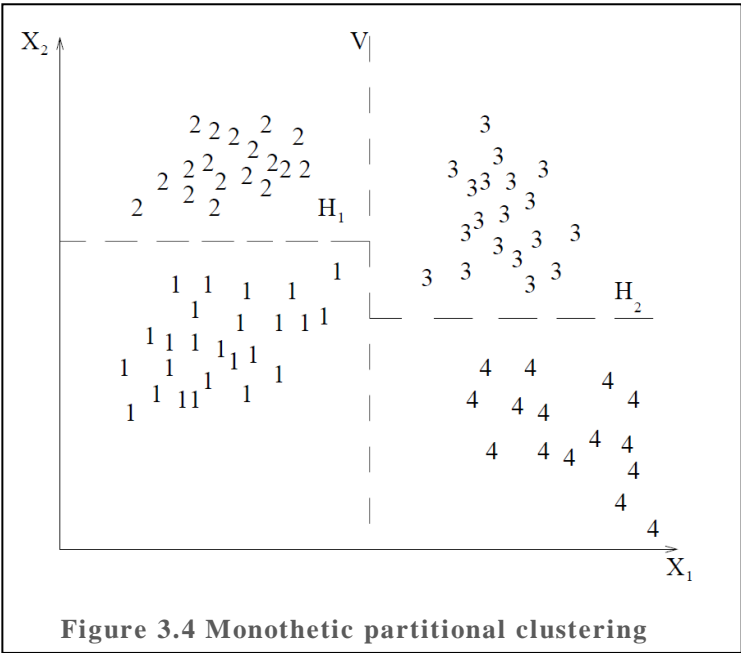


At the top level, there is a distinction between hierarchical and partitional approaches (hierarchical methods produce a nested series of partitions, while partitional methods produce only one).

The taxonomy shown in Figure 3.3 must be supplemented by a discussion of cross-cutting issues that may (in principle) affect all of the different approaches regardless of their placement in the taxonomy.

- Agglomerative *vs.* divisive: This aspect relates to algorithmic structure and operation. An agglomerative approach begins with each pattern in a distinct (singleton) cluster, and successively merges clusters together until a stopping criterion is satisfied. A divisive method begins with all patterns in a single cluster and performs splitting until a stopping criterion is met.
- Monothetic *vs.* polythetic: This aspect relates to the sequential or simultaneous use of features in the clustering process. Most algorithms are polythetic; that is, all features enter

into the computation of distances between patterns, and decisions are based on those distances. A simple monothetic algorithm reported in [4] considers features sequentially to divide the given collection of patterns. This is illustrated in Figure 3.4.



Here, the collection is divided into two groups using feature x_1 ; the vertical broken line V is the separating line. Each of these clusters is further divided independently using feature x_2 , as depicted by the broken lines H_1 and H_2 . The major problem with this algorithm is that it generates $2d$ clusters where d is the dimensionality of the patterns. For large values of d ($d > 100$ is typical in information retrieval applications [7]), the number of clusters generated by this algorithm is so large that the data set is divided into uninterestingly small and fragmented clusters.

—Hard vs. fuzzy: A hard clustering algorithm allocates each pattern to a single cluster during its operation and in its output. A fuzzy clustering method assigns degrees of membership in several clusters to each input pattern. A fuzzy clustering can be converted to a hard clustering by assigning each pattern to the cluster with the largest measure of membership.

—Deterministic vs. stochastic: This issue is most relevant to partitional approaches designed to optimize a squared error function. This optimization can be accomplished using traditional techniques or through a random search of the state space consisting of all possible labelings.

—Incremental vs. non-incremental: This issue arises when the pattern set to be clustered is large, and constraints on execution time or memory space affect the architecture of the

algorithm. The early history of clustering methodology does not contain many examples of clustering algorithms designed to work with large data sets, but the advent of data mining has fostered the development of clustering algorithms that minimize the number of scans through the pattern set, reduce the number of patterns examined during execution, or reduce the size of data structures used in the algorithm's operations.

3.4.1 Partitional Algorithms

A partitional clustering algorithm obtains a single partition of the data instead of a clustering structure, such as the dendrogram produced by a hierarchical technique. Partitional methods have advantages in applications involving large data sets for which the construction of a dendrogram is computationally prohibitive. A problem accompanying the use of a partitional algorithm is the choice of the number of desired output clusters. A seminal paper [8] provides guidance on this key design decision. The partitional techniques usually produce clusters by optimizing a criterion function defined either locally (on a subset of the patterns) or globally (defined over all of the patterns). Combinatorial search of the set of possible labelings for an optimum value of a criterion is clearly computationally prohibitive. In practice, therefore, the algorithm is typically run multiple times with different starting states, and the best configuration obtained from all of the runs is used as the output clustering.

3.4.1.1 Squared Error Algorithms

The most intuitive and frequently used criterion function in partitional clustering techniques is the squared error criterion, which tends to work well with isolated and compact clusters. The squared error for a clustering L of a pattern set X (containing K clusters) is

$$E^2(X, L) = \sum_{j=1}^K \sum_{i=1}^{n_j} \|x_i^{(j)} - c_j\|^2,$$

where $x_i^{(j)}$ is the i^{th} pattern belonging to the j^{th} cluster and c_j is the centroid of the j^{th} cluster.

The k -means is the simplest and most commonly used algorithm employing a squared error criterion [9]. It starts with a random initial partition and keeps reassigning

the patterns to clusters based on the similarity between the pattern and the cluster centers until a convergence criterion is met (e.g., there is no reassignment of any pattern from one cluster to another, or the squared error ceases to decrease significantly after some number of iterations). The k -means algorithm is popular because it is easy to implement, and its time complexity is $O(n)$, where n is the number of patterns. A major problem with this algorithm is that it is sensitive to the selection of the initial partition and may converge to a local minimum of the criterion function value if the initial partition is not properly chosen.

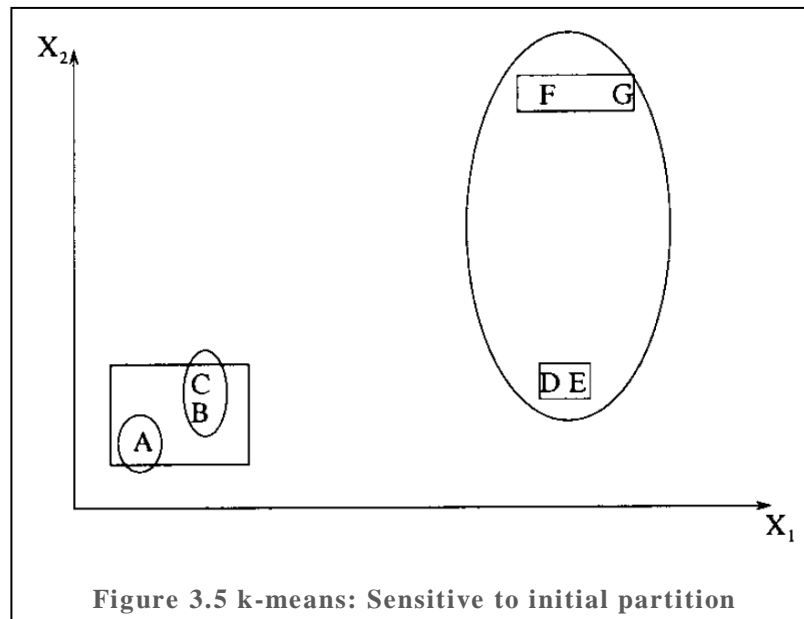


Figure 3.5 shows seven two-dimensional patterns. If we start with patterns A, B, and C as the initial means around which the three clusters are built, then we end up with the partition $\{\{A\}, \{B, C\}, \{D, E, F, G\}\}$ shown by ellipses. The squared error criterion value is much larger for this partition than for the best partition $\{\{A, B, C\}, \{D, E\}, \{F, G\}\}$ shown by rectangles, which yields the global minimum value of the squared error criterion function for a clustering containing three clusters. The correct three-cluster solution is obtained by choosing, for example, A, D, and F as the initial cluster means.

SQUARED ERROR CLUSTERING METHOD

- (1) Select an initial partition of the patterns with a fixed number of clusters and cluster centers.
- (2) Assign each pattern to its closest cluster center and compute the new cluster centers as the centroids of the clusters. Repeat this step until convergence is achieved, i.e., until the cluster membership is stable.

- (3) Merge and split clusters based on some heuristic information, optionally repeating step 2.

k-MEANS CLUSTERING ALGORITHM

- (1) Choose k cluster centers to coincide with k randomly-chosen patterns or k randomly defined points inside the hypervolume containing the pattern set.
- (2) Assign each pattern to the closest cluster center.
- (3) Recompute the cluster centers using the current cluster memberships.
- (4) If a convergence criterion is not met, go to step 2. Typical convergence criteria are: no (or minimal) reassignment of patterns to new cluster centers, or minimal decrease in squared error.

Several variants [4] of the k -means algorithm have been reported in the literature. Some of them attempt to select a good initial partition so that the algorithm is more likely to find the global minimum value.

Another variation is to permit splitting and merging of the resulting clusters. Typically, a cluster is split when its variance is above a pre-specified threshold, and two clusters are merged when the distance between their centroids is below another pre-specified threshold. Using this variant, it is possible to obtain the optimal partition starting from any arbitrary initial partition, provided proper threshold values are specified. The well-known ISODATA [10] algorithm employs this technique of merging and splitting clusters. If ISODATA is given the “ellipse” partitioning shown in Figure 3.5 as an initial partitioning, it will produce the optimal three-cluster partitioning. ISODATA will first merge the clusters {A} and {B,C} into one cluster because the distance between their centroids is small and then split the cluster {D,E,F,G}, which has a large variance, into two clusters {D,E} and {F,G}.

Another variation of the k -means algorithm involves selecting a different criterion function altogether. The *dynamic clustering* algorithm (which permits representations other than the centroid for each cluster) was proposed in [5], and [11] and describes a dynamic clustering approach obtained by formulating the clustering problem in the framework of maximum-likelihood estimation. The regularized Mahalanobis distance was used in [12] to obtain hyper-ellipsoidal clusters.

3.5 k-Medoids Clustering

In [2] a new algorithm for K-medoids clustering is proposed, which runs like the K-means algorithm and tests several methods for selecting initial medoids. The proposed algorithm calculates the distance matrix once and uses it for finding new medoids at every iterative step.

Suppose that n objects having p variables each should be grouped into k ($k < n$) clusters, where k is assumed to be given. Let us define j^{th} variable of object i as X_{ij} ($i = 1, \dots, n; j = 1, \dots, p$). The Euclidean distance is used as a dissimilarity measure although other measures can be adopted. The Euclidean distance between object i and object j is given by

$$d_{ij} = \sqrt{\sum_{a=1}^p (X_{ia} - X_{ja})^2}, i = 1, \dots, n; j = 1, \dots, n$$

The proposed algorithm is composed of the following three steps.

Step 1: (Select initial medoids)

1-1. Calculate the distance between every pair of all objects based on the chosen dissimilarity measure (Euclidean distance in this case).

1-2. Calculate v_j for object j by:

$$v_j = \frac{\sum_{i=1}^n d_{ij}}{\sum_{l=1}^n d_{il}}, j = 1, \dots, n$$

1-3. Sort v_j 's in ascending order. Select k objects having the first k smallest values as initial medoids.

1-4. Obtain the initial cluster result by assigning each object to the nearest medoid.

1-5. Calculate the sum of distances from all objects to their medoids.

Step 2: (Update medoids)

Find a new medoid of each cluster, which is the object minimizing the total distance to other objects in its cluster. Update the current medoid in each cluster by replacing with the new medoid.

Step 3: (Assign objects to medoids)

3-1. Assign each object to the nearest medoid and obtain the cluster result.

3-2. Calculate the sum of distance from all objects to their medoids. If the sum is equal to the previous one, then stop. Otherwise, go back to the Step 2.

The above algorithm is a local heuristic that runs just like K-means clustering when updating the medoids. In Step 1, a method of choosing the initial medoids is proposed. This method tends to select k most middle objects as initial medoids. The performance of the algorithm may vary according to the method of selecting the initial medoids.

3.6 Summary

In this chapter, data clustering is introduced. After discussing preliminaries, some well known techniques are provided. Finally, a k-medoid clustering algorithm is discussed in detail. In our proposed work, the algorithm in section 3.5 is used as a basis for our application. Several modifications, especially for the choice of initial cluster centers were added in our work discussed later.

3.7 References

- [1] Jain, Anil K., M. Narasimha Murty, and Patrick J. Flynn. "Data clustering: a review." *ACM computing surveys (CSUR)* 31.3 (1999): 264-323.
- [2] Park, Hae-Sang, and Chi-Hyuck Jun. "A simple and fast algorithm for K-medoids clustering." *Expert systems with applications* 36.2 (2009): 3336-3341.
- [3] Jain, A. K. and Dubes, R. C. "Algorithms for Clustering Data," Prentice-Hall advanced reference series. Prentice-Hall, Inc., Upper Saddle River, NJ. 1988,
- [4] Anderberg, Michael R. *Cluster analysis for applications: probability and mathematical statistics: a series of monographs and textbooks*. Vol. 19. Academic press, 2014.
- [5] Diday, Edwin, and J. C. Simon. "Clustering analysis." *Digital pattern recognition*. Springer, Berlin, Heidelberg, 1976. 47-94..
- [6] Michalski, Ryszard S., and Robert E. Stepp. "Automated construction of classifications: Conceptual clustering versus numerical taxonomy." *IEEE Transactions on pattern analysis and machine intelligence* 4 (1983): 396-410.

- [7] Salton, Gerard. "Developments in automatic text retrieval." *science* 253.5023 (1991): 974-980.
- [8] Dubes, Richard C. "How many clusters are best?-an experiment." *Pattern Recognition* 20.6 (1987): 645-663.
- [9] MacQueen, James. "Some methods for classification and analysis of multivariate observations." *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. No. 14. 1967.
- [10] Ball, Geoffrey H., and David J. Hall. *ISODATA, a novel method of data analysis and pattern classification*. Stanford research inst Menlo Park CA, 1965.
- [11] Diday, Edwin. "The dynamic clusters method in nonhierarchical clustering." *International Journal of Computer & Information Sciences* 2.1 (1973): 61-88.
- [12] Mao, Jianchang, and Anil K. Jain. "A self-organizing network for hyperellipsoidal clustering (HEC)." *Ieee transactions on neural networks* 7.1 (1996): 16-29.

4

Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning is considered as one of three machine learning paradigms, alongside supervised learning and unsupervised learning.

4.1 Learning

The basic components of *learning* include acquiring new, modifying or reinforcing existing knowledge, behaviors, skills, values or preferences and at times may include synthesizing different types of information. Humans, animals and some machines have the ability to learn. Progress with the passage of time tends to follow learning curves. Learning isn't a compulsion; it is contextual. It occurs gradually, slowly builds upon and gets shaped by previous knowledge. In that light, learning can be expressed as a process, instead of a collection of factual and procedural knowledge. Learning brings upon changes in an organism and these produced changes are relatively permanent [6]. According to 95% people, learning about new things is a confidence booster [7]. In the last 100 years, we have moved from the Industrial Age to the Knowledge Age, passing through the Information age. The next century should see the ability to obtain, assimilate and apply the right knowledge effectively. Past qualifications would no longer be the sole criteria for judging someone, it would probably be replaced by our capacity to learn and adapt to the future.

4.2 Machine Learning

Machine learning is a scientific discipline that explores the construction and study of algorithms that can learn from data [1]. Such algorithms generally work by constructing a model from example inputs and using them to make forecasts and decision making, [2] instead of abiding by strict static program instructions. Machine learning is quite similar to and often clashes with computational statistics; another discipline that specializes in prediction making.

In 1959, Arthur Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed" [3]. Tom M. Mitchell provided a widely quoted, more formal definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E " [4].

Machine learning enables the computers to learn without being explicitly programmed. It is a type of Artificial Intelligence (AI). It mainly aims at developing those computer programs which can teach themselves to grow and change as and when exposed to new data. Machine learning targets at computer algorithms for learning to carry out all activities. For instance, one may be interested in learning to complete a task, or to make accurate predictions, or to behave intelligently. The learning that is carried out is mostly based on observations or data, i.e. examples, direct experiences, or instructions. So, machine learning mainly targets at learning to do things in a better manner in the future compared to how they were done in the past. The machine learning mainly emphasizes on automatic methods, i.e. the basic goal is to construct learning algorithms that would do learning automatically without human intervention or assistance. The machine learning paradigm can be viewed as programming by example. Instead of programming the computer to solve the task directly, via machine learning, we can implement ways by which the computer would come up with its own program based on examples provided by the user. It is next to impossible to be able to build an intelligence system that would enable facilities such as language or vision that generally are associated with intelligence, without using learning to get there. Solving these tasks would be really troublesome without it. Considering learning as the core of intelligence, we can't consider a system as intelligent if it is not capable of learning. Machine learning mainly constitutes of AI, along with which it also comprises of other fields, especially statistics, mathematics, physics, theoretical computer science and more.

4.2.1 Examples of Machine Learning Problems

There are many examples of machine learning problems. Here are few examples:

- Optical Character Recognition (OCR): Categorize images of handwritten characters by the letters represented.
- Face Detection: Find faces in images or indicate if a face is present.
- Spam Filtering: Identify email messages as spam or non-spam.
- Topic Sorting: Categorize news articles as to whether they are about politics, sports, entertainment, etc.

- **Speech Interpretation:** Within the context of a limited domain, determine the meaning of something spoken by a speaker to the extent that it can be classified into one of a fixed set of categories.
- **Medical Diagnosis:** Diagnose a patient as a sufferer or non-sufferer of some disease.
- **Customer Segmentation:** Predict which customers will respond to a particular promotion or not.
- **Fraud Detection:** Identify credit card transactions which may be fraudulent in nature.

There are many learning problems, but if we could make the goal to behave intelligently or to make intelligent decisions, it would be a more qualified learning scenario. For instance, teaching a robot to navigate through its environment without colliding with anything.

4.3 Types of Machine Learning

Machine learning tasks are classified into three broad categories, depending upon its nature of learning signal or feedback available to a learning system. These are [5]:

- *Supervised learning:* The computer is presented with example inputs and their desired outputs, given by an instructor, and the goal is to practically implement the inputs to derive the desired outputs.
- *Unsupervised learning:* no labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself.
- *Reinforcement learning:* the computer program interacts with dynamic environment in which it performs the given goal, without an instructor explicitly telling it whether it has come close to its goal or not. For example learning to play a game by playing against an opponent [2].

In between supervised and unsupervised learning there is semi-supervised learning, where the instructor gives a partial training signal, for example, training set with some of the target outputs missing.

4.4 Reinforcement Learning

Reinforcement Learning (RL) mainly aims at how an agent ought to take actions in an environment to maximize notions of long-term reward. RL algorithms strive to find policies that maps states of the world to the actions which the agent ought to take in those states. RL differs from the supervised learning in which correct input-output pairs are never presented, nor sub-optimal actions explicitly corrected.

RL is an area of machine learning which is inspired by behaviorist psychology and is concerned with reactions of software agents in an environment to maximize some notion of cumulative reward. The problem, due to its applicability in various departments, is studied under many other fields such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics, and genetic algorithms. Approximate dynamic programming is the field in which RL methods are studied under operation research and control literature. How equilibrium would arise under bounded rationality is the basic point of target of the RL under game theory.

In machine learning, the environment is typically formulated as a Markov decision process (MDP) [17] as dynamic programming techniques is used by most of the RL algorithms under this context. The main difference between the classical techniques and RL algorithm is that the latter works without the knowledge of MDP and they target large MDPs where exact methods become baseless. The main focus of RL is on on-line performance, which mainly comprises finding a balance between exploration and exploitation. The multi-armed bandit problem [18] and infinite MDPs are the main sources of studying the exploration vs. exploitation trade off in RL.

4.4.1 Definition

RL is a branch of Machine learning henceforth being a major component of AI as well. Machines and software agents are permitted to automatically determine the ideal behavior within a specific context which helps in maximizing its performance. Reinforcement signal is the simple reward feedback which the agent uses to learn its behavior. This problem

is faced by most algorithms. RL can be defined by problems which are specific in nature and the solutions to these problems are classified as RL algorithms. When faced with a problem, the agent should be able to select the best action based on the position of the agent.

4.4.2 Motivation

Keeping the feedback from the environment as its base, RL permits the machine or software to learn its behavior. This behavior can be adapted, gradually with the passage of time or it can be learnt all at once. Some RL algorithms can converge to the global optimum if the problem is modeled with care; this is the ideal behavior that maximizes the reward. The need for a human expert in the domain of application gets curtailed by this learning scheme. Lesser time will be spent for finding a solution, as there would be no need for complex rules with Expert Systems, and the only requirement would be someone familiar with RL.

