

## Innovation Centre for Education

# Algorithm for Intelligent Systems and Robotics



## Student Guide

Course code GAI06SG190v1.0

## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

Approach®

HACMP™

Insight®

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

VMware is a registered trademark or trademark of VMware, Inc. or its subsidiaries in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

### December 2019 edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

# Contents

Trademarks .....	ix
Course description .....	xI
<b>Unit 1. System Modeling .....</b>	<b>1-1</b>
Unit objectives .....	1-2
Introduction (1 of 2) .....	1-3
Introduction (2 of 2) .....	1-5
Biological and cognitive paradigms for robot design (1 of 8) .....	1-7
Biological and cognitive paradigms for robot design (2 of 8) .....	1-9
Biological and cognitive paradigms for robot design (3 of 8) .....	1-11
Biological and cognitive paradigms for robot design (4 of 8) .....	1-13
Biological and cognitive paradigms for robot design (5 of 8) .....	1-15
Biological and cognitive paradigms for robot design (6 of 8) .....	1-17
Biological and cognitive paradigms for robot design (7 of 8) .....	1-19
Biological and cognitive paradigms for robot design (8 of 8) .....	1-21
Declarative-Procedural-Reflexive hierarchy for decision making and control .....	1-22
Articulated robots (1 of 2) .....	1-24
Articulated robots (2 of 2) .....	1-26
Joint-Link (Denavit-Hartenberg) transformations (1 of 4) .....	1-28
Joint-Link (Denavit-Hartenberg) transformations (2 of 4) .....	1-29
Joint-Link (Denavit-Hartenberg) transformations (3 of 4) .....	1-31
Joint-Link (Denavit-Hartenberg) transformations (4 of 4) .....	1-32
Mobile ground robots .....	1-33
Uninhabited ground robots (1 of 2) .....	1-35
Uninhabited ground robots (2 of 2) .....	1-37
Intelligent agents (1 of 4) .....	1-39
Intelligent agents (2 of 4) .....	1-40
Intelligent agents (3 of 4) .....	1-42
Intelligent agents (4 of 4) .....	1-43
Open-loop and closed-loop systems (1 of 4) .....	1-44
Open-loop and closed-loop systems (2 of 4) .....	1-45
Open-loop and closed-loop systems (3 of 4) .....	1-46
Open-loop and closed-loop systems (4 of 4) .....	1-47
Checkpoint (1 of 2) .....	1-49
Checkpoint (2 of 2) .....	1-50
Question bank .....	1-51
Unit summary .....	1-52
<b>Unit 2. Artificial Intelligence for Robotics Engineering .....</b>	<b>2-1</b>
Unit objectives .....	2-2
Artificial intelligence .....	2-3
Well-known definitions for AI .....	2-5
Birth of AI .....	2-6
Basic terminologies .....	2-7
Applications of AI .....	2-9
The AI problems and techniques .....	2-11
Real world problems .....	2-13

State space search . . . . .	2-1
Explicit vs. Implicit state space . . . . .	2-1
State space search notations . . . . .	2-1
Search problem and problem space . . . . .	2-1
Representation of search problems . . . . .	2-1
State space search example . . . . .	2-1
Pegs and disks problem (1 of 4) . . . . .	2-1
Pegs and disks problem (2 of 4) . . . . .	2-1
Pegs and disks problem (3 of 4) . . . . .	2-1
Pegs and disks problem (4 of 4) . . . . .	2-1
8 queen's problem . . . . .	2-1
N queens problem formulation (1 of 3) . . . . .	2-1
N queens problem formulation (2 of 3) . . . . .	2-1
N queens problem formulation (3 of 3) . . . . .	2-1
8 puzzle problem . . . . .	2-1
State space search example: Tic-tac-toe problem . . . . .	2-1
Production systems (1 of 2) . . . . .	2-1
Production systems (2 of 2) . . . . .	2-1
Commutative production system . . . . .	2-1
Problem characteristics . . . . .	2-1
Search paradigm . . . . .	2-1
Classification of search algorithms . . . . .	2-1
General terminologies (1 of 3) . . . . .	2-1
General terminologies (2 of 3) . . . . .	2-1
General terminologies (3 of 3) . . . . .	2-1
Uninformed search algorithms . . . . .	2-1
Breadth first search . . . . .	2-1
BFS illustration (1 of 3) . . . . .	2-1
BFS illustration (2 of 3) . . . . .	2-1
BFS illustration (3 of 3) . . . . .	2-1
Depth first search . . . . .	2-1
DFS illustrations (1 of 3) . . . . .	2-1
DFS illustrations (2 of 3) . . . . .	2-1
DFS illustrations (3 of 3) . . . . .	2-1
Depth Limited Search (DLS) . . . . .	2-1
Depth first iterative deepening search and bi-directional search . . . . .	2-1
Comparing the uninformed search algorithms . . . . .	2-1
Informed search algorithms . . . . .	2-1
Heuristic search techniques . . . . .	2-1
Generate-and-test . . . . .	2-1
Hill climbing . . . . .	2-1
Algorithm for simple hill climbing . . . . .	2-1
Best first search . . . . .	2-1
Knowledge representation (1 of 2) . . . . .	2-1
Knowledge representation (2 of 2) . . . . .	2-1
Knowledge representation languages . . . . .	2-1
Framework for knowledge representation . . . . .	2-1
Knowledge representation schemes . . . . .	2-1
Properties and schemes for knowledge representation . . . . .	2-1
Relational based knowledge representation scheme . . . . .	2-1
Inheritable knowledge representation scheme . . . . .	2-1

Inferential knowledge representation scheme .....	2-88
Declarative/procedural knowledge .....	2-89
Planning (1 of 2) .....	2-91
Planning (2 of 2) .....	2-93
Representation of states, goals and actions .....	2-95
Goal stack planning (1 of 2) .....	2-97
Goal stack planning (2 of 2) .....	2-99
Checkpoint (1 of 2) .....	2-103
Checkpoint (2 of 2) .....	2-104
Question bank .....	2-105
Unit summary .....	2-106
<b>Unit 3. Components of an Intelligent Robotic System .....</b>	<b>3-1</b>
Unit objectives .....	3-2
Introduction to robotics .....	3-3
Types of robots .....	3-5
Classification of robots .....	3-7
Components of robot (1 of 4) .....	3-9
Components of robot (2 of 4) .....	3-10
Components of robot (3 of 4) .....	3-11
Components of robot (4 of 4) .....	3-12
Manipulation arms .....	3-13
Merits and demerits of robot types with different geometries .....	3-14
Wrists .....	3-15
Robot kinematics (1 of 3) .....	3-16
Robot kinematics (2 of 3) .....	3-18
Robot kinematics (3 of 3) .....	3-19
Homogenous transformation modelling convention (1 of 2) .....	3-20
Homogenous transformation modelling convention (2 of 2) .....	3-21
Example of forward kinematics (1 of 4) .....	3-22
Example of forward kinematics (2 of 4) .....	3-23
Example of forward kinematics (3 of 4) .....	3-24
Example of forward kinematics (4 of 4) .....	3-25
Inverse kinematics (1 of 6) .....	3-26
Inverse kinematics (2 of 6) .....	3-27
Inverse kinematics (3 of 6) .....	3-28
Inverse kinematics (4 of 6) .....	3-29
Inverse kinematics (5 of 6) .....	3-30
Inverse kinematics (6 of 6) .....	3-31
Algebraic solution approach: Example (1 of 6) .....	3-32
Algebraic solution approach: Example (2 of 6) .....	3-33
Algebraic solution approach: Example (3 of 6) .....	3-34
Algebraic solution approach: Example (4 of 6) .....	3-35
Algebraic solution approach: Example (5 of 6) .....	3-36
Algebraic solution approach: Example (6 of 6) .....	3-37
Advanced robotics (1 of 5) .....	3-38
Advanced robotics (2 of 5) .....	3-40
Advanced robotics (3 of 5) .....	3-41
Advanced robotics (4 of 5) .....	3-42
Advanced robotics (5 of 5) .....	3-43
Machine intelligence: Architectures, controllers and applications .....	3-44

Architectures for intelligent control (1 of 2)	3-1
Architectures for intelligent control (2 of 2)	3-2
Machine learning	3-3
Machine learning: Rule-based control (1 of 3)	3-3
Machine learning: Rule-based control (2 of 3)	3-4
Machine learning: Rule-based control (3 of 3)	3-5
Machine learning: Machine learned control	3-6
Machine learning: Reinforcement learning	3-7
Advanced control systems for robotic arms	3-8
Kinematic and dynamic control	3-9
Intelligent gripping systems	3-10
Overview of the Salford theories (1 of 2)	3-11
Overview of the Salford theories (2 of 2)	3-12
Need and provision of fingertip sensor system	3-13
Computer software package implementation (1 of 2)	3-14
Computer software package implementation (2 of 2)	3-15
Force feedback control in robots and its application to decommissioning	3-16
Force feedback strategies	3-17
Introduction to mobile robots	3-18
Environment capturing with common sensors	3-19
CCD cameras (1 of 2)	3-20
CCD cameras (2 of 2)	3-21
CCD Vs. CMOS	3-22
Sonar sensors (1 of 2)	3-23
Sonar sensors (2 of 2)	3-24
Optoelectronic sensors	3-25
Sensor integration	3-26
Qualitative approaches (1 of 2)	3-27
Qualitative approaches (2 of 2)	3-28
Quantitative approaches	3-29
Bayes statistics	3-30
Kalman filter	3-31
Machine vision system	3-32
Phases of a machine vision system (1 of 2)	3-33
Phases of a machine vision system (2 of 2)	3-34
Tool condition monitoring systems	3-35
Neural networks for tool condition monitoring systems	3-36
Basic understanding of neural networks	3-37
Representational power of perceptrons	3-38
Architecture of neural networks	3-39
Single-layer feed-forward architecture	3-40
Multiple-layer feed-forward architecture	3-41
Recurrent or feedback architecture	3-42
Mesh architecture	3-43
The perceptron training rule	3-44
Gradient descent and the delta rule	3-45
Gradient descent algorithm	3-46
Stochastic approximation to gradient descent (1 of 2)	3-47
Stochastic approximation to gradient descent (2 of 2)	3-48
Multilayer networks and back-propagation algorithm	3-49
The back-propagation algorithm	3-50

Multiple principal component fuzzy neural networks .....	3-119
Fuzzy classification and uncertainties in tool condition monitoring .....	3-121
Checkpoint (1 of 2) .....	3-123
Checkpoint (2 of 2) .....	3-124
Question bank .....	3-125
Unit summary .....	3-126
<b>Unit 4. Robot Operating System (ROS) .....</b>	<b>4-1</b>
Unit objectives .....	4-2
Real and simulated robots .....	4-3
Robot Operating System (ROS) .....	4-5
ROS basics and architecture .....	4-6
The File system level .....	4-8
Files and folders in a sample package of ROS .....	4-10
ROS packages .....	4-11
ROSbash .....	4-12
package.xml .....	4-13
ROS messages .....	4-14
ROS services .....	4-15
The computational graph level (1 of 2) .....	4-16
The computational graph level (2 of 2) .....	4-18
The community level .....	4-20
Debugging and visualization (1 of 4) .....	4-21
Debugging and visualization (2 of 4) .....	4-22
Debugging and visualization (3 of 4) .....	4-24
Debugging and visualization (4 of 4) .....	4-25
Using sensors and actuators (1 of 3) .....	4-26
Using sensors and actuators (2 of 3) .....	4-28
Using sensors and actuators (3 of 3) .....	4-30
3D modeling and simulation (1 of 2) .....	4-32
3D modeling and simulation (2 of 2) .....	4-34
Computer vision .....	4-36
Checkpoint (1 of 2) .....	4-38
Checkpoint (2 of 2) .....	4-39
Question bank .....	4-40
Unit summary .....	4-41
<b>Unit 5. Navigation, SLAM and Speech Recognition and Synthesis .....</b>	<b>5-1</b>
Unit objectives .....	5-2
Navigation (1 of 3) .....	5-3
Navigation (2 of 3) .....	5-5
Navigation (3 of 3) .....	5-6
Simultaneous localization and mapping .....	5-9
Setting up rviz for navigation stack .....	5-11
Adaptive Monte Carlo Localization .....	5-13
Avoiding obstacles .....	5-14
Speech recognition and synthesis .....	5-15
Checkpoint (1 of 2) .....	5-17
Checkpoint (2 of 2) .....	5-18
Question bank .....	5-19
Unit summary .....	5-20

# Course description

## Algorithms for Intelligent Systems and Robotics

### Purpose

This course is designed to explain the concepts of system modelling and principles of control systems. This course embeds the concept of artificial intelligence in robotics. This course helps to gain knowledge of various components of intelligent robotic system. The course introduces the concept of Robot Operating System (ROS), navigation, speech recognition and synthesis.

### Audience

B.Tech in computer science engineering.

### Prerequisites

Basic programming skills, basic knowledge on artificial intelligence.

### Objectives

After completing this course, you should be able to:

- Gain knowledge on the process of system design
- Gain an insight into the AI problems and techniques
- Learn about the kinematic and dynamic control concept with a focus on intelligent gripping systems
- Understand the operating system concepts for robotics
- Understand the concept of speech recognition and synthesis and implement it

### References

- Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig, c 1995 Prentice-Hall, Inc.
- An Introduction to Robotics, Dr. Bob Williams, williar4@ohio.edu, Mechanical Engineering, Ohio University, EE/ME 4290/5290 Mechanics and Control of Robotic Manipulators, © 2019 Dr. Bob Productions
- Toward Intelligent Flight Control by Robert F. Stengel, Fellow, IEEE Transactions on Systems, Man and Cybernetics Vol 23, No 6, Nov/Dec 1993
- Designing the mind of a social robot by Nicole Lazzeri, Daniele Mazzei, Lorenzo Cominelli, Antonio Cisternino and Danilo Emilio De Rossi, Appl. Sci. 2018, 8, 302; doi:10.3390/app8020302
- <https://robots.ieee.org/learn/types-of-robots/>
- <https://docs.fetchrobotics.com/gazebo.html>
- <https://www.pirobot.org/blog/001https://robots.ieee.org/learn/types-of-robots/>
- <https://docs.fetchrobotics.com/gazebo.html>
- <https://www.pirobot.org/blog/0014/4/>

# **Unit 1. System Modeling**

## **What this unit is about**

This unit aims at gaining knowledge on system modeling and how it has evolved over the years. This unit focuses on how various AI concepts are being used for developing systems, which mimic biological processes and cognitive processes for achieving desired goals.

## **What you should be able to do**

After completing this unit, you should be able to:

- Understand the concept of system modeling
- Gain knowledge on the process of system design
- Understand the goals and principles of intelligent systems
- Gain an insight into various types of robots as systems

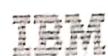
## **How you will check your progress**

- Checkpoint

## **References**

IBM Knowledge Center

## Unit objectives



IBM ICE (Innovation Centre for Education)

After completing this unit, you should be able to:

- Understand the concept of system modeling
- Gain knowledge on the process of system design
- Understand the goals and principles of intelligent systems
- Gain an insight into various types of robots as systems

---

Figure 1-1. Unit objectives

AIR011.0

### Notes:

Unit objectives are as stated above.

# Introduction (1 of 2)

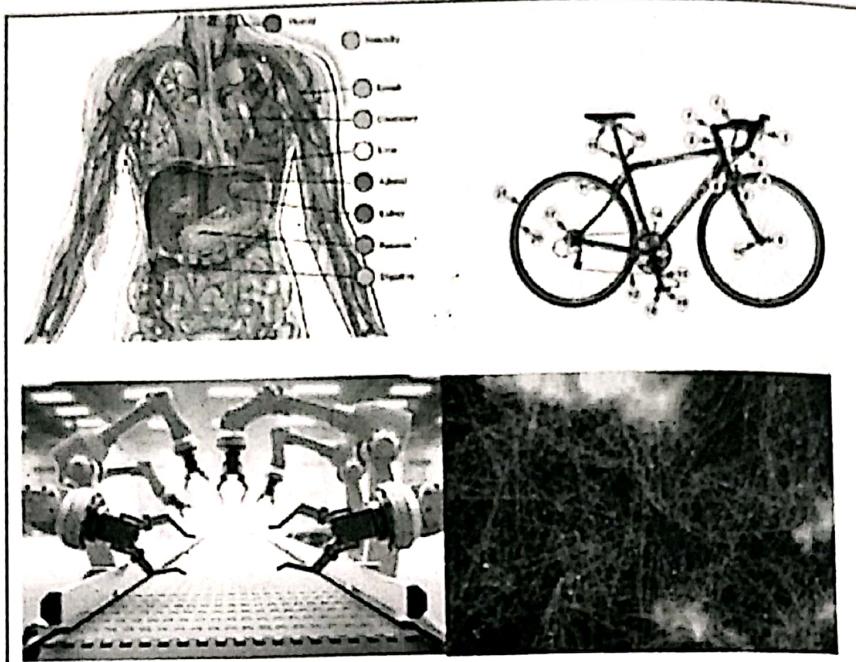


Figure: Systems found in the real world

Source: <http://totalhumanbodysystems.blogspot.com/2012/12/summary-of-effects-therapeutic-effect.html>  
[http://www.notesandsketches.co.uk/Mechanical\\_systems.html](http://www.notesandsketches.co.uk/Mechanical_systems.html)

<https://www.robotics.org/blog-article.cfm/Pick-and-Place-Robots-What-Are-They-Used-For-and-How-Do-They-Benefit-Manufacturers/>  
<https://siliconangle.com/2018/11/01/graph-databases-get-boost-market-leader-neo4j-raises-80m/>

Figure 1-2. Introduction (1 of 2)

AIR0110

## Notes:

The figures in the above slide consist of four systems out of many systems generally found in the world. A system in general is a collection of components or elements that are organized for a common goal or purpose. In other words, it is an organized, purposeful structure that consists of interrelated and interdependent components. These components or elements directly or indirectly influence one another in order to achieve the goal of the system. We see four systems namely:

- Human Body system
- A Mechanical system
- A Pick and place robot
- A Graph database system

All of these systems have:

- Inputs and outputs
- Feedback mechanisms
- Have an internal steady state
- Display characteristic properties as a whole which are different from the properties of the individual component
- Well defined boundaries as required by the users/designers.

Systems could be a part of a larger systems, sometimes termed as subsystems. The system might stop functioning or may show a different behavior when a component is changed or removed. Such systems allow us to understand and interpret the universe as a meta-system comprising of many interlinked systems. Although these four systems look very different, they have great deal of similarity. At the most basic level, the systems can be classified broadly into two categories:

**Closed systems:** Mostly theoretical systems.

Such systems don't interact with the environment and are thus not affected/influenced by its surroundings. In such systems only the components within the system are of significance. For example: a sealed bottle – nothing can enter or exit the bottle but whatever is inside can interact.

**Open Systems:** Mostly real-world systems. Such systems interact with the environment and the boundaries of such systems allow exchange of energy, materials and information, For example: a Locomotive engine: lot of interaction between the system and the real world.

The design process for any system, involves the following stages:

**Needs statement:** The design process begins with a needs statement from the client or the customer. This may be identified by various methods to arrive at the needs of potential users.

**Analysis of the Needs statement:** The first stage in developing a design is to find out the nature of the problem by analyzing it. This is the most important stage and not defining the problem accurately will lead to wasted time.

**Requirements specifications:** Requirements specification is a document that can be prepared from the analysis. This document states the problem, constraints on the solution and the criteria's to be used to judge the quality of the design.

**Generation of possible solutions:** Outline solutions are prepared which are worked out in sufficient details so as to capture the present state of the art, proposed approach(s) to obtain the solution, along with the cost analysis.

**Production of a detailed design:** In this phase the detail of the selected design has to be worked out. This generally requires the production of the prototypes or mock-ups in order to determine the optimal details of the design.

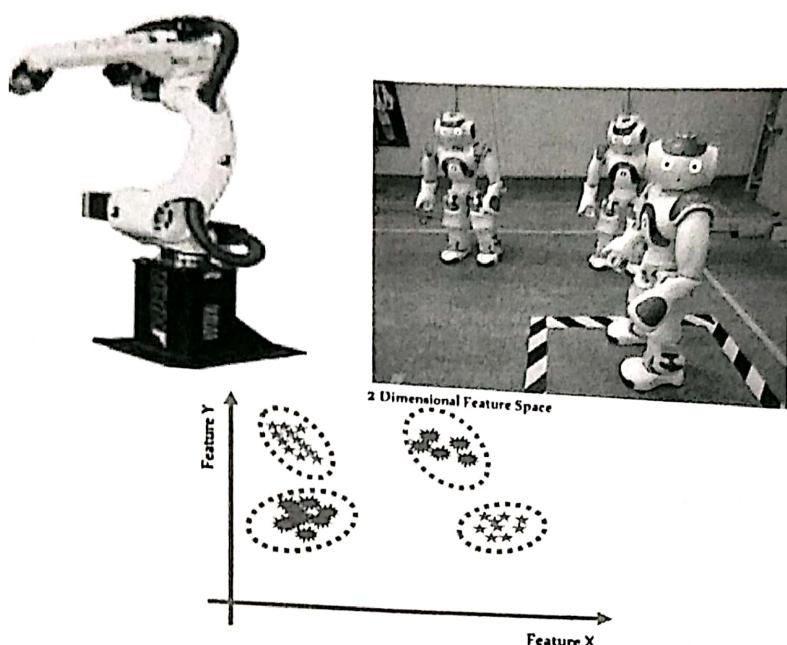
**Production of working drawings:** The selected design is then translated into working drawings circuit diagrams, flowcharts, etc. so that the product can be developed or made.

It is important to note that the design process may not follow a waterfall model which flows on stage by stage but may require an iterative process of going back to various stages during the solution development.

## Introduction (2 of 2)



IBM ICE (Innovation Centre for Education)



**Figure: Intelligent Systems**

Source: <https://robohub.org/why-ethical-robots-might-not-be-such-a-good-idea-after-all/>  
 Copyright 2017 by Robert Stengel. All rights reserved. For educational use only. <http://www.princeton.edu/~stengel/MAE345.html>  
<https://www.petersincak.com/news/why-i-do-not-believe-in-error-backpropagation/>

Figure 1-3. Introduction (2 of 2)

AIR011

### Notes:

Figure shows the various concepts required for developing an intelligent system and robots developed as a result of these concepts. Intelligent systems are those systems that perform useful important functions to achieve desired goals. These desired goals could be reached by taking into account the current state/knowledge of the system. Such systems emulate biological and cognitive processes to process the information to achieve the set objectives or goals.

Since such systems tend to emulate various biological processes, they are designed to learn from experience and different examples. There are various methods by which such systems could be made to learn, which we will cover in subsequent chapters. In machine learning terminology this is synonymous with "training". One of the important characteristic features of intelligent systems is adaptability to a real changing dynamic environment.

In the past fifty years there has been lot of research in the area of Artificial Intelligence (AI) and there has been lot of research going on presently as to how these can be applied to make intelligent systems and machines. Researchers in AI belong to two broad categories: The ones who believe in building systems with human like intelligence (Strong AI) and the others who believe that the machines can only exhibit intelligent behavior and there would always be limits to what a machine can achieve, and thus would be inferior to human beings.

Strong AI comes with its potential benefits and concerns. Some researchers fear that if strong AI becomes a reality, as it is being perceived, AI may become more intelligent than humans leading to singularity. The idea of singularity predicts that the Strong AI will be so intelligent that it can alter itself to pursue its own goals without human intervention harming humans (as shown in the movie I, robot).

Is it possible to program strong AI with moral values to prevent such outcomes? Research into such issues on ethical and moral dilemmas could help prevent future robots to turn against us or to decide if such systems could ever be modeled.

Weak AI on the other hand is able to simulate human consciousness. John Searle's Chinese room thought experiment nicely illustrated the concept of weak AI. This experiment showed that a person outside a room might be able to have a conversation in Chinese with a person inside a room who is instructed on how to respond to conversations in Chinese. The person inside the room appears to be speaking Chinese, but in reality, he couldn't actually speak or understand a word of it if the instructions are not being fed. That's because the person is good at following instructions. The person inside the room might appear to be having a Strong AI – but on the contrary he only has a Weak AI.

Weak AI systems have specific intelligence and they do not have general intelligence. An AI system that is capable of telling you how to drive from point A to point B would be incapable of playing a game of chess.

Weak AI helps turn big data into usable information by detecting patterns for making predictions. Examples include:

- Apple's Siri, and Google Assistant – Technology that answers user's spoken questions.
- Facebook's News feed – based on user's interest.

**Email Spam Filter:** Filters the emails based on previous emails and preferences: Here the mailbox uses an algorithm to learn which messages are likely to be spam, then redirects them from inbox to a spam mail folder. Learning in intelligent systems is through the knowledge generated, which is a result of the data analyzed and the insights, obtained from the analysis.

Thus, Learning: Data + Insight -> Knowledge

**Biomimetics:** Definition of biomimetics from the Merriam-Webster's dictionary is "the study of the formation, structure, or function of biologically produced substances and materials (such as enzymes or silk) and biological mechanisms and processes (such as protein synthesis or photosynthesis) especially for the purpose of synthesizing similar products by artificial mechanisms, which mimic natural ones". This is also used for designing and developing intelligent systems, which involve robots.

## Biological and cognitive paradigms for robot design (1 of 8)



Figure: Robots in real world

Source: <https://interestingengineering.com/the-history-of-robots-from-the-400-bo-archytas-to-the-boston-dynamics-robot-dog>

Figure 1-4. Biological and cognitive paradigms for robot design (1 of 8)

Alt0110

### Notes:

Karel Čapek coined the term "Robot" in his play R.U.R (Rossum's Universal Robots) whose first performance was held on January 25<sup>th</sup> 1921 in Prague. In this play an inventor by name Rossum creates a set of workers from biological parts to replace humans in any job. Čapek described these workers as robots (derived from Czech word Robota). Three Science fiction movies Metropolis, The Day the Earth Stood Still and Forbidden Planet; shifted the focus to robots as mechanical in origin unlike in Čapek's play. Isaac Asimov, a science fiction writer and a professor in his collections I, Robot (1950's) introduced Three Laws of robotics:

- **First Law:** A robot may not injure a human being or through inaction allow a human being to come to harm.
- **Second Law:** A robot must obey orders given it by human beings except where such orders would conflict with the First Law.
- **Third Law:** A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Gradually with the computers gaining prominence during the industrial revolution and after industrial automation became the order of the day, robotic arms were installed in various industries for assembling parts. This shift from human like mechanical creatures to various shapes is due to reality and design required to complete a specific task.

For example:

- A robot that delivers meals to the patients in the hospital looks like a cart not like a nurse.
- Robot vacuum cleaners look like vacuum cleaners not janitors.

An Intelligent Robot is a mechanical creature that can function autonomously. "Intelligent" here implies that the robot doesn't do things in a mindless repetitive way unlike factory automation. "Mechanical creature" acknowledges that the technology uses mechanical building blocks. Robot is not same as a computer though it might use a computer as a building block. "Function autonomously" indicates that robot can operate, self-contained, under all reasonable conditions without any human operators intervention. Autonomy also means that a robot can adapt to changes in its environment or itself and still continue to reach its goal.

This concept was nicely captured in the movie Terminator (1984). Even after losing one eye (camera), skin and flesh burnt off (external coverings), it continued to pursue its target.

Where robots can be used?

Robots can be used for any application that can be thought of. Generally robots are suited for applications where:

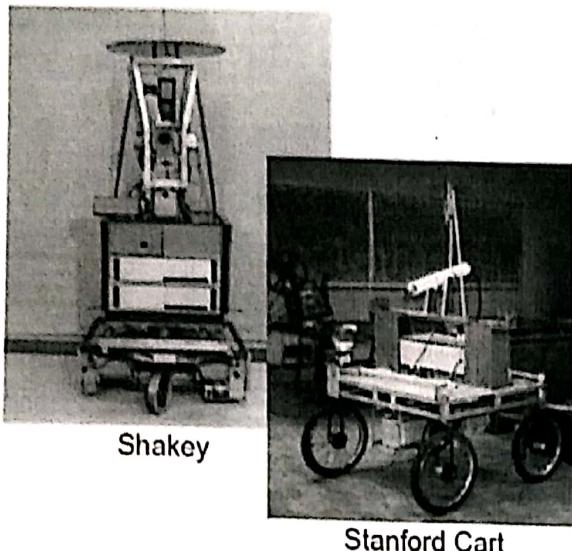
- The environment in which a persons work poses a significant risk like nuclear, space and military.
- Applications where workers can create inefficiencies because of economics or menial nature like agriculture and service industry.
- For humanitarian uses where risk is high like demining, rescue and search operations.

The following applications can be implemented if the Intelligence incorporated in the robots has the following essential abilities:

- The robots should respond flexibly to unforeseen situations. In a sense they should be able to derive sense of ambiguity or contradictions.
- The robots should be able to recognize relative importance of the information and knowledge.
- They should be able to find similarities and differences among things.
- They should be able to synthesize new ideas from old concepts.

# Biological and cognitive paradigms for robot design (2 of 8)

IBM ICE (Innovation Centre for Education)



IT WAS ALREADY TOO LATE WHEN JIMMY  
REALIZED HE HAD FORGOTTEN TO  
CONVERT HIS UNITS BACK INTO THE  
METRIC SYSTEM.

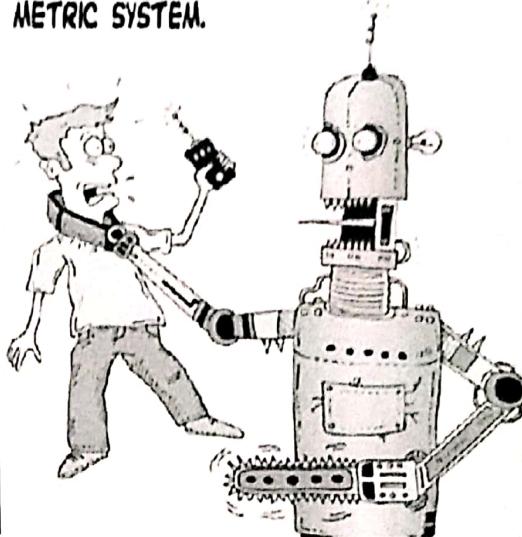


Figure: Various paradigms

Source: <https://www.cpp.edu/~ftang/courses/CS521/notes/hierarchical%20control.pdf>

Figure 1-5. Biological and cognitive paradigms for robot design (2 of 8)

AIR011.0

## Notes:

"A paradigm is a philosophy or set of assumptions and/or techniques which characterize and approach to a class of problems". It is both a way to look at the environment along with tools for solving problems. No single paradigm is always right. The same problem could be solved using different approaches. For example consider calculus problems. Differentiating in Cartesian coordinate's could solve some of these problems whereas it becomes easier to solve them if they are solved in polar coordinates. These coordinate systems represent two different paradigms for viewing and manipulating a problem. Both of these approaches produce correct results, but one takes less work for certain set of problems. Thus applying the right paradigm makes problem solving easier.

Before we discuss the methods and principle to develop the mind or intelligence of a robot, we should understand what do we mean by the word mind in context of the robots. Here the mind is a computational infrastructure that is designed to control a robot so that it can interpret and convey human readable cues during interaction.

This complex requirement to create cognitive architecture for robots require knowledge from fields as diverse as psychology, affective computing, computer science and AI. Psychology provides information on how people react/respond to stimuli, which helps in modeling the robot's behavior. Computer science helps in developing the system software that controls the behavior of the robot and its interaction with the environment and its surroundings. Affective computing focuses on interpreting the emotional states of the humans so that the robots can adopt their behavior and states according to them.

AI helps the researchers to develop algorithms and models to make the robots learn from human behaviors, to process information in the context of the present environment and to determine the action to be taken at that given moment. Apart from the cognitive paradigms, while developing robots we must ensure that we must adhere to the various biological models as per neurosciences. According to this the human intelligence is not a monolithic model. Humans generally perceive the external world not in isolation but in parallel with their internal states and multiple internal representations.

Developing pure rational reasoning is not sufficient for making decisions since human beings also show cognitive deficits without emotional capabilities. This bio-inspired paradigm has changed the AI from computational perspective of problem solving, search techniques and knowledge representations to understanding biological systems, abstracting principles of intelligent behavior for building intelligent systems.

From the technical viewpoint to implement the biological and cognitive architecture we need to have the following:

- A modular architecture that is distributed with multiple abstract and physical layers, with parallel processing and distributed computing capabilities.
- An imperative control architecture which is aimed at controlling motors, reading sensor data, calculating kinematics and signal processing.
- A hardware independent platform that can be easily adapted to various robotics platforms and consequently used in various setups.
- A high-level deliberative reasoning architecture to implement the robot's emotional and behavioral models.
- A pattern-matching engine which is able to conduct search and analysis procedures that are not easy to implement with just mathematical analyses.
- An easy-to-use and intuitive language that allows behavioral psychologists and neuroscientists to convert their theoretical models easily into executable scripts in the cognitive architecture.
- A perception system, which can extract empathic, emotional, and high-level parameters from the perceived scene/environment.
- An object-oriented system on which heterogeneous data types can be easily managed.

Over the years, many approaches/architectures have been used in AI to control robots. The three most common paradigms are: hierarchical, reactive and hybrid deliberate/reactive paradigm. All of them are defined by the relationship among the three primitives, i.e., SENSE, PLAN, and ACT, and by the processing of the sensory data by the system. Let's discuss all the three paradigms in the next few slides.