4.4.3 Methodology

There are many different solutions to the problem of which the most popular one is allowing the software agent to select an action that could maximize the reward not only in the immediate future but in the long-term too. Such algorithms are known to have infinite horizon. Learning to estimate the value of a particular state is the way to do it practically. Gradually with the passage of time, this estimate is adjusted by propagating a part of the next state's reward. Trying all the states and actions are tried sufficient number of times, the optimal policy can be defined; the action which maximizes the value of the next state is selected.

4.4.4 Limitations

RL research still faces many challenges. The complexity of the problem may be very high as it is often too memory expensive to store values of each state. The main target of research is to minimize the impact of consequences of introducing imperfect value estimation

on the quality of the solution. Problems generally are very modular; to avoid learning everything again modularity could be introduced. Although hierarchical approaches are common place for this, it is very challenging to get it done automatically.

4.4.5 Applications

Owing to the generalness of the problem specification, the possible applications of RL are abundant. A great amount of problems in AI can be fundamentally mapped to a decision process. Very effortlessly, this theory can be put to use in many different domain specific problems; hence it has a distinct advantage. Some of its practical usages include controlling robotic arms to find the most efficient motor combination, to robot navigation where collision avoidance behavior can be learnt by negative feedback from bumping into obstacles. Being traditionally defined as a sequence of decisions, logic games are best suited to RL: games such as poker, backgammon, othello, and chess have been almost tackled successfully.

4.5 Fundamentals of Reinforcement Learning

Reinforcement Learning mainly aims at maximizing the rewards by getting an agent to act in the world. For example, consider teaching a monkey a new trick: you can't directly dictate terms to it, you can instead reward or punish it if it does the right or wrong thing respectively. The credit assignment problem mainly deals with the monkey figuring out what it did that made it get the reward or punishment. Similar methods can be used to train computers to do many tasks, such as playing backgammon or chess, scheduling jobs, and controlling robot limbs.

To understand the framework of RL, we should introduce the following terminologies:

- a) **Autonomous Agent:** The subject which helps learn the environment is called Autonomous agent [8].
- b) **Environment:** The place in which the agent executes its learning procedure is called environment.

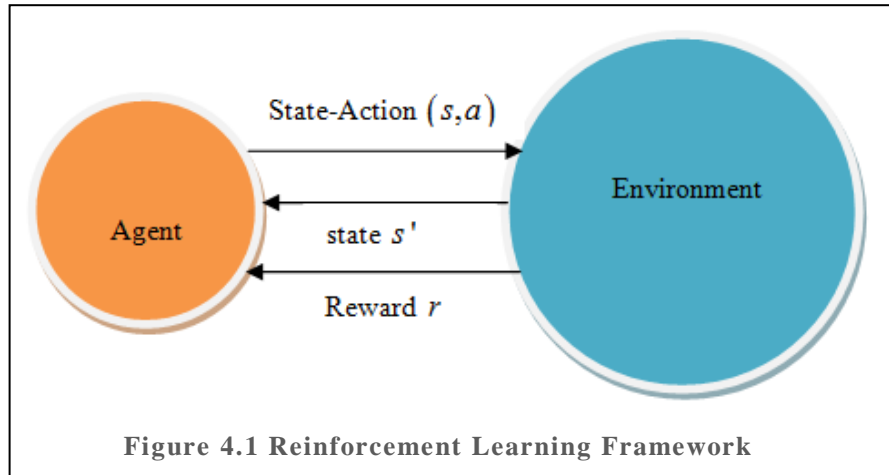
- c) **State:** The components that make up the environment are known as state. An agent can reside in a state. We may assume that the agent can't reside in between two states. The agent learning the environment in the perspective of its goal position is equally important.
- d) **Goal State:** Goal State is a pre-dominant part of the environment. This may be defined as the target zone of the agent.
- e) **Action:** It mainly constitutes the action space of the agent viz. the kind of actions it can take on its current state.
- f) **Reward or Penalty:** The feedback given to the agent by the environment after executing an action is either reward or penalty. This is a numeric value which signifies the performance of the agent while executing that action. This reward function is decided by an external expert. Generally, the reward received by the agent depends on its position with respect to the goal state.

The basic reinforcement learning consists of:

1. a set of discrete states S ;
2. action set A at each state S ;
3. rules that determine the scalar immediate reward $r(S, A)$ after a transition;
4. rules that describe what the agent observes, i.e., next state S' ;

The rules are stochastic in nature. The observation mainly revolves around the scalar immediate reward associated with the last transition.

The main purpose of reinforcement learning agent is to interact with its environment in discrete time steps. At each time t , the agent receives an observation o_t , which includes the reward r_t . It then selects an action a_t from the various actions available, which is then sent to the environment. The environment moves to a new state s_{t+1} and the reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}) is procured. Collecting maximum reward is the main target of a RL agent. Selecting a particular action as a function of the history and to randomize its action selection is up to the agent. We can formalize the RL problem as shown in Fig. 4.1:



Comparing the performance of an agent to another agent which acts optimally leads to a difference in performance which gives rise to the notion of regret. In order to obtain a optimum point, the agent should plan for long term benefits for e.g. to maximize future income I should be going to school now, even though an immediate monetary reward associated with this might be negative.

In RL tasks the objective is defined by the reward. A poorly chosen reward function may lead to force the learning system not to converge or to converge to a policy that does not achieve the desired task. The degree of delay in the reward signal classifies RL problems into three classes:

- a. Immediate reward;
- b. Delayed reward;
- c. Pure-delayed reward;

Immediate reward problems rewards immediately and completely describes the value of taking an action in a particular state. Card game snap is an example of an immediate reward problem. In this when a player notices that two cards of the same value have been laid down in succession the player shouts ‘snap’ and slaps the pile of cards with their hand. While deciding to snap or not, the current state of the cards is all that matters; Future planning is not required. Therefore, these problems should be solved with the discount factor set to zero. Immediate reward problems occur rarely as affecting the environment is a time taking process.

Delayed reward problems constitute a wider range of tasks. This class of problems rewards at every step; it doesn't evaluate the action in a particular state completely. The best example of this is the stock market where some actions yield immediate monetary rewards or losses, but decisions are taken keeping the future in mind. Another example can be the robotics tasks. The rewards must be greater than zero as it has to propagate back through time.

In *pure-delayed reward* problems the reward for every step of the trial except possibly the last step is the same where a terminal reward is given that indicates success or failure. Playing backgammon is an example of this [19]. In backgammon, success or failure is decided at the end of the game. Rewards are given for achieving sub-goals within the game. A pure-delayed reward makes sure that the learning system will aim at achieving the main goal instead of the sub-goals. Pure-delayed reward problems aren't a realistic model of the real world even though they are the most studied class of RL problems: in real world tasks there is usually some partial evaluation, or progress estimation, available before the end of the trial [20]. Treating all tasks as pure-delayed reward problems makes learning impractically slow. The quality and timeliness of solutions from by RL systems are mostly affected by the choice of reward function and state representation.

Thus, RL is more compatible to situations where long-term versus short-term reward tradeoff exists. It has been applied successfully to various problems including robot control [25], elevator scheduling [21], backgammon [22] and checkers [16].

RL gains power from mainly two components which are the use of samples to optimize performance and the use of function approximation to deal with large environments. Due to these components, RL can be used in large environments in any of the following situations:

- A model of the environment is known, but an analytic solution is not available;
- Only a simulation model of the environment is given (the subject of simulation based optimization);
- The only way to collect information about the environment is by interacting with it.

The initial two situations could be considered planning problems since some form of the model is available, while the last one could be considered as a genuine RL problem.

4.5.1 Exploration vs Exploitation

The basic requirement of RL problem is clever exploration mechanisms. Selecting random actions without referring an estimated probability distribution is most likely to lead to very poor performance. Due to the lack of algorithms that would probably scale well with the number of states or scale to problems with infinite state spaces, practically people use simple exploration methods. One such method is ϵ -greedy, in which the agent chooses the action that according to it has the best long-term effect taking probability $1-\epsilon$, and it chooses an action uniformly at random. Here, $0 < \epsilon < 1$ is a tuning parameter, which is sometimes changed, either according to a fixed schedule disabling the agent to explore more with passage of time or adaptively based on some heuristics [23].

The major problem arising by using RL is the tradeoff between exploration and exploitation. If an agent has procured a reward by trying a certain action in the past, then repeating the same would yield a reward again. In doing so, the agent is exploiting what is known to receive a reward. Exploring can turn out to be a profitable tactic as trying other possibilities may produce an even better reward. To make the RL agent learn successfully there has to be a balance between exploration and exploitation. The best way to strike a balance is to try various actions while progressively favoring those which produce the most reward.

4.5.2 On-Policy and Off-Policy Learning

On-Policy methods learn the value of the policy that is used to make decisions. The results derived by executing actions determined by policies are used to update value functions. These policies are generally soft and non-deterministic in nature [32]. Here, soft indicates that it ensures that there is always an element of exploration to the policy. The policy is lenient enough not to choose the action that derives most reward. Three common policies are used, ϵ -soft, ϵ -greedy and soft-max.

Off-Policy methods can learn different policies for behavior and estimation. There is too much exploration going on as the behavior policy is usually soft natured. Using hypothetical actions, off-policy algorithms can update the estimated value functions, those

which have not actually been tried which is contradictory to on-policy methods where update value functions are based strictly on experience. Off-policy algorithms have the power to separate exploration from control whereas on-policy algorithms can't. So, an agent trained in off-policy methods can learn tactics that it did not exhibit during the learning phase.

4.5.3 Action Selection Policies

There are three common policies used for action selection [32]. The main target of these policies is to balance the tradeoff between exploitation and exploration without exploiting what has been learnt so far.

In ϵ -greedy, the action with the highest reward is chosen viz. is called the greediest action. In small time intervals, an action is selected randomly, with a small probability ϵ . This action is independent of state-action value estimates and is selected uniformly. This method makes sure that if sufficient trials are done, each action will be tried infinite number of times, resulting in the discovery of optimal actions. ϵ -soft is very similar to ϵ -greedy. The action with the best reward gets selected having probability $1-\epsilon$ and at other times a random action gets selected uniformly.

The major drawback of ϵ -greedy and ϵ -soft is that they select random actions uniformly. The probability of the worst possible action being selected is equal to that of the second best. In *soft-max* this problem doesn't arise as it is healed by assigning a rank or weight to each of the actions, according to their action-value estimate. A random action is selected with regards to the weight associated with each action, making the worst option unavailable for selection. Hence it is the best alternative in cases where the worst actions are extremely unfavorable.

4.6 Q-Learning Algorithm for single agent systems

Knowing all possible states and action set of an agent is the most basic form of Q learning algorithm [9]. When an action is executed at a state, the position of the next state is also known, the state transition probability is one. This criterion forms the Deterministic Q-Learning Algorithm.

4.6.1 Deterministic Q-Learning Algorithms

Considering the world map containing n states from s_1 to s_n , and on each state the agent can execute a maximum of m actions, a_1 to a_m . At each state-action pair the agent receives an immediate reward from the environment, which is denoted by $r(s,a)$ where a is the action applied over state s . The agent selects its next state from the current state by using a policy. The policy should be able to maximize the cumulative reward of the agent. Such policy is called optimal policy. The cumulative reward for such optimal policy is denoted as,

$$P^*(s) = \arg \max [r(s,a) + \gamma V^*(\delta(s,a))] \quad (...1)$$

where γ is a discount factor and $\delta(s,a)$ is the next state due to action a at state s . So the optimal policy [10] can be easily found out if γ and δ are known to us.

The argument inside the $\arg \max$ is known as Q-evaluation function, which is given by,

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a)) \quad (...2)$$

In recursive manner Q-evaluation function can be given as,

$$V^*(s) = \max_{a'} Q(s,a') \quad (...3)$$

Thus, from (2) and (3) we derived

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a') \quad (...4)$$

where $s' = \delta(s,a)$

Determinist Q-learning Algorithm:

For each state $s \in \{s_1 \cdots s_n\}$

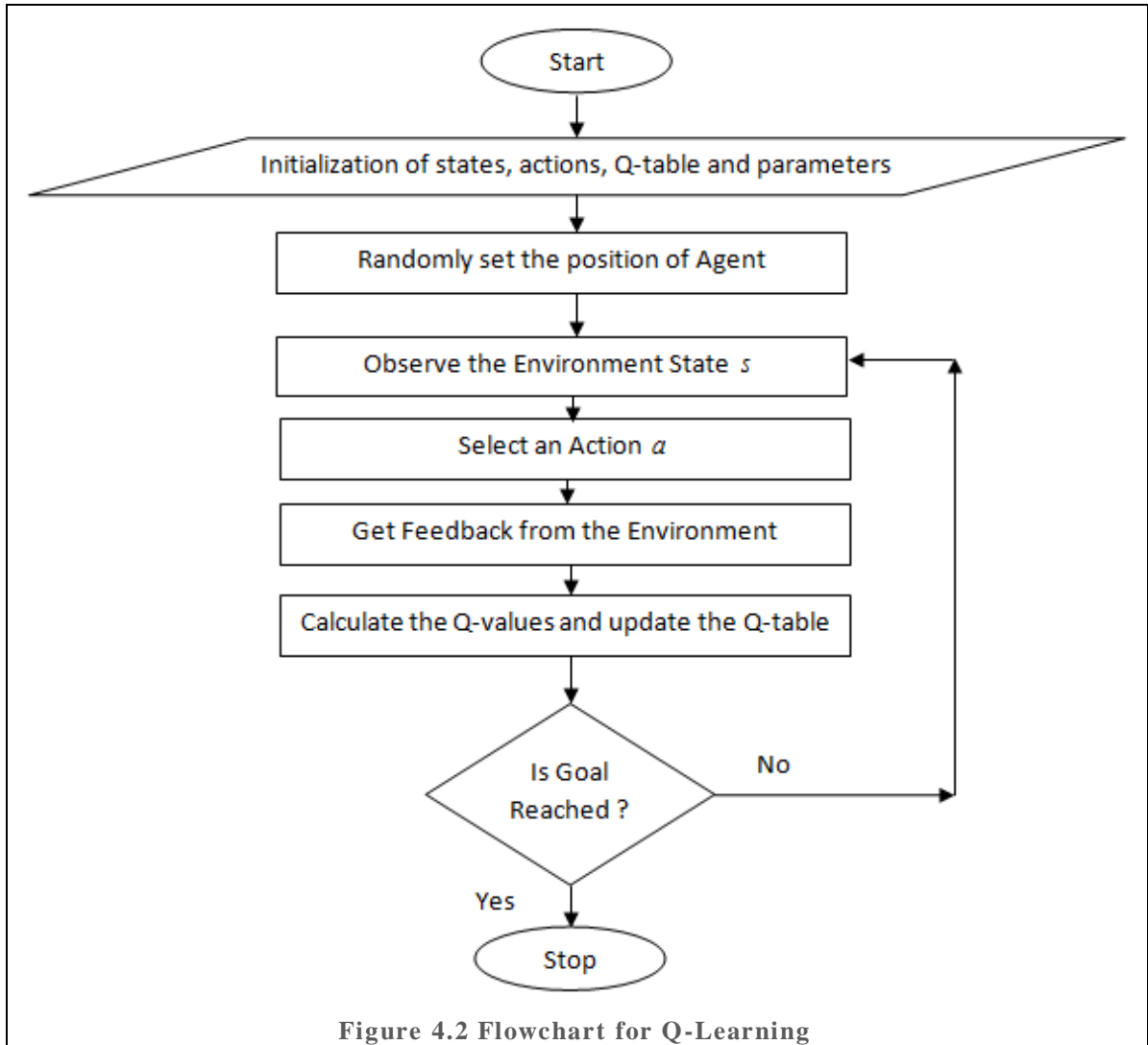
```

For each action  $a \in \{a_1 \cdots a_m\}$ 
    Initialize  $Q(s, a) \leftarrow 0$ ;
End For.
End For.
Observing the current state  $s$  ;
Repeat
    Select an action  $a \in \{a_1, a_2, a_3, \dots, a_m\}$  and execute it;
    Receive an immediate reward  $r(s, a)$  ;
    Observe new state  $s' = \delta(s, a)$  ;
    Update:
        
$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a') ;$$

         $s \leftarrow s' ;$ 
For Ever.

```

In the first step of the algorithm the state-action Q table is initialized to zero. In the next step, an action is executed on a certain state which results in a state transition and the agent receives a reward $r(s, a)$. The Q table is updated as using (4). The process is ideally continued forever, but in practice the iteration gets stopped when the difference of Q-values at a state between successive iteration becomes very small. At that time the Q-learning algorithm is said to be converged. It is important to note that the agent actually starts to learn when it reaches its goal for the first time. As the agent performs executes random action, some may argue that this sort of algorithm should not be used. But the beauty of this algorithm lies in the fact that due to this process of learning, the agent knows every possible path to reach its goal but the path it actually takes during the planning phase is the optimal path. The whole environment is learned from the goal state's complete perspective. As all possible solution to the problem is known by the agent, it can select its optimum path from any different starting state provided the goal state remains same. Flow chart for Q learning algorithm is shown in Fig. 4.2



4.6.2 Non-deterministic Reinforcement Learning

In deterministic learning, the immediate reward function and the state transition are fixed whereas in non-deterministic the parameters are probabilistic [12]. Whenever an action is executed in a state S the state transition forces the agent to move to any one of the possible states s_1, s_2, \dots, s_n . This transition from s to a state s' can be given by the probability function $P(S'|S, a)$. Immediate reward is received from the environment after the state transition, but in this case the next states are unknown hence the reward function $r(S, a)$ is probabilistic.

The probability distribution r and δ are functions of S , such a situation is non deterministic Markov decision process [13].

It is difficult to generalize the learning behavior of the agent due to the multiplicity in transitions from one state. Considering the expected value of the non deterministic outcome of the discounted cumulative reward received is a simple way to handle the problem by applying this policy:

$$V^p(S_t) = E \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \right] \quad (...5)$$

Here E stands for expectation operator.

The non-deterministic learning equation can be derived as the following:

$$Q(S, a) = E \left[r(S, a) + \gamma V^*(\delta(S, a), a) \right]$$

or
$$Q(S, a) = E \left[r(S, a) + \gamma E \left[V^*(\delta(S, a), a) \right] \right]$$

or
$$Q(S, a) = E \left[r(S, a) + \gamma \sum_{S'} P(S' | S, a) V^*(S') \right]$$

The above equation can be expressed recursively as follows:

$$Q(S, a) = E \left[r(S, a) \right] + \gamma \sum_{S'} P(S' | S, a) \max_{a'} Q(S', a) \quad (...6)$$

Due to the expectation operator E , $r(S, a)$ will change every time the transition $\langle S, a \rangle$ is repeated. It will change $Q(S', a)$ repeatedly even though the state-action table is initialized correctly. To overcome this problem, an alternative training rule can be used. The new rule works by using the decaying weighted average of the current Q-value and the revised estimate.

$$Q_n(S, a) \leftarrow (1 - \alpha_n) Q_{n-1}(S, a) + \alpha_n \left[r + \max_{a'} Q_{n-1}(S', a') \right] \quad (...7)$$

Here, $\alpha_n = \frac{1}{1 + \text{revisit}_n(s, a)}$ is an important parameter known as the learning rate [14].

$\text{revisit}_n(s, a)$ is the number of times the (s, a) pair is revisited. It is clearly seen that when n increases α_n decreases, which means the first part in the right hand side of the above expression gains more importance than the second term.

It means when $n \rightarrow \infty$.

$$Q_n'(s, a) = Q_{n-1}'(s, a) \quad (...8)$$

This parameter helps in the convergence of non-deterministic learning process.

4.6.3 Stochastic Reinforcement Learning

Under Stochastic reinforcement learning [15], the current state and the action executed over can't determine the next state. The next state is a random variable where the current state and action together gives the probability density of it. This notion of RL is quite realistic. Let us consider a simple example; suppose an agent is residing in a state and it has decided to execute a left action. The agent may not be able to go to the left state because of the rough terrain, or else some mechanical failure may be forcing it to reside in its own state. These kinds of uncertainties can't be implemented in deterministic learning. The deterministic transition function δ is now changed to $\bar{\delta}: s \times a \times s \rightarrow [0,1]$ where $\bar{\delta}$ is the probabilistic transition function and the state space is assumed to be countable. After action a_t is executed at state s_t , the probability that the next state, s_{t+1} , belongs to a region $s_{t+1} \subseteq S$ is:

$$P(s_{t+1} \in \{s_1, s_2, \dots, s_n\} | s_t, a_t) = \int \bar{\delta}(s_t, a_t, s') ds' \quad (...9)$$

After transition to a new state s_{t+1} a reward r_{t+1} is received this is given by,

$$r_{t+1} = \bar{\rho}(s, a, s') \quad (...10)$$

In countable state space equation reduces to,

$$P(s_{t+1} \in s' | s_t, a_t) = \bar{\delta}(s_t, a_t, s') \quad (...11)$$

For any s and a , $\bar{\delta}$ must satisfy the property $\sum_{s'} \bar{\delta}(s_t, a_t, s') = 1$. In the stochastic case, the Markov property should fulfill s and a fully determine the probability density of the next state.