# Biological and cognitive paradigms for robot design (3 of 8)

IBM ICE (Innovation Centre for Education)

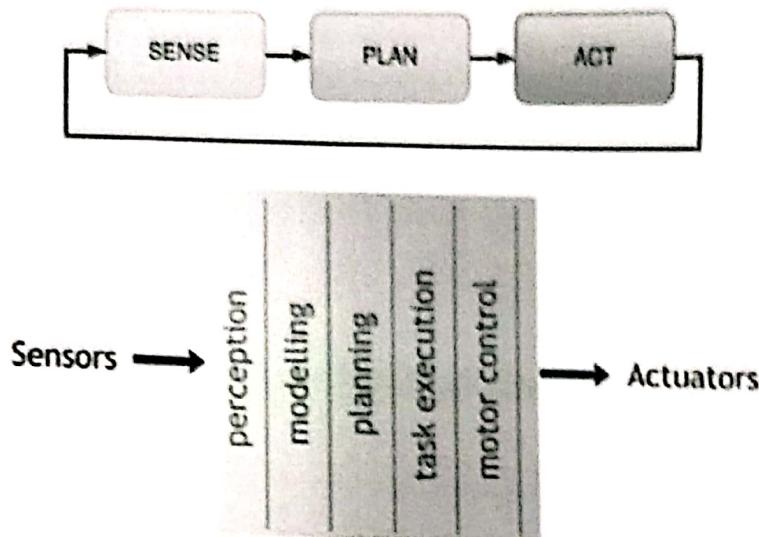


Figure: Hierarchical Paradigm

Source: Article: Designing the mind of a social robot by Nicole Lazzeri, Daniele Mazzei, Lorenzo Cominelli, Antonio Cisternino and Danilo Emilio De Rossi, Appl. Sci. 2018, 8, 302; doi:10.3390/app8020302

Figure 1-6. Biological and cognitive paradigms for robot design (3 of 8)

AIR0110

## Notes:

The most widely used implementation till 1980's for organizing the intelligence in robots. The robotic paradigm is generally defined by:

- The relationship between the three primitives namely Sense, Plan and Act.
- How the data from the sensors is processed and distributed.

As shown in the figure above the hierarchical paradigm as the name implies is orderly and sequential. These are the following steps:

- The robot senses its environment and constructs a global world map.
- The robot plans for all the action that might be needed to reach its goal.
- The robot acts to carry out the planned actions in the previous step.

Once the robot has carried out the Sense-Plan-Act step it again begins the same cycle of the above 3 steps iteratively till it reaches its goal. As shown in the figure the observation from all the sensors are coupled into a single global data structure and which represents the world model. This world model consists of:

- An a priori, representation of the environment the robot is operating in. For example the map of a building or the terrain.
- Information from the sensors. For example, the present position is in the room 103.
- Any other knowledge needed to complete a task. For example, all the cups need to be lifted.

Creating a monolithic single representation that can store all of this information can be a real challenge. Even with the increased computing speeds, hierarchical, logic based approach was not very satisfactory for navigational tasks which requires a fast response time.

Shakey the first AI robot as shown in figure needed a generalized algorithm for planning and to accomplish its goals. A general problem solver method called strips was developed. This uses means-end analysis approach. In this approach if the robot can't accomplish the task or reach the goal in one-step or movement, it picks an action, which will reduce the difference between the present state versus the goal state. This is inspired by the human cognitive behavior of trying to reach as close as to the complete solution in case the complete solution is not realizable. Strips use a recursive approach. This means that in a step if the goal is not reached, it identifies the sub goals and then tries to attain the final goal from iteratively attaining the sub goals.

Implementations of strips requires the designer to create a set up which has the following information:

- A model representation of the world.
- A difference table which has the operators, various preconditions, addition, and deletion lists.
- An evaluator to calculate the difference.

The steps in executing Strips are:

Step 1: The difference between the goal state and the initial state is calculated using the difference evaluator. If there is no difference and goal is reached, terminate.

Step 2: If there is a difference found, reduce the difference by selecting the appropriate operator from the difference table, which reduces the difference.

Step 3: If the goal is not achieved make a new sub goal and store the original goal by pushing it on a stack. Recursively reduce that difference by repeating step 2 and 3.

Step 4: When all preconditions are met, update a copy of the world model. All the steps need to be recursively carried out till the goal is reached.

The best known architectures of the Hierarchical paradigms are the Nested Hierarchical Controller (NHC) developed by Meystel and NIST Real-time Control System (RCS) developed by Albus.

# Biological and cognitive paradigms for robot design (4 of 8)

- Nested hierarchical controller

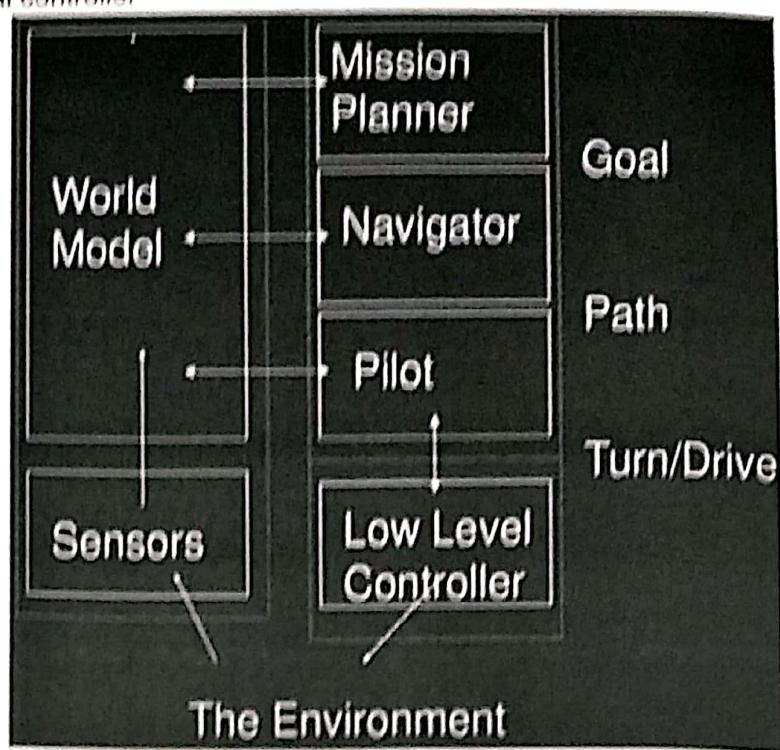


Figure: Nested hierarchical Controller

Source: CS4630: Intelligent Robotics and Perception, Planning (Chapter 2) slides by Ticker Balch

Figure 1-7. Biological and cognitive paradigms for robot design (4 of 8)

AIR011.0

## Notes:

The nested hierarchical controller has three components, which carry out the tasks for Sense, Plan and Act. The robot gathers information about its environment using the sensors and combines these to form a world model data structure. This step completes the Sense activity. The world model based on the requirement also contains the a priori knowledge about the world. Once the world model has been created, the robot can plan what actions it should take. Any plan to navigate consists of three steps:

- Planner.
- Navigator.
- Pilot.

Each of these three modules has access to the world model. The last step, the pilot module generates specific actions for the robot to do. For example turn right, move straight with a velocity of 0.7 m/s for 2 seconds and then turn left. These actions are translated into actual control signals by the low level controller. The low level controller along with the actuators together completes the Act portion in this architecture. Major contributions of the NHC were the concept of decomposition of the planning step into three subsystems aimed at providing navigation: the planner, the navigator and the pilot.

Some of the advantages of the NHC are:

- Unlike the strips, it interleaves planning and acting.
- The robot comes up with a plan, executes it, and then changes it if the world model is different than expected.
- The subsystems are inherently hierarchical in intelligence and the scope.
- Hierarchically the planner is smarter than the navigator, which is smarter than the pilot. That means that the planner is responsible for a higher-level abstraction than the pilot.

Some of the disadvantages of the NHC are:

- The planning function is appropriate for the navigational tasks only.
- The role of pilot to control end effectors is not very clear.
- Most of the NHC was implemented under simulation environments and cases of actual robots using it are not widely found in literature. This was because the hardware costs were huge during the period when hierarchical model was developed.

# Biological and cognitive paradigms for robot design (5 of 8)

IBM ICE (Innovation Centre for Education)

NIST Real time control system

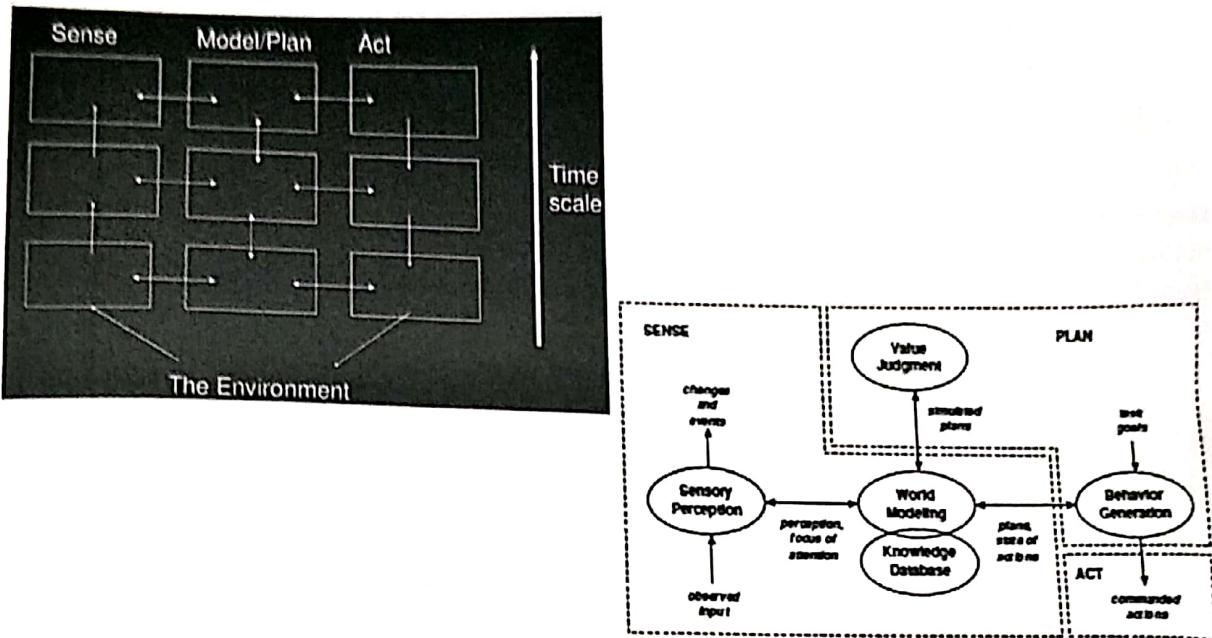


Figure: Real-time Control systems(RCS)

Source: CS4630: Intelligent Robotics and Perception; Planning (Chapter 2) slides by Ticker Balch  
Introduction to AI Robotics by Robin R Murphy, A Bradford Book, The MIT Press Cambridge, Massachusetts, London, England

Figure 1-8. Biological and cognitive paradigms for robot design (5 of 8)

AIR011.0

## Notes:

Jim Albus at the National Bureau of Standards (National Institute of Standards and Technology) understood the discontent between the industry and the manufacturers of the robots because of no common set of design standards and terms. This led to the development of a very detailed architecture called the Real-time Control Systems (RCS) as shown in figure. This served as a guide for the manufacturers who wanted to add intelligence to their robots. The concept of NHC was used during the development of RCS.

The sensing activities are grouped into the sensory perception module. The sensory inputs are passed to the world-modeling module, which constructs a global map using the knowledge base and any other domain knowledge. One of the steps, which this sensory perception module introduces, which was not present in the NHC, is between the sensor and the world modeling. This pre-processing step is often referred to as the feature extraction.

The value judgment and the behavior generation modules are associated with the "Plan" activity. The value judgment module plans, simulates and checks that is actually a working plan. Once it is sure it passes on the information to the behavior generation module, which converts the plan into actions on which the robot Acts. The behavior generation is a more generic module unlike the Pilot in NHC, which concentrated more on the navigational tasks. Many of the robots prototypes were built using the RCS approach. At that time RCS served as a blueprint for many of the robotic projects carried out in industries and universities.

Both NHC and RCS architectures are best suited for semi-autonomous controlled robots. This is because the human operator can easily provide the world map, decide the goal, decompose it into plan and then into actions, which the lower level controller along with robot will carry out. As the AI advanced the human intervention reduced and the task was to just tell the goal or the mission to be accomplished. This improved architecture is called NASREM and is still being used.

**Advantages of the Hierarchical paradigm:**

- Majority of the robots developed 2 decades ago, used this paradigm. This was because the robots developed were for specific applications rather than generic robots.
- It provides an ordered relationship between Sense, Plan and Act modules.

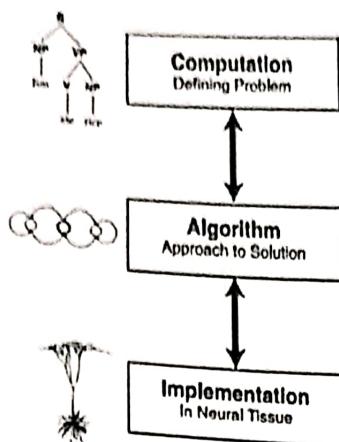
**Disadvantages of the Hierarchical paradigm:**

- The plan module was not very effective.
- The planning and sensing mechanisms were not connected.
- This architecture was not very good in handling uncertain or ambiguous situations, signals, sensor noise etc.

# Biological and cognitive paradigms for robot design (6 of 8)

- Computational theory

## A. Classic Marr Framework



## C. Comparative Computational Approach

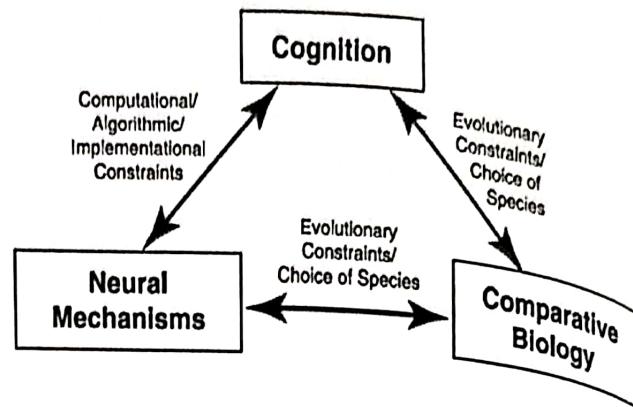


Figure: Computational Theory

Source: <http://dx.doi.org/10.1016/j.prev.2014.04.005>, 1571-0645/© 2014 Published by Elsevier B.V., Open access under CC BY-NC-ND license.  
[https://www.socsci.ucl.edu/~lpearl/courses/psych156A\\_2012spring/lectures/MidtermReviewbw.pdf](https://www.socsci.ucl.edu/~lpearl/courses/psych156A_2012spring/lectures/MidtermReviewbw.pdf))

Figure 1-9. Biological and cognitive paradigms for robot design (6 of 8)

AIR0110

## Notes:

Many robotic researchers who use AI turn to the biological sciences for number of reasons. This is because the biological sciences domain provides various aspects of intelligence. For example, till recently sensor fusion was considered a very active area of research and lot of approaches for robots were developed. Biological research showed that animals including humans perform sensor fusion, although it was a totally different mechanism than that most researchers had considered. Further principles of human intelligence animals consider the open world/environment they live in unlike the robots, which assume the closed world environment. Lot of researchers have tried to correlate intelligence of biological and cognitive sciences to life form/robot made of silicon.

Majority of these concepts lead to an abstract intelligent system, which are termed as Agents. In OOPs terminology agent is the superclass and persons and robots are derived from it. To decide on the level of intelligence of the software and its correspondence to the biological science, a theory called computational theory was developed by famous neuro-physiologist David Marr. He was instrumental in correlating the biological vision processes into new techniques for computer vision. According to this computational theory we have three levels:

**Level 1:** What is the problem to be solved? (Computational level): Suppose we are interested in building a robot to search for survivors in a quake hit area. From the biological sciences we know that the mosquitoes are very good at finding people. Mosquitoes provide a solution that it is possible for a computationally simple agent to find a human being using heat. At Level 1, thus the agents can share a common functionality or purpose.

**Level 2:** What (abstract) set of rules solves the problem? (Algorithmic level): This level can be thought of as creating a flow chart with black boxes. Each box represents a transformation of an input into an output. Now from biology we know that the mosquito zero in on the human beings by sensing the heat of a human. So, the mosquito on sensing a hot area flies towards it. We can model this process as:

if input=thermal image, output=steering may be one of the possible commands.

The black box is how the mosquito transforms the input into the output. One of the approaches might be to take the centroid of the thermal image and move towards that. If the hot patch moves, the thermal image will change with the next sensory update, and a new steering command will be generated. This might not be exactly how the mosquito actually steers, but it presents an idea of how a robot could duplicate the functionality. This level gives us more time to focus on the process rather than the implementation, since a search and rescue robot might have wheels. At this level, the agents can exhibit common processes.

**Level 3:** How are those rules physically implemented? (Implementation Level): This level of the computational theory focuses on describing how each of the black box at level 2, is implemented. For example, in a mosquito, the steering commands might be implemented with a special type of neural network, while in a robot, it might be implemented with an algorithm, which computes the angle between the centroid of heat and where the robot is currently pointing.

Likewise, a researcher interested in thermal sensing might examine the mosquito to see how it is able to detect temperature differences in such a small package; electro-mechanical thermal sensors weigh close to a pound! At Level 3, agents may have little or no commonality in their implementation.

# Biological and cognitive paradigms for robot design (7 of 8)

IBM ICE Innovation Center for Education

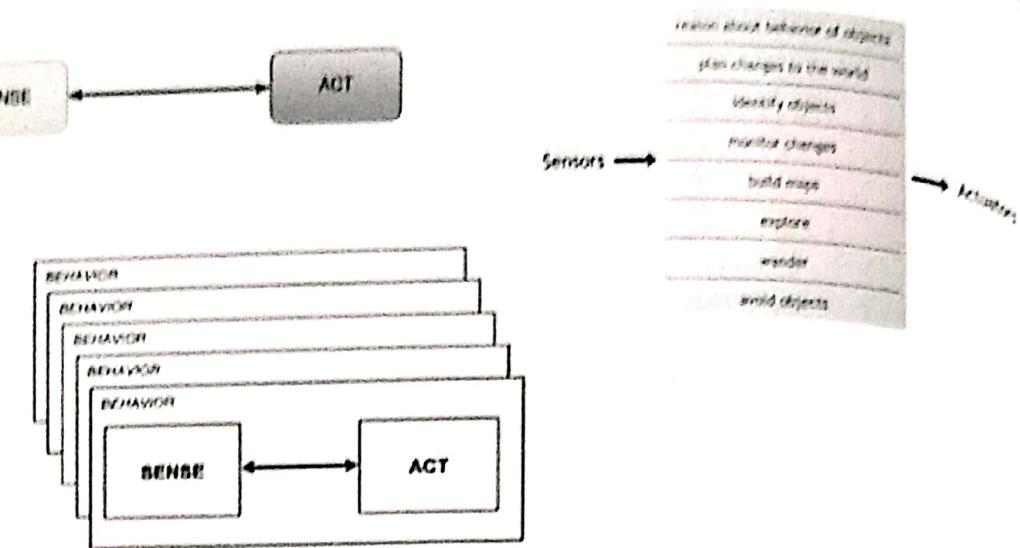


Figure: Reactive paradigm

Source: Article: Designing the mind of a social robot by Nicole Lazzeri, Daniele Mazzoni, Lorenzo Cominelli, Antonio Cisternino and Danilo Emilio De Rossi, Appl. Sci. 2018, 8, 302; doi:10.3390/app8020302  
Introduction to AI Robotics by Robin R Murphy, A Bradford Book, The MIT Press Cambridge, Massachusetts, London, England

Figure 1-10. Biological and cognitive paradigms for robot design (7 of 8)

## Notes:

The concept of reactive paradigm emerged in late 1980's. This paradigm is considered important because majority of the robots were designed using this and the recent advancement in hybrid reactive-deliberative paradigms also use this concept. One of the major attributes of this paradigm is that all actions are based on behaviors. These behaviors elicit a direct mapping of sensory inputs to a pattern of motor actions to complete the task. From the mathematical modeling perspective this means that behaviors are modeled as transfer function, which transforms the input from the sensors into commands executed by the actuator. As shown in the figure, the reactive paradigm doesn't have the plan module instead the sense and the act modules along with the behaviors are tightly coupled. All the tasks are executed as the result of the behaviors either in sequence or concurrently.

Rodney Brooks in 1986 developed the best-known representative of the reactive paradigm called the subsumption architecture for controlling a robot. This architecture is based on the fact that the cognition can be observed simply using perceptive and action systems that interact directly with each other along with a feedback loop through its environment. The subsumption architecture focuses on the idea of eliminating centralized control structures to build a robot control system with varied levels of competence. Each section/layers of the behavior-based controller is responsible for producing one or few independent behaviors. Except the bottom layer, all layers presuppose the existence of the lower layers, but none of the layers presuppose the existence of the higher layers.

In other words, using this bottom-up approach, each stage of the system development is able to operate. This architecture ensures that a minimal control system can be created for the lowest hardware level functionality of the robot and additional levels of competence can be added on the top whenever required without affecting the whole system. Figure shows an example of subsumption architecture with a behavior-based decomposition.

These are the following characteristic features of architecture that follow the reactive paradigm:

- Robots are considered as situated agents operating in an ecological environment. Situated agent means that the robot is an integral part of the environment. A robot has its own intentions and goals. When a robot takes an action, it changes the world, and it receives instantaneous feedback about the environment through sensing. What the robot senses affects its goals and how it attempts to meet them, generating a new cycle of actions.
- Behaviors serve as the basic building blocks for all robotic actions, and the overall behavior of the robot are emergent. Behaviors are independent, computational entities and operate concurrently. There is no explicit "controller" module, which determines what will be done, or functions, which call other functions. There may be a coordinated control program in the schema of a behavior, but there is no external controller of all behaviors for a task.
- Only local, behavior-specific sensing is permitted. The use of explicit abstract representational knowledge in perceptual processing, even though it is behavior-specific, is avoided. Any sensing that does require representation is expressed in ego-centric (robot-centric) coordinates. This eliminates unnecessary processing to create a world model, and then extract the position of obstacles relative to the robot.
- This architecture follows good software design principles. The concept of modularity of these behaviors supports the decomposition of a task into associated behaviors. The behaviors are tested independently, and behaviors may be assembled from primitive behaviors.
- Animal models of behavior are the basis for these systems for a particular behavior. Unlike earlier days of AI it is very much acceptable under the reactive paradigm to use animals as a motivation for a particular behavior.

## Biological and cognitive paradigms for robot design (8 of 8)



IBM ICE (Innovation Centre for Education)



- The FACE cognitive architecture based on the hybrid deliberative/reactive paradigm.

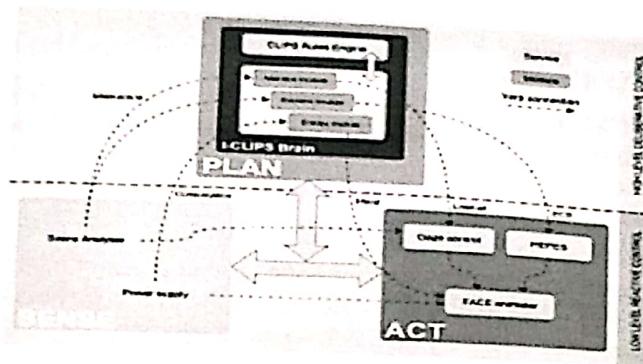


Figure: Hybrid deliberative/Reactive paradigm

Source: Article: Designing the mind of a social robot by Nicole Lazzeri, Daniele Mazzei, Lorenzo Cominelli, Antonio Cisternino and Danilo Emilio De Rossi, Appl. Sci. 2018, 8, 302; doi:10.3390/app8020302)

Figure 1-11. Biological and cognitive paradigms for robot design (8 of 8)

AIR011.0

### Notes:

Since the reactive paradigm did not have planning or any reasoning functions, as a result, a robot with this kind of control architecture could not select the best possible behavior to accomplish a task or a goal on the basis of some specific criteria. Thus, during the 1990s, AI robotic researchers tried to reintroduce the PLAN low-level control actions. Such architectures that used reactive behaviors and incorporated planning activities were said to be using a hybrid deliberative/reactive paradigm as shown in figure.

The Hybrid deliberative/Reactive paradigm has the Plan, then Sense-Act. The robot first plans to decompose a task into subtasks and it then decides what type of behavior is needed to appropriately complete the subtasks. Planning is done at one step, while sensing and acting are done together. Here the system is conceptually divided into a reactive layer and a deliberative layer. According to various literatures available the robot cognitive system can be divided in two main blocks:

- The low-level reactive control.
- The high-level deliberative control.

# Declarative-Procedural-Reflexive hierarchy for decision making and control

IBM

IBM ICE (Innovation Centre for Education)

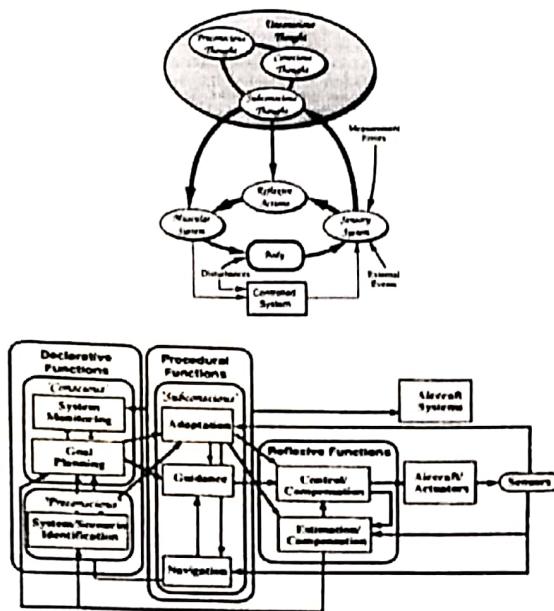


Figure: Cognitive/Biological Control Behavior

Source: Toward Intelligent Flight Control by Robert F. Stengel, Fellow, IEEE Transactions on Systems, Man and Cybernetics Vol 23, No 6, Nov/Dec 1993

Figure 1-12. Declarative-Procedural-Reflexive hierarchy for decision making and control

AIR011.0

## Notes:

According to the biological sciences and from the information processing perspective, human thoughts can be identified into four types:

- Conscious.
- Preconscious.
- Subconscious.
- Unconscious.

Conscious thoughts are thoughts that require our focus, awareness, reflection and occupy our attention. These thoughts perform declarative processing of the person's knowledge or beliefs. It leads to the language, philosophy, and emotions etc. of the person. Unconscious thought describes that information of the perceptual system that go unattended and this memory are lost as a result of displacement or display. Unconscious thoughts are further divided into subconscious thoughts and preconscious thoughts.

Subconscious thoughts are the procedural knowledge that is below our awareness, but the most important for the implementation of intelligent behaviors. This thought helps us to communicate with other parts of the body and the outside world thus helping us to show skills that are learnt like athletics, art, craft etc.

Preconscious thought is pre-attentive declarative processing, which operates on large chunks of information at a symbolic level. This plays a major role in institution and judgment, to channelize the long term and implicit memory.

In terms of human body and specifically central nervous system, it supports a hierarchy of automatic and intelligent functions with declarative actions at the top level, procedural actions at the middle level and reflexive action at the lowest/bottom level. This concept of declarative, procedural and reflexive functions could be used to model intelligent control behavior as shown in figure.

The Conscious thought module governs the overall system by performing declarative functions, receiving information and transmitting commands through the subconscious thought module, which is itself capable of performing procedural actions. Conscious thought is also linked to the Preconscious thought, which can perform symbolic declarative functions and is also alerted to the pending tasks by Subconscious thought. All these three modules are treated as unconscious thought that is termed as long-term memory.

The subconscious thought module receives the information from the sensory system and conveys commands to the muscular system. Reflexive actions provide low-level regulation in parallel with the high-level functions, which responds to stimuli and coordinates various control actions. Control actions produce Body motion and can affect an external controlled system, as is the case of piloting an aircraft. In general, learned control functions, Body motion can be correlated to the model of controlled system behavior. The Body and the controlled systems are both directly or indirectly subjected to disturbances; for example, turbulence would affect the aircraft directly and the pilot indirectly. The sensory system observes external events as well as body and controlled system motions, and it is subject to measurement errors.

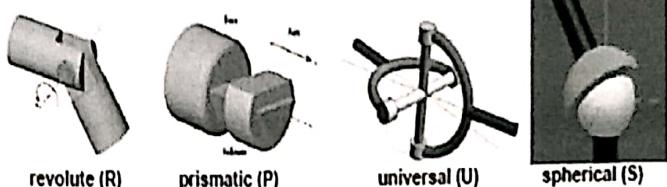
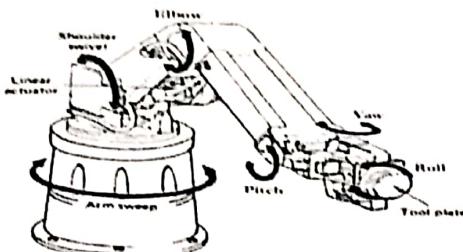
Human cognition provides paradigms for control system design. The important observation is that learning requires error or incompleteness. There is nothing new to be gained by observing a perfectly operating known process. In context of a control system, any operation should be started using the best available knowledge of the process and complete control resources. Consequently, learning is not even desirable or necessary in a flight control system. Biological adaptation however is a slow process, and proper changes in behavior can be made only if there is prior knowledge available.

The suggested intelligent flight control system structure has super-blocks identifying declarative, procedural, and reflexive functions; these contain the classical Guidance, Navigation, and Control (GNC) Systems functions plus new functions related to decision making, prediction, and learning. Within the super blocks, higher-level functions are identified as conscious, preconscious, and subconscious attributes as a working analog for establishing a computational hierarchy.

## Articulated robots (1 of 2)

IBM

IBM ICE (Innovation Centre for Education)



Common Robot Joint Examples (1, 1, 2, and 3-dof, respectively)

Figure: Articulated robot and the common joints

Source: <https://www.engineering.com/AdvancedManufacturing/ArticleID/12192/Articulated-Robot-Market-Worth-USD7958-Billion-by-2022.aspx>

An Introduction to Robotics, Dr. Bob Williams, williar4@ohio.edu, Mechanical Engineering, Ohio University, EE/ME 4290/5290 Mechanics and Control of Robotic Manipulators, © 2019 Dr. Bob Productions

Figure 1-13. Articulated robots (1 of 2)

AIR011.0

### Notes:

A robot is a machine, which is capable of movement/physical motion for interacting with the environment. Physical interactions involve locomotion, manipulation and any other activity, which changes the state of the environment or the state of the robot relative to the environment. Before we go into the details of articulated robots let's look some definitions:

- Robot: An electromechanical device with multiple degrees of freedom that is programmable to accomplish a variety of tasks.
- Degrees of Freedom: The number of independent motions, which a robot can make. This is also termed as Mobility.
- Manipulator: An electromechanical device capable of interacting with its environment.
- End-Effector: The tool, gripper or any other device mounted at the end of the manipulator, for accomplishing the task.
- Workspace: The volume in space that the robot's end effectors can reach, both in orientation and position.
- Position: The translational location of an object (Straight-Line).
- Orientation: The rotational location of an object (Angular). For example an airplane's orientation is measured by roll, pitch and yaw angles.
- Pose: Position and orientation taken together.

- Link: A rigid piece of material connecting joints in a robot.
- Joint: The device which allows relative motion between two links in a robot as shown in the figure.
- Kinematics: The study of motion without regard to forces/torques.
- Dynamics: The study of motion with regard to forces/torques.
- Actuator: Provides force/torque for robot motion.
- Sensor: A device that reads actual variables in robot motion for use in control.
- Haptics: Haptic interfaces give human operators the sense of touch and forces from the computer, either in virtual or real, remote environments.
- Anthropomorphic: Designed to appear like human beings.

Some of the technical robotic terms that are often used are as given below:

- **Speed:** Speed is the amount of distance per unit time, which the robot moves, usually specified in inches/sec or meters/sec. The speed is usually specified at a specific load.
- **Load bearing capacity:** Load bearing capacity is the maximum weight-carrying capacity of the robot.
- **Accuracy:** Accuracy is the ability of a robot to go to the specified position without making a mistake.
- **Repeatability:** Repeatability is the ability of a robot to repeatedly position itself when asked to perform a task multiple times.
- **Precision:** Precision is the fitness with which a sensor can report a value.

**Industrial robot as defined by ISO 8373:2012:** An automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications.

Some of the terms used in the definition are:

- Reprogrammable: Designed so that the programmed motions or auxiliary functions can be changed without physical alteration.
- Multipurpose: Capable of being adapted to a different application with physical alteration.
- Physical alteration: Alteration of the mechanical system (the mechanical system does not include storage media, ROMs, etc.)
- Axis: Direction used to specify the robot motion in a linear or rotary mode.

Classification of Robots as per Mechanical Structure:

- Linear/Cartesian robot: robot whose arm has three prismatic joints and whose axes are coincident with cartesian coordinate system.
- Selective Compliance Articulated Robot Arm (SCARA) robot: A robot, which has two parallel rotary joint to provide compliance in a plane.
- Articulated robot: a robot whose arm has at least three rotary joints.
- Parallel robot: a robot whose arms have concurrent prismatic or rotary joints.
- Cylindrical robot: a robot whose axes form a cylindrical coordinate system.