4.7 Summary

In this chapter, we discuss about machine learning. Reinforcement learning, a branch of machine learning is mainly addressed and an overview of different types of Reinforcement learning is provided.

4.8 References

- [1] Kohavi, R., & Provost, F. (1998). Glossary of terms. *Machine Learning*, 30(2-3), 271-274.
- [2] Bishop, C. M. (2006). *Pattern recognition and machine learning* (Vol. 4, No. 4). New York: springer.
- [3] Simon, P. (2013). *Too Big to Ignore: The Business Case for Big Data*. John Wiley & Sons.
- [4] Mitchell, T. M. (1997). *Machine learning*. WCB.
- [5] Russell, S., Norvig, P., & Intelligence, A. (1995). *A modern approach*. Artificial Intelligence. Prentice-Hall, Englewood Cliffs, 25.
- [6] Schacter, D. L., Gilbert, D. T., & Wegner, D. M. (2009). *Introducing psychology*. Macmillan.
- [7] Beinart, S., & Smith, P. (1998). *National adult learning survey 1997*. Sudbury: DfEE.
- [8] Kordic, V. *Autonomous Agents*.
- [9] Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4), 279-292.
- [10] Schapire, R. (2003). *COS 511: Foundations of Machine Learning Markov Decision Process*.
- [11] Lenz, J. (2003). *Reinforcement Learning and the Temporal Difference Algorithm*.
- [12] Konar, A. (2006). *Computational intelligence: principles, techniques and applications*. Springer Science & Business Media.
- [13] Fard, M. M., & Pineau, J. (2009). MDPs with non-deterministic policies. In *Advances in neural information processing systems* (pp. 1065-1072).
- [14] Even-Dar, E., & Mansour, Y. (2004). Learning rates for Q-learning. *The Journal of Machine Learning Research*, 5, 1-25.

- [15] Busoniu, L., Babuska, R., De Schutter, B., & Ernst, D. (2010). Reinforcement learning and dynamic programming using function approximators (Vol. 39). CRC press.
- [16] Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning. MIT Press.
- [17] White, D. J. (1993). A survey of applications of Markov decision processes. Journal of the Operational Research Society, 1073-1096.
- [18] Lai, T. L. (1987). Adaptive treatment allocation and the multi-armed bandit problem. The Annals of Statistics, 1091-1114.
- [19] Tesauro, G. (1992). Practical issues in temporal difference learning (pp. 33-53). Springer US.
- [20] Mataric, M. J. (1994). Reward functions for accelerated learning. In Machine Learning: Proceedings of the Eleventh international conference (pp. 181-189).
- [21] Brand, M., & Nikovski, D. (2004, April). Optimal parking in group elevator control. In Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on (Vol. 1, pp. 1002-1008). IEEE.
- [22] Tesauro, G. (1995). Temporal difference learning and TD-Gammon. Communications of the ACM, 38(3), 58-68.
- [23] Tokic, M., & Palm, G. (2011). Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In KI 2011: Advances in Artificial Intelligence (pp. 335-346). Springer Berlin Heidelberg.
- [24] Deisenroth, M. P., Neumann, G., & Peters, J. (2013). A Survey on Policy Search for Robotics. Foundations and Trends in Robotics, 2(1-2), 1-142.
- [25] Das, P., Sadhu, A. K., Vyas, R. R., Konar, A., & Bhattacharyya, D. (2015, February). Arduino based multi-robot stick carrying by Artificial Bee Colony optimization algorithm. In Computer, Communication, Control and Information Technology (C3IT), 2015 Third International Conference on (pp. 1-6). IEEE

5

Non Monotonic Learning

Most logic systems employ classical logic with an incremental learning method. This chapter demonstrates drawbacks of such learning systems, and introduces Non-monotonic learning as an alternative. Non-monotonic learning is flexible to newer inputs which may or may not conform to existing knowledge and either modifies the general perception or introduces specialization according to the requirement of the application in practice.

5.1 Introduction

Generalization is not everything in learning. New experience can often require the specialization of over-general beliefs [1]. For example imagine that you believed that all birds fly. We might write this as

$$\textit{Flies}(x) \leftarrow \textit{Bird}(x)$$

If you were told that although an emu is a bird it cannot fly you would have to specialize to deal with this exception. In this chapter we discuss various ways in which this could be done. In the following we will assume that an incremental learning algorithm receives new examples one at a time, its belief set being revised after each example. The belief set is generalized when it does not cover a new example. On the other hand it is specialized when contradicted by a new example. Most work in machine learning is based on the related notions of generalization and specialization. However, most treatment of the topic of specialization has been within the context of non-incremental learning algorithms [4], [5], [6], and [10]. Clearly an incremental learning algorithm as described above changes the coverage of its beliefs non-monotonically. However, many of the developers of logic-based machine learning algorithms to date have avoided non-monotonic logic representations. This leads to various problems, as it is not possible in general to preserve correct information when incrementally specializing within a classical logic framework. This impasse can be easily avoided by learning algorithms which employ a non-monotonic knowledge representation.

Muggleton and Buntine [9] have described an algorithm called CIGOL which learns unrestricted first-order Horn clause theories on the basis of unit clause examples presented one at a time. CIGOL as described in [9] is not incremental in the sense of the description above. Although it generalizes from positive examples it does not specialize when it encounters negative examples which contradict its theory. In practice this has led to difficulties. As reported in [8], by over-generalizing CIGOL managed to outstrip the performances of both humans and propositional learning algorithms when incrementally learning a chess concept from randomly selected examples. However, having over-generalized and reached a performance level of around 90%, CIGOL was not able to produce 100% performance since it could not specialize the concept definition. In this chapter we lay the theoretical foundations for incremental specialization techniques.

5.2 Non-monotonic formalisms

One of the most common approaches to non-monotonic reasoning is based on the "Closed World Assumption" (CWA) inference rule. According to the CWA if a ground atom A is not a logical consequence of a theory then infer \bar{A} [3]. The CWA can be implemented in two different ways. First, by adding additional completion axioms to the theory and applying standard theorem proving techniques. This approach is exemplified by "predicate completion" [3] and "circumscription" [7]. Second, by employing "Negation by Failure" (NF) where a modified theorem prover infers \bar{A} whenever the attempt to prove a ground atom A finitely fails. The second approach is used within the logic programming language Prolog for which NF has been shown to be equivalent to "predicate completion" [3].

5.2.1 Closed World Specialization

As we stated in the previous section the logic programming language Prolog uses "negation by failure". In classical logic the ground literal \bar{A} is provable from theory T if and only if $T \vdash \bar{A}$. In Prolog the ground goal $notp(A)$ is provable from T if and only if A is an atom and $T \not\vdash A$. In this section we will represent clauses in the notation of Edinburgh Prolog [2]. For example the clause $P(x) \leftarrow Q(x) \wedge R(x)$ is written in Prolog as

$$p(X) : -q(X), r(X)$$

But note that the prolog clause:

$$p(X) : -q(X), notp(r(X))$$

states that $p(X)$ is provable if $q(X)$ is provable and $r(X)$ is not provable. The literal to the left of the ":-" is called "head" of the clause, while the set of literals to the right of the ":-" is called the body of the clause. Prolog proofs are carried out using SLDNF-resolution [3]. The following is an algorithm for our Closed World Specialization technique.

Closed World Specialization Algorithm

Input: Set of clauses T and ground atom A such that $T \vdash A$ and A is incorrect.

Let C be the clause in T which is resolved with \bar{A} in the SLDNF-resolution proof of $T \vdash A$.

Let θ be the substitution for variables in C produced by the SLDNF-resolution proof of $T \vdash A$.

If the body of C contains a literal $notp(B)$ then

Let $\bar{T} = T \cup \{B\theta\}$

Else

Let $\{V_1, \dots, V_n\}$ be the domain of θ

Let q be a predicate symbol not found in T

Let Hd be the head of C

Let Bd be the body of C

Let B be $q(V_1, \dots, V_n)$

Let $\bar{T} = T - \{C\} \cup \{Hd : \neg(Bd \cup notp(B))\} \cup \{B\theta\}$

Output: \bar{T}

Consider the following example where this algorithm is applied on the "bird" example discussed previously.

Example

Let T be the set of clauses $\{flies(X) : \neg bird(X), bird(eagle) : -, bird(emu) : -\}$ and the true ground atoms in M , the intended interpretation of T , be a superset of $\{flies(eagle), bird(eagle), bird(emu)\}$, where $A = flies(emu)$ is not true in M . T is incorrect with respect to M since the clause $C = (flies(X) : \neg bird(X))$ is incorrect with substitution $\theta = \{X / emu\}$. The body of C does not contain a literal $notp(B)$. $\{X\}$ is the domain of θ . Let q be flightless".

B is flightless(X) and \bar{T} is $\{(flies(X) : \neg bird(X), notp(flightless(X))), (bird(eagle) : -), (bird(emu) : -), (flightless(emu) : -)\}$.

Thus, $flies(sparrow)$ can be proved from $\bar{T} \cup \{bird(sparrow) : -\}$.

5.3 Summary

In this chapter a brief overview regarding the concept of Non-monotonic learning is provided. The principle and its requirement in current learning systems is also explored. The idea of incorporating new information for unlearning existing knowledge and learning new knowledge has been used in the novel work proposed in this thesis.

5.4 References

- [1] Bain, Michael, and Stephen Muggleton. *Non-monotonic learning*. Turing Institute, 1990.
- [2] W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, Berlin, 1981.
- [3] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1984.
- [4] R. Michalski and J. Larson. Selection of most representative training examples and incremental generation of vll hypotheses: the underlying methodology and the description of programs ESEL and AQ11. UIUCDCS-R 78-867, Computer Science Department, Univ. of Illinois at Urbana-Champaign, 1978.
- [5] Michalski, Ryszard S. "A theory and methodology of inductive learning." *Machine learning*. Springer, Berlin, Heidelberg, 1983. 83-134.
- [6] Mitchell, Tom M. "Generalization as search." *Artificial intelligence* 18.2 (1982): 203-226.
- [7] M.R.Genesereth and N.J. Nilsson. *Logical foundations of artificial intelligence*. Morgan-Kaufmann, San Mateo, CA, 1987.
- [8] Muggleton, Stephen, et al. "An experimental comparison of human and machine learning formalisms." *Proceedings of the sixth international workshop on Machine learning*. Morgan Kaufmann, 1989.
- [9] Muggleton, Stephen, and Wray Buntine. "Machine invention of first-order predicates by inverting resolution." *Machine Learning Proceedings 1988*. Morgan Kaufmann, 1988. 339-352.

[10] Quinlan, J. Ross. "Learning efficient classification procedures and their application to chess end games." *Machine learning*. Springer, Berlin, Heidelberg, 1983. 463-482.

6

Microsoft Kinect Sensor



Microsoft Kinect is a sensor system consisting of a RGB camera, a depth image camera and a microphone array. Originally developed for video gaming purpose, the system was developed by Microsoft in cooperation with PrimeSense.

6.1 Microphone array

The microphone array assembled in Kinect consists of four single microphones, which are arranged along the front side of the sensor. This microphone array can be used to detect the position of a sound source. Since the microphone array was not used during this thesis, it will not be described in more detail.

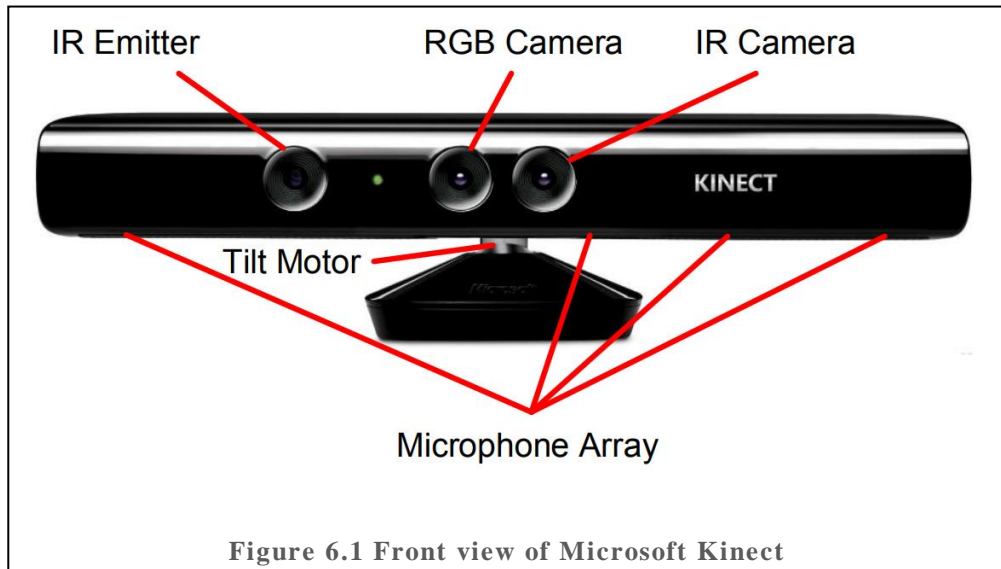


Figure 6.1 Front view of Microsoft Kinect

6.2 RGB Camera

The RGB camera is arranged central on Kinect's front side as shown in figure 6.1. The assembled camera sensor is a CMOS4 image sensor MT9M112 produced by Micron [3]. The whole camera system is completely integrated on one chip. It has a maximum resolution of 1.3 mega pixels (1280×1024) at a frame rate of 15 fps. At 640×512 the sensor reaches a frame rate of 30 fps. Each pixel has a size of $2.8 \mu\text{m} \times 2.8 \mu\text{m}$. The Kinect sensor provides only a maximum resolution of 1280×960 at a frame rate of 12 fps in RGB format with 24 bit color depth. Additionally, the RGB camera integrated in Kinect can be operated at a resolution of 640×480 with different color formats and frame rates (RGB with 24 bit color depth at 30 fps, YUV5 at 15 fps, and YUV RAW at 15 fps). According to the specification the RGB camera has a focal length of 2.9 mm and a field of view of 43° vertical and 57° horizontal.

Table 6.1 RGB Camera Specifications

Parameter	Value
Pixel size	2.8 μm \times 2.8 μm
Max. Resolution	1280 pixel \times 960 pixel
Max. sample rate	30 fps
FOV ⁶	ca. 43° \times 57°
focal length f	2.9 mm

6.3 Depth Image Camera

More interesting than the RGB camera is the depth image camera of Kinect. This camera supplies an image, where each pixel does not carry color information but depth information. The interesting thing of the depth image camera system used for Kinect is that it only uses one IR⁷ camera instead of a stereo camera system. The advantage of a stereo camera system is that it supplies both, color and depth information (the Kinect depth camera only supplies depth information). A disadvantage is the high computation effort, which is needed to find similarities in the two stereo images to get the depth information out of it. This high computation effort probably is the reason why Kinect does not use a stereo camera system. Kinect uses an IR camera, as already mentioned, in combination with an IR projector to retrieve depth information out of the scene. The IR camera is arranged right beside the RGB camera and the IR projector is placed in a distance of about 7 cm to the IR camera. The IR projector projects a pseudo random (but known) pattern onto the scene. This pattern is recorded by the IR camera. Since the camera records the pattern from a different angle of view, there will be a distortion of the recorded pattern based on the 3D position of the reflecting objects. E.g. if the pattern is reflected by an object, which is far away from the projector, it will be more stretched then for reflections on closer objects. Besides that, if the pattern is reflected on planes, which are not parallel to the projector’s plane, the pattern will be distorted. Since the projection pattern is known by Kinect, the depth information can be calculated directly out of the distortion by triangulation. Therefore no similarities in both images have to be found first as it is in a stereo camera system. Only the already known pattern has to be correlated with the recorded IR image. There exists a lot of publications about analysis of the Kinect depth sensor [2],[4],[5], hence in this thesis no further research on accuracy analysis was done.

All given information in the following sections rely on specifications given by Microsoft and already published works.

6.3.1 IR Pattern

In [4] the IR pattern is analyzed on its properties very well. The pseudo random pattern is generated by some kind of grating, which scatters the IR light source into a defined pattern. [4] shows that the whole pattern consists of one small pattern in the center, which then repeatedly is arranged eight times around it. Since the pattern has a pseudo random (white) structure, it has very appropriate correlation properties and thus each speckle in the pattern can be determined very well by correlation.

6.3.2 Specifications

The IR sensor used in the system is the MT9M001 from the company Aptina [1]. It has a resolution of 1280×1024 at a frame rate of 30 fps. The pixel data is supplied with a resolution of 10 bit. In the Kinect system, again, only $1280 \text{ pixel} \times 960 \text{ pixel}$ are used and a 2×2 pixel binning is performed. The pixel binning results in an effective pixel size of $10.4 \mu\text{m} \times 10.4 \mu\text{m}$. Kinect offers three different resolutions for the depth image (80×60 , 320×240 , and 640×480), which run all at a frame rate of 30 fps. At all 7 infrared resolutions the depth information is quantized by 12 bit (unsigned). But tests showed that the actual resolution at far distances is much lower. This comes especially from interpolation of the depth value of undefined pixels out of the values of its neighbourhood pixels. In the default mode the depth information ranges from about 0.8 m to 4 m and in the near mode from 0.5 m to 3 m. The focal length of the camera is given with 5.9 mm and thus the field of view is almost the same as that of the RGB camera.

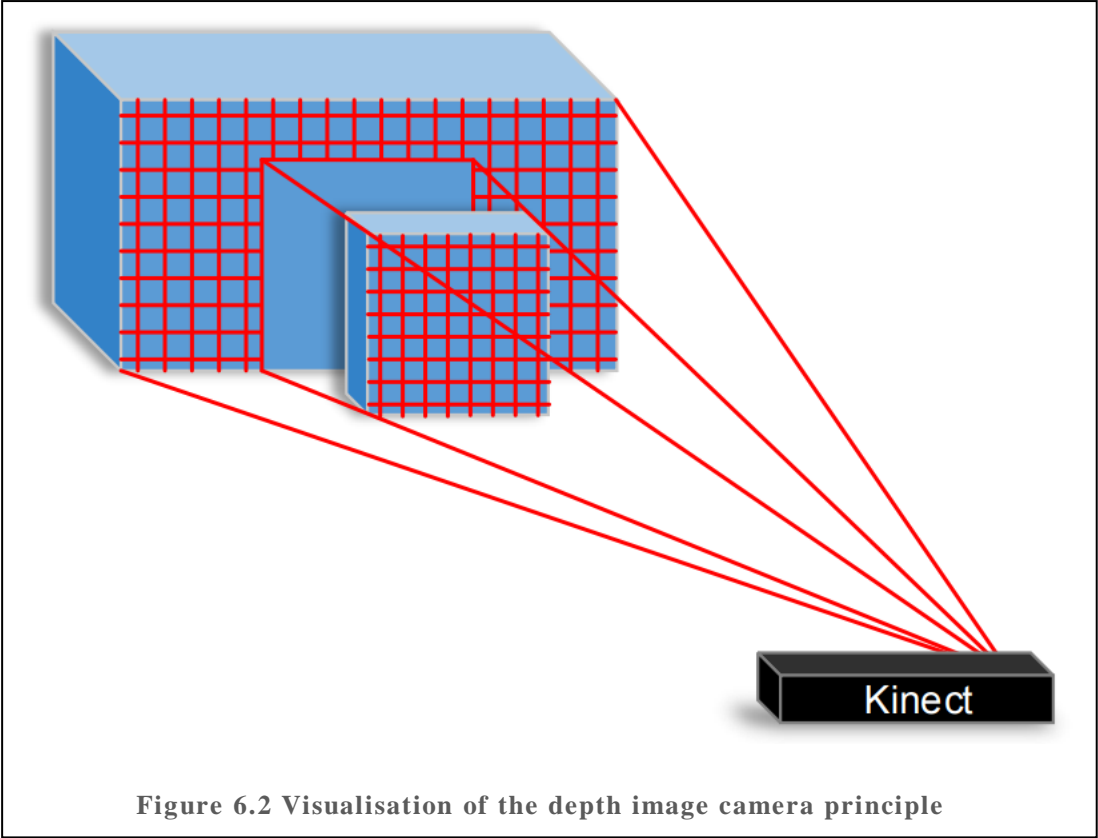
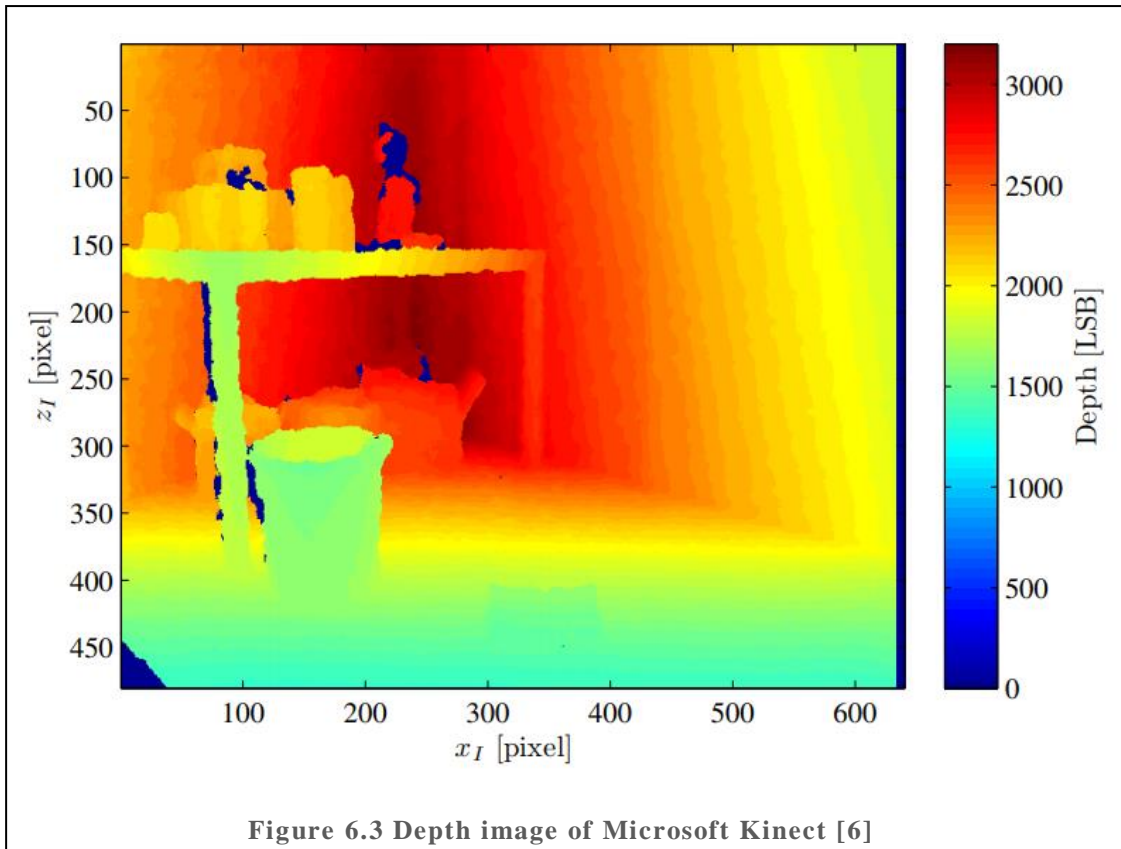