## Articulated robots (2 of 2)

IBM ICE (Innovation Centre for Education)

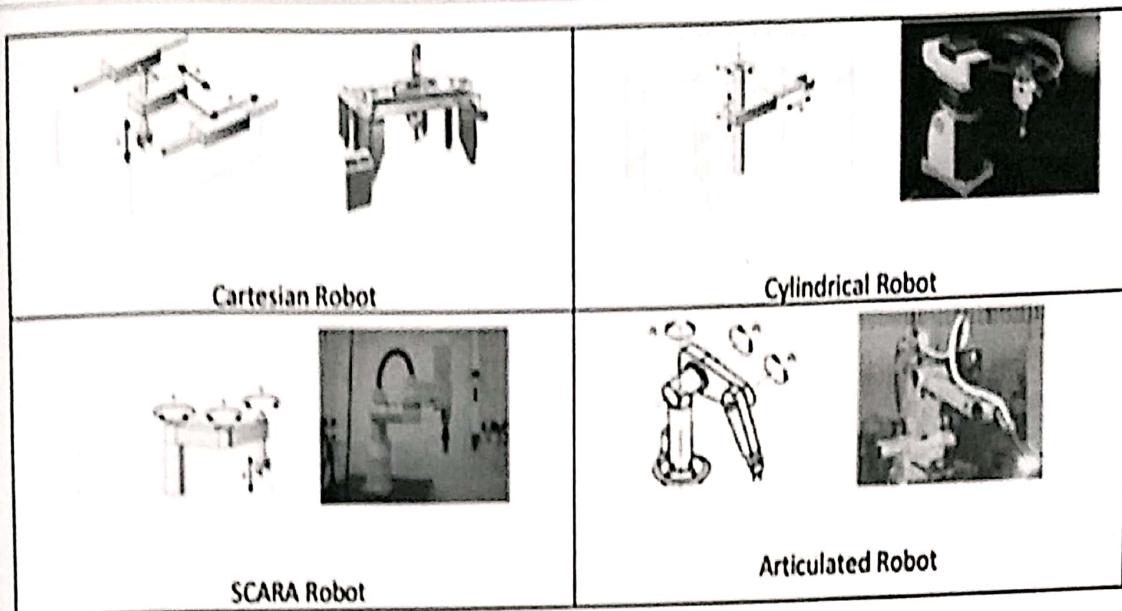


Figure: Common Robot Designs

Source: An Introduction to Robotics, Dr. Bob Williams, williar4@ohio.edu, Mechanical Engineering, Ohio University, EE/ME 4290/5290  
 Mechanics and Control of Robotic Manipulators, © 2019 Dr. Bob Productions

Figure 1-14. Articulated robots (2 of 2)

AIR011.0

### Notes:

#### Cartesian Robot:

Cartesian robots have three linear axes of movement (X, Y, Z). They are constructed of three mutually orthogonal P joints, with variable lengths  $L_1$ ,  $L_2$ ,  $L_3$ . Used for pick and place tasks and to move heavy loads. Also called Gantry Robots, they can trace rectangular volumes in 3D space.

#### Cylindrical Robot:

Cylindrical robot positions are controlled by a variable height  $L_1$ , an angle  $\theta_2$ , and a variable radius  $L_3$  (P joint, R joint, P joint). These robots are commonly used in assembly tasks and can trace concentric cylinders in 3D space.

#### SCARA Robot:

SCARA robots have two R joints  $\theta_1$  and  $\theta_2$ , plus a P joint  $d_3$  perpendicular to that plane of motion, to achieve a 3D xyz workspace. R joint angle  $\theta_4$  is the single-rotation SCARA robot wrist. These are common tabletop assembly robots.

#### Articulated Robot:

Articulated robots resemble the human arm in their 3D motion (they are anthropomorphic). They have three R joints, with three variable angles  $\theta_1$ ,  $\theta_2$ , and  $\theta_3$ , representing the human body waist, 1-DOF shoulder, and elbow joints.

They are versatile robots, but have more difficult kinematics and dynamics control equations than other serial robots. All of these robot architectures may be used with a variety of robot wrists to provide the orientation Dof.

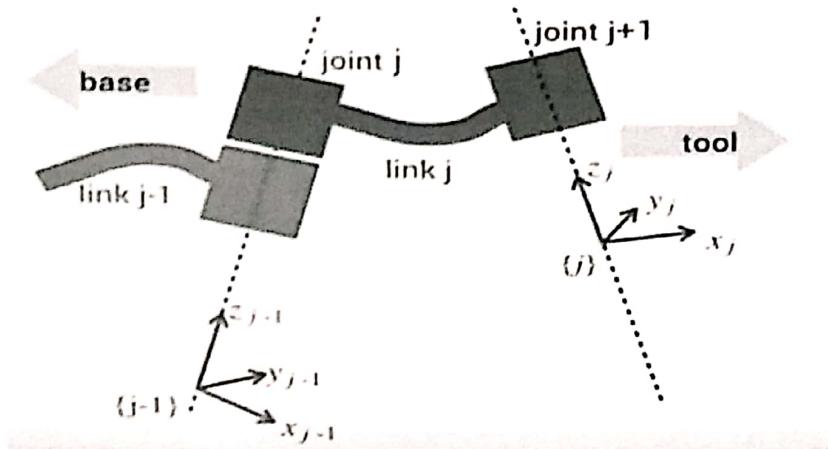
### Main Components of Industrial Robots:

- Arm or Manipulator:
  - The main anthropomorphic element of a robot.
  - In most cases the degrees of freedom depends on the arm.
  - The work volume or reach mostly depends on the functionality of the Arm.
- End effectors.
- Device attached to the robot's wrist to perform a specific task. For example Grippers
  - Mechanical Grippers.
  - Suction cups or vacuum cups.
  - Magnetized gripper.
  - Hooks.
  - Scoops (to carry fluids).
- Drive Mechanism.
- Device attached to the robot's wrist to perform a specific task. For example Tools
  - Spot Welding gun
  - Arc Welding tools
  - Spray painting gun
  - Drilling Spindle
  - Grinders, Wire brushes
  - Heating torches
- Controller.
  - It controls the whole robot.
  - Custom features: e.g. sensors and transducers
  - Tactile sensors (touch sensors, force sensors, tactile array sensors)
  - Proximity and range sensors (optical sensors, acoustical sensors, electromagnetic sensors)
  - Miscellaneous sensors (transducers and sensors which sense variables such temperature, pressure, fluid flow, thermocouples, voice sensors)
  - Machine vision systems

# Joint-Link (Denavit-Hartenberg) transformations (1 of 4)

IBM

IBM ICE (Innovation Centre for Education)



$${}^{j-1}\xi_j \sim A = R_z(\theta_j) T_z(d_j) T_x(a_j) R_x(\alpha_j) \quad ---1.1$$

Figure: Denavit-Hartenberg notation

Source: Denavit-Hartenberg notation | Robot Academy

Figure 1-15. Joint-Link (Denavit-Hartenberg) transformations (1 of 4)

AIR011.0

## Notes:

Denavit and Hartenberg in 1955 wrote an article on a general way of describing serial link mechanisms. Since most of the robots have this type of link mechanisms, so Denavit and Hartenberg (DH) is most applicable to these classes of mechanisms. In general they developed a general theory to describe an articulated sequence of joints.

# Joint-Link (Denavit-Hartenberg) transformations (2 of 4)

IBM ICE (Innovation Centre for Education)

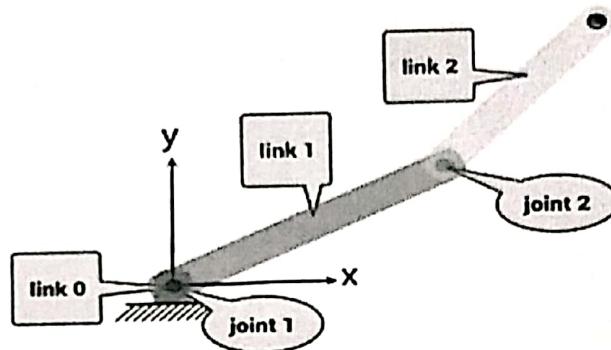


Figure: Serial link manipulator

Source: Denavit-Hartenberg notation | Robot Academy

Figure 1-16. Joint-Link (Denavit-Hartenberg) transformations (2 of 4)

AIR011.0

## Notes:

A key aspect of DH notation is that each joint in the robot is described by four parameters. Let's start with a basic serial-link manipulator as shown in figure. As seen in the figure each joint is attached to the previous joint via a link. Each of the links is rigid and the joints can either be revolute (rotational) or prismatic (sliding). Every joint connects two links and every link connects two joints expect the first and last link. The first link is the base of the robot, which does not move and is referred as link 0 and the last link is the end effector. So in general we have N joints with N+1 Links.

The fundamental concept to the DH notation is to attach a coordinate frame to the far end of every link of the robot. As shown in figure, we can see link j shown in blue and we attach the coordinate frame j to the far end of link j that is the end of the link that's closest to the robots tool. We then describe the pose of that link frame w.r.t the link frame of the previous joint. This is a relative pose.

In the DH notation the link transform is represented by a homogeneous transformation matrix denoted by letter A and it comprises a number of elementary transformations as shown in equation 1.1. This equation shows that we have a rotation around the Z-axis, a translation along the Z-axis, a translation along the X-axis and a rotation around the X- axis. It allows us to describe the relationship between the 2 link coordinate frames by simply 4 parameters,  $\theta$ ,  $d$ ,  $a$  and  $\alpha$ . We have now thus moved from frame  $j-1$  to the frame  $j$  by applying the 4 elementary transformations i.e. 2 translations and 2 rotations.

The concept of representing translational and rotational components which are described by 3 numbers each by just 4 parameters is that the DH notation places some constraints on where we can place the coordinate frames.

The following are the constraints:

- X-axis of frame J intersects the Z-axis of frame J-1.
- X-axis of frame J is perpendicular to the Z-axis of frame J-1.

Although, there are only 6 degrees of freedom in a relative pose. Since we've introduced 2 constraints means that we can describe this using only 4 parameters. The relative pose from the frame of 1 link to the next is described by 4 elementary transformations. For an n-link robot, we can stack groups of those elementary transformations and each group contains 4 parameters, which described the relationship between 1 link frame and the next. The great advantage of the Denavit-Hartenberg notation is that it allows us to very concisely describe a robot. So, for the 2-link robot, simply a table discussed in the next slide can describe it.

## Joint-Link (Denavit-Hartenberg) transformations (3 of 4)

- Denavit-Hartenberg parameters for a 2 link robot

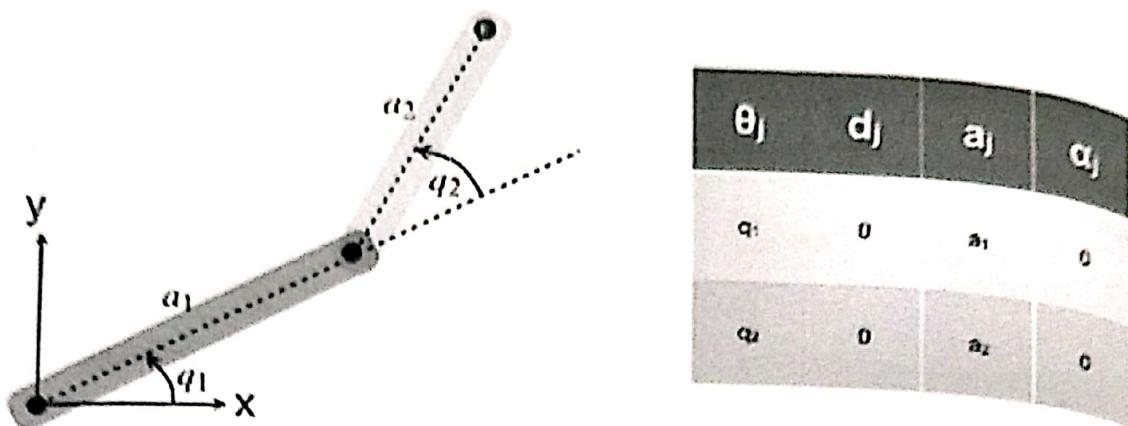


Figure: Two-Link Robot and Kinematics of the Robot

Source: Denavit-Hartenberg notation | Robot Academy

Figure 1-17. Joint-Link (Denavit-Hartenberg) transformations (3 of 4)

AIRC11

### Notes:

In the table as shown in the figure, we have one column for each of the DH parameters and we have one row for each joint of the robot. The joint variables  $q_1$  and  $q_2$  lie in the theta column because they are revolute joints. The D values are all 0. There are no translations along the Z-axis because this robot exists in a plane and the 2 link lengths appear in the A column and the alpha values are all equal to 0. So, this very compact table completely describes what we call the kinematics of the robot. Let us look at a few more cases where there are different numbers of joints as in next figure.

# Joint-Link (Denavit-Hartenberg) transformations (4 of 4)

IBM

IBM ICE (Innovation Centre for Education)

Denavit-Hartenberg parameters for a 2 link robot

- All revolute joints (RRRRRR)

$$T_E = R_z(\theta_1) T_z(d_1) T_x(a_1) R_x(\alpha_1) \underset{\text{revolute joint}}{R_z(\theta_2) T_z(d_2) T_x(a_2) R_x(\alpha_2)} \cdots \underset{\text{revolute joint}}{R_z(\theta_N) T_z(d_N) T_x(a_N) R_x(\alpha_N)}$$

- Revolute and prismatic joints (RPRRRR)

$$T_E = R_z(\theta_1) T_z(d_1) T_x(a_1) R_x(\alpha_1) \underset{\text{revolute joint}}{R_z(\theta_2) T_z(d_2) T_x(a_2) R_x(\alpha_2)} \cdots \underset{\text{prismatic joint}}{R_z(\theta_N) T_z(d_N) T_x(a_N) R_x(\alpha_N)} \underset{\text{revolute joint}}{R_z(\theta_{N+1}) T_z(d_{N+1}) T_x(a_{N+1}) R_x(\alpha_{N+1})}$$

Figure: Two cases (i) with revolute joints only (ii) Revolute and prismatic joints

Source: Denavit-Hartenberg notation | Robot Academy

Figure 1-18. Joint-Link (Denavit-Hartenberg) transformations (4 of 4)

AIR011.0

## Notes:

For an n-link robot, we can stack groups of these elementary transformations and each group contains 4 parameters, which described the relationship between 1 link frame and the next. If the robot has got all revolute joints then, the joint angles correspond to the theta values shown. So, these are the joint variables. They change as the robot moves. All the other parameters, the D's, the A's and the alpha's are all constant. They're a function of the mechanical design of the particular robot. Let's consider a robot that has got a prismatic joint. Its second joint is prismatic. Since the first joint is revolute, we substitute  $q_1$  in here. For the second joint, which is prismatic, we substitute  $q_2$  in here. For a joint like this,  $\theta_2$  is a constant just like  $a_2$  and  $\alpha_2$ . They are the function of the mechanical design of this particular robot.

# Mobile ground robots



Figure: Mobile Ground Robots

Source: An Introduction to Robotics, Dr. Bob Williams, williar4@ohio.edu, Mechanical Engineering, Ohio University, EE/ME 4290/5290 Mechanics and Control of Robotic Manipulators, © 2019 Dr. Bob Productions

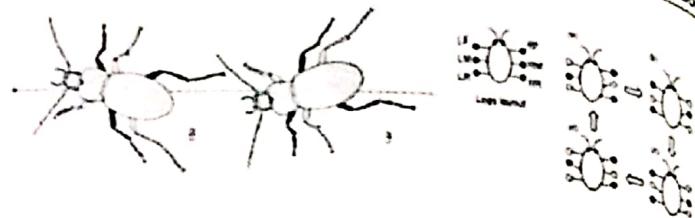


Figure: Legged Robots (gaits) (Ref: Gullan et al., The Insects: An outline of entomology, 2005; Iida et al., Science Direct, 63, 2008)

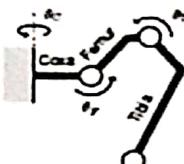


Figure: Hexapod Walking robot (Ref: Introduction to Robotics-Lecture 1, Jan Faigl, Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague)

Figure 1-19. Mobile ground robots

## Notes:

Mobile robots for locomotion use wheels, legs or other means and hence navigate around the workspace. These robots are used as hospital nurses, vacuum cleaners, lawn movers, etc. These robots require good sensors to see the workspace and avoid collisions for getting the work done. Locomotion generally refers as to how the robot moves from one location to another location. The most typical effectors and actuators for ground robots are wheels and legs.

### Locomotion – Wheel Robots

One of the simplest wheeled robots is a differential drive robot. It has two driven wheels on a common axis. It may use a castor wheel (or ball) for stability. The simple motion control can be realized in a turn-move like schema. Further motion control using path following or trajectory following approaches with feedback controller based on the position of the robot to the path/trajectory.

### Locomotion – Legged Robots (Gaits)

Gait is a way as how a legged robot moves. A gait defines the order how the individual legs lift and lower and also places of the foot tip on the ground. The properties of gaits are stability, speed, energy efficiency, robustness and simplicity. A typical gait for hexapod walking robot is tripod, which is stable as all times at least three legs, are on the ground.

### Locomotion - Hexapod Walking Robot

The figure shows a hexapod robot with six identical legs each with 3 DOF. Each leg consists of three parts called Coxa, Femur and Tibia. The movement is a coordination of the stance and swing phases of the legs defined by the gait. A stride is a combination of the leg movement with the foot tip on the ground (stance phase) and the leg movement in a particular direction (swing phase) within one gait cycle. Thus various gaits can be created by different sequences of stance and swing phases.

# Uninhabited ground robots (1 of 2)

IBM ICE (Innovation Centre for Education)

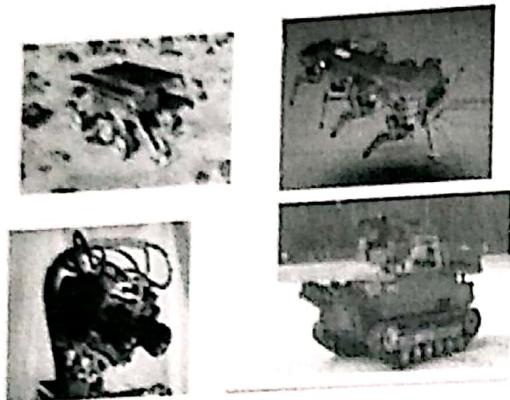


Figure: Uninhabited Ground Robots

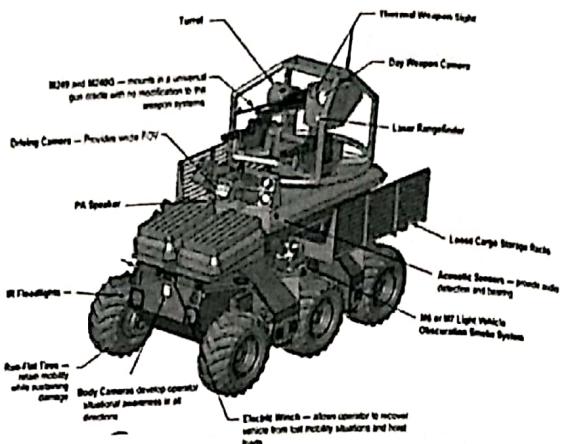


Figure: DOD robotic vehicle: Gladiator

Source: Military Robotics: Malignant machines or the path to Peace? Presented By: Dr. Robert Finkelstein President, Robotic Technology Inc., revised January 2010

Figure 1-20. Uninhabited ground robots (1 of 2)

AIR011

## Notes:

These types of robots are generally used in dull, dirty and dangerous terrains. They are also used in hostile and hazardous environments. Other rationales for using such robots are as given below:

- No need to encase and protect humans in vehicles since these are used which are smaller, lighter, less expensive.
- More survivable: Small signature.
- More maneuverable: Faster, higher acceleration.
- Faster response time: Pre-positioning.
- Riskier maneuvers and tactics.
- Fearless and aggressive: Not deterred by near misses.
- No need for sleep or rest.
- Fewer personnel can supervise more systems.

This slide introduces the robotic ground vehicles, which are generally referred to as unmanned ground vehicles (UGV), but sometimes also include air and sea robotic vehicles. The figure shows the DOD robotic vehicle: Gladiator. As seen this has Run-Flat tires, Body cameras, Electric winch, Driving camera, PA speaker, M249 and M240G gun cradle, day weapon camera, laser range finder, loose cargo storage racks, Acoustic sensors, obscuration smoke system.

## Uninhabited ground robots (2 of 2)

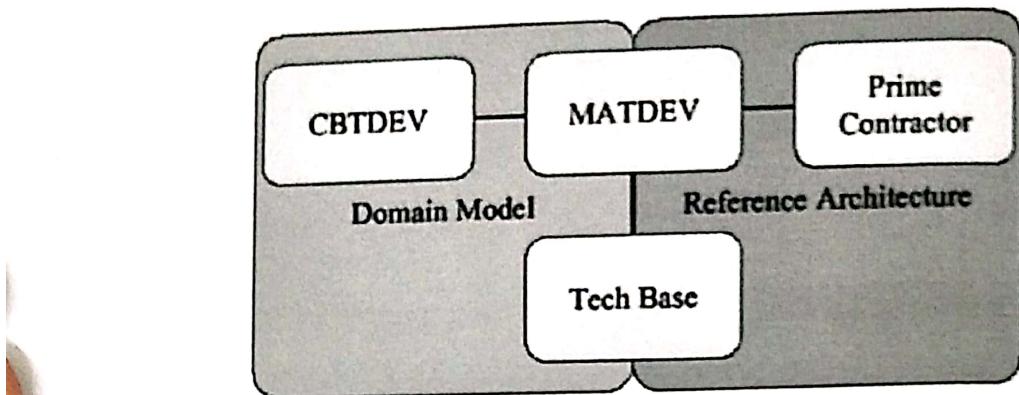


Figure: JAUGS Reference Architecture

Source: Military Robotics: Malignant machines or the path to Peace? Presented By: Dr. Robert Finkelstein President, Robotic Technology Inc., revised January 2010

Figure 1-21. Uninhabited ground robots (2 of 2)

AIR0110

### Notes:

JAUGS is a reference architecture developed by DOD's Joint Robotics Program (JRP). JAUGS is a very broad framework since it encompasses a wide variety of autonomy and operational scenarios required for defense specific unmanned ground systems. The JAUGS architecture does not specify the design or implementation; rather, it is a framework for designing architectures fulfilling the following design conditions:

- Independent of vehicle/platform.
- Mission independence.
- Independent of computer hardware used.
- Independent of technology being used.

JAUGS is an architecture specification primarily concentrating on the implementation and control issues which are necessary to ensure interoperability and exchangeability of unmanned vehicles and robots. This will ultimately lead to a situation where a group of JAUGS robots would not only have standardized communications (e.g., TCP/IP) but a standardized modus operandi (i.e., basis of operation).

A common interface would allow for improved interoperability and portability so that the features can be upgraded simply, and also new options can be added. The JAUGS architecture depends on the JAUGS Domain model as shown in figure. This model outlines the requirements for the user ("in the language of the user"), defines system topology (i.e., operational systems, nodes, components), specifies interface issues, provides requirements on methods for message passing, and standards to support component integration.

The JAUGS "domain model," in the present form, focuses on the remote driving of large unmanned ground vehicles and not on micro-robots. Also, the JAUGS model also does not allow for separation or distinction based upon internal and external system operations, which is the most important aspect in multi-robot coordination. The specifications are still being designed and revised by the JAUGS Working Group for Standard Operating Procedure and currently only constrains solutions that are within the "area of compliance"

The JAUGS model has both advantages and disadvantages. Some of the advantages of the JAUGS architecture are: an architecture open to the community, reduction in life cycle costs, focus on "new" requirements/engineering system, and a framework for technology insertion. However, the disadvantages of the JAUGS system (in its current form) are: that it is too highly linked to the deliberative architecture/model, it does not have enough flexibility to properly handle architectures without world models (e.g., reactive), and it appears applicable only to large ground systems and not the multi-robot domain type systems.

## Intelligent agents (1 of 4)

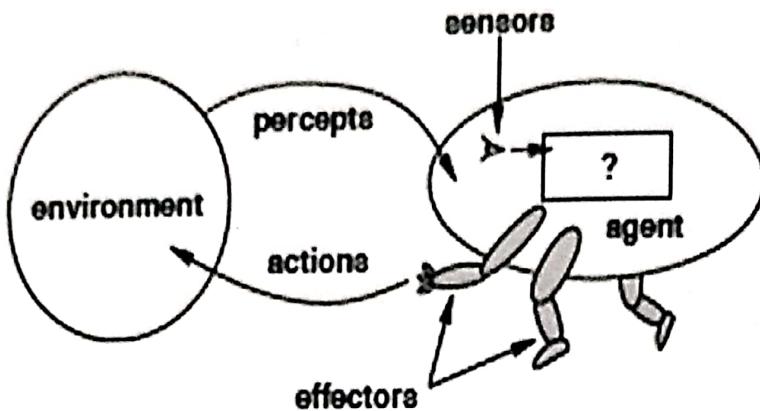


Figure: Agents(Ref: Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig, c 1995 Prentice-Hall, Inc.)

Source: Military Robotics: Malignant machines or the path to Peace? Presented By: Dr. Robert Finkelstein President, Robotic Technology Inc., revised January 2010

Figure 1-22. Intelligent agents (1 of 4)

AIR011.0

### Notes:

An agent is either a software/hardware that perceives its environment through sensors and acts upon that environment through actuators. For example if human is considered as an agent, he/she has eyes, ears and other sensory organs acting as sensors and other body parts as effectors. A robotic agent has cameras, infrared range finders as the sensors and various motors as effectors. A software agent has the information encoded as binary digits as percepts and actions. So we have:

- Human agents.
- Robotic agents.
- Software agents.

Let's now look at some of the commonly used terminologies:

**Percept:** The agent's perceptual inputs.

**Percept sequence:** The complete history of everything the agent has perceived.

**Agent function:** Maps any given percept sequence to an action  $[f: p^* \rightarrow A]$ .

**The agent program:** Runs on the physical architecture to produce  $f$ .

**Agent** = architecture + program.

## Intelligent agents (2 of 4)

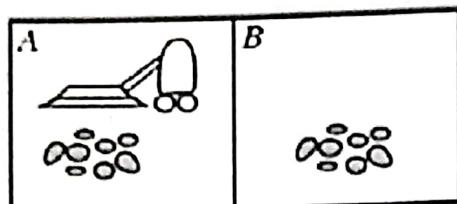


Figure: Vacuum-cleaner World

Source: CS 420:Artificial Intelligence Chapter 2 :Intelligent Agents

Percept sequence	Action
[A, Clean]	
[A, Dirty]	
[B, Clean]	
[B, Dirty]	
[A, Clean], [A, Clean]	
[A, Clean], [A, Dirty]	
...	
[A, Clean], [A, Clean], [A, Clean]	
[A, Clean], [A, Clean], [A, Dirty]	
...	

?

Figure: Simple Agent Function

Source: CS 420:Artificial Intelligence Chapter 2 :Intelligent Agents

Figure 1-23. Intelligent agents (2 of 4)

AIR011.0

### Notes:

Let's us now try to understand this with an example shown in figure.

In the above example the Percepts are the location and contents, example [A, Dirty] and the actions might be move left, move right, suck and No operation. The simple agent function as in figure could be perceived as shown.

An agent should do the right thing based on what it perceives and the actions it can perform. The right actions will cause the agent to be successful. The performance measure is the objective criterion for success of the agent's behavior. In this case it can be amount of dirt cleaned within certain time or some credits for cleaning some area per unit time. A general rule for rationality is to measure what one wants rather than how one thinks the agent should behave.

For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has. Thus we should remember that a rational agent is one that does the right thing.

**A simple agent that cleans a square if it is dirty and moves to the other square if not?**

Now the question arises as to whether is it rational?

We have the following Assumption:

- Performance measure: 1 point for each clean square at each step.
- Environment is known a priori.
- Actions = {left, right, stuck, no-op}.
- Agent is able to perceive the location and dirt in that location.

Given different assumption, it might not be rational anymore.

## Intelligent agents (3 of 4)

**IBM**  
IBM ICE (Innovation Centre for Education)

Performance Measure	Environment	Actuators	Sensors
safe, fast, legal, comfortable trip, maximize profits	roads, other traffic, pedestrians, customers	steering, accelerator, brake, signal, horn, display	camera, sonar, speedometer, GPS, odometer, engine sensors, keyboard, accelerator

Figure: PEAS for Taxi Driver Example

Source: CS 420:Artificial Intelligence Chapter 2 :Intelligent Agents

Figure 1-24. Intelligent agents (3 of 4)

AIR011.0

### Notes:

Specifying the task environment is always the first step in designing agent. The most important thing to keep in mind is PEAS, where:

P stands for Performance, E for Environment, A for Actuators and S for sensors. So we consider a Taxi and the driver as an example, the following information of PEAS will be very useful.

As we know already that Agent = Architecture + Program, the job of AI is to design the agent program that implements the agent function mapping percepts to various actions. The aim is to find a way to implement the rational agent function concisely. Thus, for the agent we choose the same skeleton such that it takes the current percepts as input from the sensors and returns an action to control the actuators. The agent program takes the current percept as an input and no other information is available from the environment whereas the agent function takes the entire percept history.

# Intelligent agents (4 of 4)

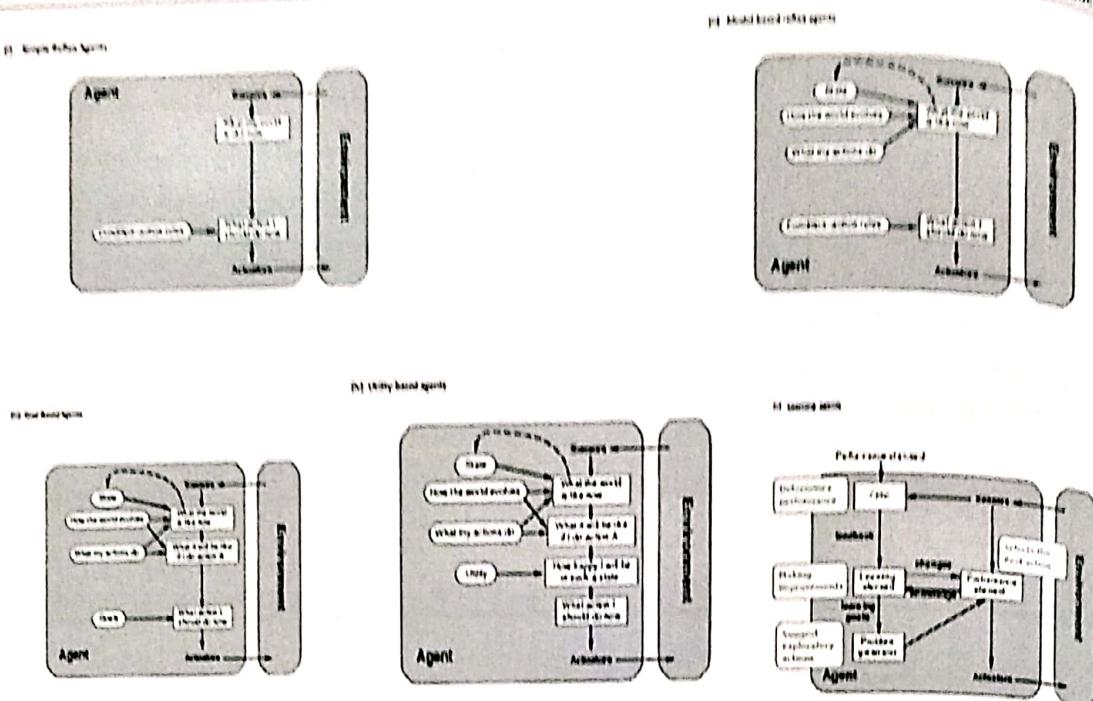


Figure: Basic Agent Types

Source: Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig, c 1995 Prentice-Hall, Inc.)

Figure 1-25. Intelligent agents (4 of 4)

AIR011.0

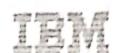
## Notes:

There are five basic agent types:

- Simple reflex agents.
- Model-based reflex agents.
- Goal-based agents.
- Utility-based agents.
- Learning agents.

Each of these can be shown with a generic block diagram and these are self explanatory to understand the essence and concept.

# Open-loop and closed-loop systems (1 of 4)



IBM ICE (Innovation Centre for Education)

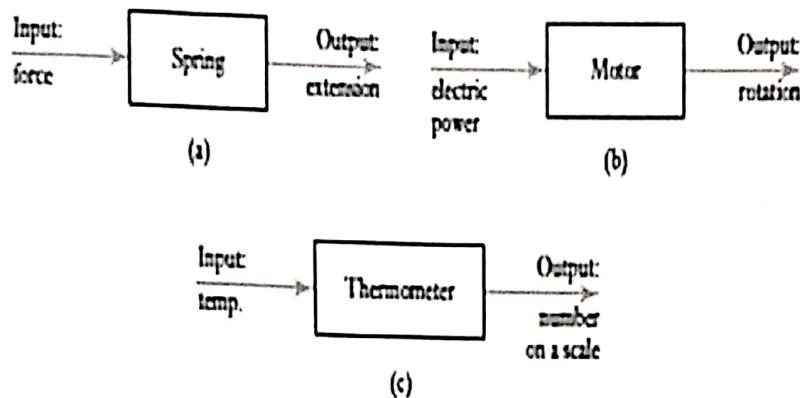


Figure: Systems found in the real world

Source: Mechatronics – Seventh Edition by William Bolton:

<https://books.google.co.in/books?id=BIV1DwAAQBAJ&pg=SA1-PA5&lpg=SA1-PA5&dq=spring+motor+and+thermometer+william+bolton&source=bl&ots=JHvk-Pto9C&sig=ACfU3U3hEsc2-sgrCe28ZUZwrqaMEEA&hl=en&sa=X&ved=2ahUKEwj19Hgy9TkAhVEvY8KHbvSBuwQ6AEwCnoECAkQAQ#v=onepage&q=spring%20motor%20and%20thermometer%20william%20bolton&f=false>

Figure 1-26. Open-loop and closed-loop systems (1 of 4)

AIR011.0

## Notes:

As shown in the figure, a system can be thought of as a black box, which has an input and an output. The most important aspect of such a system is the relationship between the output and the input rather than what's inside the black box. Modeling is the term used when we express the behavior of a real system using mathematical equations. These equations represent the relationship between the inputs and outputs from the system. For ex: A spring if considered as a system will have force as the input and extension as the output. The input force  $F$  and the output extension  $x$  can be modeled using the relationship  $F = kx$  where  $k$  is a constant. Similarly a motor may be considered as a system with electric power as the input and rotation as the output. So also in (c) we have a thermometer that is a system used for measurement. The input is the temperature and the output is the number on the scale.