Table 6.2 Depth image camera specification

Parameter	Value
Pixel size	10.4 μm \times 10.4 μm
Max. Resolution	640 pixel \times 480 pixel
Max. sample rate	30 fps
FOV	ca. 43° \times 57°
focal length f	5.9 mm
Depth quantization	12 bit
Depth range (near mode)	ca. 0.5 m to 3 m
Depth range (far mode)	ca. 0.8 m to 4 m



6.3.3 Depth Image

As mentioned before, the depth image camera supplies depth information for each pixel with a resolution of 12 bit. Figure 6.3 from [6] shows a sample of such a depth image. In the image pixels with small depth values are blue and pixels with high depth values are red. The dark blue areas have got the depth value -1 , which means they do not carry depth information. The reason that these areas do not carry any depth information is the lack of IR reflection in the camera direction. This can either result when the surface is shadowed from the IR pattern by any other object or when not enough IR light is reflected, which is based on the orientation and texture of the surface. Glossy or matt black surfaces, for example, basically do not reflect any light into the camera direction and therefore are inappropriate for depth determination with Kinect. Points, which are in a too close distance to the Kinect sensor are also represented by the value -1 . Points, which are too far away from the sensor have got the maximum depth value ($2^{12} - 1$). Some experiments showed that the accuracy of the depth information decreases with increasing depth. While at about 0.5 m depth the accuracy is in a range of about ± 5 mm, it decreases to about ± 2.5 cm at 3 m depth. Since the density of the IR pattern also decreases with increasing depth,

the depth information is interpolated at high depth values. That causes further inaccuracy for high depth values and a jumping depth information of neighbourhood pixels.

6.4 Kinect SDK

Beside the official Microsoft Kinect SDK there exist some open source projects for Kinect software development, as already mentioned. For this thesis the official SDK was used since it offers a really good online support and supports all needed functions. Beside those needed functionality a lot more tools are offered. The current version of Kinect SDK offers libraries for C# and C++ programming. It offers a skeletal tracking mode, which returns 3D coordinates for each body joint, which was significantly used. The SDK supports speech recognition as well as face tracking and offers accelerometer data of the Kinect sensor. The development of Software using the Microsoft Kinect SDK is very simple. Basically all functions to control the Kinect and to read out the data from it are implemented in one class in the SDK library. This class is called `KinectSensor` and can be used in a C# as well as in C++ program. For all Kinect devices, which are connected to the computer, an object of the class `KinectSensor` is created automatically. Kinect itself as well as single functions, like the depth or RGB camera, can be activated or deactivated just by calling a member function of the `KinectSensor` class. For both, the depth camera as well as the RGB camera exist objects in `KinectSensor` by which the cameras can be controlled and images can be read out. These objects are of the class `ColorStream`, respectively `DepthStream` and are used for example to set the image resolution or format.

6.5 Skeletal Tracking using Kinect

Skeletal tracking was extensively done in our proposed work. While the details of the work is included in the later chapter, and the codes are provided in the accompanying disc, a basic open-source [7] skeletal tracking program is provided here, which could act as a starting point for people new to this particular application or Kinect in general.

The following is a fairly simple tutorial that shows how to get basic information about human bodies in the view of the Kinect. It shows how to extract the 3D positions of the body's joints, which can then be further processed to do things as simple as drawing a skeleton, to things as complex as gesture recognition.

Global Variables

As a global variable, we will keep an array of all the Joints in the last seen body.

```
// Body tracking variables
Vector4 skeletonPosition[NUI_SKELETON_POSITION_COUNT];
```

There are `NUI_SKELETON_POSITION_COUNT = 20` joints tracked by the Kinect. Each joint has one `Vector4` in the array describing its 3D location in camera coordinates. For example, to find the position of the right elbow, use `skeletonPosition[NUI_SKELETON_POSITION_ELBOW_RIGHT]`

Kinect Initialization

We add two new parts to our Kinect initialization. First, we request skeletal tracking data when we initialize the Kinect, and then we enable tracking.

```
bool initKinect(){
// ...
    sensor->NuiInitialize(
        NUI_INITIALIZE_FLAG_USES_DEPTH_AND_PLAYER_INDEX
    | NUI_INITIALIZE_FLAG_USES_COLOR
    | NUI_INITIALIZE_FLAG_USES_SKELETON);

    sensor->NuiSkeletonTrackingEnable(
        NULL,
0// NUI_SKELETON_TRACKING_FLAG_ENABLE_SEATED_SUPPORT for only upper body
    );
// ...
}
```

By default, the Kinect expects people to be standing approximately facing the sensor for best tracking. It can track seated people facing the sensor (e.g. on a couch) if you pass in the appropriate flag as the second argument to `NuiSkeletonTrackingEnable`

Getting joint data from the Kinect

Getting skeletal data is simpler than getting color or depth data - no locking or anything. We simply grab a `NUI_SKELETON_FRAME` from the sensor. Joint tracking can be very noisy, so to reduce the amount of jitter in joint positions over time, we also call a built in smoothing function.

```
void getSkeletalData(){
    NUI_SKELETON_FRAME skeletonFrame = {0};
    if(sensor->NuiSkeletonGetNextFrame(0, &skeletonFrame) >= 0){
        sensor->NuiTransformSmooth(&skeletonFrame, NULL);
// Process skeletal frame (see below)...
    }
}
```

The Kinect can track up to `NUI_SKELETON_COUNT` people simultaneously (in the SDK, `NUI_SKELETON_COUNT == 6`). The skeleton data structure, `NUI_SKELETON_DATA` can be accessed in the `SkeletonData` array field of the frame. Note that each skeleton may not necessarily refer to an actual person that the Kinect can see, and the first tracked body might not necessarily be the first array element. Thus, we need to check whether or not each of the elements of the array is a tracked body. To do this, we check the tracking state of the skeleton.

```
// Loop over all sensed skeletons
for(int z =0; z < NUI_SKELETON_COUNT;++z){
const NUI_SKELETON_DATA& skeleton = skeletonFrame.SkeletonData[z];
// Check the state of the skeleton
if(skeleton.eTrackingState == NUI_SKELETON_TRACKED){
// Get skeleton data (see below)...
}
}
```

Once we have a valid, tracked skeleton, then we can just copy the all the joint data into our array of joint positions. We also check the tracking state of each individual joint, since the Kinect may not be able to track all of the joints (such as if the user's arms are behind their back or otherwise occluded). We use the `w`-coordinate of the `Vector4` joint positions to keep track of whether or not it is a valid, tracked joint.

```
// For the first tracked skeleton
{
// Copy the joint positions into our array
for(int i =0; i < NUI_SKELETON_POSITION_COUNT;++i){
skeletonPosition[i]= skeleton.SkeletonPositions[i];
if(skeleton.eSkeletonPositionTrackingState[i]==
NUI_SKELETON_POSITION_NOT_TRACKED){
skeletonPosition[i].w =0;
}
}
return;// Only take the data for one skeleton
}
```

6.6 Summary

In this chapter the basic sensors of Kinect and its functionalities are discussed. After that, as an introduction to the proposed work, an example program is provided. Although the language, libraries used, and application is different from the work discussed in chapter 8, this should serve as a good introduction to programming for skeletal tracking.

6.7 References

- [1] Aptina Imaging, 3080 North 1st Street, San Jose, CA 95134, United States. MT9M001 Monochrome Image Sensor, 2011.
- [2] K. Khoshelham. Accuracy analysis of Kinect depth data. In ISPRS Workshop Laser Scanning, volume XXXVIII, pages 133–138, Aug. 2011.
- [3] Micron, 8000 S. Federal Way, P.O. Box 6, Boise, ID 83707-0006, United States. MT9M112 CMOS Image Sensor System-on-Chip, 2005.
- [4] J. Schares, L. Hoegner, and Stilla U. Geometrische Untersuchung zur Tiefengenaugigkeit des KinectSensorsystems. In Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation (DGPF) e.V., volume 21, pages 372–380, 2012.
- [5] J. Smisek, M. Jancosek, and T. Pajdla. 3d with Kinect. In Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on, pages 1154–1160, Nov. 2011.
- [6] Zeller, Niclas (2013) Obstacle detection using Microsoft Kinect (Masters’ thesis). Retrieved from: https://digital.library.ryerson.ca/islandora/object/RULA%3A2998/datastream/OBJ/download/Obstacle_detection_using_Microsoft_Kinect.pdf
- [7] https://github.com/kyzyx/Tutorials/tree/master/KinectSDK/4_SkeletalTracking

7

Jaco Robot Arm



Jaco Robot arm manufactured by Kinova is an ultimate solution for rehabilitative robotics. It is a 6 axis robotic manipulator with a three fingered hand. The arm had 6 degrees of freedom in total with maximum reach of 90 cm radius sphere and maximum speed of 30cm/sec. It employs three sensors force, position and acceleration. This arm is suitable for a person with disability of upper arm and can be attached to a wheel chair. This chapter provides a through description of engineering behind it and also provides a description of programing of this arm by manufacturer provided Application Programming Interface (API).

7.1 Introduction

Jaco is a light weight robot arm made by KINOVA and it is mainly employed for rehabilitation purpose but it has versatile usage and especially in robotics research it has got significant importance. Following sections will describe the specification and necessary details for operating the JACO arm.

7.1.1 Specification

Table 7.1 Jaco Arm Specifications

ENVIRONMENT	GENERAL
	<ul style="list-style-type: none">• Can be operated within the temperature Range 0-30° C.• Can withstand little rainfall for limited time (IPX2 rating).• Must be used under normal atmospheric pressure.• Can be stored within temperature range 0-50° C.• Can with stand maximum relative humidity 55%.
	INPUT POWER
	<ul style="list-style-type: none">• Input voltage is within 18 V to 29 V DC.• Input Current is 2A in normal use and 10 A maximum.• It is powered by external power supply, it does not have any internal power source.
	OUTPUT POWER
ELECTRICAL	<ul style="list-style-type: none">• Output voltage is 24V+/- 20% DC• Output current is 1.5A (continuous) and 5A maximum.
	POWER SUPPLY
	<ul style="list-style-type: none">• Input power supply is 100-240 V AC, 50 Hz, 2 A.• Output Power 24V dc, 5 A.
MECHANICAL	GENERAL
	<ul style="list-style-type: none">• Total weight of the arm is 5.6 kg.• Maximum Payload is 1.5 kg at midrange and 1kg at full range.• Maximum reach is 90 cm.

N I C A L	<ul style="list-style-type: none">• Maximum linear arm speed is 15cm/sec.• Maximum linear acceleration 1.6m/sec².
	GRIPPER
	<ul style="list-style-type: none">• Simultaneous usage of 3 fingers.• Independent control of each finger is possible.• Finger force is limited to 7N.
F I R M W A R E	GENERAL
	<ul style="list-style-type: none">• Independent control of each axis is possible.• Position and error check in every 0.01 s.• Automatic system recovery on occurrence of fault.

7.2 **Precautions**

- Mobile and any other electronic gadgets must be switched off within the room where the JACO is operating. This is done to avoid electromagnetic interference.
- When the power is turned off, the JACO arm will fall on itself and may damage itself, depending on its position at the time of disconnection. Be sure to support its wrist before turning the power off.
- We must wait for the Jaco arm to reach room temperature before using it.
- Home button must not be pressed when it is holding anything.

7.3 **Movement Of Jaco Arm**

Jaco arm is very easy to move and it can be moved by Joystick and as well as by Computer programming. Jaco arm can be moved in Cartesian mode and in Angular mode. In Cartesian mode user only moves the end effector and all the joints are oriented in according manner. Jaco arm is capable of 16 different movements.

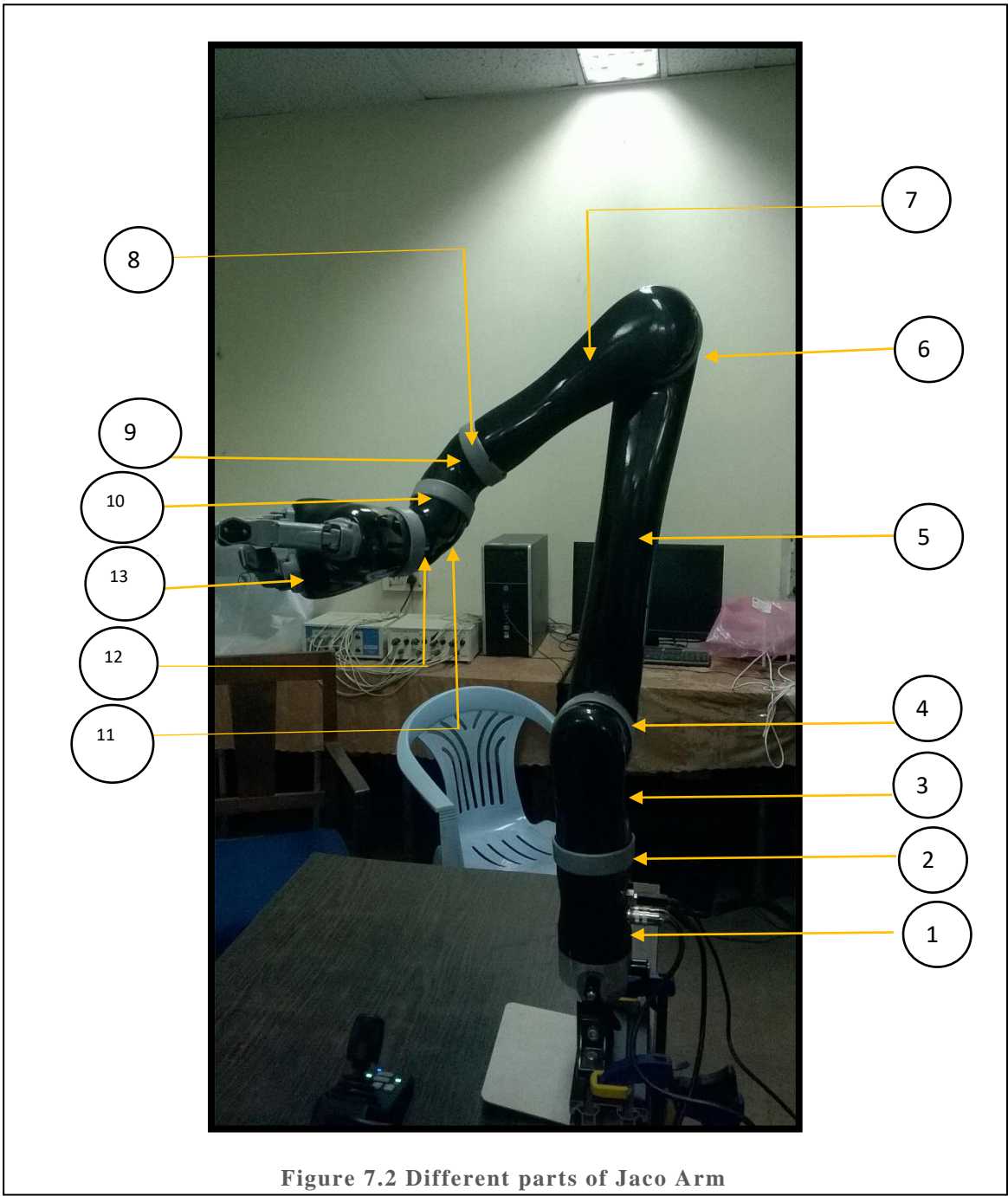
In the *Translation mode* we control the position of hand in space. Translation X is designed for left/right movement, translation Y is designed for front/back movement and translation Z is for up/down movement.

In *Wrist mode* we change the orientation of arm around the centre point of the arm. Lateral orientation refers to a thumb/index circular movement of the wrist around the reference point. Vertical orientation refers to a top/bottom circular movement of the wrist around the reference point. Wrist rotation refers to a circular movement of the hand around itself.

Drinking mode must be used with the *wrist rotation mode*. When we are operating arm in drinking mode, we must provide the offset in height and length to produce rotation that will compensate when it is used for drinking anything from glass.

There is another mode called *Finger mode* which is used to control or opening and closing of three fingers.

7.4 Different Parts Of Jaco Arm



1	Fixed Base	6	Actuator 3	11	Wrist 2
2	Actuator 1	7	Forearm	12	Actuator 6
3	Base	8	Actuator 4	13	Finger
4	Actuator 2	9	Wrist 1		
5	Arm	10	Actuator 5		

7.5 Actuator Limitation

Table 7.2 Minimum and Maximum Position of Joint Angles

	MINUMUM POSITION	MAXIMUM POSITION
Axis 1	-1000°	+1000°
Axis 2	+42°	+318°
Axis 3	+17°	+343°
Axis 4	-1000°	+1000°
Axis 5	-1000°	+1000°
Axis 6	-1000°	+1000°
Finger 1	posSwitch	posSwitch+60,0
Finger 2	posSwitch	posSwitch+60,0
Finger 3	posSwitch	posSwitch+60,0

7.6 External Connection

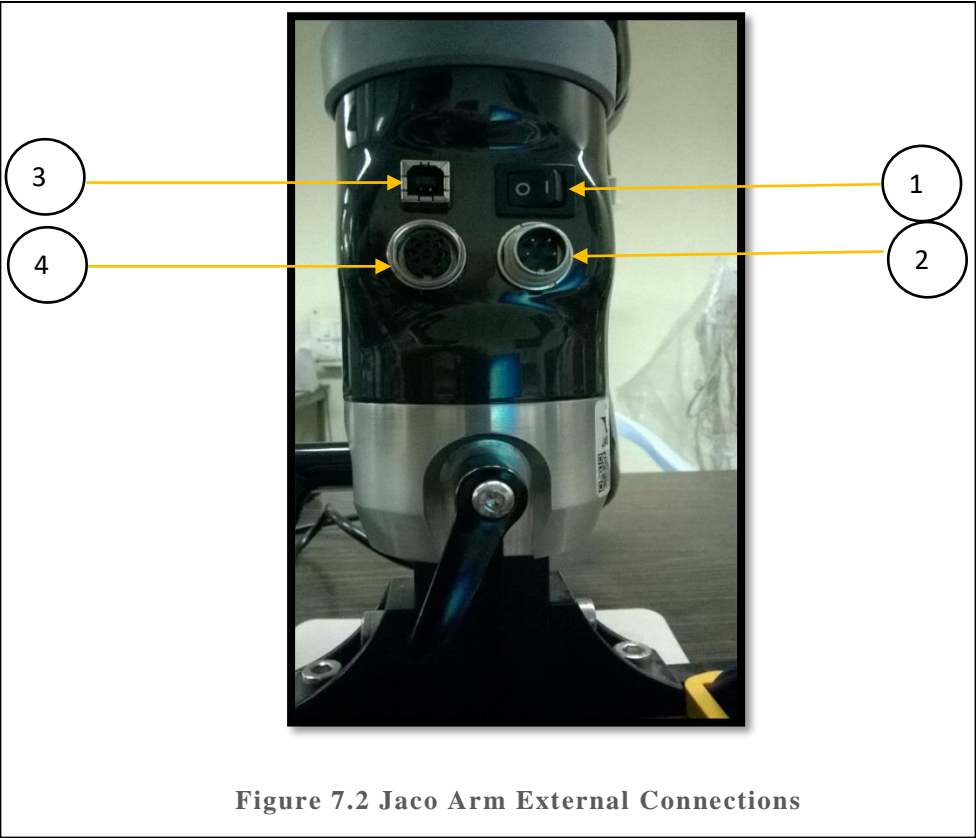


Table 7.3 Description of different parts of external connection

PART NO	DESCRIPTION
1	Power Switch to ON/OFF Jaco.
2	Power Supply from external source via adapter.
3	USB port to connect Jaco to computer.
4	Control port to connect with joystick.