## Open-loop and closed-loop systems (2 of 4)

- System modeling

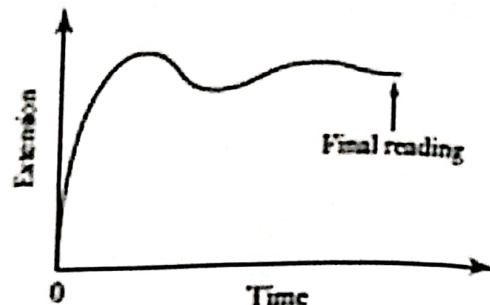


Figure: Response to an input for a spring

Source: Ref: Mechatronics – Seventh Edition by William Bolton:

<https://books.google.co.in/books?id=BIV1DwAAQBAJ&pg=SA1-PA5&lpg=SA1-PA5&dq=spring+motor+and+thermometer+william+bolton&source=bl&ots=JHVK-Pt09C&sig=ACfU3U3hEsc2-sgrCe28ZUZwrqaMEEA&hl=en&sa=X&ved=2ahUKewj19Hgy9TkAhVEvY8KHbvSBuwQ6AEwCnoECAkQAQ#v=onepage&q=spring%20motor%20and%20thermometer%20william%20bolton&f=false>

Figure 1-27. Open-loop and closed-loop systems (2 of 4)

AIR011.0

### Notes:

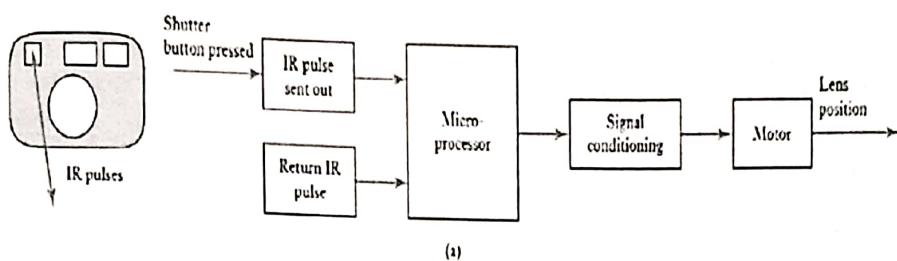
The response of any system to an input is not instantaneous. The relation between the input force  $F$  and its output, extension  $x$  is given by  $F = kx$ . This relationship is when a steady-state condition occurs. This means that before the spring settles to its steady state extension value, there are oscillations that will occur, implying that the response of the system is a function of time, as shown. Thus to know the behavior of the system when any input is applied to them, we need to create models for systems which relate the output to the input. This relationship should be able to let us track how the output will vary with time till it finally settles down at the steady state.

## Open-loop and closed-loop systems (3 of 4)

IBM

IBM ICE (Innovation Centre for Education)

- Digital camera



**Figure: Digital Camera**

Source: lecture slides of Dr. Atul Thakur, Module I.pdf

Figure 1-28. Open-loop and closed-loop systems (3 of 4)

AIR011.0

### Notes:

This example shows a digital camera as a system, which has a microprocessor as a controller and various other components like IR pulse emitter, signal-conditioning module and a motor. The system designers are required to create a model, which will show the relationship between the rays entering the camera and the lens position.

## Open-loop and closed-loop systems (4 of 4)

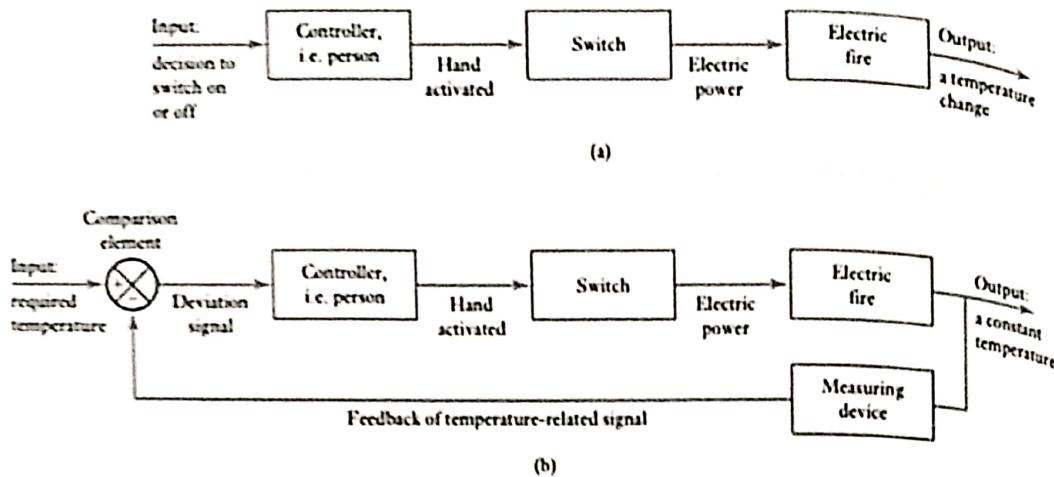


Figure: Open and closed loop system

Source: Ref: Mechatronics – Seventh Edition by William Bolton:

<https://books.google.co.in/books?id=BIV1DwAAQBAJ&pg=SA1-PA5&lpg=SA1-PA5&dq=spring+motor+and+thermometer+william+bolton&source=bl&ots=JHvk-Pt09C&sig=ACfU3U3hEsc2-sgrICeu28ZUZwrqaMEEA&hl=en&sa=X&ved=2ahUKEwj19Hgy9TkAhVEvY8KHbvSBuwQ6AEwCnoECAkQAA#v=onepage&q=spring%20motor%20and%20thermometer%20william%20bolton&f=false>

Figure 1-29. Open-loop and closed-loop systems (4 of 4)

AIR0110

### Notes:

A control system can be thought of as a system which can be used to:

- Control a variable/parameter at some value, e.g., a AC where the temperature is set at a particular value.
- Control the sequence of events, e.g., a washing machine set to a particular wash cycle, which represents a sequence of events, appropriate to that type of cloth.
- Control whether an event has occurred, e.g., a safety lock on a machine where it cannot be operated without a guard.
- There are two basic forms of control system, one being called **open loop** and the other **closed loop**.

The difference between these two can be illustrated with a simple example. Consider an electric heater which has a selection switch which allows a 1 kW or a 2 kW heating rod to be selected. A person can select a 1 kW rod or a 2 kW rod based on how much heating is required. If there are sudden changes in the room temperature (opening a window), there is no way the heating can be adjusted to compensate. This is an example of open-loop control system where there is no information feed back to the element(s) to adjust it and maintain a required constant temperature.

The electric heating system with the heating element could be made a closed-loop system if the person senses the temperature using a thermometer and switches the 1 kW and 2 kW elements on or off, based on the difference between the actual temperature and the required temperature, to maintain the temperature of the room constant.

In such a scenario there is feedback i.e., the input to the system is being adjusted according to whether its output is the required temperature. Here the input to the switch depends on the deviation of the actual temperature from the required temperature, which is being determined by a comparison element – the person in this case. Figure illustrates these two types of system.

**Actuation systems** are the elements of control systems which are responsible for transforming the output of a microprocessor or control system into a controlling action on a machine or device. Thus, for example, we might have an electrical output from the controller which has to be transformed into a linear motion to move a load. Another example might be where an electrical output from the controller has to be transformed into an action which controls the amount of liquid passing along a pipe.

#### **Closed form and probabilistic path planning**

Probabilistic Path Planner's (PPP) can be generally described, without considering any specific type of a robot. The idea here during constructing a roadmap is that a data structure is incrementally constructed in a probabilistic way, and the same data structure during querying, is further used for solving individual path planning problems.

The data-structure constructed during the roadmap construction is an undirected graph  $G = (V, E)$ , where the nodes  $V$  are probabilistically generated and the edges  $E$  represent the (simple) feasible paths. These simple paths, are referred to as local paths, which are computed by a local planner. A local planner  $L$  is a function which takes two configurations as input arguments, and returns an edge (path) connecting them, which is feasible without any obstacles. Guaranteed probabilistic completeness of the global planner is achieved when proper choice of the local planner is carried out. If, given two configurations  $x$  and  $y$ , and the path  $L(x, y)$  is collision-free, then we say that  $L$  connects from  $x$  to  $y$ .

During querying, given a start configuration  $x$  and a goal configuration  $y$ , we try to connect  $x$  and  $y$  to suitable nodes  $s$  and  $g$  in  $V$ . Then we perform a graph search to find a sequence of edges in  $E$  connecting  $s$  to  $g$ , and then we transform this sequence into the most feasible path. Thus, the paths generated in the query phase are basically just concatenations of local paths, and therefore the properties of the global paths are introduced by the local planner.

## Checkpoint (1 of 2)

### Multiple choice questions:

1. Which of these can be called a system.
  - a) Human body
  - b) Pick and place robot
  - c) Graph database
  - d) All of these
  
2. Who coined the term robot.
  - a) Karel Capek
  - b) Romi K
  - c) Isac Asimo
  - d) None of the above
  
3. NHC means\_\_\_\_\_.
  - a) Nested Home Controller
  - b) Nested Hierarchical Controller
  - c) New Home controller
  - d) None of the above

Figure 1-30. Checkpoint (1 of 2)

AIR011

### Notes:

Write your answers here:

- 1.
- 2.
- 3.



## Checkpoint (2 of 2)

Fill in the blanks:

1. \_\_\_\_\_ architecture are best suited for semi-autonomous controlled robots
2. Device attached to the robot's wrist to perform a specific task are called \_\_\_\_\_
3. A robot, which has two parallel rotary joints to provide compliance in a plane is called \_\_\_\_\_.
4. \_\_\_\_\_ is the ability of a robot to repeatedly position itself when asked to perform a task multiple times.

True or False:

1. Reactive paradigm did not have planning or any reasoning functions. True/False
2. The Subconscious Thought module receives the information from the Sensory System and conveys commands to the Muscular System. True/False
3. Kinematics means the study of motion without regards to the forces or torques. True/False

Figure 1-31. Checkpoint (2 of 2)

AIR011.0

### Notes:

Write your answers here:

Fill in the blanks:

- 1.
- 2.
- 3.
- 4.

True or false:

- 1.
- 2.
- 3.

# Question bank

## Two mark questions:

1. Name the main components of industrial robots.
2. What parts does the DOD's robotic vehicle have?
3. What is an agent?
4. What is a system?

## Four mark questions:

1. Discuss digital camera's system representation.
2. Describe open and closed loop systems.
3. Explain JAUGS reference architecture.
4. Write short note on mobile ground robots?

## Eight mark questions:

1. Explain the concept of joint link transformations.
2. Explain the five basic agents in brief.

Figure 1-32. Question bank

AIR01

## Notes:

# Unit summary



IBM ICE (Innovation Centre for Education)

**Having completed this unit, you should be able to:**

- Understand the concept of system modeling
- Gain knowledge on the process of system design
- Understand the goals and principles of intelligent systems
- Gain an insight into various types of robots as systems

---

Figure 1-33. Unit summary

AIR011.0

## Notes:

Unit summary is as stated above.

## Unit 2. Artificial intelligence for robotics engineering

### What this unit is about

This unit helps to provide an understanding on the basics of Artificial Intelligence (AI). This unit helps to gain knowledge on the AI problems and techniques. The concept of state space search and production systems is also presented here. This unit provides an understanding on heuristic search techniques and knowledge representation.

### What you should be able to do

After completing this unit, you should be able to:

- Gain knowledge on the basics of Artificial Intelligence (AI)
- Gain an insight into the AI problems and techniques
- Learn about the state space search and production systems
- Understand the concept of problem characteristics and search paradigm
- Learn about heuristic search techniques and knowledge representation

### How you will check your progress

- Checkpoints

### References

IBM Knowledge Center

## Unit objectives

After completing this unit, you should be able to:

- Gain knowledge on the basics of Artificial Intelligence (AI)
- Gain an insight into the AI problems and techniques
- Learn about the state space search and production systems
- Understand the concept of problem characteristics and search paradigm
- Learn about heuristic search techniques and knowledge representation

Figure 2-1 Unit objectives

### **Notes:**

Unit objectives are as stated above.

AIRPORT

## Artificial Intelligence



IBM ICE (Innovation Centre for Education)

- Artificial Intelligence (AI) is the study and design of intelligent agents.
- AI is defined as the science and engineering of making machines intelligent.
- Science of making machines do things that would require intelligence.
- It is the intelligence of machines and the branch of computer science that aims to create it.
- It focuses on developing hardware and software systems that solve problems and accomplish tasks.

Figure 2-2. Artificial Intelligence

AIR011.0

### Notes:

#### Artificial Intelligence:

"Artificial Intelligence (AI) is the study and design of intelligent agents. AI is defined as the science and engineering of making machines intelligent with the help of intelligent agents programs". AI is concerned with the design of intelligence in an artificial artifacts and artificial devices. The artificial devices are designing by integrating intelligence into them. We have also seen numerous definitions on AI. It is a branch of computer science which aims in creating intelligence where intelligence is a computational part of machines to achieve goals.

The goal of AI is to make machines (example: computers) do tasks which currently human beings do better. Intelligence is related to tasks demanding higher mental processes such as creativity, problem solving pattern recognition, classification, learning, induction deduction, language understanding etc. Some intelligent behaviors include perceiving environment, acting in a complex world, reasoning to solve problems, discover hidden knowledge, thinking abstractly using analogies (comparison of similarity and differences). Generally, the purpose of AI is to enable computers behave like human beings and imitate the reasoning power of human beings.

In a way, artificial intelligence study is attempting to understand and build of intelligent agents. Artificial intelligence is a branch of the field of computer and information science. It focuses on developing hardware and software systems that solve problems and accomplish tasks that if accomplished by humans would be considered a display of intelligence. The field of AI includes studying and developing machines such as robots, automatic pilots for airplanes and spaceships, and smart military weapons. Europeans tend to use the term Machine Intelligence (MI) instead of the term AI.

## **Student Notebook**

---

The theory and practice of AI is leading to the development of a wide range of artificially intelligent tools. These tools working under the guidance of human, sometimes without external guidance and are also able to solve or help to solve a steadily increasing range of problems. Over the past 50 years, ai has produced a number of results that are important to students, teachers, our overall educational system, and to our society. AI is related to tasks involving higher mental processes.

For example in creativity, solving problems, pattern recognition, classification, learning, induction, deduction, building analogies, optimization, language processing, knowledge and many more. AI is the computational part of the ability to achieve goals.



IBM ICE (Innovation Centre for Education)

## Well-known definitions for AI

- Intelligence is the computational part of the ability to achieve goals in the world.
- Varying kinds and degrees of intelligence occur in people, many animals.
- AI is the study of the mental faculties through the use of computational models.
- An intelligent agent is a system that studies and acts on the basis of environment.

Figure 2-3. Well-known definitions for AI

AIR011.0

### Notes:

#### The following are some of the well-known definitions for AI:

In the early 1960s, Marvin Minsky indicated that "Artificial Intelligence is the science of making machines do things that would require intelligence if done by men." In *Unified Theories of Cognition*, Allen Newell defines intelligence as the degree to which a system approximates a knowledge-level system. Perfect intelligence is defined as the ability to bring all the knowledge a system has at its disposal to bear in the solution of a problem (which is synonymous with goal achievement). This may be distinguished from ignorance, a lack of knowledge about a given problem space.

AI is the computer-based exploration of methods for solving challenging tasks that have traditionally depended on people for solution. Such tasks include complex logical inference, diagnosis, visual recognition and comprehension of natural language, game playing, explanation, and planning. Artificial Intelligence (AI) is the intelligence of machines and the branch of computer science that aims to create it. Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines. AI is the study of the mental faculties through the use of computational models. AI is the study of how to make computers do things which, at the moment, people do better. AI is the study and design of intelligent agents, where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success.

## Birth of AI



IBM ICE (Innovation Centre for Education)

- Another influential figure in AI is John McCarthy.
- McCarthy and Minsky, Claude Shannon, and Nathaniel Rochester brought together US researchers interested in the study of intelligence.
- Two researchers from Carnegie Tech, Allen Newell and Herbert Simon created a reasoning program, the logic theorist (LT).

Figure 2-4. Birth of AI

AIR011.0

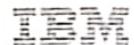
### Notes:

#### The birth of artificial intelligence:

Another influential figure in AI is John McCarthy. After graduation, McCarthy moved to Dartmouth College, which was to become the official birthplace of the field. McCarthy convinced Minsky, Claude Shannon, and Nathaniel Rochester to help him bring together US researchers interested in automata theory, neural nets, and the study of intelligence. Two researchers from Carnegie Tech, Allen Newell and Herbert Simon, rather stole the show. Although the others had ideas and in some cases programs for particular applications such as checkers, Newell and Simon already had a reasoning program, the logic theorist (LT), about which Simon claimed, "we have invented a computer program, a capable mind-body problem". Soon after the workshop, the program was able to prove most of the theorems of Russell and Whitehead's Principia Mathematica.

Russell was reportedly delighted when Simon showed him that the program had come up with a proof for one theorem that was shorter than the one in Principia. The editors of the journal of symbolic logic were less impressed; they rejected a paper coauthored by Newell, Simon, and Logic Theorist. The Dartmouth workshop did not lead to any new breakthroughs, but it did introduce all the major figures to each other. For the next 20 years, the field is dominated by these people and their students and colleagues at MIT, CMU, Stanford, and IBM. Perhaps the longest-lasting thing to come out of the workshop was an agreement to adopt McCarthy's new name for the field: Artificial Intelligence. Perhaps "computational rationality" would have been better, but "AI" has stuck.

## Basic terminologies



IBM ICE (Innovation Centre for Education)

- An agent is something that perceives and acts in an environment.
- The performance measure evaluates the behavior of the agent in an environment.
- The agent program implements the agent function.
- Goal directed agent is a type of intelligent agent.

Figure 2-5. Basic terminologies

AIR011.0

### Notes:

#### Basic terminologies:

An agent is something that perceives and acts in an environment. The agent function for an agent specifies the action taken by the agent in response to any percept sequence. A rational agent acts so as to maximize the expected value of the performance measure, given the percept sequence it has seen so far. A task environment specification includes the performance measure, the external environment, the actuators, and the sensors. In designing an agent, the first step must always be to specify the task environment as fully as possible. Task environments vary along several significant dimensions. They can be fully or partially observable, deterministic or stochastic, episodic or sequential, static or dynamic, discrete or continuous, and single-agent or multi-agent. The agent program implements the agent function. There exists a variety of basic agent-program designs, reflecting the kind of information made explicit and used in the decision process. The designs vary in efficiency, compactness, and flexibility.

The appropriate design of the agent program depends on the nature of the environment. Simple reflex agents respond directly to percepts, whereas model-based reflex agents maintain internal state to track aspects of the world that are not evident in the current percept. Goal-based agents act to achieve their goals, and utility-based agents try to maximize their own expected "happiness". Goal directed agent is a type of intelligent agent. A goal directed agent needs to achieve certain goals. Such an agent selects its actions based on the goal it has. Many problems can be represented as a set of states and a set of rules of how one state is transformed to another. Each state is an abstract representation of the agent's environment. It is an abstraction that denotes a configuration of the agent.

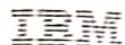
Any agent has:

- Initial state: the description of the starting configuration of the agent.
- An action/ operator take the agent from one state to another state. A state can have any number of successor states.
- A plan is a sequence of actions.
- A goal is a description of a set of desirable states of the world. Goal states are often specified by a goal test in which any goal state must satisfy.

Let us look at few examples of goal directed agents:

- 15-puzzle: the goal of an agent working on a 15-puzzle problem may be to reach a configuration which satisfies the condition that the top row has the tiles 1, 2 and 3. The details of this problem will be described later.
- The goal of an agent may be to navigate a maze and reach the home position. The agent must choose a sequence of actions to achieve the desired goal.
- The agent must choose a sequence of actions to achieve the desired goal.
- All agents can improve their performance through learning.

## Applications of AI



IBM ICE (Innovation Centre for Education)

- Applications of AI:
  - Autonomous planning and scheduling.
  - Game playing.
  - Autonomous control.
  - Diagnosis.
  - Logistics planning.
  - Robotics.
  - Language understanding and problem solving.

Figure 2-6 Applications of AI

AIR011.0

### Notes:

#### Autonomous planning and scheduling:

A hundred million miles from earth, NASA's remote agent program became the first on-board autonomous planning program to control the scheduling of operations for a spacecraft. Remote agent generated plans from high-level goals specified from the ground, and it monitored the operation of the spacecraft as the plans were executed-detecting, diagnosing, and recovering from problems as they occurred.

#### Game playing:

IBM's deep blue became the first computer program to defeat the world champion in a chess match when it bested Garry kasparov by a score of 3.5 to 2.5 in an exhibition match. Kasparov said that he felt a "new kind of intelligence" across the board from him.

#### Autonomous control:

- The Alvin computer vision system was trained to steer a car to keep it following a lane.
- It was placed in CMU's navlab computer-controlled minivan and used to navigate across the united states-for 2850 miles it was in control of steering the vehicle 98% of the time.
- A human took over the other 2%, mostly at exit ramps.

- Navlab uses video cameras that transmit road images to Alvin, which then computes the best direction to steer, based on experience from previous training runs.

**Diagnosis:**

- Medical diagnosis programs based on probabilistic analysis have been able to perform at the level of an expert physician in several areas of medicine.
- Heckerman describes a case where a leading expert on lymph-node pathology scoffs at a program's diagnosis of an especially difficult case.
- The creators of the program suggest he ask the computer for an explanation of the diagnosis.
- The machine points out the major factors influencing its decision and explains the subtle interaction of several of the symptoms in this case.
- Eventually, the expert agrees with the program.

**Logistics planning:**

- During the Persian gulf crisis of 1991, U.S. Forces deployed a dynamic analysis and replanning tool, DART, to do automated logistics planning and scheduling for transportation.
- This involved up to 50,000 vehicles, cargo, and people at a time, and had to account for starting points, destinations, routes, and conflict resolution among all parameters.
- The AI planning techniques allowed a plan to be generated in hours that would have taken weeks with older methods.
- The Defense Advanced Research Project Agency (DARPA) stated that this single application more than paid back DARPA's 30-year investment in AI.

**Robotics:**

- Many surgeons now use robot assistants in microsurgery.
- Hipnav is a system that uses computer vision techniques to create a three-dimensional model of a patient's internal anatomy and then uses robotic control to guide the insertion of hip replacement prosthesis.

**Language understanding and problem solving:**

Proverb is a computer program that solves crossword puzzles better than most humans, using constraints on possible word fillers, a large database of past puzzles, and a variety of information sources including dictionaries and online databases such as a list of movies and the actors that appear in them.



## The AI problems and techniques

- What is an AI technique?
- Properties of knowledge which are less desirable.
- A problem can be defined formally by following components:
  - ⇒ The initial state.
  - ⇒ A description of the possible actions.
  - ⇒ The goal test.
  - ⇒ A path cost.
  - ⇒ A solution.

Figure 2-7. The AI problems and techniques

AIR0115

### Notes:

#### AI problems and techniques:

Artificial intelligence techniques are pervasive and are too numerous to list and AI problems span a very broad spectrum. Frequently, when a technique reaches mainstream use, it is no longer. This phenomenon is described as the AI effect. AI techniques appear to have very little in common except that they are hard. For all varieties of problems corresponding solutions are available. The concept of intelligence is not independent but it depends on the concept of knowledge, i.e., We should have knowledge to apply intelligence.

The following are the properties of knowledge which are less desirable:

- It is voluminous
- It is hard to characterize
- It is dynamic i.e. Constantly changing
- It differs from data

#### Problems and problem spaces

A problem can be defined formally by following components:

- The initial state that the agent starts in.
- A description of the possible actions available to the agent.

The most common formulation uses a successor function. Together, the initial state and successor function implicitly define the state space of the problem—the set of all states reachable from the initial state. The state space forms a graph in which the nodes are states and the arcs between nodes are actions. A path in the state space is a sequence of states connected by a sequence of actions. The goal test, which determines whether a given state is a goal state. Sometimes there is an explicit set of possible goal states, and the test simply checks whether the given state is one of them.

For example, in chess, the goal is to reach a state called "checkmate", where the opponent's king is under attack and cannot escape. A path cost function that assigns a numeric cost to each path. The problem-solving agent chooses a cost function that reflects its own performance measure. A solution to a problem is a path from the initial state to a goal state. Solution quality is measured by the path cost function, and an optimal solution has the lowest path cost among all solutions.

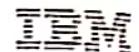
2-12 AIR01

---

© Copyright IBM Corp. 2019

Course materials may not be reproduced in whole or in part  
without the prior written permission of IBM.

## Real world problems



IBM ICE (Innovation Centre for Education)

- Any real-world problem specified by the following:
  - States.
  - Initial state.
  - Successor function.
  - Goal test.
  - Path cost.
- Example problems:
  - TSP: traveling salesperson problem.
  - Routing in computer networks.
  - VLSI layout problem.
  - Robot navigation.

Figure 2-8. Real world problems

AIR0110

### Notes:

#### Real world problems:

We have already seen how the route-finding problem is defined in terms of specified locations and transitions along links between them. Route-finding algorithms are used in a variety of applications, such as routing in computer networks, military operations planning, and airline travel planning systems. These problems are typically complex to specify.

Consider a simplified example of an airline travel problem specified as follows:

- **States:** Each is represented by a location (e.g., An airport) and the current time.
- **Initial state:** This is specified by the problem.
- **Successor function:** This returns the states resulting from taking any scheduled flight (perhaps further specified by seat class and location), leaving later than the current time plus the within-airport transit time, from the current airport to another.
- **Goal test:** Are we at the destination by some pre-specified time?

**Path cost:** This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on. Commercial travel advice systems use a problem formulation of this kind, with many additional complications to handle the byzantine fare structures that airlines impose. Any seasoned traveler knows, however, that not all air travel goes according to plan. A really good system should include contingency plans-such as backup reservations on alternate flights to the extent that these are justified by the cost and likelihood of failure of the original plan.

**TSP (Traveling Salesperson Problem):**

The traveling salesperson problem is a touring problem in which each city must be visited exactly once. The aim is to find the *shortest* tour. The problem is known to be np-hard, but an enormous amount of effort has been expended to improve the capabilities of tsp algorithms. In addition to planning trips for traveling salespersons, these algorithms have been used for tasks such as planning movements of automatic circuit-board drills and of stocking machines on shop floors.

**VLSI layout problem:**

A VLSI layout problem requires positioning millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield. The layout problem comes after the logical design phase, and is usually split into two parts:

- Cell layout.
- Channel routing.

In cell layout, the primitive components of the circuit are grouped into cells, each of which performs some recognized function. Each cell has a fixed footprint (size and shape) and requires a certain number of connections to each of the other cells. The aim is to place the cells on the chip so that they do not overlap and there is room for the connecting wires to be placed between the cells. Channel routing finds a specific route for each wire through the gaps between the cells.

**Robot navigation:**

Robot navigation is a generalization of the route-finding problem described earlier. Rather than a discrete set of routes, a robot can move in a continuous space with (in principle) an infinite set of possible actions and states. For a circular robot moving on a flat surface, the space is essentially two-dimensional. When the robot has arms and legs or wheels that must also be controlled, the search space becomes many-dimensional. Advanced techniques are required just to make the search space finite.

## Explicit vs. Implicit state space

- Explicit state space:
  - This would require a table such as the following

State	Action	Resulting State
$s_0$	$act_{s_0}$	$s_1$
$s_0$	$act_{s_1}$	$s_2$
$s_0$	$act_{s_2}$	$s_3$
$s_0$	$act_{s_3}$	—

- To generate the state space implicitly, the agent needs to know both initial state and operators.

Figure 2-10. Explicit vs. Implicit state space

AIR0110

### Notes:

#### Explicit vs. Implicit state space

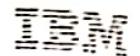
##### Explicit state space:

One possible representation of the effect and precondition of actions is to explicitly enumerate the states and, for each state, specify the actions that are possible in that state and, for each state-action pair, specify the state that results from carrying out the action in that state. This would require a table as shown above. The first tuple in this relation specifies that it is possible to carry out action  $act_{s_0}$  in state  $s_0$ , and, if it were to be carried out in state  $s_0$ , the resulting state would be  $s_1$ . Thus, this is the explicit representation of the actions in terms of a graph. This is called a **state-space graph**.

But this is not a good representation for three main reasons.

- There are usually too many states to represent, to acquire, and to reason with.
- Small changes to the model mean a large change to the representation. Modeling another feature means changing the whole representation. For example, to model the level of power in the robot, so that it can recharge itself in the lab, every state has to change.
- There is also usually much more structure and regularity in the effects of actions. This structure can make the specification of the preconditions and the effects of actions, and reasoning about them, more compact and efficient.

## State space search



IBM ICE (Innovation Centre for Education)

- State space search is a process used in the field of computer science, including Artificial Intelligence (AI)
- A state space search includes:
  - A state.
  - An operator.
  - The initial state.
  - The goal state.
- Explicit state space: One possible representation of the effect and precondition of actions is to explicitly enumerate the states.
- Search is the process of considering various possible sequences of operators applied to the initial state and finding out a sequence which culminates in a goal state.

Figure 2-9 State space search

AIR011.0

### Notes:

#### State space search:

State space search is a process used in the field of computer science, including AI, in which successive configurations or states of an instance are considered, with the goal of finding a goal state with a desired property. The problems are often modeled as a state space, a set of states that a problem can be in. The set of states forms a graph where two states are connected if there is an operation that can be performed to transform the first state into the second. The state space search often differs from traditional computer science search methods because the state space is implicit: the typical state space graph is too large to generate and store in memory. Instead, nodes are generated as they are explored, and typically discarded thereafter. A solution to a combinatorial search instance may consist of the goal state itself, or of a path from some initial state to the goal state. Hence, we formulate a problem as a state space search by showing the legal problem states, the legal operators, and the initial and goal states.

Typically, a state space search includes:

- A state which is defined by the specification of the values of all attributes of interest in the world.
- An operator that changes one state into the other; it has a precondition which is the value of certain attributes prior to the application of the operator, and a set of effects, which are the attributes altered by the operator .
- The initial state is where you start.
- The goal state is the partial description of the solution.

## Explicit vs. Implicit state space

- Explicit state space:
  - This would require a table such as the following:

State	Action	Resulting State
$s_7$	$act_{47}$	$s_{54}$
$s_7$	$act_{14}$	$s_{23}$
$s_{54}$	$act_5$	$s_{10}$
—	—	—

- To generate the state space implicitly, the agent needs to know both initial state and operators.

Figure 2-10. Explicit vs. Implicit state space

AIR011.0

### Notes:

#### Explicit vs. Implicit state space

##### Explicit state space:

One possible representation of the effect and precondition of actions is to explicitly enumerate the states and, for each state, specify the actions that are possible in that state and, for each state-action pair, specify the state that results from carrying out the action in that state. This would require a table as shown above. The first tuple in this relation specifies that it is possible to carry out action  $act_{47}$  in state  $s_7$ , and, if it were to be carried out in state  $s_7$ , the resulting state would be  $s_{54}$ . Thus, this is the explicit representation of the actions in terms of a graph. This is called a **state-space graph**.

But this is not a good representation for three main reasons:

- There are usually too many states to represent, to acquire, and to reason with.
- Small changes to the model mean a large change to the representation. Modeling another feature means changing the whole representation. For example, to model the level of power in the robot, so that it can recharge itself in the lab, every state has to change.
- There is also usually much more structure and regularity in the effects of actions. This structure can make the specification of the preconditions and the effects of actions, and reasoning about them, more compact and efficient.

The state space may be explicitly represented by a graph. But more typically the state space can be implicitly represented and generated when required. To generate the state space implicitly, the agent needs to know:

- The initial state.
- The operators and a description of the effects of the operators.

An operator is a function which "expands" a node and computes the successor node(s). In the various formulations of the n-queen's problem, note that if we know the effects of the operators, we do not need to keep an explicit representation of all the possible states, which may be quite large. A production system (or production rule system) is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about behavior. These rules, termed productions, are a basic representation found useful in automated planning, expert systems and action selection. A production system provides the mechanism necessary to execute productions in order to achieve some goal for the system.

## State space search notations



IBM ICE (Innovation Centre for Education)

- An initial state is agent's configuration at the beginning.
- An *action* or an operator takes to the successor state.
- A state can have a number of successor states.
- A *plan* is a sequence of actions.
- The path cost represents cost of a plan.

Figure 2-11. State space search notations

AIR011.0

### Notes:

#### State space search notations

An initial state is the description of the starting configuration of the agent. A *plan* is a sequence of actions. The cost of a plan is referred to as the path cost. The path cost is a positive number, and a common path cost may be the sum of the costs of the steps in the path. Now let us look at the concept of a search problem. Problem formulation means choosing a relevant set of states to consider, and a feasible set of operators for moving from one state to another. *Search* is the process of considering various possible sequences of operators applied to the initial state and finding out a sequence which culminates in a goal state.