7.7 Classic DH parameters frame position

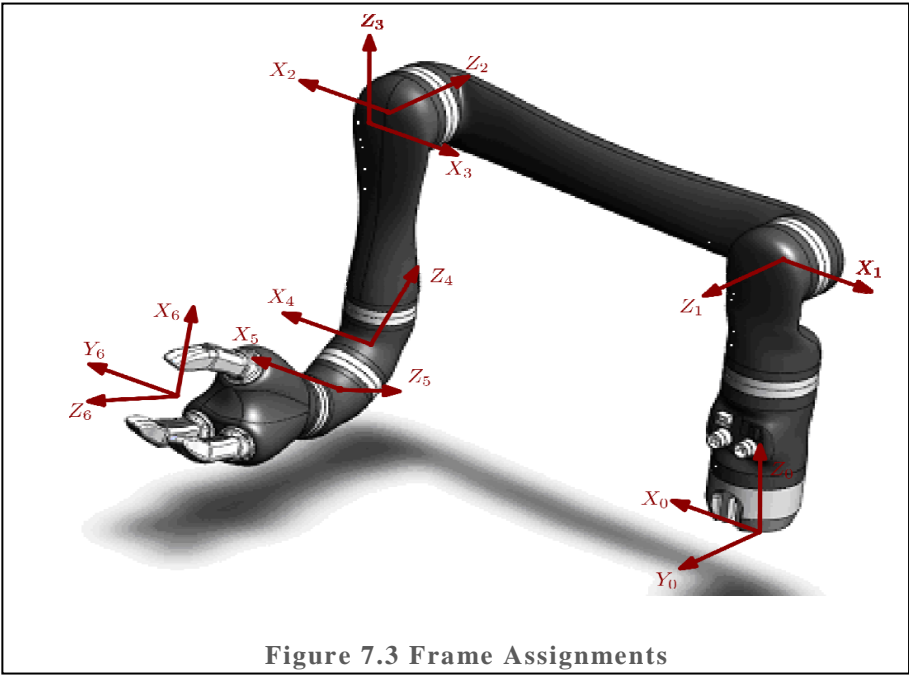


Figure 7.3 Frame Assignments

7.8 Joystick

The Kinova’s standard controller is a 3 axis joystick mounted on a support which includes 5 independent push buttons and 4 auxiliary inputs (on the back side).

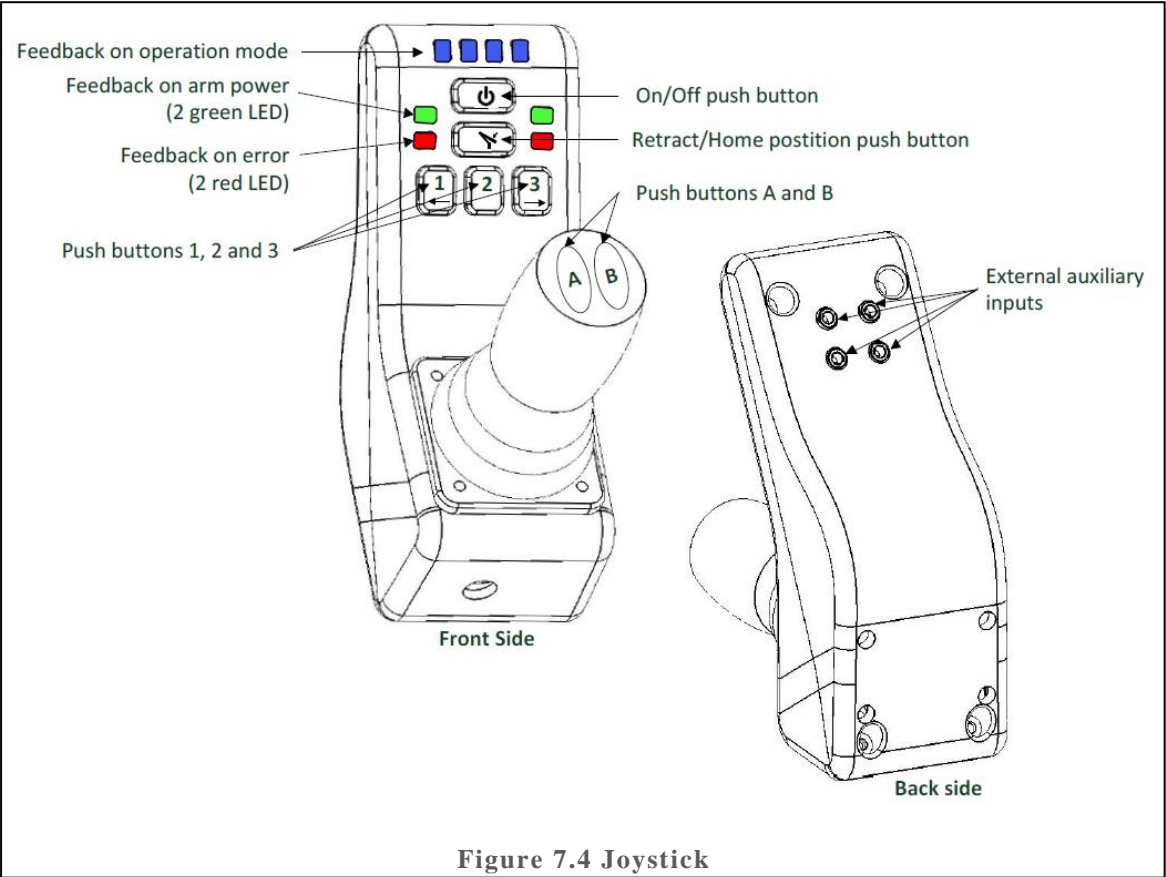


Figure 7.4 Joystick

There are two operation mode in Jaco joystick, i.e 2 axis and 3 axis mode. This joystick can control the jaco arm either in 3 axis direction or in 2 axis direction. In 2 axis mode rotation of joystick is disabled. Following table briefly describes the use of joy stick buttons.

Table 7.4 Description of joystick buttons

BUTTON	ONE CLICK	HOLD FOR 2 SECONDS
POWER	Deactivate/activate joystick	Change joystick mode (2axis or 3 axis)
HOME		Hold until desired position is reached
3 AXIS		
1	Deactivate/ Activate drinking mode	
2	-----	Set position
3	-----	Hold to go to pre set position
A	Change to Finger mode	Decrease speed
B	Change to translation/ wrist mode	Increase speed
Ext 1	Change to Finger mode	Decrease speed
Ext 2	Change to translation/ wrist mode	Increase speed

Ext 3	-----	Home position
Ext 4	Deactivate/ Activate drinking mode	-----
2 AXIS		
1	Deactivate/ Activate drinking mode	
2	Change to wrist orientation/ finger mode	Decrease speed
3	Change to translation X/Y & translation Z/wrist orientation mode	Increase speed
A	-----	-----
B	-----	-----
Ext 1	Reach Wrist orientation & Finger mode	Decrease speed
Ext 2	Reach Translation-X/Y & Translation-Z/Wrist rotation mode	Increase speed
Ext 3	-----	Home position
Ext 4	Deactivate/Activate drinking mode	



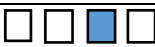

7.8.1 Visual Feedback

Joystick has 3 different colour LED which gives the states of present control mode. These LEDs are given below:

- BLUE- feedback about control mode.
- GREEN- feedback on arm power.
- RED- feedback on occurrence of error.

a) BLUE LED

Table 7.5 Blue LED condition

BLUE LIGHT		CONTROL MODE
3 AXIS		Translation (X-Y-Z)
		Wrist
		Finger
		Drinking Mode

	<div><div></div><div></div><div></div><div></div></div>	Disabled
2 AXIS	<div><div></div><div></div><div></div><div></div></div>	Translation X-Y
	<div><div></div><div></div><div></div><div></div></div>	Translation Z
	<div><div></div><div></div><div></div><div></div></div>	Wrist Orientation
	<div><div></div><div></div><div></div><div></div></div>	Finger
	<div><div></div><div></div><div></div><div></div></div>	Drinking Mode
	<div><div></div><div></div><div></div><div></div></div>	Disabled

If none of the Blue LED is glowing then controller must be in disabled mode, then we need to switch it ON.

b) GREEN LED

Table 7.6 Green LED condition

CONDITION	STATUS
Flashing	Jaco has been just turned ON and not yet ready to be used.
Solid	Jaco is ready to use.

c) RED LED

Table 7.7 Red LED condition

CONDITION	STATUS
Flashing	Put down any object grabbed by Jaco and bring the arm to HOME position and must wait until RED light is gone
Solid	Jaco is in fault and Jaco must be switched OFF and again turn it ON again.

7.9 Communication with PC

The important part in robotics is communication of Jaco with the personal computer. Here also cross-compiled executable code need to be transferred to the robot. To do that only wired communication via USB is used here. There is no provision of wireless communication in case of Jaco arm.

7.10 Interface using API

After installing JacoSoft software provided by the manufacturer, the file structure will look like below on windows platform and it will be found inside ‘Kinova’ folder on application data folder.

Table 7.8 Folder description

Folder’s name	Description
Kinova_Root	Kinova root folder
Kinova_Root \ Product	This is where all Kinova products are installed. It is located at: Kinova_Root\
Product \ Licenses	This is where all licence related files are kept.
Product \ Jacosoft	This is the root folder of Jacosoft.
Jacosoft \ API	This is the root folder of all Kinova API.
API \ Data	This is a folder where all serialized data are stored.
Data \ Config	This is a folder where you can store serialized configurations.
Data \ Logs	This is the root folder of all log files.
Logs \ LogErrors	This is where all errors log files are stored.
Logs \ Activities	This is where all activities log files are stored.
Data \ Positions	This is a folder where you can store serialized position.
Data \ Profiles	This is the root folder of all stored profiles.
Data \ Trajectories	This is a folder where you can store trajectories.

7.10.1Libraries

Jaco’s API include set of libraries which must be included at the time programming to communicate with Jaco arm. Along with that we must copy the external DLL folders at the same place of our application. All the libraries are given below.

- Kinova.API.Jaco.dll
- Kinova.DLL.Data.dll
- Kinova.DLL.SafeGate.dll

- Kinova.DLL.Tools.dll
- Kinova.DLL.USBManager.dll
- Kinova.DLL.TestData.dll
- Kinova.DLL.ReportBuilder

7.11 Programming

Programming API is available both in C# and C++. The examples below show the development of Jaco Program using C# programming language.

All the APIs that have been provided, can be classified into 3 groups of functionalities which can be accessed through an object called “manager”. Manager can be accessed by defining CJacoArm object.

Three groups of managers are given below:

- Configuration manager
- Control manager
- Diagnostic manager

7.11.1 Writing Jaco Application

1. At first open the Visual Studio and create a C# console application with a main function and here we will name it “Hello Jaco”. It will look like below;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace HelloWorld
{
    class HelloJaco
    {
        static void Main(string[] args)
        {
        }
    }
}
```

```
}
}
```

2. Initialize an instance of an object CJacoArm and it will be our entrance path to API. Here we will provide constructor an encrypted password.
3. We must include all the required DLL (digitally linked libraries) files to our project and at the same time we must make sure that we have copied all DLL files inside the project folder and External_DLL folder is at same location of our application.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Kinova.API.Jaco;
using Kinova.DLL.SafeGate;
namespace HelloWorld
{
    class HelloJaco
    {
        static void Main(string[] args)
        {
            string MyValidPassword = "Actual Password";
            try
            {
                //An Object that represents the robotic arm Jaco.
                CJacoArmJaco = new CJacoArm(Crypto.GetInstance().Encrypt(MyValidPassword));
                //Is Jaco ready to communicate?
                if (Jaco.JacoIsReady())
                {
                    System.Console.WriteLine("\n\nHelloJaco ! ! !");
                }
            }
            else
            {
                System.Console.WriteLine("Jaco is not ready to communicate.");
            }
        }
        catch (Exception ex)
        {
            System.Console.WriteLine("Exception during execution of the example. Verify " +
```

```

"your API installation, verify if your Jaco is " +
"connected and verify that " +
"you have a valid password.");
}
System.Console.WriteLine("End of the example...");
System.Console.ReadKey();
}

```

4. Here the line 'catch(Exception ex)' catches all the exception and display a message when particular exception is hit.
5. The program will display Hello Jaco on the screen if the Jaco is ready to communicate otherwise it will display "Jaco is not ready to communicate."

7.11.2 Client Configurations

Basic configuration of Jaco arm can be retrieved by using Jaco API and we can also modify it and send back to Jaco. Here the main function will look like this:

```

static void Main(string[] args)
{
    string MyValidPassword = "MyValidPassword";
    try
    {
        //An Object that represents the robotic arm Jaco.
        CJacoArmJaco = new CJacoArm(Crypto.GetInstance().Encrypt(MyValidPassword));
        CClientConfigurationsconfig = new CClientConfigurations();
        //Is Jaco ready to communicate?
        if (Jaco.JacoIsReady())
        {
            config = Jaco.ConfigurationsManager.GetClientConfigurations();
        }
        else
        {
            System.Console.WriteLine("Jaco is not ready to communicate.");
        }
    }
    catch (Exception ex)
    {

```

```
System.Console.WriteLine("Exception during execution of the example. Verify " +
"your API installation, verify if your Jaco is " +
"connected and verify that " +
"you have a valid password.");
}
System.Console.WriteLine("End of the example...");
System.
```

Here the line ‘CClientConfigurationsconfig = new CClientConfigurations()’ defines an object which is a member of Kinova.DLL.Data.Jaco.Config and this object contains all data specific to a Jaco’s client. The line ‘configJaco.ConfigurationsManager.GetClientConfigurations()’ has been used here to access the configuration manager and to get all the client configuration.

Now we can display the client configuration and also can modify the configuration.

```
if (Jaco.JacoIsReady())
{
//Get the data from Jaco
config = Jaco.ConfigurationsManager.GetClientConfigurations();
//Display the Data
System.Console.WriteLine(" Name : " + config.ClientName);
System.Console.WriteLine("Max linear speed : " + config.MaxLinearSpeed);
System.Console.WriteLine(" Organization : " + config.Organization);
config.ClientName = "Jaco ";
config.MaxLinearSpeed = 0.1f;
config.Organization = "MI6";
//We send the new data to Jaco
Jaco.ConfigurationsManager.SetClientConfigurations(config);
//Get the new data from Jaco
config = Jaco.ConfigurationsManager.GetClientConfigurations();
//Display the new Data
System.Console.WriteLine(" Name : " + config.ClientName);
System.Console.WriteLine("Max linear speed : " + config.MaxLinearSpeed);
System.Console.WriteLine(" Organization : " + config.Organization);
}
```

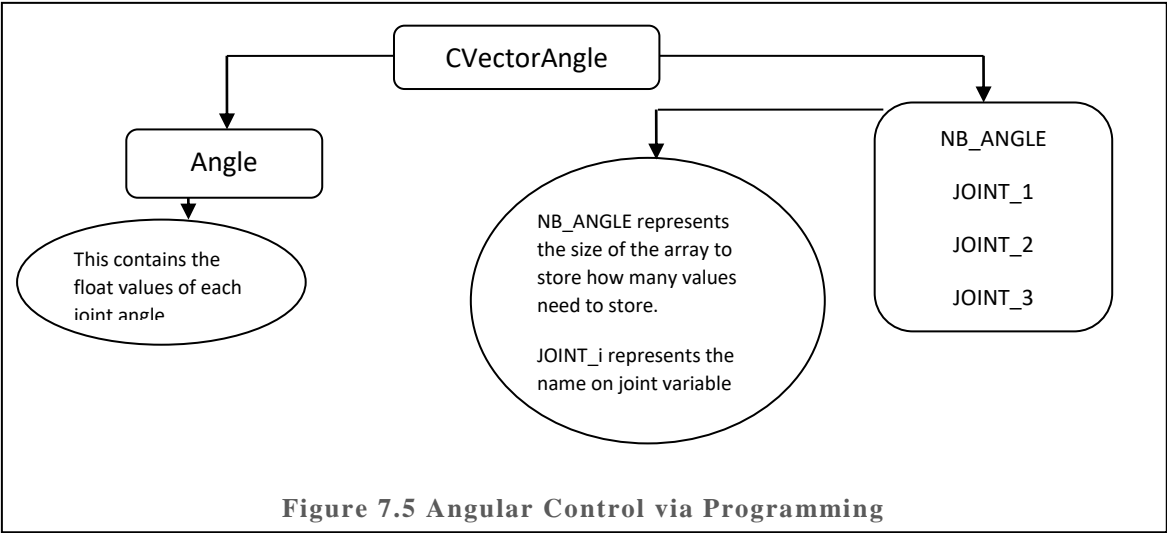
The above code first displays the present configuration of Jaco arm as client name, maximum linear speed and organization, then we have modified the said parameters and now it will display the new configuration.

7.12 Control Types

7.12.1 Angular Control Type

This type of control is pretty simple. It consists of moving the robotic arm joint by joint by specifying angles to each of them. The most important thing you need to know about this type of control is that there is no advance security and it is quite possible for the arm to hurt itself. The value of angle of each joint must be limited within the range.

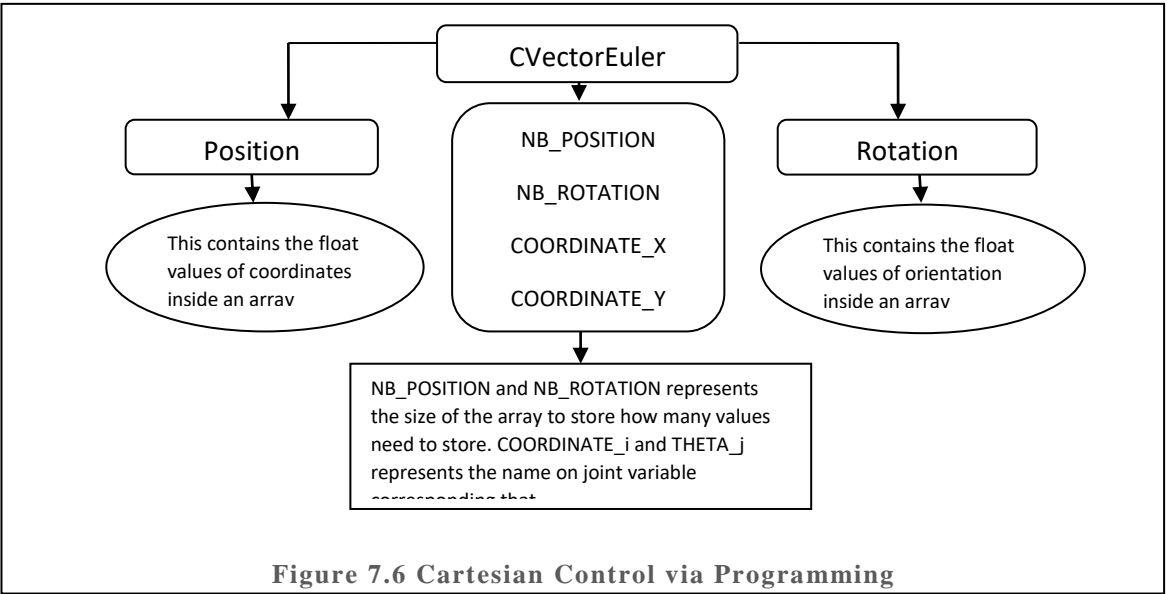
The data structure *CVectorAngle* is involved to deal with this type of control. For example to get the current value of all the joints in Jaco use the function *GetJointPosition* which returns a object of *CVectorAngle* type contains all 6 joints values in degree. The fields inside the *CVectorAngle* object is named as *JOINT_i* where i is the number of joint. All these are of float data type.



7.12.2Cartesian Control Type

This type of control is more intuitive than the angular control and it use the onboard kinematics system of Jaco. Basically, one will specify a coordinate with its orientation and Jaco goes there if it can.

Most of the time, when we deal with cartesian position, the data structure *CVectorEuler*is involved. The main purpose of this class is to hold information about the cartesian position of Jaco. The fields inside the *CVectorEuler* object are COORDINATE_X, COORDINATE_Y, COORDINATE_Z, THETA_X, THETA_Y, THETA_Z. All these are of float data type.



7.13 Summary

In this chapter, detailed discussion regarding the physical dimensions and limitations of the Jaco robotic arm has been carried out. The Jaco programming API and control types have also been introduced. The Jaco arm forms the crux of the experiment carried out for the work proposed in this thesis, and thus this chapter would help the reader with little previous knowledge about the Jaco arm, gain some understanding about it.

7.14 References

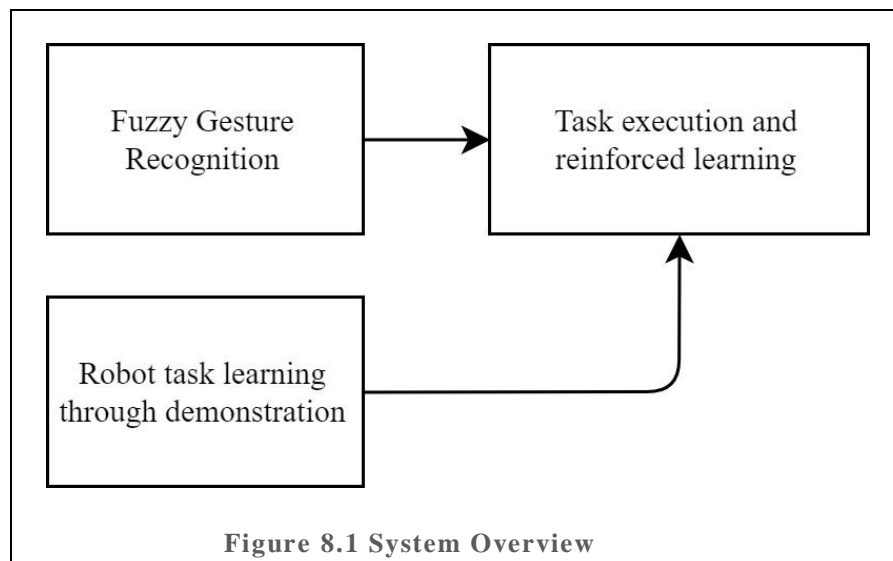
- [1] JacoAPI_ProgrammingGuide published by Kinova.
- [2] Jaco User Guide Research published by Kinova.
- [3] Getting Started Guide for Jaco published by Kinova.
- [4] Jacosoft User Guide published by Kinova.
- [5] Jaco DH parameters guide published by Kinova.
- [6] Jaco maintenance Guide published by Kinova.

8

Experiment and Analysis

8.1 System Framework

The proposed system is structured into separate functional blocks, each of which is concerned with processing the received data from the previous block, manipulating it accordingly and passing it onto the subsequent block. This functional grouping provides the advantage of fine-tuning any individual block according to requirement that is independent of the functioning of other blocks. Fig. 8.1 shows the three basic blocks proposed and their interconnections.

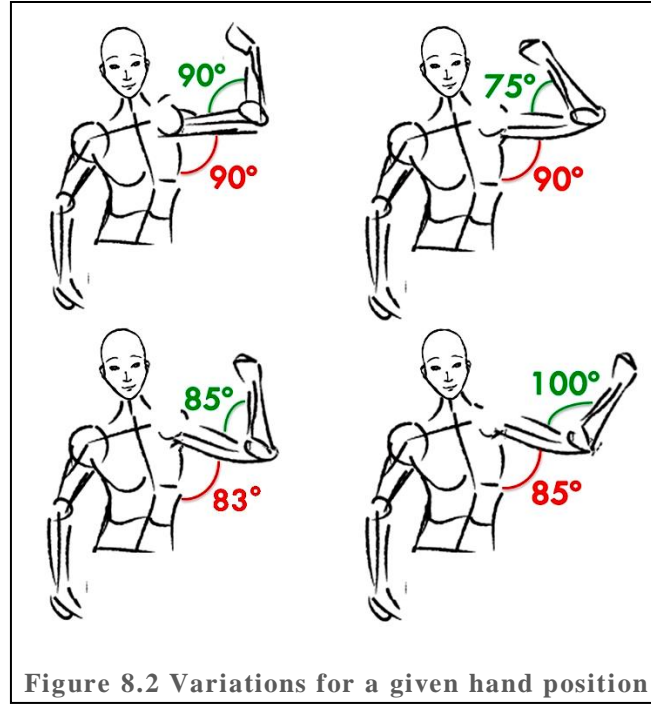


The user input is taken in through a Microsoft Kinect device, which can detect human skeletons and provide information about the joint positions of the same in a three dimensional space [1]. The information about the angles between certain joints is extracted from this raw data using basic co-ordinate transformations. The raw data in this work constituted a number of basic hand positions in a three dimensional space corresponding to a number of basic angle sets between parts of the hand. Angles are named based on the joint where they are created. As an example, the elbow joint angle is the angle between the arm and the forearm. Though the generalized structure of the proposed system allows for inclusion of more hand positions, we feel five positions should suffice to explain the methodology presented in this text.

In this work, the angles made at the shoulder and elbow joints in the right hand are used as features for the gestures that are defined later. The fuzzy gesture recognition block inputs the samples for these features for different subjects which are then passed on to a clustering system. The clustering system clusters the obtained data, thus recognizing the basic hand positions. The statistical distribution of data points inside the clusters are then used to set up an interval type-2 fuzzy system during the training phase. The tracking phase, involving the task execution and reinforced learning, starts with accepting a single input through the Kinect device and recognizing the gesture it belongs to by evaluating the fuzzy membership functions for all the gestures and choosing the one with the highest membership. This information is passed onto a learning system which learns the provided gesture and takes an action accordingly. In the robot task learning through demonstration block, an action is demonstrated by a skilled person under the observation of a Kinect device. The device records a subject performing a pre-defined trajectory movement which is then fed into a Jaco Kinova robot arm. This work uses a form of non-monotonic learning for the purpose. The system incorporates dynamism in training, and can thus be easily trained as and when required (for example when it has difficulty in recognizing a hand position).

The motivation for using a fuzzy system to model the individual hand positions derives from the fact that a subject may not remember the configuration of the limb for a single hand position accurately during different trials. Moreover, different subjects may represent a certain hand position a bit differently than others. This introduces fuzziness in the system, where every hand position can have both intra-subject and inter-subject variations.

Fig. 8.2 highlights the points made in the preceding paragraph. The ideal scenario, which considers a shoulder angle of 90 degrees and an elbow angle of 90 degrees, represents a hand position (for the first subject). However, the same hand position may be represented differently by different subjects as portrayed in the figure for the other three subjects. All these variations can be efficiently captured and recognized by using a fuzzy system.



8.2 Experiment

8.2.1 Feature Selection from Input Data and Training

Microsoft's Kinect device is the sensor used to capture the input data from subjects. Kinect can obtain the three dimensional co-ordinate positions of several joints in a human body which can be used to construct a skeleton of the same. Since the captured positions of the joints in the skeleton would vary for different subjects, based on several factors like distance of the subject from the Kinect device, his orientation with respect to the capture area of the Kinect device, his height and the length of his limbs, they are not suitable to be considered as features. The angle between various skeletal parts, for example the spine and the arm, is a more universal descriptor for a given hand position as it is independent of the said factors and is thus considered.

Let, $\vec{A}, \vec{O}, \vec{F}, \vec{S}$ be the position vectors of the elbow joint, shoulder joint, wrist joint and waist respectively in the capture area of the Kinect device, as shown in Fig. 8.3. The origin of these

position vectors is the center of the capture area for the Kinect device. \vec{V}_a is used to represent the arm as a vector, \vec{V}_f to represent the forearm as a vector, and \vec{V}_s to represent the spine as a vector.

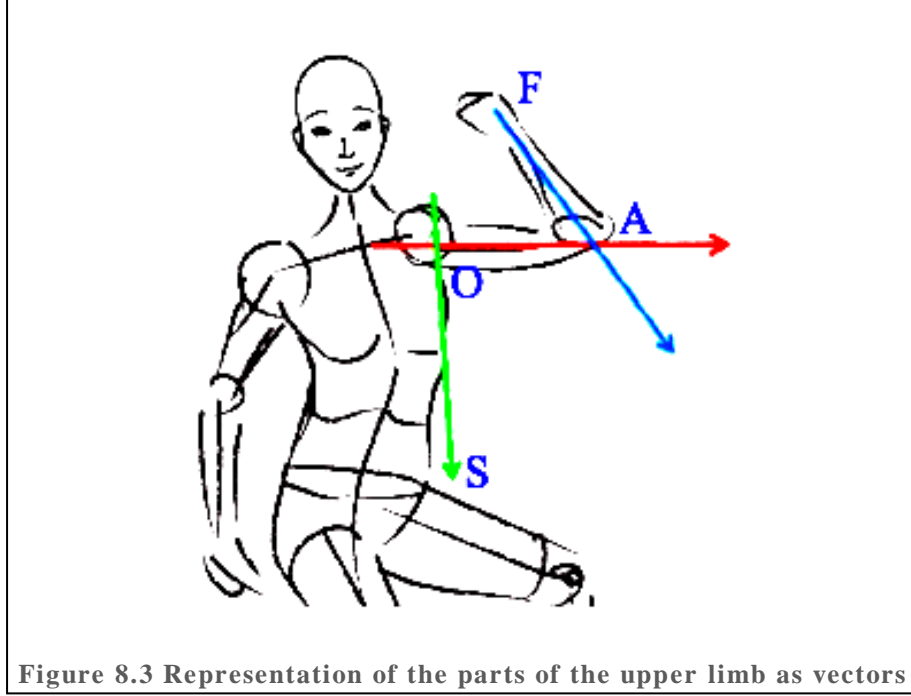


Figure 8.3 Representation of the parts of the upper limb as vectors

$$\vec{V}_a = \vec{A} - \vec{O} \quad (1)$$

$$\vec{V}_f = \vec{F} - \vec{A} \quad (2)$$

$$\vec{V}_s = \vec{S} - \vec{O} \quad (3)$$

In this work, the angle between the spine and the arm, called the shoulder angle (a_1), and the angle between the arm and the forearm, called the elbow angle (a_2), are considered as features. These angles must lie between 0 degree and 180 degrees. The angles are obtained using the following relations:

$$a_1 = \arccos \frac{|\vec{V}_a \cdot \vec{V}_s|}{|\vec{V}_a| |\vec{V}_s|} \quad (4)$$

$$a_2 = \arccos \frac{|\vec{V}_f \cdot \vec{V}_a|}{|\vec{V}_f| |\vec{V}_a|} \quad (5)$$

These features are extracted from the raw data for each subject for a large number of observations. The data obtained from each subject is stored separately to facilitate the formation of the upper and lower bounds in the type 2-interval fuzzy set to be constructed later. Thus a set of vectors is constructed of the form:

$$A_{n,i} = \begin{bmatrix} {}^n a_{i,1} & {}^n a_{i,2} \end{bmatrix} \quad (6)$$

Here n varies from 1 to N , where N denotes the number of subjects used during training, and i varies from 1 to I , where I denotes the number of observations for each subject. ${}^n a_{i,1}$ is the angle between the spine and the arm, i.e., the shoulder angle, and ${}^n a_{i,2}$ is the angle between the arm and the forearm, i.e., the elbow angle for the i^{th} observation of the n^{th} subject.

8.2.2 Clustering and Construction of Interval Type-2 Fuzzy Sets

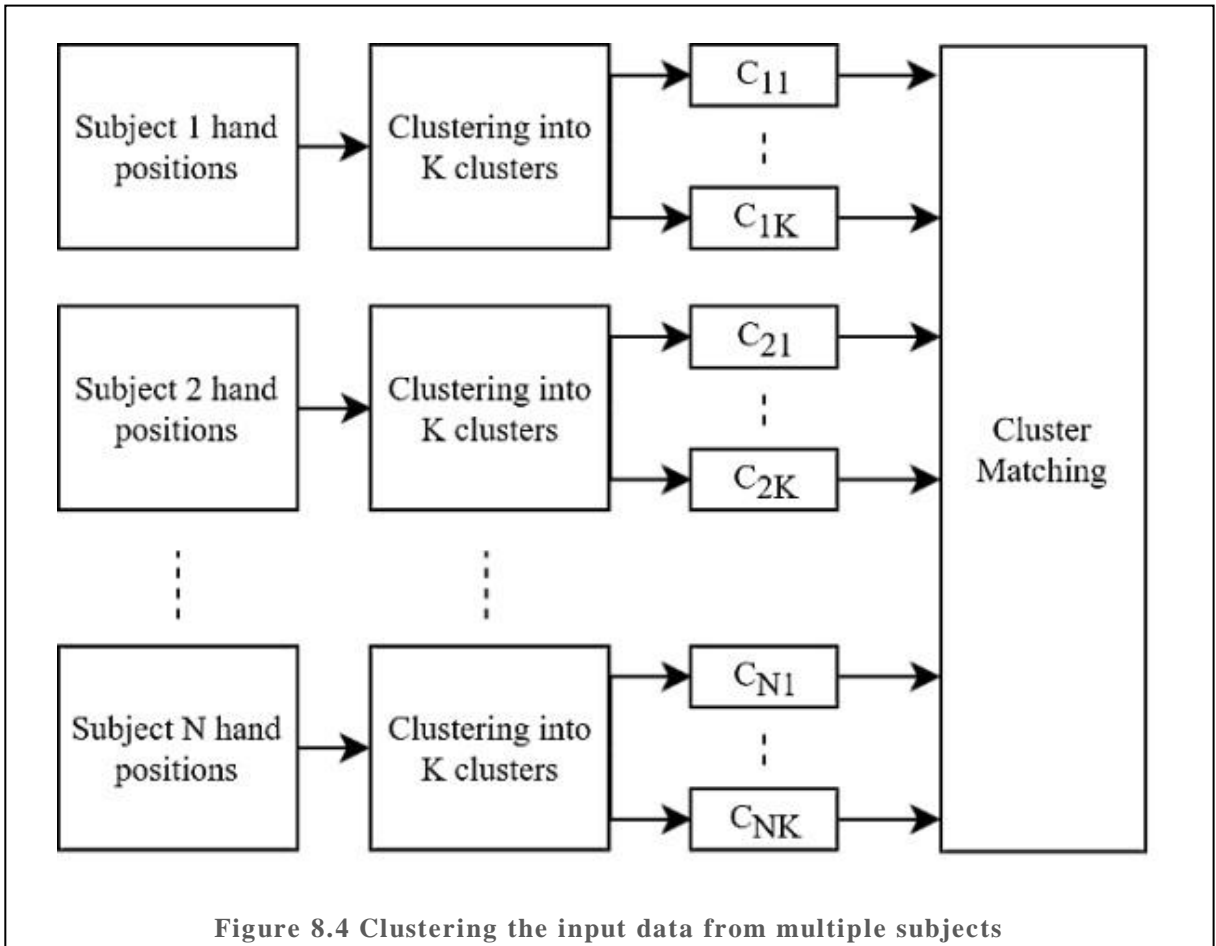
The recorded data from the preceding section contains I observations for each of the N subjects. However, for a particular subject, all I observations do not correspond to a single hand position, but multiple number of hand positions mixed together. To construct fuzzy memberships for individual hand positions, a clustering process is first carried out. Each obtained cluster corresponds to a single hand position for a given subject.

The obtained data is clustered using the K-medoids clustering technique [3], where the initial seed points are selected in a probabilistic manner, as put forward in [4]. All the data points obtained from all the subjects are considered for clustering the raw data into K clusters.

In this work, three cluster centers were considered (i.e., $K=3$) while using the K-medoids algorithm for both the shoulder and elbow angles. These two angles were combined together to form five different hand positions.

After the clustering has been completed for each subject, the data points in each cluster for each subject are forwarded to a cluster matching system. The cluster matching system

matches clusters from different subjects based on their Euclidean distances from one another and assigns similar cluster numbers to the matched clusters. For example as seen in Fig. 8.4, the first cluster of the first subject, C_{11} might correspond to the same hand position that the third cluster of the second subject, C_{23} represents. Thus, C_{11} and C_{23} would represent the same hand positions, albeit by different subjects and are assigned the same cluster number.



Clusters having similar cluster numbers are considered for construction of the interval type-2 fuzzy sets [2] as portrayed in Fig. 8.5.

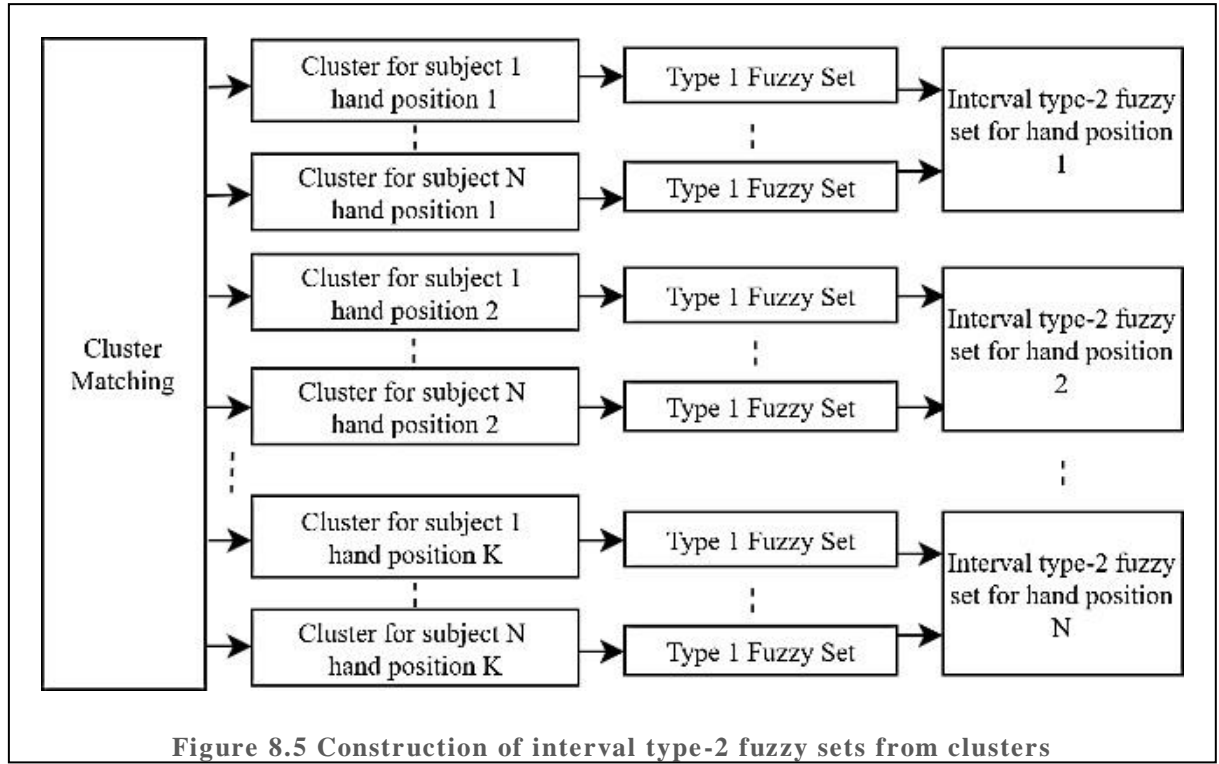


Figure 8.5 Construction of interval type-2 fuzzy sets from clusters

Within clusters having same cluster numbers, the data points for each subject are taken separately and their mean and standard deviation are calculated. For each such subject, a Gaussian Membership distribution is obtained using the mean and standard deviation for that subject:

$$\mu(x) = \exp\left[-\frac{(x - \mu_{xn})^2}{2\sigma_{xn}^2}\right] \quad (7.1)$$