# Search problem and problem space

IBM ICE (Innovation Centre for Education)

- Search plays a major role in solving many AI problems.
- The problems that are addressed by AI search algorithms fall into three general classes:
  - Single-agent path-finding problems.
  - Two-player games, and
  - Constraint-satisfaction problems.
- Problem space represents connections between states.
- Schematic representation of search problem:

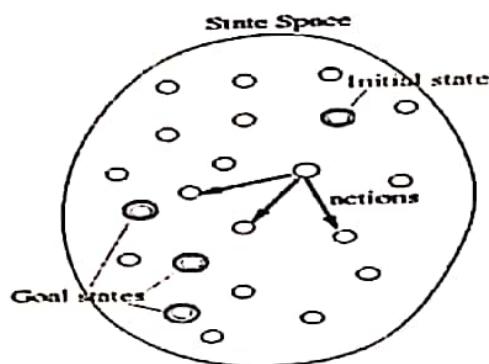


Figure 2-12. Search problem and problem space

AIR011.0

## Notes:

### Search problem

Search plays a major role in solving many AI problems. Search is a universal problem-solving mechanism in AI. In many problems, sequence of steps required to solve is not known in advance but must be determined by systematic trial-and-error exploration of alternatives. The problems that are addressed by AI search algorithms fall into three general classes: single-agent path-finding problems, two-player games, and constraint-satisfaction problems

**Single-agent path-finding problems:** Classic examples in the AI literature of path-finding problems are sliding-tile puzzles, Rubik's cube and theorem proving. The single-tile puzzles are common test beds for research in AI search algorithms as they are very simple to represent and manipulate. Real-world problems include the traveling salesman problem, vehicle navigation, and the wiring of VLSI circuits. In each case, the task is to find a sequence of operations that map an initial state to a goal state.

**Two-player games:** Two-player games are two-player perfect information games. Chess, checkers, and Othello are some of the two-player games. **Constraint satisfaction problems:** eight queens' problem is the best example. The task is to place eight queens on an 8\*8 chessboard such that no two queens are on the same row, column or diagonal. Real-world examples of constraint satisfaction problems are planning and scheduling applications.

### **Problem space**

Problem space is a set of states and the connections between them to represent the problem. Problem space graph is used to represent a problem space. In the graph, the states are represented by nodes of the graph, and the operators by edges between nodes. Although most problem spaces correspond to graphs with more than one path between a pair of nodes, for simplicity they are often represented as trees, where the initial state is the root of the tree. The cost of the simplification is that any state that can be reached by two different paths will be represented by duplicate nodes thereby increasing the tree size.

The benefit of using tree is that the absence of cycles greatly simplifies many search algorithms:

- We are now ready to formally describe a search problem which is schematically depicted in the following figure.
- The search problem is to find a sequence of actions which transforms the agent from the initial state to a goal state  $g \rightarrow g$ . A search problem is represented by a 4-tuple  $\{S, s_0, A, G\}$ .
- Here,  $s$  is the set of states,  $s_0 \rightarrow s$ , is an initial state,  $a: s \rightarrow s$  are operators/ actions that transform one state to another state and  $g$  is a goal, a set of states.  $G \rightarrow s$ .
- This sequence of actions is called a solution plan. It is a path from the initial state to a goal state.
- A plan  $p$  is a sequence of actions. i.e.,  $P = \{a_0, a_1, \dots, a_n\}$  which leads to traversing a number of states  $\{s_0, s_1, \dots, s_{n+1}, g\}$ . A sequence of states is called a path.
- The cost of a path is a positive number.
- In many cases the path cost is computed by taking the sum of the costs of each action.

## Representation of search problems

- A search problem is represented using a directed graph.
- Searching process: the generic searching process can be very simply described in terms of the following steps:

Do until a solution is found or the state space is exhausted.

1. Check the current state.
2. Execute allowable actions to find the successor states.
3. Pick one of the new states.
4. Check if the new state is a solution state.
5. If it is not, the new state becomes the current state and the process is repeated.

Figure 2-13. Representation of search problems

AIR011.0

### Notes:

#### Representation of search problems

A search problem is represented using a directed graph:

- The states are represented as nodes.
- The allowed actions are represented as arcs.
- We are now ready to formally describe a search problem which is schematically depicted in the above figure.

## State space search example

- We will now illustrate the searching process with the help of an example. Consider the problem depicted in Figure.

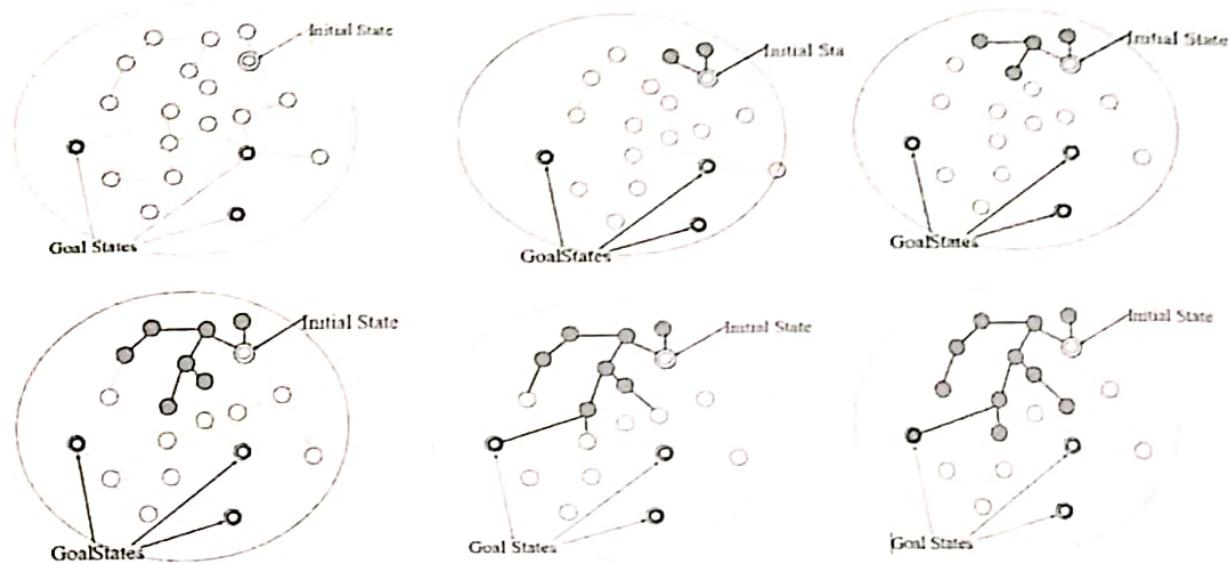


Figure 2-14. State space search example

AIR011.0

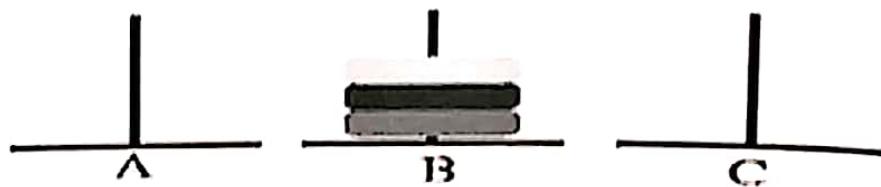
### Notes:

#### Examples

Here,  $s_0$  is the initial state. The successor states are the adjacent states in the graph. There are three goal states. The two successor states of the initial state are generated. The successors of these states are picked, and their successors are generated. Successors of all these states are generated. The successors are generated. A goal state has been found. The above example illustrates how we can start from a given state and follow the successors and be able to find solution paths that lead to a goal state. The grey nodes define the search tree. Usually the search tree is extended one node at a time. The order in which the search tree is extended depends on the search strategy.

## Pegs and disks problem (1 of 4)

- Figure: Pegs and Disks problem.



- Figure: The initial state is illustrated below.

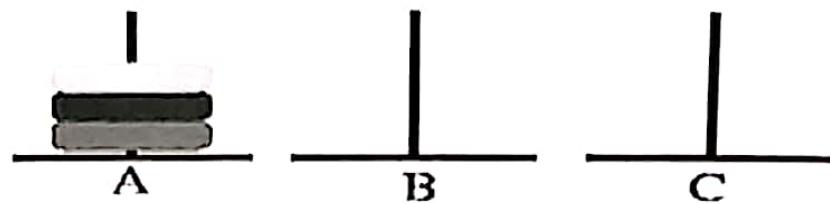


Figure 2-15 Pegs and disks problem (1 of 4)

AIR0110

### Notes:

#### Example problems

##### Pegs and disks problem

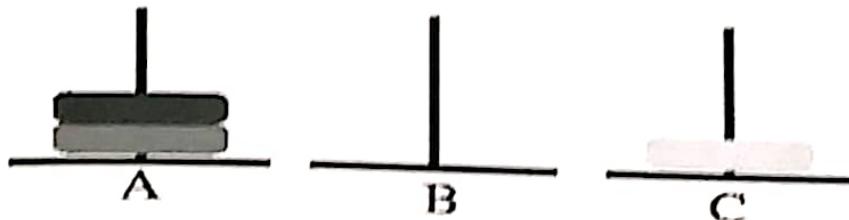
We have 3 pegs and 3 disks. Operators: one may move the topmost disk on any needle to the topmost position to any other needle. In the goal state all the pegs are in the needle b as shown in the figure1. Initial state of the peg and disk problem in figure2.

## Pegs and disks problem (2 of 4)

IBM

IBM VCE (Instructional Materials for Education)

- Now we will describe a sequence of actions that can be applied on the initial state.
- Step 1: Move A  $\rightarrow$  C



- Step 2: Move A  $\rightarrow$  B

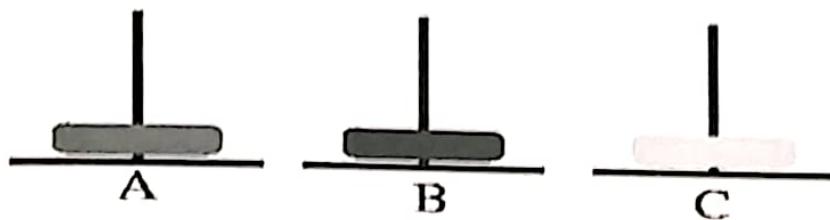


Figure 2-16. Pegs and disks problem (2 of 4)

AIR011.0

### Notes:

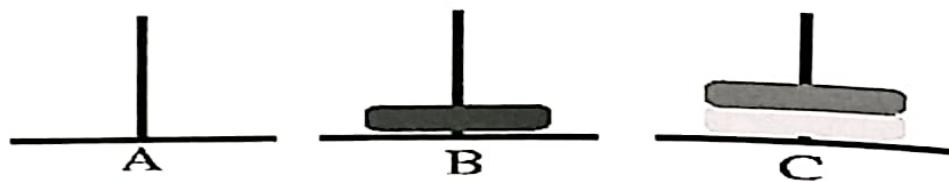
Now we will describe a sequence of actions that can be applied on the initial state.

Step 1: move A  $\rightarrow$  C, peg and disk status after movement.

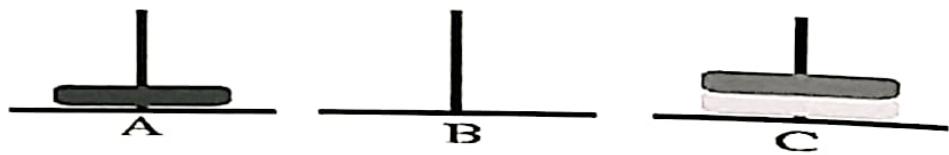
Step 2: move A  $\rightarrow$  B, peg and disk status after movement.

## Pegs and disks problem (3 of 4)

- Step 3: Move A → C



- Step 4: Move B → A



- Step 5: Move C → B

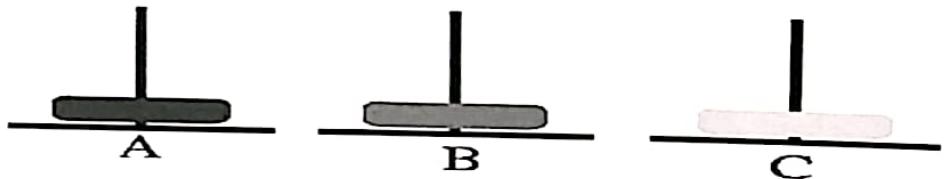


Figure 2-17. Pegs and disks problem (3 of 4)

AIR0110

### Notes:

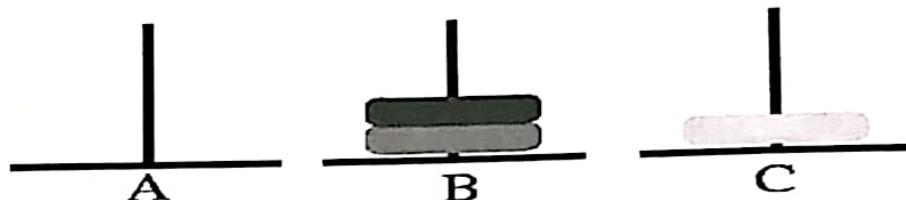
Step 3: move A → C, peg and disk status after movement.

Step 4: move B → A, peg and disk status after movement.

Step 5: move C → B, peg and disk status after movement.

## Pegs and disks problem (4 of 4)

- Step 6: Move A → B



- Step 7: Move C → B

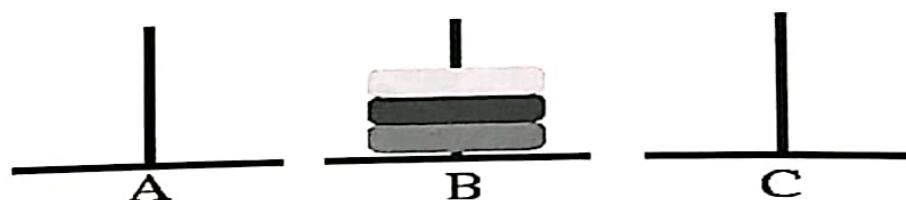


Figure 2-18. Pegs and disks problem (4 of 4)

AIR011.0

### Notes:

Step 6: move A → B, peg and disk status after movement.

Step 7: move C → B, peg and disk status after movement.

## 8 queen's problem

- Eight Queen's problem:

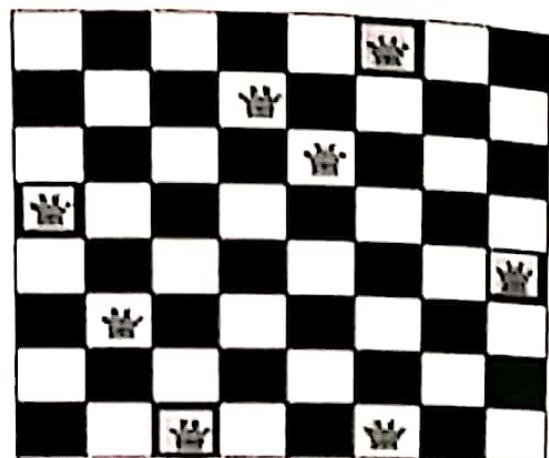
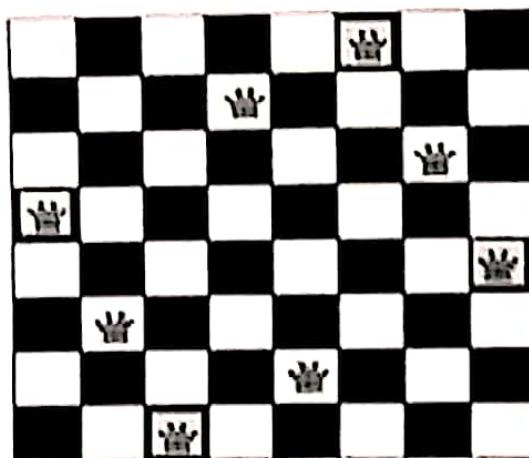


Figure 2-10 8 queen's problem

Albert

### Notes:

#### Eight queen's problem:

The problem is to place 8 queens on a chessboard so that no two queens are in the same row, column or diagonal. The picture shows a solution of the 8-queens problem. The picture on the right is not a valid solution, because some of the queens are attacking each other. How do we formulate this in problem terms of a state-space search problem? The problem formulation involves deciding the representation of the states, selecting the initial state representation, the description of the operators, and the successor states. We will show in next, that we can formulate the search problem in several different ways for this problem.

## N queens problem formulation (1 of 3)

- In this problem, initially there will not be any queen in the board.

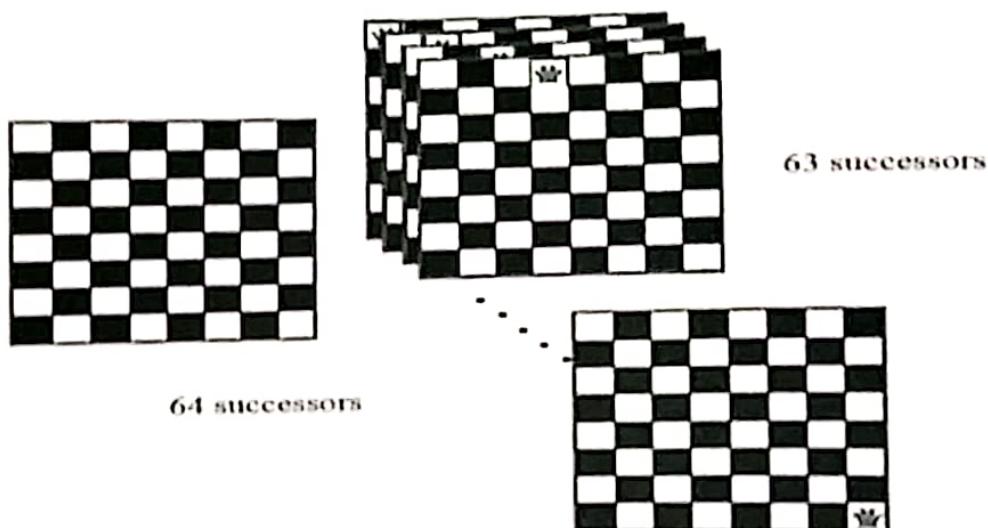


Figure 2-20. N queens problem formulation (1 of 3)

AIR011.0

### Notes:

#### N queens' problem formulation:

- States:** Any arrangement of 0 to 8 queens on the board.
- Initial state:** 0 queens on the board.
- Successor function:** Add a queen in any square.
- Goal test:** 8 queens on the board, none are attacked.

The initial state has 64 successors. Each of the states at the next level has 63 successors, and so on. We can restrict the search tree somewhat by considering only those successors where no queen is attacking each other. To do that, we have to check the new queen against all existing queens on the board. The solutions are found at a depth of 8.

## N queens problem formulation (2 of 3)

- In this problem, all the queens will be at column 1 as shown in the figure:

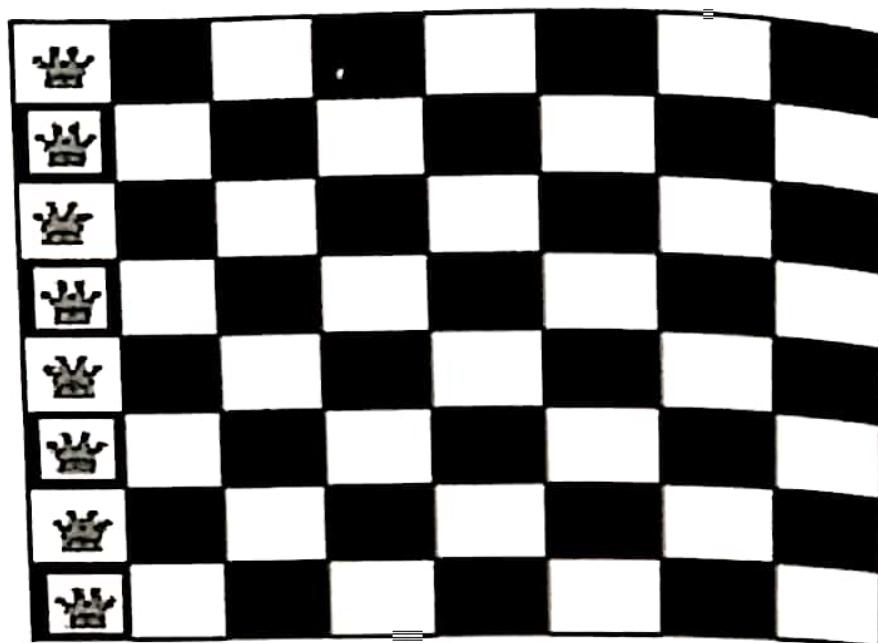


Figure 2-21 N queens problem formulation (2 of 3)

### Notes:

#### N queens problem formulation:

- States:** Any arrangement of 8 queens on the board.
- Initial state:** All queens are at column 1.
- Successor function:** Change the position of any one queen.
- Goal test:** 8 queens on the board, none are attacked.

If we consider moving the queen at column 1, it may move to any of the seven remaining columns

## N queens problem formulation (3 of 3)

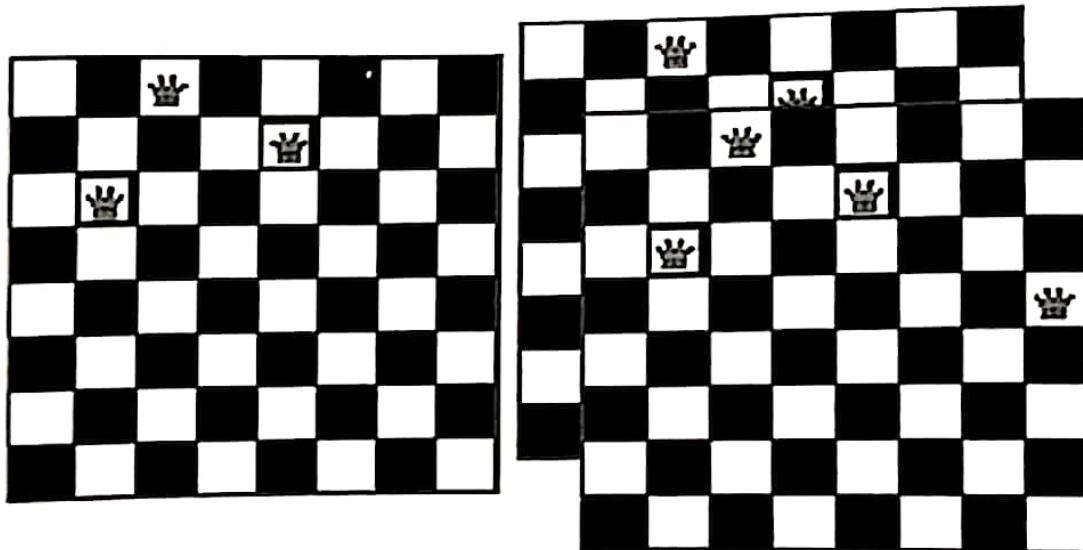


Figure 2-22. N queens problem formulation (3 of 3)

AIR011.0

### Notes:

#### N queens problem formulation:

- **States:** Any arrangement of  $k$  queens in the first  $k$  rows such that none are attacked.
- **Initial state:** 0 queens on the board.
- **Successor function:** It adds a queen to the  $(k+1)$ th row so that none are attacked.
- **Goal test:** 8 queens on the board, none are attacked.

## 8 puzzle problem

- 8 puzzle problem:
  - States
  - Operators/Action
  - Goal Test
  - Path Cost
- A small portion of the state space of 8-puzzle is shown below.

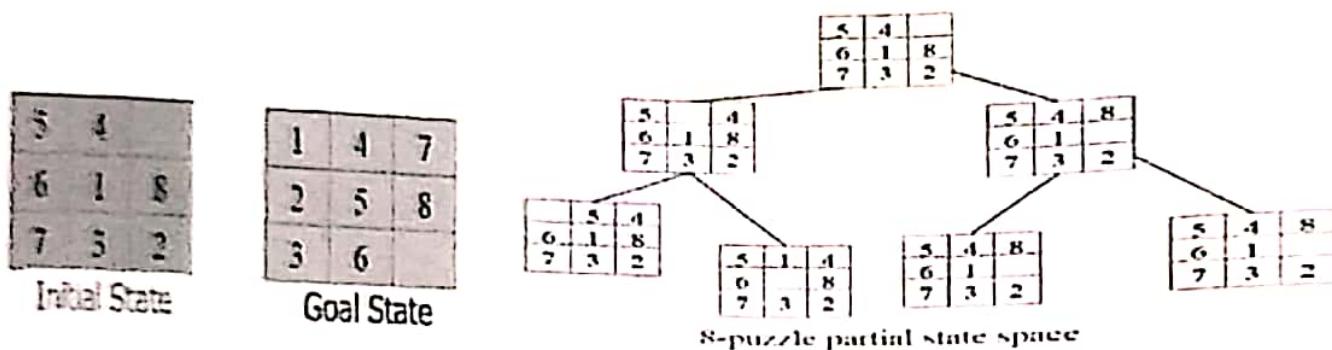


Figure 2-23 8 puzzle problem

AIR011.0

### Notes:

#### 8 puzzle problem:

In the 8-puzzle problem we have a  $3 \times 3$  square board and 8 numbered tiles. The board has one blank position. Blocks can be slid to adjacent blank positions. We can alternatively and equivalently look upon this as the movement of the blank position up, down, left or right. The objective of this puzzle is to move the tiles starting from an initial position and arrive at a given goal configuration. The 15-puzzle problems is similar to the 8-puzzle. It has a  $4 \times 4$  square board and 15 numbered tiles.

The state space representation for this problem is summarized below:

- **States:** A state is a description of each of the eight tiles in each location that it can occupy.
- **Operators/action:** The blank moves left, right, up or down.
- **Goal test:** The current state matches a certain state (e.g. One of the ones shown on previous slide).
- **Path cost:** Each move of the blank costs 1.

A small portion of the state space of 8-puzzle is shown in figure. Note that we do not need to generate all the states before the search begins. The states can be generated when required.

## State space search example: Tic-tac-toe problem

IBM

IBM iSeries Innovation Center for Education

- Tic-tac-toe is a game that involves two players who play alternately.
- Player one puts a X in an empty position.
- Player 2 places an O in an unoccupied position.
- The player who can first get three of his symbols in the same row, column or diagonal wins.
- Water jug problem:
  - Initial state.
  - Goal test.
  - Successor function.

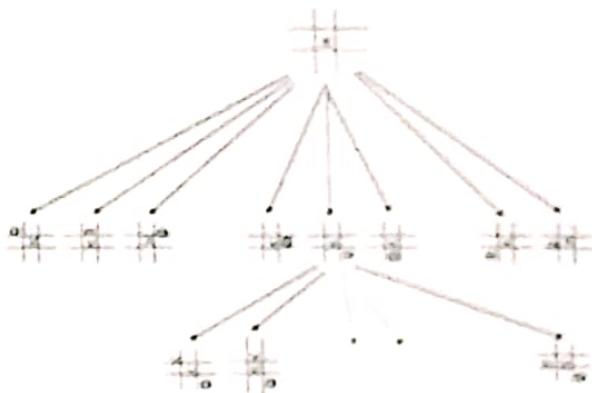


Figure 2-24. State space search example: Tic-tac-toe problem

AIR011.0

### Notes:

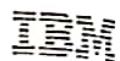
#### Tic-Tac-Toe problem:

Another example we will consider now is the game of tic-tac-toe. This is a game that involves two players who play alternately. Player one puts a X in an empty position. Player 2 places an O in an unoccupied position. The player who can first get three of his symbols in the same row, column or diagonal wins. A portion of the state space of tic-tac-toe is depicted above. Now that we have looked at the state space representations of various search problems, we will explore briefly the search algorithms later.

#### Water Jug problem:

You have three jugs measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You need to measure out exactly one gallon.

- **Initial state:** All three jugs are empty.
- **Goal test:** Some jug contains exactly one gallon.
- **Successor function:** Applying the **action** *transferto* jugs i and j with capacities  $C_i$  and  $C_j$  and containing  $G_i$  and  $G_j$  gallons of water, respectively, leaves jug i with  $\max(0, G_i - (C_j - G_j))$  gallons of water and jug j with  $\min(C_j, G_i + G_j)$  gallons of water. Applying the **action** *fill* to jug i leaves it with  $C_i$  gallons of water.
- **Cost function:** Charge one point for each gallon of water transferred and each gallon of water filled.



## Production systems (1 of 2)

- Production consist of two parts: A sensory precondition (or "IF" statement) and an action (or "THEN").
- The production is triggered when production's precondition and current state matches.
- A production system consists of four basic components:
  - A set of rules.
  - One or more knowledge databases.
  - A control strategy.
  - A rule applier.
- A production system consists of rules and factors. Knowledge is encoded in a declarative from which comprises of a set of rules of the form:
  - Situation → Action; SITUATION that implies ACTION.
  - Example: IF the initial state is a goal state THEN quit.

Figure 2-25. Production systems (1 of 2)

AIR011.0

### Notes:

#### Production systems:

Productions consist of two parts: a sensory precondition (or "if" statement) and an action (or "then"). If a production's precondition matches the current state of the world, then the production is said to be triggered. If a production's action is executed, it is said to have fired. A production system also contains a database, sometimes called working memory, which maintains data about current state or knowledge, and a rule interpreter. The rule interpreter must provide a mechanism for prioritizing productions when more than one is triggered.

#### A production system consists of four basic components:

A set of rules of the form  $c_i \rightarrow a_i$ , where  $c_i$  is the condition part and  $a_i$  is the action part. The condition determines when a given rule is applied, and the action determines what happens when it is applied. One or more knowledge databases that contain whatever information is relevant for the given problem. Some parts of the database may be permanent, while others may temporary and only exist during the solution of the current problem. The information in the databases may be structured in any appropriate manner. A control strategy that determines the order in which the rules are applied to the database and provides a way of resolving any conflicts that can arise when several rules match at once. A rule applier which is the computational system that implements the control strategy and applies the rules.

## Production systems (2 of 2)



IBM ICE (Innovation Centre for Education)

- The major components of an AI production system are:
  - A global database.
  - A set of production rules and
  - A control system.
- Advantages of production systems:
  - Provide an excellent tool for structuring AI programs.
- Disadvantages of production systems:
  - Very difficult to analyze the flow of control within a production system.
- Production systems describe the operations that can be performed in a search for a solution to the problem:
  - Monotonic production system.
  - Commutative production system.
  - Partially commutative, Monotonic.

Figure 2-26. Production systems (2 of 2)

AIR011.0

### Notes:

#### Production systems:

A production system consists of rules and factors. Knowledge is encoded in a declarative form which comprises of a set of rules of the form:

Situation → action, SITUATION that implies ACTION.

Example: If the initial state is a goal state then quit.

The major components of an AI production system are:

- A global database.
- A set of production rules.
- A control system.

The goal database is the central data structure used by an AI production system.

#### The production system

The production rules operate on the global database. Each rule has a precondition that is either satisfied or not by the database. If the precondition is satisfied, the rule can be applied.

Application of the rule changes the database. The control system chooses which applicable rule should be applied and ceases computation when a termination condition on the database is satisfied. If several rules are to fire at the same time, the control system resolves the conflicts. Four classes of production systems:

- A monotonic production system.
- A non-monotonic production system.
- A partially commutative production system.
- A commutative production system.

#### **Advantages of production systems**

- Production systems provide an excellent tool for structuring AI programs.
- Production systems are highly modular because the individual rules can be added, removed or modified independently.
- The production rules are expressed in a natural form, so the statements contained in the knowledge base should be a recording of an expert thinking out loud.

**Disadvantages of production systems:** One important disadvantage is the fact that it may be very difficult analyse the flow of control within a production system because the individual rules don't call each other. Production systems describe the operations that can be performed in a search for a solution to the problem.

They can be classified as follows:

**Monotonic production system:** It is a system in which application of a rule never prevents later application of another rule that could also have been applied at the time the first rule was selected.

**Non-monotonic production system:** In non-monotonic production system the above criterion is not true.

## Commutative production system

- A Commutative production system is a production system that is both monotonic and partially commutative.
- Partially Commutative, Monotonic:
  - Useful for solving ignorable problems.
  - Can be implemented without backtrack capability.
- Non-Monotonic, Partially Commutative:
  - Useful for problems in which changes occur but can be reversed.
  - Example: Robot Navigation.
- Not partially Commutative:
  - Problems in which irreversible change occurs.
  - Example: chemical synthesis.

Figure 2-27. Commutative production system

AIR011.0

### Notes:

#### Commutative production system:

A partially commutative production system has a property that if the application of a particular sequence of rules transform state  $x$  into state  $y$ , then any permutation of those rules that is allowable, also transforms state  $x$  into state  $y$ . A commutative production system is a production system that is both monotonic and partially commutative.

#### Partially commutative, monotonic:

These production systems are useful for solving ignorable problems. Example: theorem proving. They can be implemented without the ability to backtrack to previous states when it is discovered that an incorrect path has been followed:

- This often results in a considerable increase in efficiency, particularly because since the database will never have to be restored, it is not necessary to keep track of where in the search process every change was made.
- They are good for problems where things do not change; new things get created.

#### Non monotonic, partially commutative:

Useful for problems in which changes occur but can be reversed and in which order of operations is not critical. Example: robot navigation, 8-puzzle, blocks world.

## Problem characteristics

- Different characteristics that are required include the following abilities:
  - The ability to act intelligently, as a human.
  - The ability to behave following "general intelligent action."
- The AI problem is analyzed by the following key dimensions:
  - Is the problem decomposable into set of independent smaller or easier sub problems?
  - Can solution steps be ignored or at least undone if they prove unwise?
  - Is the problem's universe predictable?

Figure 2-28. Problem characteristics

AIR0110

### Notes:

#### Problem characteristics

In order for something to be considered an "artificial intelligence," there are a few different characteristics that are required. Some of these characteristics include the following abilities:

- The ability to act intelligently, as a human.
- The ability to behave following "general intelligent action."
- The ability to artificially simulate the human brain.
- The ability to actively learn and adapt as a human.
- The ability to process language and symbols.

As can be seen from just these few examples, artificial intelligence primarily concerns the ability for a computer to mimic human intelligence. That is its key characteristic. The AI problem is analyzed by the following key dimensions:

- Is the problem decomposable into set of independent smaller or easier sub problems?
- Can solution steps be ignored or at least undone if they prove unwise?
- Is the problem's universe predictable?
- Is a good solution to the problem obvious without comparison to all other possible solutions?

Suppose the robot has the following ops: go north (n), go east (e), go south (s), go west (w). To reach its goal, it does not matter whether the robot executes the N-N-E or N-E-N.