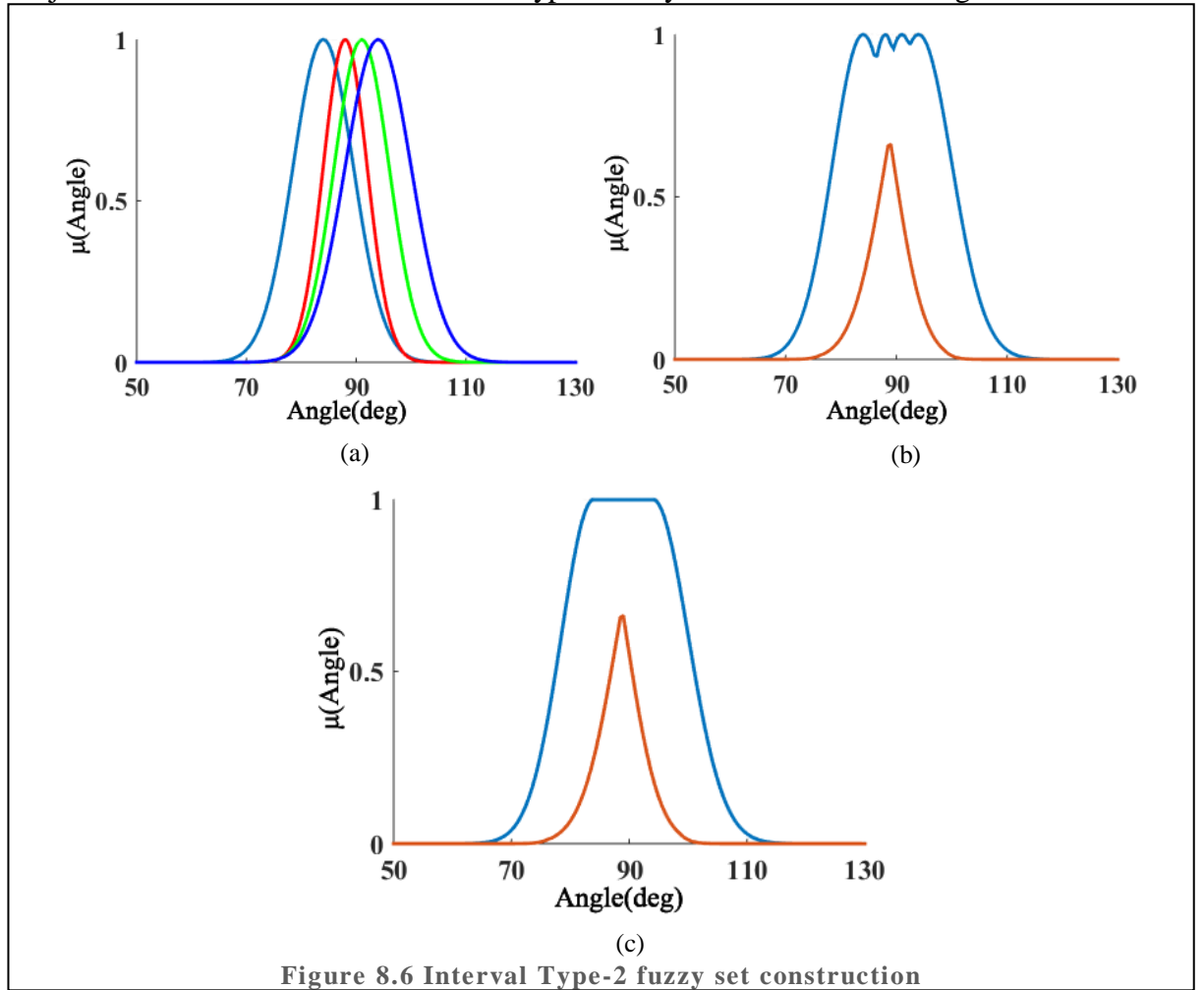
$$\mu(y) = \exp\left[-\frac{(y - \mu_{yn})^2}{2\sigma_{yn}^2}\right] \quad (7.2)$$

Here, μ_{xn} and σ_{xn} are the mean and standard deviation for the observations of the angle between the spine and the arm for the n^{th} subject within a single cluster, μ_{yn} and σ_{yn} are the mean and standard deviation for the angle between the arm and the forearm for the same subject within a single cluster. x and y represent the captured angles between the spine and the arm, and between the arm and the forearm respectively. Thus, from within a cluster, several

such distribution functions are obtained. These distribution functions are used to construct the interval type-2 fuzzy set for each hand position by obtaining the maximum and minimum distribution functions from these distributions. In this work, each captured angle has been discretized into steps of 0.5 degree, within the range of 0 degree to 180 degrees, thereby limiting each angle to have 360 possible values. Thus, for every joint angle considered in all of the hand positions, we have a fuzzy set of the form:

$$A_k = \{\mu_k \mid k\} \quad (8)$$

Here, A_k is the fuzzy set for the k^{th} joint angle, and μ_k is the corresponding membership value, obtained using the Gaussian function (7). During training, every time a new subject provides a hand position that has a joint angle k , a new μ_k is created. The A_k 's of different subjects are used to construct an interval type 2 fuzzy set as illustrated in Fig. 8.6.



All the membership functions of different subjects (a) are considered to form the upper and lower distribution functions for the type-2 fuzzy set (b). The functions are then smoothened out to obtain the final fuzzy membership (c).

The clusters obtained for each angle in a hand position thus capture both inter and intra subject variance and can be used to efficiently detect them.

8.2.3 Robot Task Learning through Demonstration

The robot is made to learn a set of simple tasks through demonstration. A complex task may be performed by executing a set of these tasks sequentially. The tasks considered in this work includes traversing a trajectory when instructed to do so. For this purpose, the robot is made to learn specific trajectories by imitating the joint positions of a human arm during the trajectory training phase. The Kinova Jaco robot arm executes trajectories as a sequence of data points in space. Thus, during the training phase, the states of the joints in a human arm is tracked at regular intervals to create a trajectory. Each of these trajectories, after training, are assigned a trajectory number.

8.2.4 Task Execution and Reinforced Learning

Once the training has been done with, any arbitrary subject can act as an input to the system. The angles used as features during training are extracted in a similar manner as specified before and passed through the fuzzy system to check which hand position it corresponds to. This is done in two steps. Firstly, for each captured joint angle, the closest trained angle is found out. This is done by locating the fuzzy set which has the maximum membership value corresponding to the input joint angle. If d_i is the joint angle input, we decide that it corresponds to the k^{th} trained angle if

$$\mu_k(d_i) = \max (\mu_j(d_i)) \forall j$$

j corresponds to the set of all trained angles. Secondly, the various joint angles considered are grouped together using a production rule to detect a hand position. Such a production rule for two joint angles can be given as follows:

If the joint angle P is close to p degrees and the joint angle Q is close to q degrees, then it corresponds to the hand position R .

If each and every hand position is detected using the aforementioned approach, a gesture can also be easily recognized owing to the fact that a gesture is basically an ordered sequence of hand positions. Once a gesture has been recognized, a task from the set of trained tasks need to be performed. This is done using the procedure highlighted in Fig. 8.7.

A set of G gestures is defined at first. Each such gesture is an ordered sequence of hand positions as explained before. A set of A actions is also defined, which was obtained during the robot task learning phase. A weight matrix W , of dimension $G \times A$ is initialized, which is constantly updated during the learning period. Each element of W is initialized to a positive value to avoid the weights getting stuck at zero during learning. The proposed method works faithfully irrespective of the initial values taken. After an input X is taken from a test subject (consisting of joint angles), the membership value of X is evaluated in each of the K fuzzy sets. The fuzzy set with the highest membership value for X indicates the detected angle from the set of K trained angles. The hand position is obtained by passing the detected angles through the production rules. The hand positions may be passed through a sequence detector to detect gestures. Once a gesture has been identified, an action a is performed from the set of actions A such that the value of W_{ga} is maximum for that particular gesture g and all actions $a \in A$. The subject is asked for the correctness of the action performed corresponding to the input gesture and the value of W_{ga} is increased adaptively for a positive response and decreased for a negative response. This helps in non-monotonic learning as the system can respond efficiently to a change of knowledge as well. The rate of un-learning an old knowledge and acquiring new knowledge, however depends explicitly on the value of an adaptive parameter (β), among other factors. The rate of learning can be controlled even more

efficiently by using different updation values for a correct action performed and a wrong action performed for a given gesture.

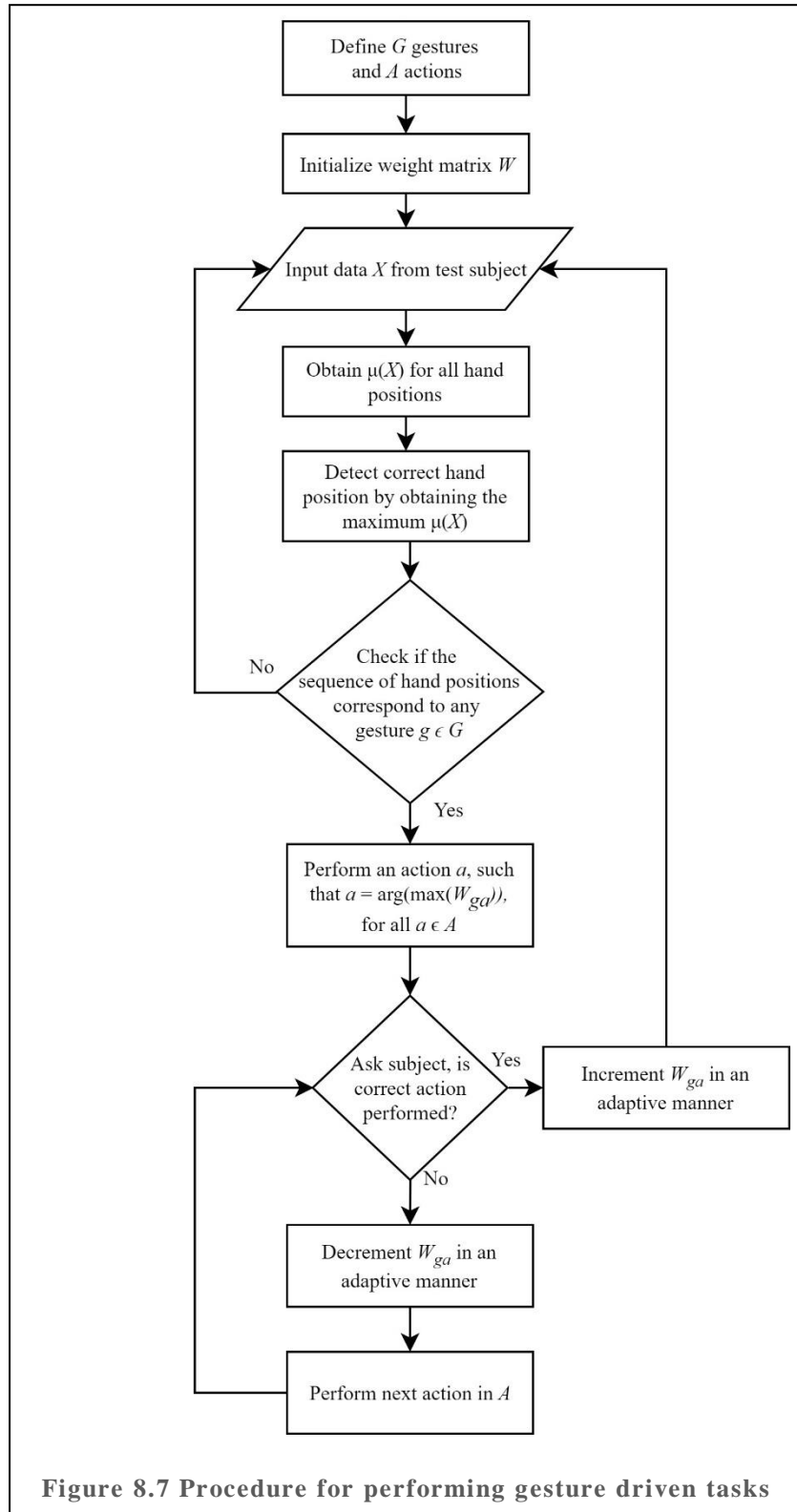


Fig. 8.8 provides a comprehensive view of the task learning process. In the learning algorithm, the parameter β is chosen based on the relative importance a subject imparts to new knowledge over existing knowledge. The choice of β is also highly application specific and plays an important role in the learning and unlearning process, as has been highlighted in the results section.

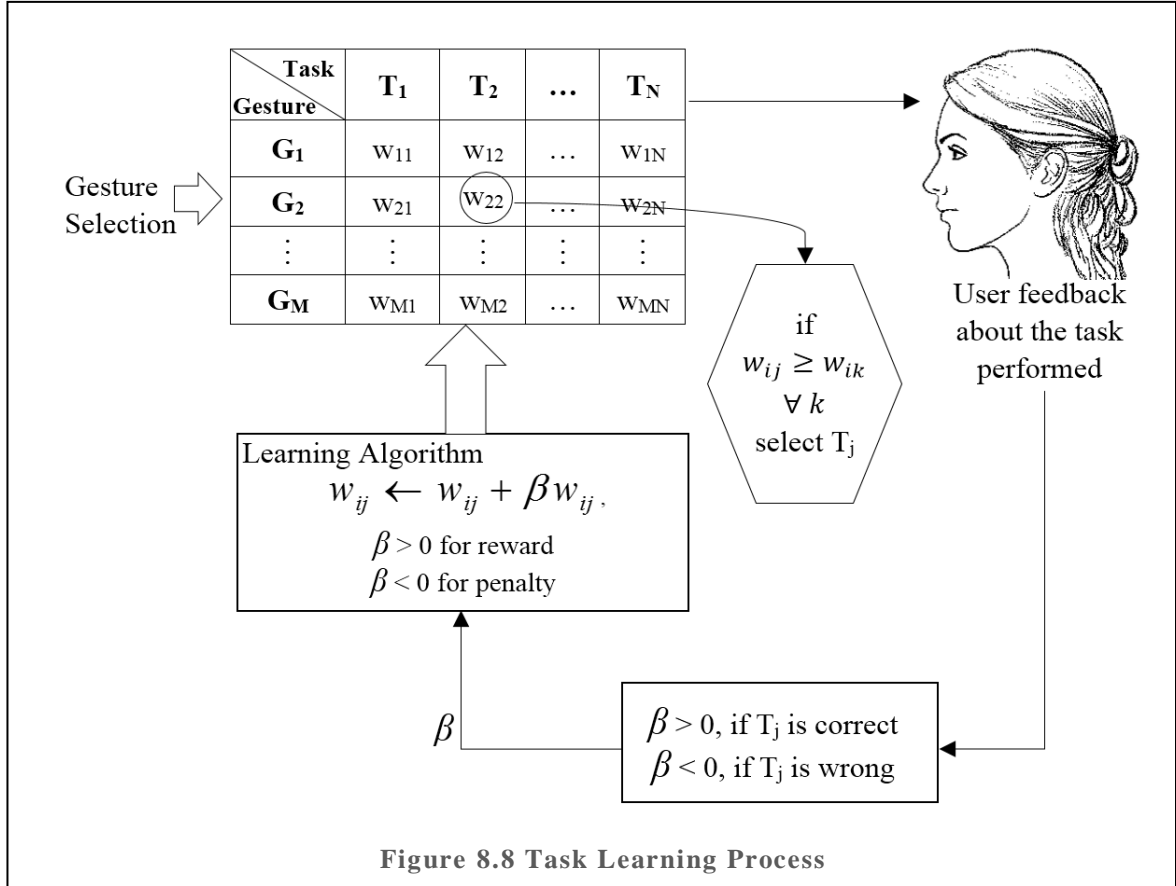


Figure 8.8 Task Learning Process

8.3 Results

8.3.1 Analysis of the Training Data

In this experiment, five unique arm positions are defined, where each position consists of two angles viz. the shoulder and elbow angles of the subject's right hand. Table 8.1 shows the angles corresponding to each arm position in the ideal case.

Table 8.1 Arm Positions (angles in degrees)

	Shoulder Angle	Elbow Angle
Position 1	180	180
Position 2	135	180
Position 3	90	180
Position 4	90	135
Position 5	90	90

10 different subjects, both men and women, of different age groups, and of different educational backgrounds are chosen for collecting the gesture data with the aim of obtaining a varied dataset. 2000 gestures of each subject over different time-frames are captured to form the dataset. Table 8.2 and Table 8.3 show the mean and variance, for each subject for every arm position when their gesture data is clustered as stated in section 8.2.2.

Table 8.2 Observations for Shoulder Angle (all values are in degrees)

	Mean	Var	Mean	Var	Mean	Var
Ideal	90	0	135	0	180	0
Sub 1	92.1	2.93	135.1	3.54	172.2	4.59
Sub 2	91.5	3.32	134.7	3.23	166.9	3.00
Sub 3	88.2	3.71	128.6	2.79	163.0	2.85
Sub 4	96.6	2.61	131.0	2.21	171.2	5.99
Sub 5	92.6	2.14	137.2	1.88	168.2	2.80
Sub 6	90.3	2.89	131.8	3.92	159.7	4.38
Sub 7	89.0	4.52	130.0	4.63	154.8	3.64
Sub 8	85.3	2.44	125.5	4.82	162.2	1.35
Sub 9	89.3	3.52	126.0	3.27	166.3	2.43
Sub 10	91.8	2.08	128.1	2.95	164.2	3.25

Table 8.3 Observations for Elbow Angle (all values are in degrees)

	Mean	Var	Mean	Var	Mean	Var
Ideal	90	0	135	0	180	0
Sub 1	97.1	3.38	138.5	3.66	168.3	2.08
Sub 2	93.0	2.46	142.3	1.98	174.8	4.62
Sub 3	89.5	4.32	141.1	4.41	172.8	2.67
Sub 4	90.8	3.39	135.2	2.31	172.0	6.58
Sub 5	101.3	4.09	129.9	2.45	169.9	1.03
Sub 6	98.1	2.22	140.2	3.05	162.4	3.42
Sub 7	91.8	2.77	133.6	5.53	163.5	2.91
Sub 8	92.7	1.81	137.5	2.00	167.1	4.33
Sub 9	94.3	3.91	134.0	4.86	165.6	2.42
Sub 10	96.4	2.51	140.8	2.60	171.0	2.79

It is seen that the mean for different subjects are marginally different, but close to the ideal case. The variance values are low as well. However, it is also noted that subjects fail to attain a perfect 180 degrees, for both the shoulder and the elbow angles, which can be attributed to muscular restrictions. From these values the fuzzy sets are formed, as discussed in section

8.3.2 Errors during Robot Task Learning

During the robot task learning phase, as explained in section 8.2.3, the robot is taught 10 distinct tasks. Each task is demonstrated by a teacher, by performing a particular pattern of hand movements. Each pattern is captured and encoded into a sequence of six pair of points. Each pair consists of a shoulder angle and an elbow angle value, which are mapped onto the second and third joint angles of the Jaco arm. Afterwards the pattern shown by the teacher and the trajectory followed by the mimicking robot are found to differ by a small margin. Fig. 8.9 displays the deviation of the Jaco arm from the teacher for one instance during training. The error values for one entire task has been shown in Table 8.4.

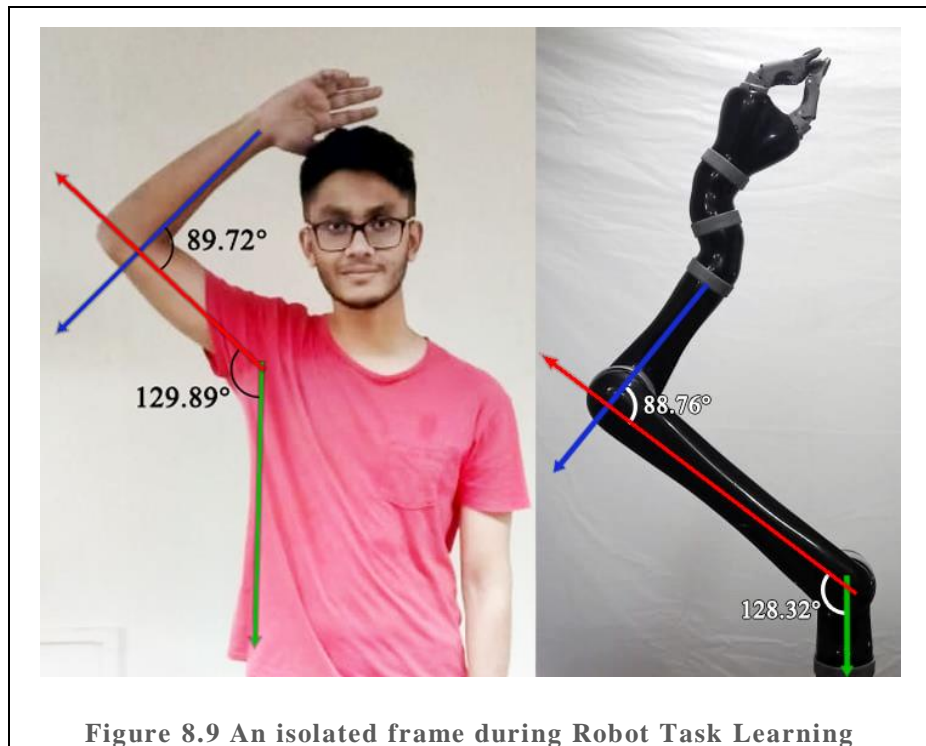


Table 8.4 Deviation of robot position from teacher (all values are in degrees)

	Path	Pt. 1	Pt. 2	Pt. 3	Pt. 4	Pt. 5	Pt. 6
Teacher	Shoulder Angle (I_a)	42.38	65.31	92.60	101.04	118.20	129.89
	Elbow Angle (II_a)	169.54	148.39	131.23	114.54	95.66	89.72
Robot	Joint 2 Angle (I_b)	43.21	66.18	92.48	101.84	117.76	128.32
	Joint 3 Angle (II_b)	168.67	147.92	130.05	112.97	94.06	88.76
Angle Errors	(I_b) - (I_a)	0.83	0.87	- 0.12	0.80	- 0.44	- 1.57
	(II_b) - (II_a)	- 0.87	- 0.47	- 1.18	- 1.57	- 1.60	- 0.96

8.3.3 Fuzzy Gesture Recognition

Fig. 8.10 shows three hand positions considered for a gesture. Thus, the subject moves his right hand through positions the three positions (from left to right) to perform a gesture. Each of the hand positions are detected based on the angles at the shoulder and the elbow.