**Not partially commutative:**

Problems in which irreversible change occurs. Example: chemical synthesis. The ops can be: add chemicals to the pot, change the temperature to  $t$  degrees. These ops may cause irreversible changes to the potion being brewed. The order in which they are performed can be very important in determining the final output  $(x+y)+z$  is not the same as  $(z+y)+x$ . Non partially commutative production systems are less likely to produce the same node many times in search process. When dealing with ones that describe irreversible processes, it is partially important to make correct decisions the first time, although if the universe is predictable, planning can be used to make that less important.

## Problem characteristics



IBM ICE (Innovation Centre for Education)

- Different characteristics that are required include the following abilities:
  - The ability to act intelligently, as a human.
  - The ability to behave following "general intelligent action."
- The AI problem is analyzed by the following key dimensions:
  - Is the problem decomposable into set of independent smaller or easier sub problems?
  - Can solution steps be ignored or at least undone if they prove unwise?
  - Is the problem's universe predictable?

Figure 2-28. Problem characteristics

AIR011.0

### Notes:

#### Problem characteristics

In order for something to be considered an "artificial intelligence," there are a few different characteristics that are required. Some of these characteristics include the following abilities:

- The ability to act intelligently, as a human.
- The ability to behave following "general intelligent action."
- The ability to artificially simulate the human brain.
- The ability to actively learn and adapt as a human.
- The ability to process language and symbols.

As can be seen from just these few examples, artificial intelligence primarily concerns the ability for a computer to mimic human intelligence. That is its key characteristic. The AI problem is analyzed by the following key dimensions:

- Is the problem decomposable into set of independent smaller or easier sub problems?
- Can solution steps be ignored or at least undone if they prove unwise?
- Is the problem's universe predictable?
- Is a good solution to the problem obvious without comparison to all other possible solutions?

- Is the desired solution a state or the world or a path to a state?
- Is large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constrain the search?
- Can a computer that is simply given the problem return the solution, or will the solution of the problem require interaction between computer and a person?
- **Is the problem decomposable?**
- A very large and composite problem can be easily solved if it can be broken into smaller problems and recursion could be used. Suppose we want to solve. Ex:  $\int x^2 + 3x + \sin 2x \cos 2x dx$
- This can be done by breaking it into three smaller problems and solving each by applying specific rules. Adding the results the complete solution is obtained.
- **Can solution steps be ignored or undone?**
- Problem fall under three classes ignorable, recoverable and irrecoverable. This classification is with reference to the steps of the solution to a problem. Consider thermo proving. We may later find that it is of no help. We can still proceed further, since nothing is lost by this redundant step. This is an example of ignorable solutions steps.
- Now consider the 8-puzzle problem tray and arranged in specified order. While moving from the start state towards goal state, we may make some stupid move and consider theorem proving. We may proceed by first proving lemma. But we may backtrack and undo the unwanted move. This only involves additional steps and the solution steps are recoverable.
- Lastly consider the game of chess. If a wrong move is made, it can neither be ignored nor be recovered. The thing to do is to make the best use of current situation and proceed.

This is an example of an irrecoverable solution steps:

- Ignorable problems. ex: Theorem proving - in which solution steps can be ignored.
- Recoverable problems. ex: 8 puzzle-in which solution steps can be undone.
- Irrecoverable problems. ex: chess- in which solution steps can't be undone.

Knowledge of these will help in determining the control structure.

- **Is the solution is universal predictable?**

Problems can be classified into those with certain outcome (eight puzzle and water jug problems) and those with uncertain outcome (playing cards) in certain – outcome problems, planning could be done to generate a sequence of operators that guarantees to a lead to a solution. Planning helps to avoid unwanted solution steps. For uncertain outcome problems, planning can at best generate a sequence of operators that has a good probability of leading to a solution. The uncertain outcome problems do not guarantee a solution and it is often very expensive since the number of solution and it is often very expensive since the number of solution paths to be explored increases exponentially with the number of points at which the outcome cannot be predicted. Thus one of the hardest types of problems to solve is the irrecoverable, uncertain – outcome problems (ex:- playing cards).

- **Is good solution absolute or relative? (Is the solution a state or a path?)**

There are two categories of problems. In one, like the water jug and 8 puzzle problems, we are satisfied with the solution, unmindful of the solution path taken, whereas in the other category not just any solution is acceptable. We want the best, like that of traveling salesman problem, where it is the shortest path. In any-path problems, by heuristic methods we obtain a solution and we do not explore alternatives. For the best-path problems all possible paths are explored using an exhaustive search until the best path is obtained.

- **Is the knowledge base consistent?**

In some problems the knowledge base is consistent and in some it is not. For example consider the case when a boolean expression is evaluated. The knowledge base now contains theorems and laws of Boolean algebra which are always true.

On the contrary consider a knowledge base that contains facts about production and cost. These keep varying with time. Hence many reasoning schemes that work well in consistent domains are not appropriate in inconsistent domains.

Ex: Boolean expression evaluation.

- **What is the role of knowledge?**

Though one could have unlimited computing power, the size of the knowledge base available for solving the problem does matter in arriving at a good solution. Take for example the game of playing chess, just the rules for determining legal moves and some simple control mechanism is sufficient to arrive at a solution. But additional knowledge about good strategy and tactics could help to constrain the search and speed up the execution of the program. The solution would then be realistic. Consider the case of predicting the political trend. This would require an enormous amount of knowledge even to be able to recognize a solution, leave alone the best. Ex: 1. Playing chess 2. Newspaper understanding

- **Does the task require interaction with the person?**

The problems can again be categorized under two heads. Solitary in which the computer will be given a problem description and will produce an answer, with no intermediate communication and with he demand for an explanation of the reasoning process. Simple theorem proving falls under this category, given the basic rules and laws, the theorem could be proved, if one exists. Ex: Theorem proving (give basic rules & laws to computer)

Conversational, in which there will be intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional informed information to the user, or both problems such as medical diagnosis fall under this category, where people will be unwilling to accept the verdict of the program, if they cannot follow its reasoning. Ex: Problems such as medical diagnosis.



## Search paradigm

- Search is a problem-solving technique that systematically explores a space of problem states.
- Search methods can be divided into systematic and nonsystematic algorithms.
- The importance of search and their properties:
  - Many goal based agents are essentially problem solving agents.
  - We need to search for a sequence of rule applications.
  - For neural network systems, we need to search for the set of connection weights.
- Four important factors/properties to consider:
  - Completeness.
  - Optimality.
  - Time complexity.
  - Space complexity.

Figure 2-29. Search paradigm

AIR0110

### Notes:

#### Search paradigm:

"Search is a problem-solving technique that systematically explores a space of problem states, i.e., successive and alternative stages in the problem-solving process. Examples of problem states might include the different board configurations in a game or intermediate steps in a reasoning process. This space of alternative solutions is then searched to find an answer. Newell and Simon (1976) have argued that this is the essential basis of human problem solving. Indeed, when a chess player examines the effects of different moves or a doctor considers a number of alternative diagnoses, they are searching among alternatives".

"Search encompasses a very general class of algorithms for exploring a problem space in order to find a solution. Virtually every problem can be solved using a search algorithm, although searching can be relatively inefficient for tasks such as arithmetic. Search methods can be divided into systematic and nonsystematic algorithms".

Dr. Martin Johnson from New Zealand defined search as: "this is the essence of search - choosing one option and putting the others aside for later, in case the first choice does not lead to a solution. We continue choosing, testing, and expanding until a solution is found or until there are no more states that can be expanded. The choice of which state to expand first is determined by the search strategy".

**The importance of search and their properties:**

It has already become clear that many of the tasks underlying AI can be phrased in terms of a search for the solution to the problem at hand.

- Many goal-based agents are essentially problem-solving agents which must decide what to do by searching for a sequence of actions that lead to their solutions.
- For production systems, we have seen the need to search for a sequence of rule applications that lead to the required fact or action.
- For neural network systems, we need to search for the set of connection weights that will result in the required input to output mapping.

How we go about each of these searches is determined by a search strategy. In this unit, we shall begin by looking at a number of uninformed (blind) search strategies, i.e. Search strategies that do not use any information about the distance to the goal. Then we will have an overview of some important informed search strategies. There are four important factors/properties to consider:

- **Completeness:** It is a solution guaranteed to be found if at least one solution exists?
- **Optimality:** It is the solution found guaranteed to be the best (or lowest cost) solution if there exists more than one solution?
- **Time complexity:** It is the upper bound on the time required to find a solution, as a function of the complexity of the problem.
- **Space complexity:** It is the upper bound on the storage space (memory) required at any point during the search, as a function of the complexity of the problem.

## Classification of search algorithms

- The different search strategies that we will consider include the following:
  - Blind Search strategies or uninformed search:
    - Depth first search.
    - Breadth first search.
    - Iterative deepening search.
    - Iterative broadening search.
  - Informed Search.
  - Constraint Satisfaction Search.
  - Adversary Search.
- Some general terminologies:
  - State Space Representations.
  - State Space Graphs.
  - Routes through State Space.
  - Search Trees.
  - Node data structure.

Figure 2-30. Classification of search algorithms

AIR0110

### Notes:

#### Classification of search algorithms

The different search strategies that we will consider include the following:

- Blind search strategies or uninformed search: depth first search, breadth first search, iterative deepening search, iterative broadening search.
- Informed search.
- Constraint satisfaction search.
- Adversary search.

We shall start with some general terminologies, and then look in more detail at some specific search algorithms.

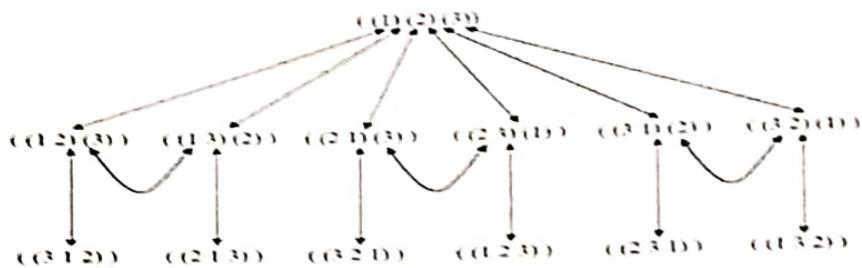
#### State space representations:

The state space is simply the space of all possible states, or configurations. Generally, of course, we prefer to work with some convenient representation of that search space. There are two components to the representation of state spaces:

- Static states.
- Transitions between states.

## General terminologies (1 of 3)

- State space graphs:
  - Example shown below.



- Routes through State Space:
  - Aim is to search for a route, or sequence of transitions, through the state space graph.

Figure 2-31. General terminologies (1 of 3)

AIR011.0

### Notes:

#### State space graphs:

If the number of possible states of the system is small enough, we can represent all of them, along with the transitions between them, in a state space graph.

#### Routes through state space:

Our general aim is to search for a route, or sequence of transitions, through the state space graph from our initial state to a goal state. Sometimes there will be more than one possible goal state. We define a goal test to determine if a goal state has been achieved. The solution can be represented as a sequence of link labels (or transitions) on the state space graph.

Note that the labels depend on the direction moved along the link. Sometimes there may be more than one path to a goal state, and we may want to find the optimal (best possible) path. We can define link costs and path costs for measuring the cost of going along a particular path, e.g. The path cost may just equal the number of links or could be the sum of individual link costs. For most realistic problems, the state space graph will be too large for us to hold all of it explicitly in memory at any one time.

## General terminologies (3 of 3)

IBM  
IBM Innovation Center for Education

- Node data structure:
  - A node used in the search algorithm is a data structure.
- The nodes that the algorithm has generated are kept in a data structure called OPEN.
- Expanding a node from open results in a closed node.
- The search process constructs a search tree, where:
  - Root is the initial state.
  - Leaf nodes are nodes.

Figure 2-33. General terminologies (3 of 3)

AIR0110

### Notes:

#### Node data structure:

A node used in the search algorithm is a data structure which contains the following:

- A state description.
- A pointer to the parent of the node.
- Depth of the node.
- The operator that generated this node.
- Cost of this path (sum of operator costs) from the start state.

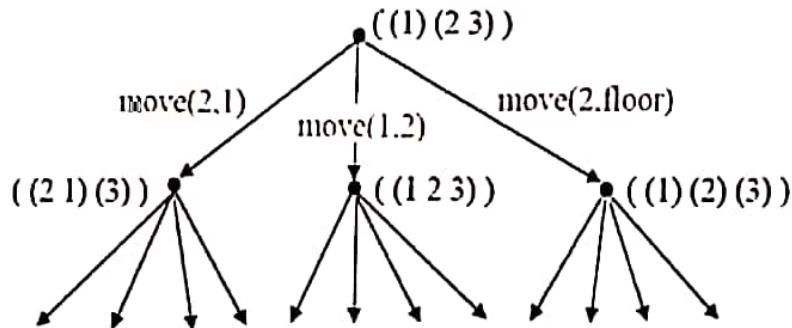
The nodes that the algorithm has generated are kept in a data structure called OPEN or fringe. Initially only the start node is in OPEN. The search starts with the root node. The algorithm picks a node from OPEN for expanding and generates all the children of the node. Expanding a node from OPEN results in a closed node. Some search algorithms keep track of the closed nodes in a data structure called CLOSED.

A solution to the search problem is a sequence of operators that is associated with a path from a start node to a goal node. The cost of a solution is the sum of the arc costs on the solution path. For large state spaces, it is not practical to represent the whole space. State space search makes explicit a sufficient portion of an implicit state space graph to find a goal node. Each node represents a partial solution path from the start node to the given node. In general, from this node there are many possible paths that have this partial path as a prefix.

## General terminologies (2 of 3)

- Search Trees:

- The root of the search tree is the search node corresponding to the initial state as shown below:



- Search tree contains:

- Root Node.
- Leaf Node.
- Ancestor/Descendant.
- Branching factor.
- Path.

Figure 2-32. General terminologies (2 of 3)

AIR0110

### Notes:

#### Search trees:

It is helpful to think of the search process as building up a search tree of routes through the state space graph. The root of the search tree is the search node corresponding to the initial state. The leaf nodes correspond either to states that have not yet been expanded, or to states that generated no further nodes when expanded. At each step, the search algorithm chooses a new unexpanded leaf node to expand. The different search strategies essentially correspond to the different algorithms one can use to select which is the next node to be expanded at each stage.

#### Search tree contains:

- Root node:** The node from which the search starts.
- Leaf node:** A node in the search tree having no children.
- Ancestor/descendant:**  $x$  is an ancestor of  $y$  if either  $x$  is  $y$ 's parent or  $x$  is an ancestor of the parent of  $y$ . If  $s$  is an ancestor of  $y$ ,  $y$  is said to be a descendant of  $x$ .
- Branching factor:** The maximum number of children of a non-leaf node in the search tree.
- Path:** A path in the search tree is a complete path if it begins with the start node and ends with a goal node. Otherwise it is a partial path.

## General terminologies (3 of 3)

- **Node data structure:**
  - A node used in the search algorithm is a data structure.
- The nodes that the algorithm has generated are kept in a data structure called OPEN
- Expanding a node from open results in a closed node.
- The search process constructs a search tree, where:
  - Root is the initial state.
  - Leaf nodes are nodes.

Figure 2-33. General terminologies (3 of 3)

AIR011.0

### Notes:

#### Node data structure:

A node used in the search algorithm is a data structure which contains the following:

- A state description.
- A pointer to the parent of the node.
- Depth of the node.
- The operator that generated this node.
- Cost of this path (sum of operator costs) from the start state.

The nodes that the algorithm has generated are kept in a data structure called OPEN or fringe. Initially only the start node is in OPEN. The search starts with the root node. The algorithm picks a node from OPEN for expanding and generates all the children of the node. Expanding a node from OPEN results in a closed node. Some search algorithms keep track of the closed nodes in a data structure called CLOSED.

A solution to the search problem is a sequence of operators that is associated with a path from a start node to a goal node. The cost of a solution is the sum of the arc costs on the solution path. For large state spaces, it is not practical to represent the whole space. State space search makes explicit a sufficient portion of an implicit state space graph to find a goal node. Each node represents a partial solution path from the start node to the given node. In general, from this node there are many possible paths that have this partial path as a prefix.

The search process constructs a search tree, where

- Root is the initial state.
- Leaf nodes are nodes: not yet expanded (i.e., In fringe) or having no successors (i.e., "Dead-ends").

Search tree may be infinite because of loops even if state space is small. The search problem will return as a solution a path to a goal node. Finding a path is important in problems like path finding, solving 15-puzzle, and such other problems. There are also problems like the n-queens problem for which the path to the solution is not important. For such problems the search problem needs to return the goal state only.

## Uninformed search algorithms

- Types of uninformed search algorithms:
  - Breadth First Search (BFS).
  - Depth First Search (DFS).
  - Depth Limited Search (DLS).
  - Depth First Iterative Deepening Search.
  - Bi-Directional Search.

Figure 2-34. Uninformed search algorithms

AIR011.0

### Notes:

#### Uninformed search algorithms

- **Breadth First Search (BFS):** BFS expands the leaf node with the lowest path cost so far and keeps going until a goal node is generated.
- **Depth First Search (DFS):** DFS expands the leaf node with the highest path cost so far and keeps going until a goal node is generated.
- **Depth Limited Search (DLS):** DLS is a variation of DFS.
- **Depth First Iterative Deepening Search:** DFIDS is a variation of DLS.
- **Bi-Directional Search:** The idea behind bi-directional search is to search simultaneously both forward from the initial state and backwards from the goal state and stop when the two BFS searches meet in the middle.

## Breadth first search

- BFS expands the leaf node with the lowest path cost.
- Algorithm and BFS tree is as shown below.

### Breadth first search

Let fringe be a list containing the initial state

Loop

if fringe is empty return failure

Node  $\leftarrow$  remove-first (fringe)

if Node is a goal

then return the path from initial state to Node

else generate all successors of Node, and

(merge the newly generated nodes into fringe)

add generated nodes to the back of fringe

End Loop

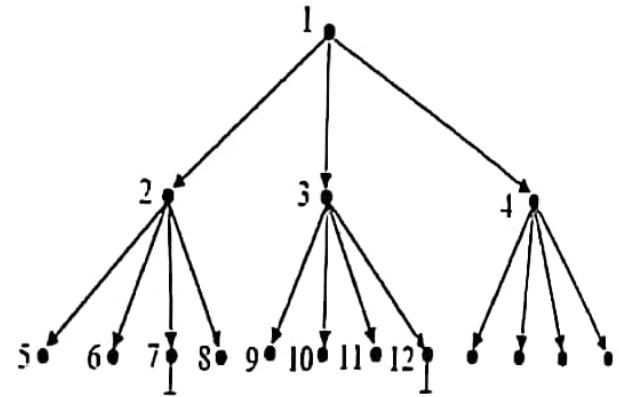


Figure 2-35. Breadth first search

AIR0110

### Notes:

#### Breadth first search (BFS)

BFS expands the leaf node with the lowest path cost so far and keeps going until a goal node is generated. If the path cost simply equals the number of links, we can implement this as a simple queue ("first in, first out"). This is guaranteed to find an optimal path to a goal state. It is memory intensive if the state space is large. If the typical branching factor is  $b$ , and the depth of the shallowest goal state is  $d$  the space complexity is  $O(b^d)$ , and the time complexity is  $O(b^d)$ .

Note that in breadth first search the newly generated nodes are put at the back of fringe or the open list. What this implies is that the nodes will be expanded in a FIFO (first in first out) order. The node that enters OPEN earlier will be expanded earlier. This amounts to expanding the shallowest nodes first.

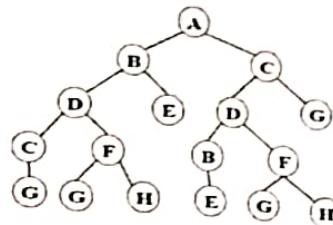
## BFS illustration (1 of 3)

IBM

IBM ICE (Innovation Centre for Education)

- We will now consider the search space in the following figure.

- Step 1: initially fringe contains only one node corresponding to the source state A.



- Step 2: A is removed from fringe.

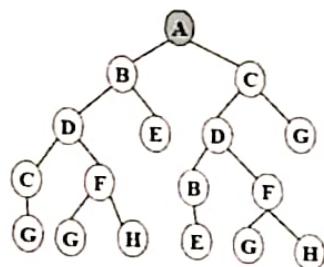


Figure 2-36. BFS illustration (1 of 3)

AIR011.0

### Notes:

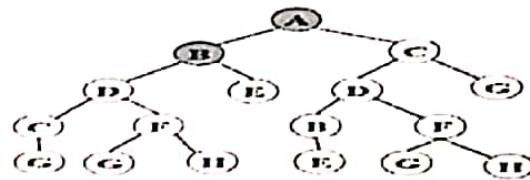
#### BFS illustration

We will now consider the search space in the following figure and show how breadth first search works on this graph.

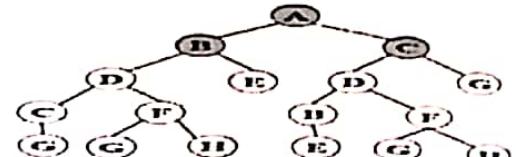
- Step 1: Initially fringe contains only one node corresponding to the source state A.
- Step 2: A is removed from fringe. The node is expanded, and its children B and C are generated. They are placed at the back of fringe.

## BFS illustration (2 of 3)

- Step 3: node B is removed from fringe and is expanded.



- Step 4: node C is removed from fringe and is expanded.



- Step 5: node D is removed from fringe.

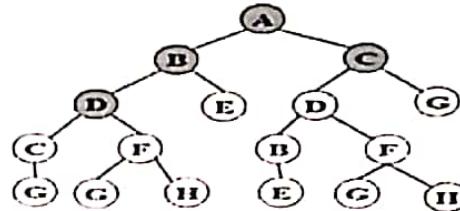


Figure 2-37. BFS Illustration (2 of 3)

AIR0110

### Notes:

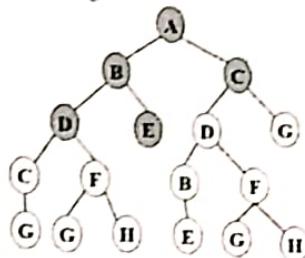
- Step 3: Node B is removed from fringe and is expanded. Its children D, E are generated and put at the back of fringe.
- Step 4: Node C is removed from fringe and is expanded. Its children D and G are added to the back of fringe.
- Step 5: Node D is removed from fringe. Its children C and F are generated and added to the back of fringe.

## BFS illustration (3 of 3)

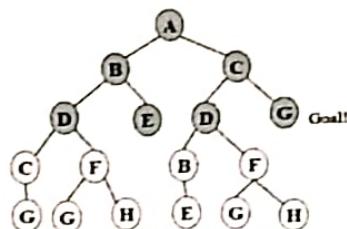
IBM

IBM ICE (Innovation Centre for Education)

- Step 6: Node E is removed from fringe.



- Step 7: D is expanded; B and F are put in OPEN.



- Step 8: G is selected for expansion.

AIR011.0

Figure 2-38. BFS illustration (3 of 3)

### Notes:

- Step 6: Node E is removed from fringe. It has no children.
- Step 7: D is expanded; B and F are put in OPEN.
- Step 8: G is selected for expansion. It is found to be a goal node. So the algorithm returns the path A C G by following the parent pointers of the node corresponding to G. The algorithm terminates.

**Advantages of breadth first search:** It finds the path of minimal length to the goal.

**Disadvantages of breadth first search:** It requires the generation and storage of a tree whose size is exponential the depth of the shallowest goal node.

## Depth first search

- DFS expands the leaf node with the highest path cost.
- Algorithm and DFS tree are shown below:

### Depth First Search

```

Let fringe be a list containing the initial state
Loop
if fringe is empty return failure
Node ← remove-first (fringe)
if Node is a goal
then return the path from Initial state to Node
else
generate all successors of Node, and
merge the newly generated nodes into fringe
add generated nodes to the front of fringe
End Loop
    
```

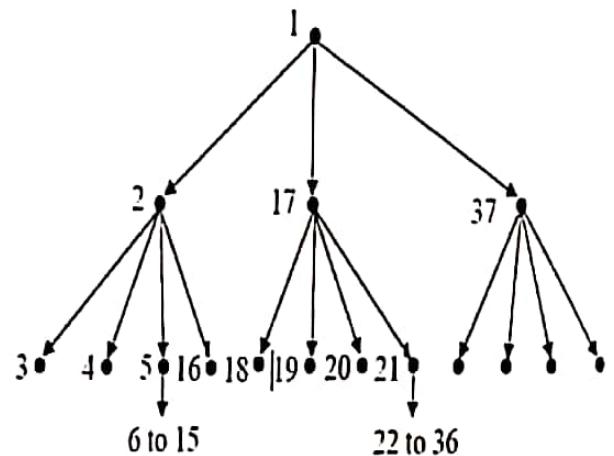


Figure 2-39. Depth first search

AIR011.0

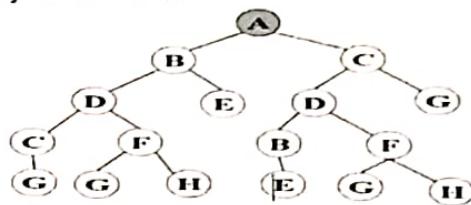
### Notes:

#### Depth first search (DFS)

DFS expands the leaf node with the highest path cost so far and keeps going until a goal node is generated. If the path cost simply equals the number of links, we can implement this as a simple stack ("last in, first out"). This is not guaranteed to find any path to a goal state. It is memory efficient even if the state space is large. If the typical branching factor is  $b$ , and the maximum depth of the tree is  $m$  (possibly  $\infty$ ) – the space complexity is  $O(b^m)$ , and the time complexity is  $O(b^m)$ . The depth first search algorithm puts newly generated nodes in the front of OPEN. This results in expanding the deepest node first. Thus the nodes in OPEN follow a LIFO order (last in first out). OPEN is thus implemented using a stack data structure.

## DFS illustrations (1 of 3)

- Let us now run depth first search on the search space for the following graph.
- Step 1: initially fringe contains only the node for A.



- Step 2: A is removed from fringe.

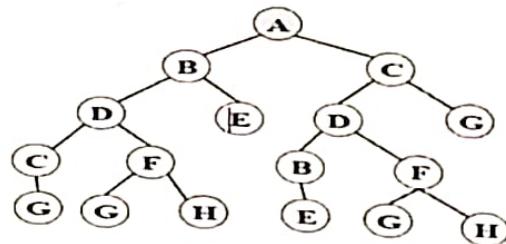


Figure 2-40. DFS illustrations (1 of 3)

AIR011.0

### Notes:

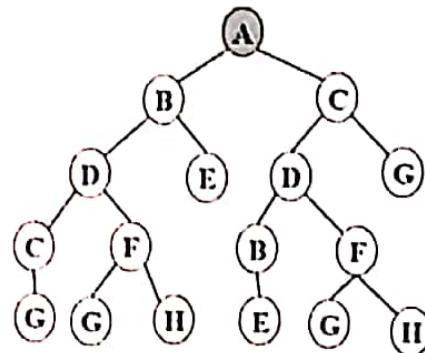
#### DFS Illustrations

- Step 1: Initially fringe contains only the node for A.
- Step 2: A is removed from fringe. A is expanded and its children B and C are put in front of fringe.



## DFS illustrations (2 of 3)

- Step 3: Node B is removed from fringe.



- Step 4: Node D is removed from fringe.

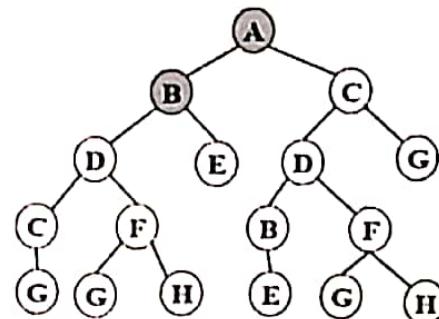


Figure 2-41. DFS illustrations (2 of 3)

AIR011.0

### Notes:

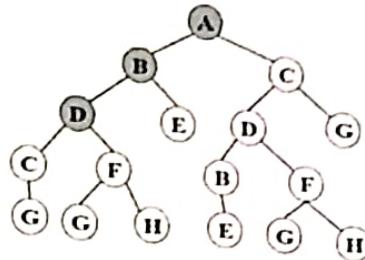
- Step 3: Node B is removed from fringe, and its children D and E are pushed in front of fringe.
- Step 4: Node D is removed from fringe, C and F are pushed in front of fringe.

**DFS illustrations (3 of 3)**

- Step 5: Node C is removed from fringe.

IBM

IBM ICE (Innovation Centre for Education)



- Step 6: Node G is expanded and found to be a goal node.

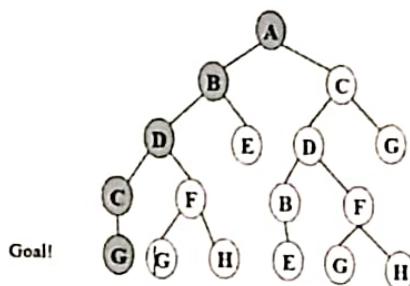


Figure 2-42. DFS illustrations (3 of 3)

AIR011.0

**Notes:**

- Step 5: Node C is removed from fringe. Its child G is pushed in front of fringe.
- Step 6: Node G is expanded and found to be a goal node. The solution path A-B-D-C-G is returned, and the algorithm terminates.

## Depth limited search (DLS)

- DLS is a variation of DFS.
- If we put a limit  $l$  on how deep a depth first search can go, then the search will terminate.

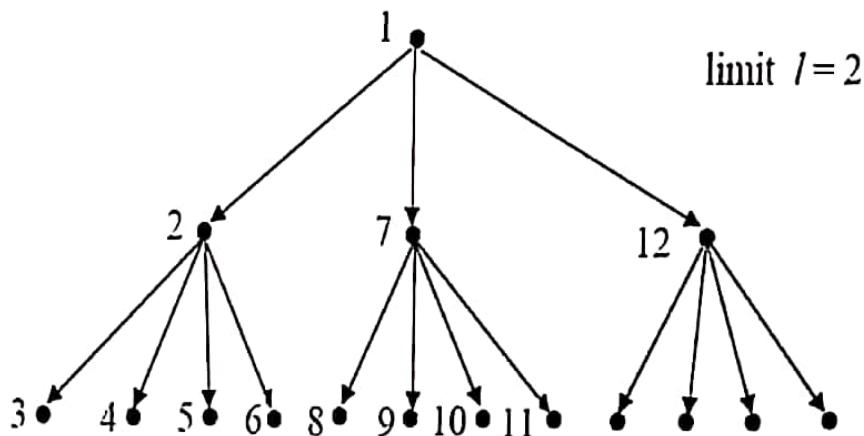


Figure: DLS tree

Figure 2-43. Depth limited search (DLS)

AIR011.0

### Notes:

#### Depth Limited Search (DLS)

DLS is a variation of DFS. If we put a limit  $l$  on how deep a depth first search can go, we can guarantee that the search will terminate (either in success or failure). If there is at least one goal state at a depth less than  $l$ , this algorithm is guaranteed to find a goal state, but it is not guaranteed to find an optimal path. The space complexity is  $O(b^l)$ , and the time complexity is  $O(b^l)$ . For most problems we will not know what is a good limit  $l$  until we have solved the problem!

## Depth first iterative deepening search and bi-directional search



IBM ICE (Innovation Centre for Education)

- Depth first iterative deepening search (DFIDS):

- DFIDS is a variation of DLS.

- Bi-directional search (BDS):

- The idea behind bi-directional search is to search simultaneously both forward and backwards.

- Repeated states:

- The possibility that we will waste time by expanding states that have already been expanded

- Fig. Bi-directional search tree:

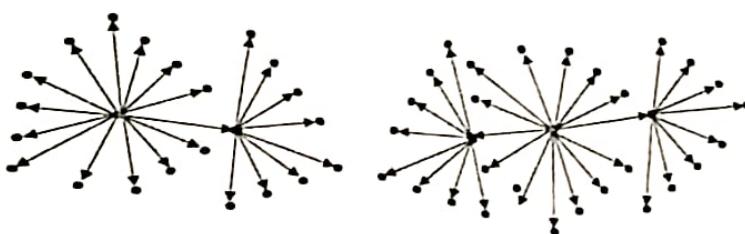


Figure 2-44. Depth first iterative deepening search and bi-directional search

AIR011.0

### Notes:

#### Depth first iterative deepening search (DFIDS)

DFIDS is a variation of DLS. The space complexity is  $O(b^d)$  as in DLS with  $l = d$ , which is better than BFS. The time complexity is  $O(b^d)$  as in BFS, which is better than DFS. If the lowest depth of a goal state is not known, we can always find the best limit  $l$  for DLS by trying all possible depths  $l = 0, 1, 2, 3, \dots$  in turn, and stopping once we have achieved a goal state. This appears wasteful because all the DLS for  $l$  less than the goal level are useless, and many states are expanded many times. However, in practice, most of the time is spent at the deepest part of the search tree, so the algorithm combines the benefits of DFS and BFS. Because all the nodes are expanded at each level, the algorithm is complete and optimal like BFS, but has the modest memory requirements of DFS. The space complexity is  $O(b^d)$  as in DLS with  $l = d$ , which is better than BFS. The time complexity is  $O(b^d)$  as in BFS, which is better than DFS.

#### Bi-directional search (BDS):

The idea behind bi-directional search is to search simultaneously both forward from the initial state and backwards from the goal state, and stop when the two BFS searches meet in the middle. This is not always going to be possible but is likely to be feasible if the state transitions are reversible. The algorithm is complete and optimal, and since the two search depths are  $\sim d/2$ , it has space complexity  $O(b^{d/2})$ , and time complexity  $O(b^{d/2})$ . However, if there is more than one possible goal state, this must be factored into the complexity.

**Repeated states:**

In the above discussion we have ignored an important complication that often arises in search processes – the possibility that we will waste time by expanding states that have already been expanded before somewhere else on the search tree. For some problems this possibility can never arise, because each state can only be reached in one way. For many problems, however, repeated states are unavoidable. The search trees for these problems are infinite, but if we can prune out the repeated states, we can cut the search tree down to a finite size, we effectively only generate a portion of the search tree that matches the state space graph.

**Avoiding repeated states:**

There are three principal approaches for dealing with repeated states:

- **Never return to the state you have just come from:** The node expansion function must be prevented from generating any node successor that is the same state as the node's parent.
- **Never create search paths with cycles in them:** The node expansion function must be prevented from generating any node successor that is the same state as any of the node's ancestors.
- **Never generate states that have already been generated before:** This requires that every state ever generated is remembered, potentially resulting in space complexity of  $O(b^d)$ .

Clearly these are in increasing order of effectiveness and computational overhead.

## Comparing the uninformed search algorithms

IBM

IBM ICE (Innovation Centre for Education)

- We can now summarize the properties of our five uninformed search strategies:

Strategy	Complete	Optimal	Time Complexity	Space Complexity
BFS	Yes	Yes	$O(b^d)$	$O(b^d)$
DFS	No	No	$O(b^m)$	$O(b^m)$
DLS	If $l \geq d$	No	$O(b^l)$	$O(b^l)$
DFIDS	Yes	Yes	$O(b^d)$	$O(b^d)$
BDS	Yes	Yes	$O(b^{d/2})$	$O(b^{d/2})$

Figure 2-45. Comparing the uninformed search algorithms

AIR011.0

### Notes:

#### Comparing the uninformed search algorithms

Simple BFS and BDS are complete and optimal but expensive with respect to space and time. DFS requires much less memory if the maximum tree depth is limited, but has no guarantee of finding any solution, let alone an optimal one. DLS offers an improvement over DFS if we have some idea how deep the goal is. The best overall is DFID which is complete, optimal and has low memory requirements, but still exponential time.



## Informed search algorithms

- Informed search uses some kind of evaluation function.
- Heuristic is a rule of thumb that probably leads to a solution.
- Heuristic Information:
  - Information about the problem includes the nature of states, cost of transforming etc.
  - Often expressed in the form of heuristic evaluation function.
- Some of the heuristic search techniques are:
  - Pure Heuristic Search.
  - A\* algorithm
  - Iterative-Deepening A\*.

Figure 2-46 Informed search algorithms

AIR0110

### Notes:

#### Informed search algorithms

Informed search uses some kind of evaluation function to tell us how far each expanded state is from a goal state, and/or some kind of heuristic function to help us decide which state is likely to be the best one to expand next. The hard part is to come up with good evaluation and/or heuristic functions. Often there is a natural evaluation function, such as distance in miles or number objects in the wrong position. Sometimes we can learn heuristic functions by analyzing what has worked well in similar previous searches.

Heuristic is a rule of thumb that probably leads to a solution. Heuristics play a major role in search strategies because of exponential nature of the most problems. Heuristics help to reduce the number of alternatives from an exponential number to a polynomial number. In artificial Intelligence, *heuristic search* has a general meaning, and a more specialized technical meaning. In a general sense, the term *heuristic* is used for any advice that is often effective but is not guaranteed to work in every case. Within the *heuristic search* architecture, however, the term *heuristic* usually refers to the special case of a heuristic evaluation function.

#### Heuristic information:

In order to solve larger problems, domain-specific knowledge must be added to improve search efficiency. Information about the problem includes the nature of states, cost of transforming from one state to another, and characteristics of the goals. This information can often be expressed in the form of heuristic evaluation function, say  $f(n,g)$ , a function of the nodes  $n$  and/or the goals  $g$ .

The simplest idea, known as greedy best first search, is to expand the node that is already closest to the goal, as that is most likely to lead quickly to a solution. This is like DFS in that it attempts to follow a single route to the goal, only attempting to try a different route when it reaches a dead end. As with DFS, it is not complete, not optimal, and has time and complexity of  $O(b^n)$ . However, with good heuristics, the time complexity can be reduced substantially.

Following is a list of heuristic search techniques:

- Pure heuristic search
- A\* algorithm
- Iterative-deepening A\*
- Depth-first branch-and-bound
- Heuristic path algorithm
- Recursive best-first search

**A\* search:** suppose that, for each node  $n$  in a search tree, an evaluation function  $f(n)$  is defined as the sum of the cost  $g(n)$  to reach that node from the start state, plus an estimated cost  $h(n)$  to get from that state to the goal state. That  $f(n)$  is then the estimated cost of the cheapest solution through  $n$ .

A\* search, which is the most popular form of best-first search, repeatedly picks the node with the lowest  $f(n)$  to expand next. It turns out that if the heuristic function  $h(n)$  satisfies certain conditions, then this strategy is both complete and optimal. In particular, if  $h(n)$  is an admissible heuristic, i.e. is always optimistic and never overestimates the cost to reach the goal, then A\* is optimal. The classic example is finding the route by road between two cities given the straight line distances from each road intersection to the goal city. In this case, the nodes are the intersections, and we can simply use the straight line distances as  $h(n)$ .

**Hill climbing/gradient descent:** the basic idea of hill climbing is simple: at each current state we select a transition, evaluate the resulting state, and if the resulting state is an improvement we move there, otherwise we try a new transition from where we are. We repeat this until we reach a goal state, or have no more transitions to try. The transitions explored can be selected at random, or according to some problem specific heuristics.

In some cases, it is possible to define evaluation functions such that we can compute the gradients with respect to the possible transitions, and thus compute which transition direction to take to produce the best improvement in the evaluation function. Following the evaluation gradients in this way is known as gradient descent. In neural networks, for example, we can define the total error of the output activations as a function of the connection weights, and compute the gradients of how the error changes as we change the weights. By changing the weights in small steps against those gradients, we systematically minimize the network's output errors.

The following are some of the issues to be considered while designing search algorithms:

- The direction in which to conduct the search (forward versus backward reasoning).
- How to select applicable rules (matching).
- How to represent each node of the search process (knowledge representation problem).

## Heuristic search techniques



- Heuristic search is an AI search technique.
- In artificial intelligence, heuristic search has both general meaning, and specialized technical meaning.
- Some of the variety of heuristic search are:
  - Generate and test.
  - Hill climbing.
  - Means ends analysis.

Figure 2-47. Heuristic search techniques

AIR0110

### Notes:

#### Heuristic search techniques and knowledge representation

Heuristic search is an AI search technique that employs heuristic for its moves. *Heuristic* is a rule of thumb that probably leads to a solution. Heuristics play a major role in search strategies because of exponential nature of the most problems. Heuristics help to reduce the number of alternatives from an exponential number to a polynomial number. In artificial intelligence, *heuristic search* has a general meaning, and a more specialized technical meaning. In a general sense, the term *heuristic* is used for any advice that is often effective but is not guaranteed to work in every case. Within the heuristic search architecture, however, the term *heuristic* usually refers to the special case of a heuristic evaluation function.

In order to solve larger problems, domain-specific knowledge must be added to improve search efficiency. Information about the problem includes the nature of states, cost of transforming from one state to another, and characteristics of the goals. This information can often be expressed in the form of heuristic evaluation function, say  $f(n,g)$ , a function of the nodes  $n$  and/or the goals  $g$ . In this unit, we provide a framework for describing search methods and several general-purpose search techniques are discussed.

The variety of heuristic search includes:

- Generate and test.
- Hill climbing.
- Best first search.
- Problem reduction.
- Constraint satisfaction.
- Means ends analysis.

The time complexity of a heuristic search algorithm depends on the accuracy of the heuristic function. For example, if the heuristic evaluation function is an exact estimator, then A\* search algorithm runs in linear time, expanding only those nodes on an optimal solution path. Conversely, with a heuristic that returns zero everywhere, A\* algorithm becomes uniform-cost search, which has exponential complexity.

In general, the time complexity of a\* search and ida\* search is an exponential function of the error in the heuristic function. For example, if the heuristic has constant absolute error, meaning that it never underestimates by more than a constant amount regardless of the magnitude of the estimate, then the running time of A\* is linear with respect to the solution cost. A more realistic assumption is constant relative error, which means that the error is a fixed percentage of the quantity being estimated. The base of the exponent, however, is smaller than the brute-force branching factor, reducing the asymptotic complexity and allowing larger problems to be solved. For example, using appropriate heuristic functions, IDA\* can optimally solve random instance of the twenty-four puzzle and Rubik's cube.

## Generate-and-test

IBM ICE (Innovation Centre for Education)

- The generate-and-test strategy is the simplest of all the approaches.

- The following is an algorithm for generate-and-test:
  - Generating a particular point in problem space.
  - Generating a path from start state.
  - Test to see if this is the actual solution by comparing.
  - If a solution has been found, quit. Otherwise return to step 1.
- The generate-and-test is a:
  - It is a depth first search.
  - It is an exhaustive search of the problem space.
  - Operate by generating solutions randomly.

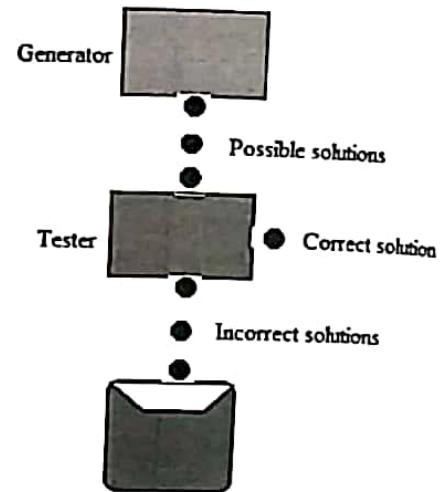


Figure: generate-and-test procedure

Figure 2-48. Generate-and-test

AIR0110

### Notes:

#### Generate-and-test

The generate-and-test strategy is the simplest of all the approaches. The following is an algorithm for generate-and-test:

- Generating a solution: for some problems, this means generating a particular point in problem space. For others, it means generating a path from start state.
- Test to see if this is the actual solution by comparing the chosen point or the end point of the chosen path to the set of acceptable goal states.
- If a solution has been found, quit. Otherwise return to step 1.

#### The generate-and-test is a:

- It is a depth first search procedure since complete solutions must be generated before they can be tested.
- In its most systematic form, it is simply an exhaustive search of the problem space.
- Operate by generating solutions randomly.
- Also called as British museum algorithm.
- If a sufficient number of monkeys were placed in front of a set of typewriters, and left alone long enough, then they would eventually produce all the works of Shakespeare.

**Systematic generate-and-test:**

While generating complete solutions and generating random solutions are the two extremes there exists another approach that lies in between. The approach is that the search process proceeds systematically but some paths that unlikely to lead the solution are not considered. This evaluation is performed by a heuristic function. Depth-first search tree with backtracking can be used to implement systematic generate-and-test procedure. As per this procedure, if some intermediate states are likely to appear often in the tree, it would be better to modify that procedure to traverse a graph rather than a tree.

**Generate-and-test and planning:**

Exhaustive generate-and-test is very useful for simple problems. But for complex problems even heuristic generate-and-test is not very effective technique. But this may be made effective by combining with other techniques in such a way that the space in which to search is restricted. An AI program DENDRAL, for example, uses plan-generate-and-test technique. First, the planning process uses constraint-satisfaction techniques and creates lists of recommended and contraindicated substructures. Then the generate-and-test procedure uses the lists generated and required to explore only a limited set of structures. Constrained in this way, generate-and-test proved highly effective. A major weakness of planning is that it often produces inaccurate solutions as there is no feedback from the world. But if it is used to produce only pieces of solutions then lack of detailed accuracy becomes unimportant.

In the case of generate-and-test algorithm, a heuristic is needed to sharpen up the search. Consider the problem of four 6-sided cubes, and each side of the cube is painted in one of four colors. The four cubes are placed next to one another and the problem lies in arranging them so that the four available colors are displayed whichever way the 4 cubes are viewed. The problem can only be solved if there are at least four sides colored in each color and the number of options tested can be reduced using heuristics if the most popular color is hidden by the adjacent cube.



## Hill climbing

- Hill climbing is a mathematical optimization technique.
- Hill climbing is good for finding a local optimum solution.
- It is used widely in artificial intelligence.
- Hill climbing can often produce a better result than other algorithms.
- Computation of heuristic function can be done.
- Hill climbing is often used when a good heuristic function is available.

Figure 2-49. Hill climbing

AIR011.0

### Notes:

#### Hill climbing

Hill climbing is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

For example, hill climbing can be applied to the travelling salesman problem. It is easy to find an initial solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much shorter route is likely to be obtained. Hill climbing is good for finding a local optimum (a solution that cannot be improved by considering a neighboring configuration) but it is not guaranteed to find the best possible solution (the global optimum) out of all possible solutions (the search space). The characteristic that only local optima are guaranteed can be cured by using restarts (repeated local search), or more complex schemes based on iterations, like iterated local search, or memory, like reactive search optimization and tabu search, or memory-less stochastic modifications, like simulated annealing.

The relative simplicity of the algorithm makes it a popular first choice amongst optimizing algorithms. It is used widely in artificial intelligence, for reaching a goal state from a starting node. Choice of next node and starting node can be varied to give a list of related algorithms. Although more advanced algorithms such as *simulated annealing* or *tabu* search may give better results, in some situations hill climbing works just as well.

Hill climbing can often produce a better result than other algorithms when the amount of time available to perform a search is limited, such as with real-time systems. It is an anytime algorithm: it can return a valid solution even if it's interrupted at any time before it ends.

Hill climbing is a variant of generate-and test in which feedback from the test procedure is used to help the generator decide which direction to move in search space. The test function is augmented with a heuristic function that provides an estimate of how close a given state is to the goal state. Computation of heuristic function can be done with negligible amount of computation. Hill climbing is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available.

#### **Mathematical model**

Hill climbing attempts to maximize (or minimize) a target function  $f(\mathbf{x})$ , where  $\mathbf{x}$  is a vector of continuous and/or discrete values. At each iteration, hill climbing will adjust a single element in  $\mathbf{x}$  and determine whether the change improves the value of  $f(\mathbf{x})$ . (Note that this differs from gradient descent methods, which adjust all of the values in  $\mathbf{x}$  at each iteration according to the gradient of the hill.) With hill climbing, any change that improves  $f(\mathbf{x})$  is accepted, and the process continues until no change can be found to improve the value of  $f(\mathbf{x})$ .  $\mathbf{x}$  is then said to be "locally optimal". In discrete vector spaces, each possible value for  $\mathbf{x}$  may be visualized as a vertex in a graph. Hill climbing will follow the graph from vertex to vertex, always locally increasing (or decreasing) the value of  $f(\mathbf{x})$  until a local maximum (or local minimum)  $x_m$  is reached.

#### **Variants of hill climbing**

In simple hill climbing, the first closer node is chosen, whereas in *steepest ascent hill climbing* all successors are compared and the closest to the solution is chosen. Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions. *Steepest ascent hill climbing* is similar to best-first search, which tries all possible extensions of the current path instead of only one.

Stochastic hill climbing does not examine all neighbors before deciding how to move. Rather, it selects a neighbor at random, and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another. Coordinate descent does a line search along one coordinate direction at the current point in each iteration. Some versions of coordinate descent randomly pick a different coordinate direction at each iteration.

Random-restart hill climbing is a meta-algorithm built on top of the hill climbing algorithm. It is also known as shotgun hill climbing. It iteratively does hill-climbing, each time with a random initial condition  $x_0$ . The best  $x_m$  is kept: if a new run of hill climbing produces a better  $x_m$  than the stored state, it replaces the stored state. Random-restart hill climbing is a surprisingly effective algorithm in many cases. It turns out that it is often better to spend CPU time exploring the space, than carefully optimizing from an initial condition.

## Algorithm for simple hill climbing

- Evaluate the initial state.
- Loop until a solution.
- Algorithm for steepest-ascent hill climbing.
- Simulated annealing.

Figure 2-50. Algorithm for simple hill climbing

AIR011.0

### Notes:

#### Algorithm for simple hill climbing

##### Steps:

- Evaluate the initial state. If it is also goal state, then return it and quit. Otherwise continue with the initial state as the current state.
- Loop until a solution is found or until there are no new operators left to be applied in the current state.
- Select an operator that has not yet been applied to the current state and apply it to produce a new state
- Evaluate the new state: if it is the goal state, then return it and quit and if it is not a goal state but it is better than the current state, then make it the current state.
- If it is not better than the current state, then continue in the loop.
- The key difference between simple hill climbing and generate-and-test is the use of evaluation function as a way to inject task specific knowledge into the control process.

#### Algorithm for steepest-ascent hill climbing

##### Steps:

- Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
- Loop until a solution is found or until a complete iteration produces no change to current state.
- Let SUCC be a state such that any possible successor of the current state will be better than SUCC.
- For each operator that applies to the current state do apply the operator and generate a new state and evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to SUCC. If it is better, then set SUCC to this state. If it is not better, leave SUCC alone.
- If the SUCC is better than the current state, then set current state to succ.

#### This simple policy has three well-known drawbacks

- **Local maxima:** A local maximum as opposed to global maximum.
- **Plateaus:** An area of the search space where evaluation function is flat, thus requiring random walk.
- **Ridge:** Where there are steep slopes and the search direction is not towards the top but towards the side.

In each of the previous cases (local maxima, plateaus & ridge), the algorithm reaches a point at which no progress is being made. A solution is to do a random-restart hill-climbing - where random initial states are generated, running each until it halts or makes no discernible progress. The best result is then chosen.

#### Simulated annealing

An alternative to a hill-climbing algorithm when stuck on a local maximum is to do a 'reverse walk' to escape the local maximum. This is the idea of simulated annealing. The term simulated annealing derives from the roughly analogous physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure. The simulated annealing process lowers the temperature by slow stages until the system "freezes" and no further changes occur.

Simulated annealing (SA) is a generic probabilistic meta-heuristic for the global optimization problem of locating a good approximation to the global optimum of a given function in a large search space. It is often used when the search space is discrete (e.g., All tours that visit a given set of cities). For certain problems, simulated annealing may be more efficient than exhaustive enumeration provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

**Differences:** The algorithm for simulated annealing is slightly different from the simple-hill climbing procedure. In this algorithm, the annealing schedule must be maintained; moves to worse states may be accepted and to maintain, in addition to the current state, the best state found so far.

#### Algorithm: simulate annealing

- Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
- Initialize best-so-far to the current state.
- Initialize t according to the annealing schedule.
- Loop until a solution is found or until there are no new operators left to be applied in the current state.
- Select an operator that has not yet been applied to the current state and apply it to produce a new state.
- Evaluate the new state. Compute:  $\Delta E = (\text{value of current}) - (\text{value of new state})$ .
- If the new state is a goal state, then return it and quit.
- If it is a goal state but is better than the current state, then make it the current state. Also set BEST-SO-FAR to this new state.
- If it is not better than the current state, then make it the current state with probability  $p'$  as defined above. This step is usually implemented by invoking a random number generator to produce a number in the range [0, 1]. If the number is less than  $p'$ , then the move is accepted. Otherwise, do nothing. Revise T as necessary according to the annealing schedule.

- Return BEST-SO-FAR as the answer.
- From implementation point of view, it is necessary to select an annealing schedule which has three components: initial value to be used for temperature; criteria that will be used to decide when the temperature will be reduced. Amount by which the temperature will be reduced.

## Best first search

- Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.
- The a\* search algorithm is an example of best-first search.
- Best-first algorithms are often used for path finding in combinatorial search.

• Algorithm:

```

OPEN = [initial state]
CLOSED = []
while OPEN is not empty
do
    1. Remove the best node from OPEN, call it n, add it to CLOSED.
    2. If n is the goal state, backtrace path to n (through recorded parents) and return
       path.
    3. Create n's successors.
    4. For each successor do:
        a. If it is not in CLOSED: evaluate it, add it to OPEN, and record its parent.
        b. Otherwise, change recorded parent if this new path is better than previous
           one.
done

```

Figure 2-51. Best first search

AIR0110

### Notes:

#### Best first search

Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule. Judea pearl described best-first search as estimating the promise of node  $n$  by a "heuristic evaluation function  $f(n)$  which, in general, may depend on the description of  $n$ , the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain." Some authors have used "best-first search" to refer specifically to a search with a heuristic that attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first. This specific type of search is called greedy best-first search. Efficient selection of the current best candidate for extension is typically implemented using a priority queue. The A\* search algorithm is an example of best-first search. Best-first algorithms are often used for path finding in combinatorial search.

The best first search procedure combines the advantages of both dfs and bfs into a single method. The DFS is good because it allows a solution to be found without all competing branches having to be expanded. The BFS is good because it does not get branches on dead end paths. One way of combining the two is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one does. At each step of the BFS search process; we select the most promising of the nodes we have generated so far. This is done by applying an appropriate heuristic function to each of them.

We then expand the chosen node by using the rules to generate its successors similar to steepest ascent hill climbing with two exceptions:

- In hill climbing, one move is selected, and all the others are rejected, never to be reconsidered. This produces the straight-line behavior that is characteristic of hill climbing.
- In bfs, one move is selected, but the others are kept around so that they can be revisited later if the selected path becomes less promising. Further, the best available state is selected in the BFS, even if that state has a value that is lower than the value of the state that was just explored. This contrasts with hill climbing, which will stop if there are no successor states with better values than the current state.

### Algorithm for BFS:

Open = [initial state], while open is not empty or until a goal is found:

#### Do

- Remove the best node from OPEN, call it n.
- If n is the goal state, backtrace path to n (through recorded parents) and return path.
- Create n's successors.
- Evaluate each successor, add it to open, and record its parent.

#### Done

- Note that this version of the algorithm is not *complete*, i.e. It does not always find a possible path between two nodes even if there is one. For example, it gets stuck in a loop if it arrives at a dead end, that is a node with the only successor being its parent. It would then go back to its parent, add the dead-end successor to the OPEN list again, and so on.

The following version extends the algorithm to use an additional closed list, containing all nodes that have been evaluated and will not be looked at again. As this will avoid any node being evaluated twice, it is not subject to infinite loops.

- Open = [initial state], closed = [], while open is not empty

#### Do

- Remove the best node from OPEN, call it n, add it to CLOSED.
- If n is the goal state, backtrace path to n (through recorded parents) and return path.
- Create n's successors.
- For each successor do:
  - If it is not in closed: evaluate it, add it to open, and record its parent.
  - Otherwise: change recorded parent if this new path is better than previous one.

#### Done

- Also note that the given pseudo code of both versions just terminates when no path is found. An actual implementation would of course require special handling of this case.

### BFS: Simple explanation

It proceeds in steps, expanding one node at each step, until it generates a node that corresponds to a goal state. At each step, it picks the most promising of the nodes that have so far been generated but not expanded. It generates the successors of the chosen node, applies the heuristic function to them, and adds them to the list of open nodes, after checking to see if any of them have been generated before. By doing this check, we can guarantee that each node only appears once in the graph, although many nodes may point to it as a successor.

## Knowledge representation (1 of 2)



IBM ICE (Innovation Centre for Education)

- Knowledge Representation (KR) is an area of artificial intelligence research.
- Knowledge Representation (KR) research involves analysis of how to reason accurately and effectively.
- The fundamental goal of knowledge representation is to facilitate reasoning, inferencing, or drawing conclusions.
- Knowledge representation in terms of five distinct roles:
  - It is most fundamentally a surrogate.
  - It is a set of ontological commitments.
  - It is a fragmentary theory of intelligent reasoning.
  - It is a medium for pragmatically efficient computation.
  - It is a medium of human expression.

Figure 2-52. Knowledge representation (1 of 2)

AIR011.0

### Notes:

#### Knowledge representation

Knowledge representation (KR) is an area of artificial intelligence research aimed at representing knowledge in symbols to facilitate inference from those knowledge elements, creating new elements of knowledge. The KR can be made to be independent of the underlying knowledge model or Knowledge Base System (KBS) such as a semantic network. Knowledge representation (KR) research involves analysis of how to reason accurately and effectively and how best to use a set of symbols to represent a set of facts within a knowledge domain. A symbol vocabulary and a system of logic are combined to enable inferences about elements in the KR to create new KR sentences.

Logic is used to supply formal semantics of how reasoning functions should be applied to the symbols in the KR system. Logic is also used to define how operators can process and reshape the knowledge. Examples of operators and operations include negation, conjunction, adverbs, adjectives, quantifiers and modal operators. The logic is interpretation theory. These elements- symbols, operators, and interpretation theory-are what give sequences of symbols meaning within a KR. A key parameter in choosing or creating a KR is its expressivity. The more expressive a KR, the easier and more compact it is to express a fact or element of knowledge within the semantics and grammar of that KR.

However, more expressive languages are likely to require more complex logic and algorithms to construct equivalent inferences. A highly expressive KR is also less likely to be complete and consistent. Less expressive KRS may be both complete and consistent. Auto-epistemic temporal modal logic is a highly expressive KR system, encompassing meaningful chunks of knowledge with brief, simple symbol sequences

(sentences). Propositional logic is much less expressive but highly consistent and complete and can efficiently produce inferences with minimal algorithm complexity. Nonetheless, only the limitations of an underlying knowledge base affect the ease with which inferences may ultimately be made (once the appropriate KR has been found). This is because a knowledge set may be exported from a knowledge model or knowledge base system into different KRS, with different degrees of expressiveness, completeness, and consistency. If a particular KR is inadequate in some way, that set of problematic KR elements may be transformed by importing them into a KBS, modified and operated on to eliminate the problematic elements or augmented with additional knowledge imported from other sources, and then exported into a different, more appropriate KR.

In applying KR systems to practical problems, the complexity of the problem may exceed the resource constraints or the capabilities of the KR system. Recent developments in KR include the concept of the semantic web, and development of xml-based knowledge representation languages and standards, including resource description framework (RDF), RDF schema, topic maps, DARPA Agent Markup Language (DAML), Ontology Inference Layer (OIL), and Web Ontology Language (OWL).

There are several KR techniques such as frames, rules, tagging, and semantic networks which originated in cognitive science. Since knowledge is used to achieve intelligent behavior, the fundamental goal of knowledge representation is to facilitate reasoning, inferencing, or drawing conclusions. A good KR must be both declarative and procedural knowledge. What is knowledge representation can best be understood in terms of five distinct roles it plays, each crucial to the task at hand:

- A knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., By reasoning about the world rather than acting in it.
- It is a set of ontological commitments, i.e., An answer to the question: in what terms should I think about the world?
- It is a fragmentary theory of intelligent reasoning, expressed in terms of three components:
  - The representation's fundamental conception of intelligent reasoning.
  - The set of inferences the representation sanctions.
  - The set of inferences it recommends.
- It is a medium for pragmatically efficient computation, i.e., The computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences.
- It is a medium of human expression, i.e., "A language in which we say things about the world".

## **Knowledge representation (2 of 2)**



IBM ICE (Innovation Centre for Education)

- Characteristics of knowledge representation:
  - Coverage.
  - Understandable by humans.
  - Consistency.
  - Efficient.
  - Easiness for modifying and updating.
  - Supports the intelligent activity which uses the knowledge base.
- Knowledge representation techniques:
  - Lists.
  - Trees.
  - Semantic networks.
  - Schemas.
  - Rule-based representations.
  - Logic-based representations.

---

Figure 2-53. Knowledge representation (2 of 2)

AIR011.0

### **Notes:**

#### **Characteristics of knowledge representation**

A good knowledge representation covers six basic characteristics:

- **Coverage:** Which means the KR covers a breadth and depth of information. Without a wide coverage, the KR cannot determine anything or resolve ambiguities.
- **Understandable by humans:** KR is viewed as a natural language, so the logic should flow freely. It should support modularity and hierarchies of classes (polar bears are bears, which are animals). It should also have simple primitives that combine in complex forms.
- **Consistency:** If john closed the door, it can also be interpreted as the door was closed by john. By being consistent, the KR can eliminate redundant or conflicting knowledge.
- **Efficient:** Easiness for modifying and updating.
- **Supports the intelligent activity which uses the knowledge base.**
- To gain a better understanding of why these characteristics represent a good knowledge representation, think about how an encyclopedia (e.g. Wikipedia) is structured. There are millions of articles (coverage), and they are sorted into categories, content types, and similar topics (understandable). It redirects different titles but same content to the same article (consistency). It is efficient, easy to add new pages or update existing ones, and allows users on their mobile phones and desktops to view its knowledge base.

## **Knowledge representation techniques**

All of these, in different ways, involve hierarchical representation of data:

- Lists: Linked lists are used to represent hierarchical knowledge
- Trees: Graphs which represent hierarchical knowledge. LISP, the main programming language of AI, was developed to process lists and trees.
- Semantic networks: Nodes and links - stored as propositions.
- Schemas: Used to represent common sense or stereotyped knowledge.
- Frames: Describe objects. Consist of a cluster of nodes and links manipulated as a whole. Knowledge is organised in slots. Frames are hierarchically organised.
- Scripts: Describe event rather than objects. Consist of stereotypically ordered causal or temporal chain of events.
- Rule-based representations: Used in specific problem-solving contexts. Involve production rules containing *if-then* or *situation-action* pairs. Specific example: problem space representations.
- Contain: Initial state, goal state, legal operators, i.e. Things you are allowed to do and operator restrictions, i.e. Factors which constrain the application of operators.
- Logic-based representations: May use deductive or inductive reasoning. Contain: facts and premises, rules of propositional logic (boolean - dealing with complete statements, rules of predicate calculus (allows use of additional information about objects in the proposition, use of variables and functions of variables, measures of certainty - may involve certainty factors (e.g. If symptom then (CF) diagnosis) which could be derived from expert estimation or from statistical data, Bayesian probability, or fuzzy logic (in which the concepts or information itself has some associated certainty value)).



IBM iSeries Innovation Center for Education

## Knowledge representation languages

- Programming is a process of representing knowledge.
- A KR language must unambiguously represent any interpretation of a sentence (logical adequacy), have a method for translating from natural language to that representation, and must be usable for reasoning.
- The main features of KR language are:
  - Object-oriented.
  - Generalization/specialization.
  - Reasoning.
  - Classification.
- Object orientation and generalization help to make the represented knowledge more understandable to humans.
- Production systems allow for the simple and natural expression of if-then rules.

Figure 2-54. Knowledge representation languages

AIR0110

### Notes:

#### Knowledge representation languages

A successful representation of some knowledge must, then, be in a form that is understandable by humans, and must cause the system using the knowledge to behave as if it knows it. The "structural ingredients" that accomplish these goals are typically found in the languages for KR, both implemented and theoretical, that have been developed over the years. It is a well-known fact that the programming is a process of representing knowledge. The medium used for this representation, programming languages, do not satisfy the criteria set down by the KR hypothesis very well. They do affect a behaviour that is consistent with the knowledge represented, but they are well known to be difficult to understand.

William Woods defines the properties of a KR language as follows: a KR language must unambiguously represent any interpretation of a sentence (logical adequacy), have a method for translating from natural language to that representation, and must be usable for reasoning. Woods' definition is merely a simplification of the KR hypothesis where "reasoning" is the only method of "engendering the behaviour that manifests that knowledge." Reasoning is essential to KR, and especially to KR languages, yet even simple reasoning capabilities can lead to serious tractability problems and thus must be well understood and used carefully.

The main features of such a language are:

- **Object-orientedness:** All the information about a specific concept is stored with that concept, as opposed, for example, to rule-based systems where information about one concept may be scattered throughout the rule base.
- **Generalization/specialization:** KR languages provide a natural way to group concepts in hierarchies in which higher level concepts represent more general, shared attributes of the concepts below.
- **Reasoning:** The ability to state in a formal way that the existence of some piece of knowledge implies the existence of some other, previously unknown piece of knowledge is important to KR. Each KR language provides a different approach to reasoning.
- **Classification:** Given an abstract description of a concept, most KR languages provide the ability to determine if a concept fits that description, this is actually a common special form of reasoning.

Object orientation and generalization help to make the represented knowledge more understandable to humans, reasoning and classification help make a system behave as if it knows what is represented. Frame-based systems thus meet the goals of the KR hypothesis. It is important to realize both the capabilities and limitations of frame-based representations, especially as compared to other formalisms. To begin with, all symbolic KR techniques are derived in one way or another from first order logic (FOL), and as a result are suited for representing knowledge that doesn't change. Different KR systems may be able to deal with non-monotonic changes in the knowledge being represented, but the basic assumption has been that change, if present, is the exception rather than the rule.

Two other major declarative KR formalisms are production systems and database systems. Production systems allow for the simple and natural expression of if-then rules. However, these systems have been shown to be quite restrictive when applied to large problems, as there is no ordering of the rules, and inferences cannot be constrained away from those dealing only with the objects of interest. Production systems are subsumed by frame-based systems, which additionally provide natural inference capabilities like classification and inheritance, as well as knowledge-structuring techniques such as generalization and object orientation.

Database systems provide only for the representation of simple assertions, without inference. Rules of inference are important pieces of knowledge about a domain frame-based systems are currently severely limited when dealing with procedural knowledge. An example of procedural knowledge would be Newton's law of gravitation - the attraction between two masses is inversely proportional to the square of their distances from each other. Given two frames representing the two bodies, with slots holding their positions and mass, the value of the gravitational attraction between them cannot be inferred declaratively using the standard reasoning mechanisms available in frame-based KR languages, though a function or procedure in a programming language could represent the mechanism for performing this "inference" quite well.

Frame-based systems that can deal with this kind of knowledge do so by adding a procedural language to its representation. The knowledge is not being represented in a frame-based way, it is being represented as C or (more commonly) LISP code. This is an important distinction - there is knowledge being encoded in those LISP functions that is not fully accessible. The system can reason with that knowledge, but not about it, in other words we can use some attached procedure to compute (or infer) the value of one slot based on some others, but we cannot ask how that value was obtained.