Figure 8.10 Hand positions for a Gesture

The hand position detection for the first position is shown in Fig. 8.11. The shoulder angle is close to 135 degrees and the elbow angle is close to 90 degrees.

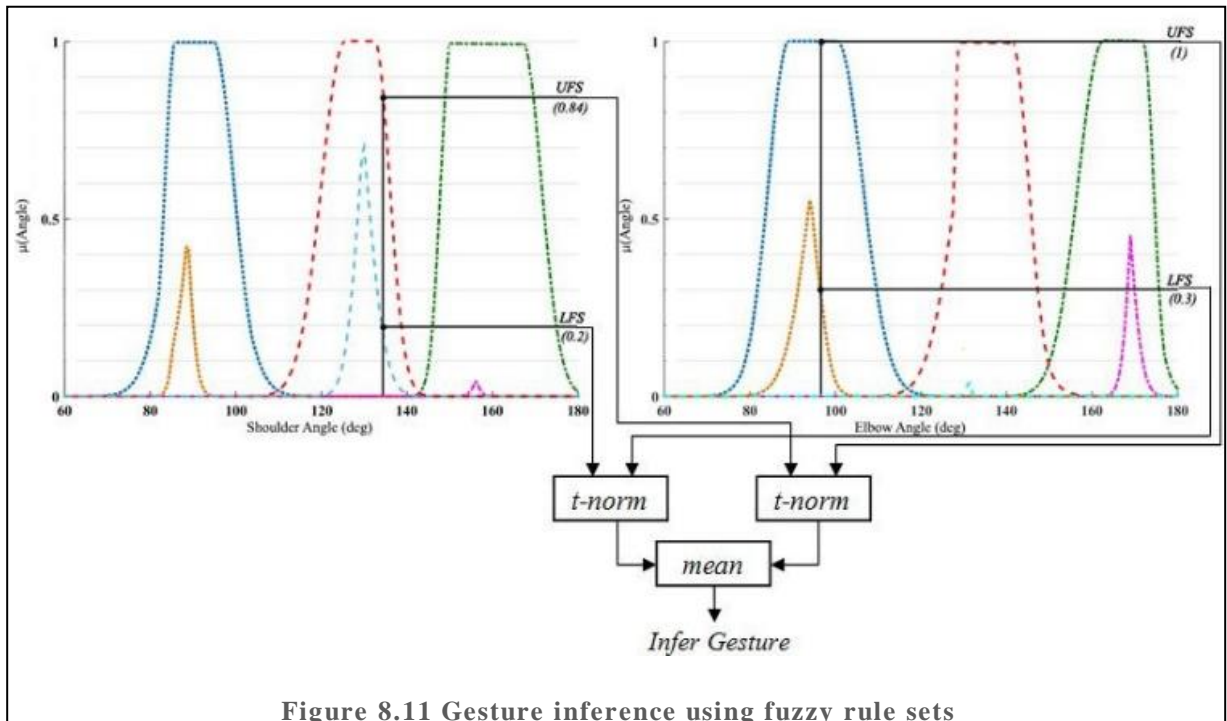
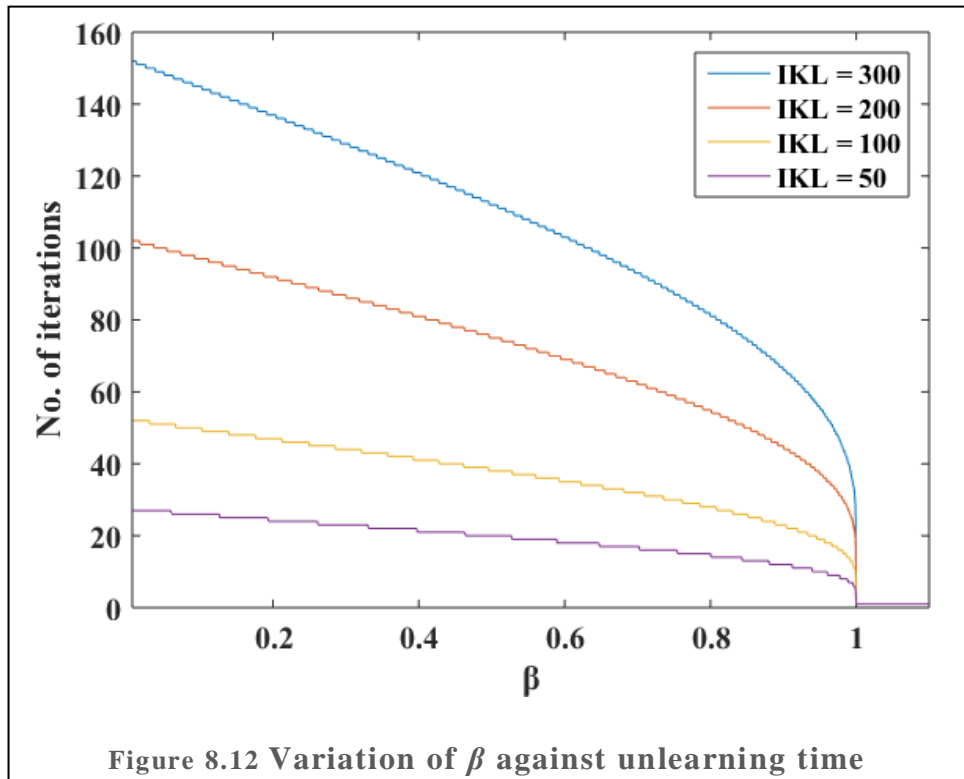


Figure 8.11 Gesture inference using fuzzy rule sets

The ‘and’ clause of the production rule as specified in section 8.2.4 is implemented by taking t-norm’s on the upper and lower firing strengths of both angles and then taking a mean of the two. This enforces the fuzzy-binary relationship between the input gesture and the output task execution.

8.3.4 Learning

The learning algorithm uses a parameter β to control the learning rate. Higher values of β guarantee faster learning, and helps in faster un-learning as well. This means the system can implement non-monotonicity in the learning process, and can add to its existing knowledge efficiently. Fig. 8.12 shows the number of iterations required to learn a new knowledge that invalidates a previous knowledge. It is seen that as β approaches 1, the number of iterations required to unlearn a previously learnt fact and learn a new fact decreases. For each curve, the initial knowledge level (IKL) is different and is kept fixed. The variation of number of iterations against β is shown for four different values of IKL.



8.4 References

- [1] M. A. Livingston, J. Sebastian, Z. Ai, and J. W. Decker, “Performance Measurements of the Microsoft Kinect Skeleton,” Virtual Reality Short Papers and Posters, 2012.
- [2] J.M. Mendel, R.I. John and F. Liu, “Interval Type-2 Fuzzy Logic Systems Made Simple,” IEEE Transactions on Fuzzy Systems, vol. 14, No. 6, December 2006.
- [3] Jin X., Han J., “K-Medoids Clustering,” Encyclopedia of Machine Learning, Springer, 2011.
- [4] D. Arthur and S. Vassilvitskii, “k-means++: The advantages of careful seeding,” Proceedings, ACM-SIAM symposium on Discrete algorithms, pp. 1027–1035, 2007.

Conclusions

Automation is the need of the hour for the want of efficiency, accuracy and uniformity. However, naturalization of its implementation, so as to make human-robot interaction similar to human-human interaction, is still in its nascent stage. This thesis proposes a novel work on recognition of imprecise gestures, and using them for complex task execution. Over 20,000 gesture data from 10 subjects were used to verify the capability of the gesture recognition scheme presented. Errors due to robot imitation of human demonstration were found to be minimal and thus can be ignored. A sequence of three hand positions was considered to constitute a gesture, and the proposed fuzzy-binary rules were found to be able to identify unknown gestures accurately. Finally, the non-monotonic adaptive learning algorithm put forward is found to work effectively where its rate of convergence is dependent only on the learning parameter β .

Future Directions

From the advent of computers to now, the digital world and its importance to the physical world has grown steadily. The digital world of mobile phones, tablets, cameras, computers, along with intelligent devices such as robots and drones are an essential part of human life now. That is why, it is imperative that efforts are made to bring the digital or virtual world closer to the physical world, so as to make their interaction more seamless, more lifelike. This thesis proposes one idea of making human-robot interaction more natural. This application can be further explored by considering even more joints, joint angles, complex dynamic gestures, overlapping gesture patterns, more complex tasks. Other learning algorithms and robots can be used to compare and contrast the variation of performance due to those factors.

Appendix

Guide to using the system

The codes of this work has been included in the CD accompanying this thesis (See inside the back cover). The following is a step by step guide on how to use the system.

1. Pre-requisites:

The work was done in a computer set up with:

- Microsoft Visual Studio 2010
- MATLAB 2015a

A Jaco robot arm manufactured by Kinova and a Microsoft Kinect Sensor is needed. One will also need the Kinova libraries (included in the CD) and Microsoft Kinect SDK (available open-source)

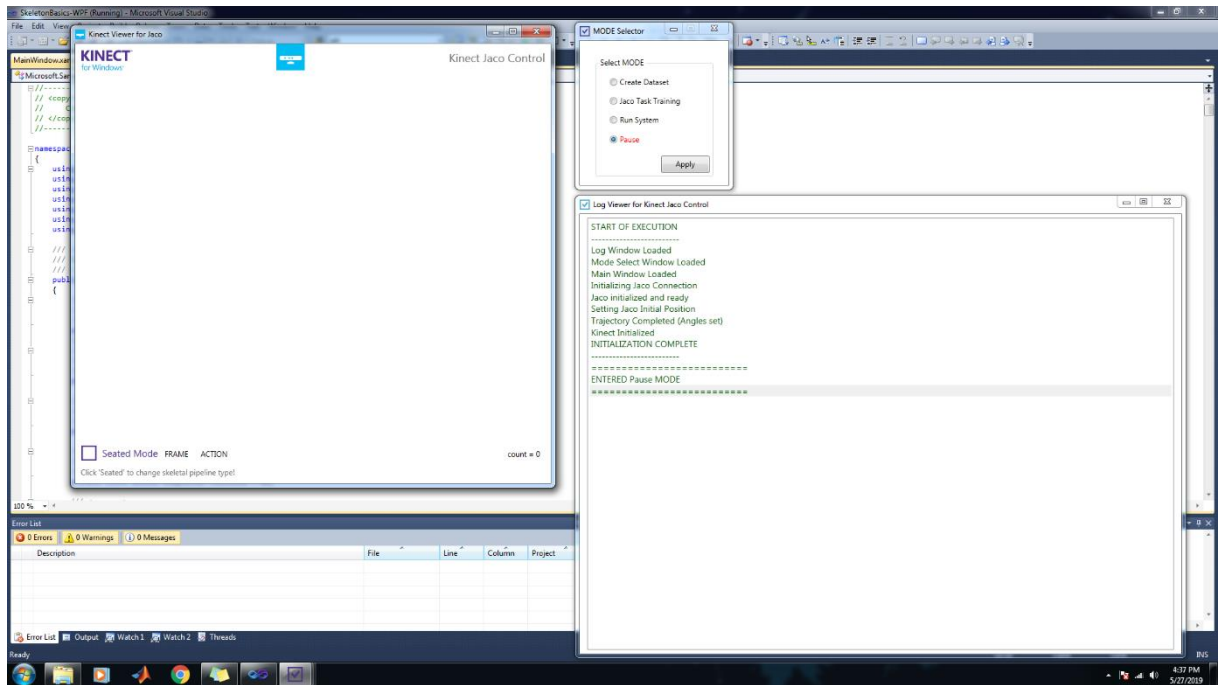
2. Setting up the project:

As the project build is provided in full, there is no need to set up a new project or change any settings. Simply open up the C# solution file and you are good to go.

The only thing needed to be taken care of is the paths. In the .cs files find the places where a path is specified. It will look something like, “D:/Kaustuv/..” and replace it with the path of the solution folder. After that build and run the solution from MainWindow.cs

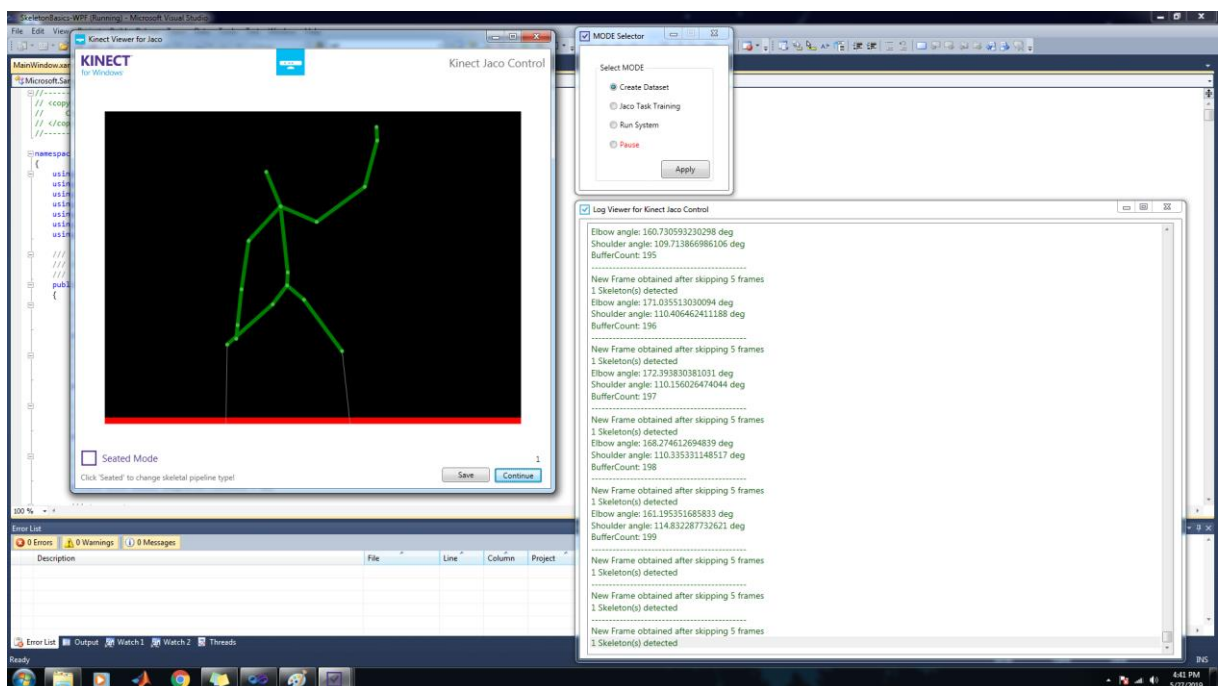
3. Home Screen:

This is what it would look like once you run the project:



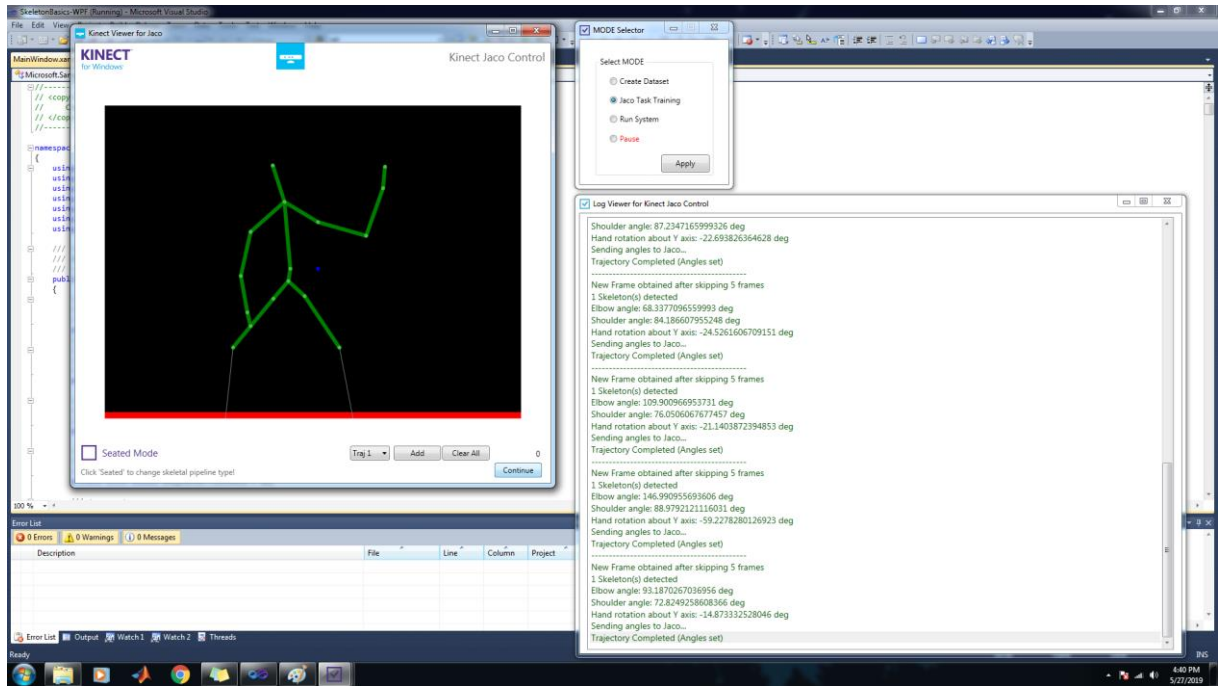
In this state the system is booted up and all the checks regarding connections are completed. However it is in pause mode.

4. Create dataset mode:



In this mode the system captures 200 gestures of the subject sitting in front of the Kinect and store it in a file as named in the code.

5. Task training mode:



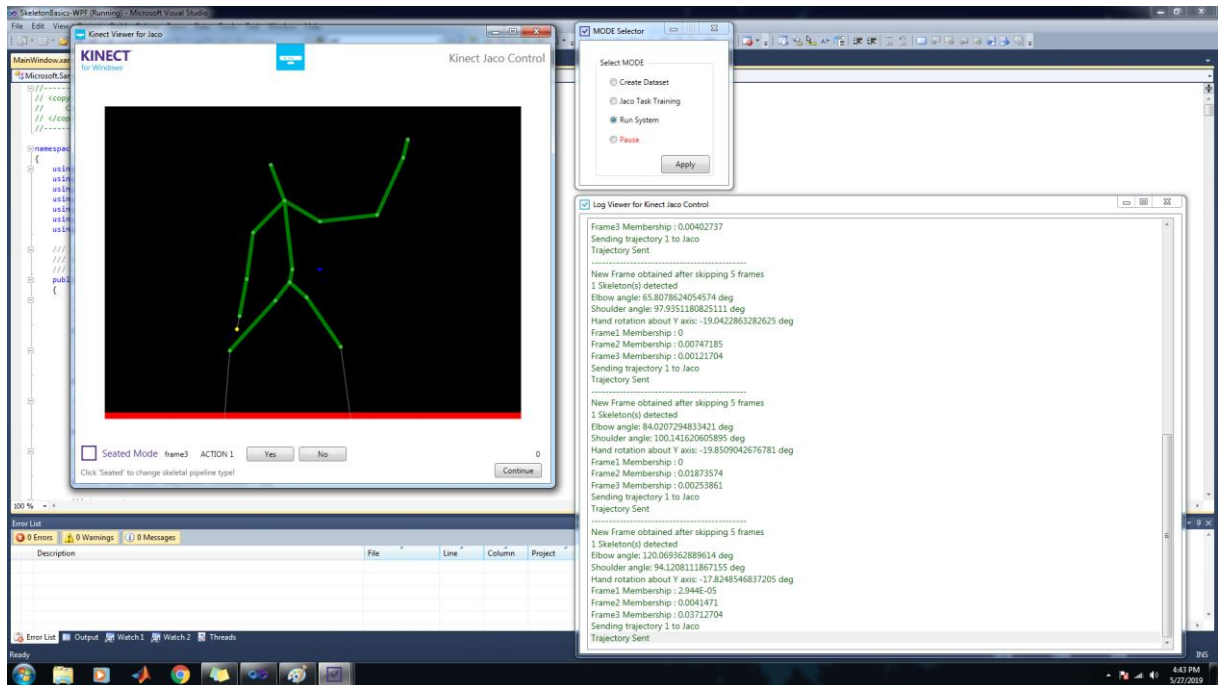
In this mode the Jaco arm follows the user's right arm and the system will store each movement pattern as a trajectory when pressed the "add" button corresponding to the desired number.

6. MATLAB code:

Before the system is ready to run, the MATLAB code needs to be executed to form the clusters and memberships.

The paths need to be specified correctly again. The input to the MATLAB code would be the stored gesture patterns for the different subjects, and the output will be written to the file specified.

7. Run System:



Specify the input path as the output from the MATLAB code, and then run the system. For each unknown gesture from the subject in front of the Kinect device, the system predicts a known gesture and according to the formulations executes a task. If the user believes that the task (termed “ACTION” in the GUI) is correct, “Yes” is pressed or else “No” is pressed. Over time the system will learn correct gesture-action pairs and become accurate.