## Framework for knowledge representation



IBM iXO Innovation Centre for Education

- **Domain modelling:**
  - Domain modelling is the field in which the application of KR to specific domains is studied and performed.
- **Ontological analysis:**
  - Ontological analysis is the process of defining this part of the model.
- **In general, ontology consists of three parts:**
  - Concept definitions.
  - Role definitions.
  - Inference definitions.
- **A role definition may have up to three parts as well:**
  - The role taxonomy.
  - Role inverses.
  - Role restrictions.
- **Classic:**
  - Classic is a frame-based knowledge representation language that belongs to the family of description logics.

Figure 2-55. Framework for knowledge representation

AIR011.0

### Notes:

#### Framework for knowledge representation

**Domain modelling:** Domain modelling is the field in which the application of KR to specific domains is studied and performed. Domain models are represented with frame-based KR languages and programs are represented with programming languages. This traditional separation between programs and domain models causes problems during the instantiation of a domain model that includes not only knowledge of the objects and attributes, but knowledge of the procedural aspects of the processes associated with the domain as well. The predominant methodologies for domain modelling clearly indicate that the instantiation of the model is the most time-consuming part, and that the most important part of instantiation is ontological analysis. Ontologies for general taxonomies of objects are abundant, and there seem to be clear guidelines for developing new ones.

**Ontological analysis:** The word ontology means "the study of the state of being." Ontology describes the states of being of a set of things. This description is usually made up of axioms that define each thing. In knowledge representation, ontology has become the defining term for the part of a domain model that excludes the instances yet describes what they can be. Ontological analysis is the process of defining this part of the model. Each knowledge representation language differs in its manner and range of expression. In general, ontology consists of three parts: concept definitions, role definitions, and further inference definitions. The concept definitions set up all the types of objects in the domain. In object-oriented terms this is called the class definitions, and in database terms these are the entities. There can be three parts to the concept definitions:

**Concept taxonomy:** The taxonomy is common to most knowledge representation languages, and through it is specified the nature of the categories in terms of generalization and specialization. Role defaults which specify for each concept what the default values are for any attributes. Role restrictions which specify for a concept any constraints on the values in a role, such as what types the values must be, how many values there can be, etc.

**Role:**

A role is an attribute of an object. In object-oriented terms it is a slot, in database terms (and even some KR languages) it is a relation. In the simplest case, a role for an object just has a value; the object mailbox-4 might have a role number-of-messages, for example, that would have a value which is a number. Roles also express relationships between objects. The same object might have a role called owner which relates mailbox-4 to the object person-2. Roles which represent relationships are unidirectional.

A role definition may have up to three parts as well:

- The role taxonomy which specifies the generalization/specialization relationship between roles. For example, an ontology for describing cars might include roles called has-engine, has-seats, and has-headlights, which relate objects that represent cars to objects that represent engines, seats, and headlights, resp. The role has-parts, then, could be expressed as the generalization of all these roles, and the result is that all the values of all the more specialized roles would also be values of the more general role.
- Role inverses which provide a form of inference that allows the addition of a role in the opposite direction when the forward link is made. For example, if the inverse of has-engine was engine-of, then when the has-engine link between the object that represents the car and the object that represents the engine is made, the engine-of link will automatically be added between the engine object and the car object.
- Role restrictions. The role itself may be defined such that it can only appear between objects of certain types (domain/range restrictions) or can only appear a specified number of times (cardinality restriction). This is the same information specified in role restriction for concepts, some representation languages consider this information to be part of the role, and some consider it to be part of the concept.

**Classic:** Classic is a frame-based knowledge representation language that belongs to the family of description logics. Classic knowledge-bases are composed of four kinds of objects: concepts, roles, rules, and individuals. Ontology in classic consists of concept taxonomy, role taxonomy, role inverses, role restrictions and defaults. The role restrictions and defaults are specified as part of the concept definitions. Classic rules are forward chaining rules.

## Knowledge representation schemes



IBM ICE (Innovation Centre for Education)

- There are four types of Knowledge representation namely Relational, Inheritable, Inferential, and Declarative/Procedural:
  - Relational based knowledge representation scheme.
  - Inheritable knowledge representation scheme.
  - Inferential knowledge representation scheme.
  - Declarative knowledge representation scheme
  - Procedural knowledge representation scheme.

Figure 2-56. Knowledge representation schemes

AIR011.0

### Notes:

#### Knowledge representation schemes

There are four types of knowledge representation namely relational, inheritable, inferential, and declarative/procedural:

- **Relational based knowledge representation scheme:** This provides a framework to compare two objects based on equivalent attributes. Any instance in which two different objects are compared is a relational type of knowledge.
- **Inheritable knowledge representation scheme:** This is obtained from associated objects. It prescribes a structure in which new objects are created which may inherit all or a subset of attributes from existing objects.
- **Inferential knowledge representation scheme:** This is inferred from objects through relations among objects. e.g., A word alone is a simple syntax, but with the help of other words in phrase the reader may infer more from a word; this inference within linguistic is called semantics.
- **Declarative knowledge representation scheme:** This is a statement in which knowledge is specified, but the use to which that knowledge is to be put is not given. e.g., Laws, people's name; these are facts which can stand alone, not dependent on other knowledge.
- **Procedural knowledge representation scheme:** A representation in which the control information, to use the knowledge, is embedded in the knowledge itself. e.g., Computer programs, directions, and recipes; these indicate specific use or implementation.

## Properties and schemes for knowledge representation

- Properties for knowledge representation systems:
  - Representational adequacy.
  - Inferential adequacy.
  - Inferential efficiency.
  - Acquisition efficiency.
- Knowledge representation schemes
  - There are four types of knowledge representation namely:
    - Relational
    - Inheritable
    - Inferential
    - Declarative/procedural..

Figure 2.57 Properties and schemes for knowledge representation

AIR0110

### Notes:

#### Properties for knowledge representation systems

The following properties should be possessed by a knowledge representation system:

- **Representational adequacy:** The ability to represent the required knowledge;
- **Inferential adequacy:** The ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original;
- **Inferential efficiency:** The ability to direct the inferential mechanisms into the most productive directions by strong appropriate guides;
- **Acquisition efficiency:** The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

### **Knowledge representation schemes**

There are four types of knowledge representation namely relational, inheritable, inferential, and declarative/procedural. Relational based knowledge representation scheme: this provides a framework to compare two objects based on equivalent attributes. Any instance in which two different objects are compared is a relational type of knowledge.

- **Inheritable knowledge representation scheme:** This is obtained from associated objects. It prescribes a structure in which new objects are created which may inherit all or a subset of attributes from existing objects.
- **Inferential knowledge representation scheme:** This is inferred from objects through relations among objects. e.g.,.. A word alone is a simple syntax, but with the help of other words in phrase the reader may infer more from a word; this inference within linguistic is called semantics.
- **Declarative knowledge representation scheme:** A statement in which knowledge is specified, but the use to which that knowledge is to be put is not given. e.g.,.. Laws, people's name; these are facts which can stand alone, not dependent on other knowledge;
- **Procedural knowledge representation scheme:** A representation in which the control information, to use the knowledge, is embedded in the knowledge itself. e.g.,.. Computer programs, directions, and recipes; these indicate specific use or implementation.

## Inheritable knowledge representation scheme

IBM

IBM Almaden Research Center for Education

- The knowledge is embodied in the design hierarchies found in all the three domains.
- The inheritance is a powerful form of inference, but not adequate.

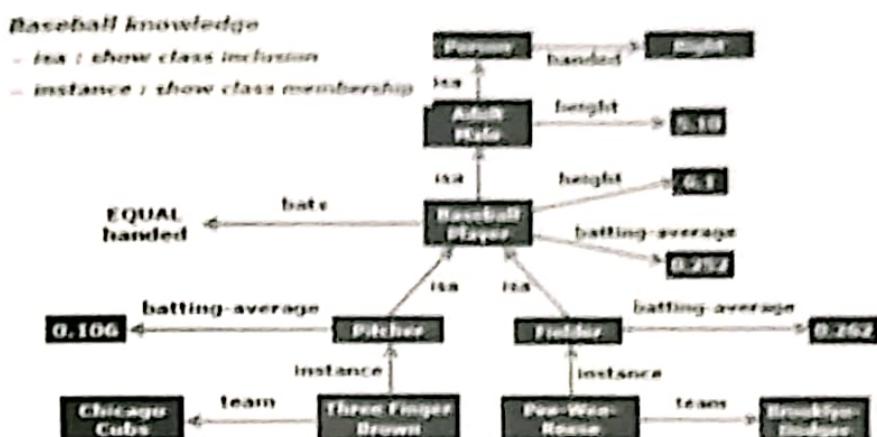


Figure: Inheritable knowledge representation (KR)

Figure 2-59. Inheritable knowledge representation scheme

AIR01

### Notes:

#### Inheritable knowledge representation scheme

Here the knowledge elements inherit attributes from their parents. The knowledge is embodied in the design hierarchies found in the functional, physical and process domains. Within the hierarchy, elements inherit attributes from their parents, but in many cases not all attributes of the parent elements be prescribed to the child elements. The inheritance is a powerful form of inference, but not adequate. The basic KR needs to be augmented with inference mechanism. The KR in hierarchical structure, shown below, is called "semantic network" or a collection of "frames" or "slot-and-filler structure". The structure shows property inheritance and way for insertion of additional knowledge.

**Property inheritance:** the objects or elements of specific classes inherit attributes and values from more general classes. The classes are organized in a generalized hierarchy. The above figure shows inheritable Knowledge Representation (KR) where the directed arrow represent attributes (i.e., instance, team) originates at object being described and terminates at object or its value. The box nodes represent objects and values of the attributes.

## Relational based knowledge representation scheme



IBM ICE (Innovation Centre for Education)

- The simplest way of storing facts is to use a relational method.
- This scheme is a:
  - Simple way to store facts.
  - Each fact about a set of objects is set out systematically in columns.
  - Little opportunity for inference.
  - Knowledge basis for inference engines.
- The table below shows a simple way to store facts. The facts about a set of objects are put systematically in columns.

Player	Height	Weight	Bats - Throws
Aaron	6-0	180	Right – Right
Maya	5-10	170	Right - Right
Ruth	6-2	215	Left - Left
Williams	6-3	205	Left - Right

Figure 2-58. Relational based knowledge representation scheme

AIR011.0

### Notes:

#### Relational based knowledge representation scheme

The simplest way of storing facts is to use a relational method where each fact about a set of objects is set out systematically in columns. Here, knowledge associates elements of one domain with another domain. The relational knowledge is made up of objects consisting of attributes and their corresponding associated values. The results of this knowledge type are a mapping of elements among different domains. This representation gives little opportunity for inference, but it can be used as the knowledge basis for inference engines.

This scheme is a:

- Simple way to store facts.
- Each fact about a set of objects is set out systematically in columns.
- Little opportunity for inference.
- Knowledge basis for inference engines.

Given the facts it is not possible to answer simple question such as:

- "Who is the heaviest player? ". but if a procedure for finding heaviest player is provided, then these facts will enable that procedure to compute an answer.
- We can ask things like who "bats – left" and "throws – right".

## Inheritable knowledge representation scheme



IBM ICE (Innovation Centre for Education)

- The knowledge is embodied in the design hierarchies found in all the three domains.
- The inheritance is a powerful form of inference, but not adequate.

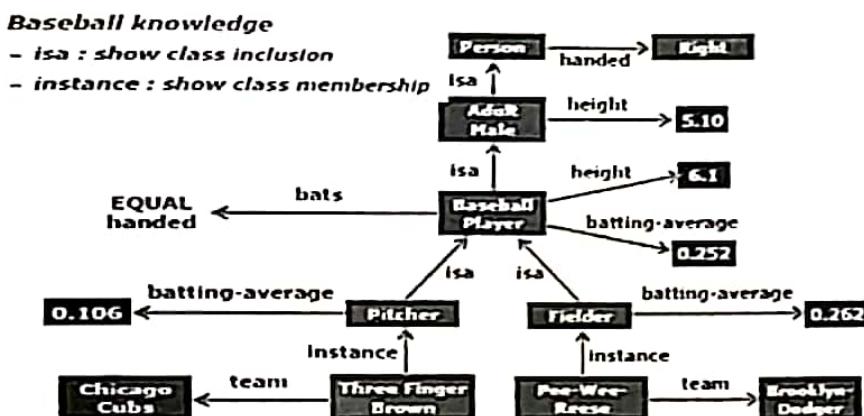


Figure: Inheritable knowledge representation (KR)

Figure 2-59. Inheritable knowledge representation scheme

AIR0110

### Notes:

#### Inheritable knowledge representation scheme

Here the knowledge elements inherit attributes from their parents. The knowledge is embodied in the design hierarchies found in the functional, physical and process domains. Within the hierarchy, elements inherit attributes from their parents, but in many cases not all attributes of the parent elements be prescribed to the child elements. The inheritance is a powerful form of inference, but not adequate. The basic KR needs to be augmented with inference mechanism. The KR in hierarchical structure, shown below, is called "semantic network" or a collection of "frames" or "slot-and-filler structure". The structure shows property inheritance and way for insertion of additional knowledge.

**Property Inheritance:** the objects or elements of specific classes inherit attributes and values from more general classes. The classes are organized in a generalized hierarchy. The above figure shows inheritable Knowledge Representation (KR) where the directed arrow represent attributes (*is a*, *instance*, *team*) originates at object being described and terminates at object or its value. The box nodes represent objects and values of the attributes.

**Algorithm: Property inheritance**

Retrieve a value v for an attribute a of an instance object o. Steps to follow:

- Find object O in the knowledge base.
- If there is a value for the attribute a then report that value.
- Else, if there is a value for the attribute instance; if not, then fail.
- Else, move to the node corresponding to that value and look for a value for the attribute a; if one is found, report it.
- Else, do until there is no value for the "is a" attribute or until an answer is found: get the value of the "is a" attribute and move to that node and see if there is a value for the attribute a; if yes, report it.

This algorithm is simple. It describes the basic mechanism of inheritance. It does not say what to do if there is more than one value of the instance or "is a" attribute.

## Inferential knowledge representation scheme



IBM ICE (Innovation Centre for Education)

- This knowledge generates new information from the given information.
- This new information require analysis of the given information to generate new knowledge.
  - Example:
    - Given a set of relations and values, one may infer other values or relations.
- The symbols used for the logic operations are: "  $\rightarrow$  " (implication), "  $\neg$  " (not), "  $\vee$  " (or), "  $\lambda$  " (and), "  $\forall$  " (for all), "  $\exists$  " (there exists).
  - Examples of predicate logic statements:
    - "Wonder" is a name of a dog : dog (wonder).
    - All dogs belong to the class of animals :  $\forall x : \text{dog}(x) \rightarrow \text{animal}(x)$ .

Figure 2-60. Inferential knowledge representation scheme

AIR011.0

### Notes:

#### Inferential knowledge representation scheme

This knowledge generates new information from the given information. This new information does not require further data gathering form source but does require analysis of the given information to generate new knowledge. Example:

- Given a set of relations and values, one may infer other values or relations.
- A predicate logic (a mathematical deduction) is used to infer from a set of attributes.
- Inference through predicate logic uses a set of logical operations to relate individual data.

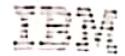
The symbols used for the logic operations are: "  $\rightarrow$  " (implication), "  $\neg$  " (not), "  $\vee$  " (or), "  $\lambda$  " (and).

Examples of predicate logic statements:

- "Wonder" is a name of a dog : dog (wonder).
- All dogs belong to the class of animals :  $x : \text{dog}(x) \rightarrow \text{animal}(x)$ .
- All animals either live on land or in water:  $x : \text{animal}(x) \rightarrow \text{live}(x, \text{land}) \vee \text{live}(x, \text{water})$ .

From these three statements we can infer that: "wonder lives either on land or on water".

**Note:** If more information is made available about these objects and their relations, then more knowledge can be inferred.



## Declarative/procedural knowledge

IBM ICE (Innovation Centre for Education)

- Declarative knowledge:
  - Based on declarative facts about axioms and domains.
- Procedural knowledge:
  - A mapping process between domains that specify "what to do when".
- The procedural knowledge:
  - May have inferential efficiency.
  - Represented as small programs.
- Advantages:
  - Heuristic or domain specific knowledge can be represented.
- Disadvantages:
  - Completeness.
  - Consistency.

Figure 2-61. Declarative/procedural knowledge

AIR011.0

### Notes:

#### Declarative/procedural knowledge representation scheme

Here, the basic idea is that the knowledge is encoded in some procedures. The small programs know how to do specific things, how to proceed. Example: A parser in a natural language has the knowledge that a noun phrase may contain articles, adjectives and nouns. It is represented by calls to routines that know how to process articles, adjectives and nouns. However, the differences between declarative/procedural knowledge are not very clear.

**Declarative knowledge:** Here, the knowledge is based on declarative facts about axioms and domains:

- Axioms are assumed to be true unless a counter example is found to invalidate them.
- Domains represent the physical world and the perceived functionality.
- Axiom and domains thus simply exists and serve as declarative statements that can stand alone.

**Procedural knowledge:** Here, the knowledge is a mapping process between domains that specify "what to do when" and the representation is of "how to make it" rather than "what it is". The procedural knowledge:

- May have inferential efficiency, but no inferential adequacy and acquisitional efficiency.
- Are represented as small programs that know how to do specific things, how to proceed.

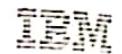
**Advantages:**

- Heuristic or domain specific knowledge can be represented.
- Extended logical inferences, such as default reasoning facilitated.
- Side effects of actions may be modeled. Some rules may become false in time. Keeping track of this in large systems may be tricky.

**Disadvantages:**

- Completeness: Not all cases may be represented.
- Consistency: Not all deductions may be correct, e.g., If we know that Fred can fly. Later we might discover that Fred is an EMU.
- Modularity is sacrificed.
- Changes in knowledge base might have far-reaching effects.
- Cumbersome control information.

## Planning (1 of 2)



IBM ICE (Innovation Centre for Education)

- The term 'planning' is typically used in AI when the problem is a "real-world" problem.
- It is very difficult and usually impossible to solve real "real-world" problems using the method of state space search.
- State space search requires complete description of the states searched and requires the search to be carried out locally.
- Planning is an important activity for autonomous agents. It is used for more than just moving objects around.

Figure 2-62. Planning (1 of 2)

AIR011.0

### Notes:

#### Planning

As we will see, many of the same issues addressed in problem solving research arise in planning research. However, the term 'planning' is typically used in AI when the problem is a "real-world" problem. The scare quotes surround "real-world" because many of the examples in the planning literature are based on block-stacking problems. Stacking blocks is perhaps a real-world problem prior to the age of four, but most of us don't spend much time stacking blocks in our respective "real-worlds". It is very difficult and usually impossible to solve real "real-world" problems using the method of state space search. State space search has two properties which contribute to this difficulty. State space search requires complete description of the states searched and requires the search to be carried out locally.

Completely describing each state that is expanded during a search can be computationally expensive when real world problems are involved. In puzzle problems such as the 15 puzzle, the total information about a current state can be represented using 16 atomic expressions that declare the occupancy status of each location together with an invariant set of expressions that describe the contiguity structure of the locations. This is a relatively small number of expressions. Compare this to the number of expressions required to represent something as mundane as the room in which you are currently located. The method of problem reduction operates on descriptions of "problem states". These problem states are partial rather than complete descriptions of a state of the world. Some variant of the method of problem reduction is used in most of the planning research that we will encounter.

The local property of state space search refers to the fact that search can proceed only by expanding states that are directly reachable from the current state. Many of the examples of problem reduction that will be examined allow for the generation of so-called "island states." An island state (or really a partial state) is a state through which the planner believes the solution must pass. For example, to attend a particular rock concert, the planner may believe that a ticket to this concert will be required. Put another way, this is a belief that the solution path will involve passing through at least one state in which the planner possesses a ticket to the concert.

The significance of island states can be appreciated if we consider the combinatorics of orderings of actions. If a solution to a problem requires only 7 actions, then there are potentially  $7!$  or 5,040 possible orderings of these actions. However, if one of these actions must occur as the fourth action and we know which action must occur in this position then there are now only  $2(3!)$  or 12 possible orderings to consider. The ability to introduce information about partial states that aren't directly reachable from a current state allows problem reduction methods to escape the limitations of the local property of state space search.

Planning is an important activity for autonomous agents. It is used for more than just moving objects around. Examples include the following:

- Planning to achieve some goal such as move an object from one location to another.
- Planning a discourse that will cause another agent to do something.
- Understanding the plans of another agent in order to be able to assist (thwart) that agent.

The purpose of planning is to find a sequence of actions that achieves a given goal when performed starting in a given state. In other words, given a set of operator instances (defining the possible primitive actions by the agent), an initial state description, and a goal state description or predicate, the planning agent computes a plan.

## Planning (2 of 2)

- A sequence of operator instances.
- A simple planning agent.
- Problem solving agents + knowledge-based agents = planning agents.
- The pseudo-code is as shown below:

```

function SIMPLE-PLANNING-AGENT (percept) returns an action
  static KB, a knowledge base (includes action descriptions)
        p, a plan, initially NoPlan
        t, a counter, initially 0, indicating time
  local variables G, a goal
                    current, a current state description

  TELL(KB, MAKE-PERCEPT-SENTENCE (percept,t))
  current :- STATE-DESCRIPTION(KB, t)
  if p = NoPlan then
    G :- ASK (KB, MAKE-GOAL-QUERY(t))
    p :- IDEAL-PLANNER(current, G, KB)
  if p = NoPlan or p is empty then action :- NoOp
  else
    action :- FIRST(p)
    p :- REST(p)
  TELL(KB, MAKE-ACTION-SENTENCE(action,t))
  t :- t+1
  return action
```

Figure 2-63. Planning (2 of 2)

AIR011.0

### Notes:

#### What is a plan?

A sequence of operator instances, such that "executing" them in the initial state will change the world to a state satisfying the goal state description. Goals are usually specified as a conjunction of goals to be achieved.

#### A simple planning agent:

Earlier we saw that problem-solving agents are able to plan ahead - to consider the consequences of sequences of actions - before acting. We also saw that knowledge-based agents can select actions based on explicit, logical representations of the current state and the effects of actions. This allows the agent to succeed in complex, inaccessible environments that are too difficult for a problem-solving agent.

#### Problem solving agents + knowledge-based agents = planning agents

We put these two ideas together to build planning agents. At the most abstract level, the task of planning is the same as problem solving. Planning can be viewed as a type of problem solving in which the agent uses beliefs about actions and their consequences to search for a solution over the more abstract space of plans, rather than over the space of situations.

**Algorithm of a simple planning agent:**

- Generate a goal to achieve.
- Construct a plan to achieve goal from current state.
- Execute plan until finished.
- Begin again with new goal.

The agent first generates a goal to achieve, and then constructs a plan to achieve it from the current state. Once it has a plan, it keeps executing it until the plan is finished, then begins again with a new goal. This is illustrated as shown in the pseudo-code above.

**Assumptions:**

A simple planning agent creates and uses plans based on the following assumptions:

- **Atomic time:** Each action is indivisible.
- **No concurrent actions allowed.**
- **Deterministic actions:** Result of each action is completely determined by the definition of the action, and there is no uncertainty in performing it in the world.
- **Agent is the sole cause of change in the world.**
- **Agent is omniscient** has complete knowledge of the state of the world.
- **Closed world assumption** everything known to be true in the world is included in a state description. Anything not listed is false.

**Components of a planning system:** Classical planners use the STRIPS (Stanford Research Institute Problem Solver) language to describe states and operators. It is an efficient way to represent planning algorithms.

# Representation of states, goals and actions

IBM ICE (Innovation Centre for Education)

IBM

- States are represented by conjunctions of function-free ground literals.
- Representation of actions:
  - Strips operators consist of three components.
- Preconditions and effects are restrictive.
- The set of operators for the "box world" example problem is shown below:

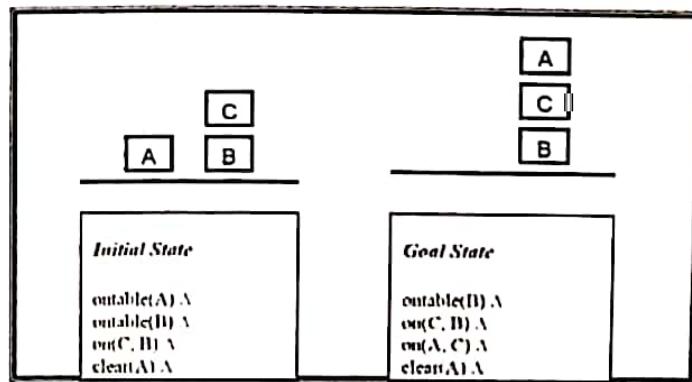


Figure 2-64. Representation of states, goals and actions

AIR011.0

## Notes:

### Representation of states and goals

States are represented by conjunctions of function-free ground literals, that is, predicates applied to constant symbols, possibly negated. An example of an initial state is:  $at(home) \wedge \neg have(milk) \wedge \neg have(bananas) \wedge \neg have(drill)$ . A state description does not have to be complete. We just want to obtain a successful plan to a set of possible complete states. But if it does not mention a given positive literal, then the literal can be assumed to be false.

- **Goals are a conjunction of literals:** Therefore the goal is  $at(home) \wedge have(milk) \wedge have(bananas) \wedge have(drill)$ .
- **Goals can also contain variables:** Being at a store that sells milk is equivalent to  $at(x) \wedge sells(x, milk)$ .

We have to differentiate between a goal given to a planner which is producing a sequence of actions that makes the goal true if executed, and a query given to a theorem prover that produces true or false if there is truth in the sentences, given a knowledge base. We also have to keep track of the changes rather than of the states themselves because most actions change only a small part of the state representation.

### Representation of actions

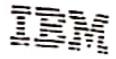
Strips operators consist of three components:

- **Action description:** What an agent returns to the environment in order to do something.
- **Precondition:** Conjunction of atoms (positive literals), that says what must be true before an operator can be applied.
- **Effect of an operator:** Conjunction of literals (positive or negative) that describe how the situation changes when the operator is applied.

An example action of going from one place to another:  $Op(\text{action: } go(\text{there}))$ , PRECOND:  $at(here) \wedge path(here, there)$  EFFECT:  $at(there) \wedge \neg at(here)$ ) the following above figure shows a diagram of the operator go(there). The preconditions appear above the action, and the effects below.

- **Operator schema:** It is an operator with variables. It is a family of actions, one for each of the different values of the variables. Every variable must have a value.
- **Preconditions and effects are restrictive:** Operator o is applicable in a state s if every one of the preconditions in o are true in s. An example is if the initial situation includes the literals.
- **At(home, path(home, supermarket))**: Then the action go (supermarket) is applicable, and the resulting situation contains the literals.
- **At (home), at (supermarket), path (home, supermarket)**: The result is all positive literals in effect (o) hold, all literals in s hold and negative literals in effect (o) are ignored. The set of operators for the "box world" example problem is shown in the figure.

## Goal stack planning (1 of 2)



IBM ICE (Innovation Centre for Education)

- Basic idea to handle interactive compound goals uses goal stacks, here the stack contains:
  - Goals.
  - Operators -- ADD, DELETE and PREREQUISITE lists.
  - A database maintaining the current situation for each operator used.
- Goal stack planning algorithm is given in the notes.

Figure 2-65. Goal stack planning (1 of 2)

AIR011.0

### Notes:

#### Definitions of operators

- Ontable(x): Block x is on top of the table
- On(x,y): Block x is on top of block y
- Clear(x): There is nothing on top of block x; therefore it can be picked up
- Handempty: You are not holding any block

Op {action: pickup(x)}

PRECOND: ontable(x), clear(x), handempty

EFFECT: holding(x), ~ontable(x), ~clear(x), ~handempty } O

{action: putdown(x)}

PRECOND: holding(x)

EFFECT: ontable(x), clear(x), handempty, ~holding(x) }

Op{action: stack(x,y)}

PRECOND: holding(x), clear(y)

EFFECT: on(x,y), clear(x), handempty, ~holding(x), ~clear(y) }

EFFECT: on(x,y), clear(x), handempty, ~holding(x), ~clear(y) }

Op {action: unstack(x,y)}

PRECOND: clear(x), on(x,y), handempty

EFFECT: holding(x), clear(y), ~clear(x), ~on(x,y), ~handempty}

### **Goal stack planning**

Basic idea to handle interactive compound goals uses goal stacks, here the stack contains:

- Goals.
- Operators: Add, delete and prerequisite lists.
- A database maintaining the current situation for each operator used.

### **Goal stack planning algorithm:**

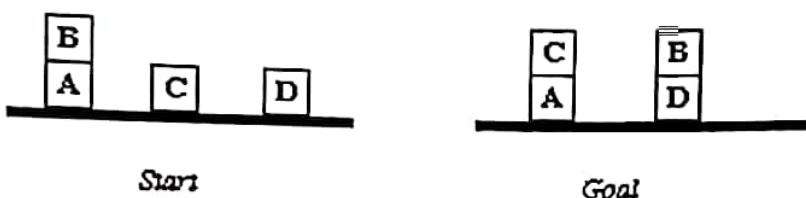
1. Initialize database by the initial state (it represents the current state).
2. Initialize the planning action list to nil.
3. Add goal to the goal stack.
4. If the top of the stack is a condition satisfied by the database, remove it from the goal stack. Go to step # 7.
5. If the top of the stack is a condition not satisfied by the database, replace it by the action (whose add list will satisfy this condition) and add preconditions of the action to the goal stack. If more than one action can be applied, use heuristic to choose the best one. Go to step #7.
6. If the top of the goal stack is the action (whose preconditions are satisfied), then remove it from the goal stack and add it to the planning action list. Insert "add conditions" of the action to the database. Remove "delete conditions" of the action from the database.
7. If the goal stack is not empty, then go to step 4, else planning action list contains the plan to achieve the goal.

## Goal stack planning (2 of 2)

IBM ICE (Innovation Centre for Education)



- Goal stack planning example is as shown in figure:



We can describe the *start* state:

ON (B, A)  $\wedge$  ONTABLE (A)  $\wedge$  ONTABLE(C)  $\wedge$  ONTABLE (D)  
 $\wedge$  ARMEMPTY

and *goal* state:

ON(C, A)  $\wedge$  ON (B, D)  $\wedge$  ONTABLE (A)  $\wedge$  ONTABLE (D)

Figure 2-66. Goal stack planning (2 of 2)

AIR011.0

### Notes:

Consider the following where wish to proceed from the *start* to *goal* state.

We can describe the *start* state:

ON (B, A) ONTABLE (A) ONTABLE(C) ONTABLE (D) ARMEMPTY *goal state*:  
 ON (C, A) ON (B, D) ONTABLE (A) ONTABLE (D)

- Initially the goal stack is the goal state.
- We then split the problem into four sub-problems.
- Two are solved as they already are true in the initial state – ONTABLE (A), ONTABLE (D).
- With the other two – there are two ways to proceed:
  - ON(C,A), ON(B,D), ON(C,A), ON(B,D), ONTABLE(A), ONTABLE(D)
  - ON(B,D), ON(C,A), ON(C,A), ON(B,D), ONTABLE(A), ONTABLE(D)

The method is to:

- Investigate the first node on the stack i.e. the top goal.
- If a sequence of operators is found that satisfies this goal it is removed, and the next goal is attempted.
- This continues until the goal state is empty.

Consider alternative 1 above further:

The first goal  $ON(C,A)$  is not true and the only operator that would make it true is  $STACK(C,A)$  which replaces  $ON(C,A)$  giving:

$B \leftarrow STACK(C,A)$

$ON(B,D)$

$ON(C,A) \wedge ON(B,D)$

$\wedge ONTABLE(A) \wedge$

$ONTABLE(D)$

$STACK$  has prerequisites that must be met which mean that block A is clear, and the arm is holding block C.  
So we must do:

$B \leftarrow CLEAR(A)$

$HOLDING(C)$

$CLEAR(A) \wedge HOLDING(C)$

$STACK(C,A)$

$ON(B,D)$

$ON(C,A) \wedge ON(B,D)$

$\wedge ONTABLE(A) \wedge$

$ONTABLE(D)$

Now top goal is false and can only be made true by un-stacking B. This leads to:

$B \leftarrow ON(B,A)$

$CLEAR(B)$

$ARMEMPTY$

$ON(B,A) \wedge CLEAR(B)$

$\wedge ARMEMPTY$

$UNSTACK(B,A)$

$HOLDING(C)$

$CLEAR(A) \wedge HOLDING(C)$

Now the first goal is true, the second is universally true, and the arm is empty. Thus all top three goals are true means that we can apply the operator  $UNSTACK(B,A)$  as all prerequisites are met. This gives us the first node in database.

$ONTABLE(A) \wedge ONTABLE(C) \wedge ONTABLE(D) \wedge HOLDING(C) \wedge CLEAR(A)$

Note as a future reference of the use of  $UNSTACK(B,A)$  that  $HOLDING(B)$  is true as well as  $CLEAR(A)$

The goal stack becomes

$HOLDING(C)$

$CLEAR(A) \wedge HOLDING(C)$

STACK(C,A)  
 ON(B,D)  
 ON(C,A) ^ ON(B,D) ^ ONTABLE(A)  
 ^ONTABLE(D)

There are two ways we can achieve HOLDING(C) by using the operators PICKUP(C) or UNSTACK(C,x) where x is an unspecified block. This leads to two alternative paths:

ON(C, x)  
 CLEAR(C)  
 ARMEMPTY  
 ON(C, x) ^ CLEAR(C)  
 ^ARMEMPTY  
 UNSTACK(C,x)  
 CLEAR(A) ^ HOLDING(C)  
 STACK(C,A)  
 ON(B,D)  
 ON(C,A) ^ ON(B,D) ^ ONTABLE(A)  
 ONTABLE(D)  
 ONTABLE(C)  
 CLEAR(C)  
 ARMEMPTY  
 ONTABLE(C) ^ CLEAR(C)  
 ^ARMEMPTY  
 PICKUP(C)  
 CLEAR(A) ^ HOLDING(C)  
 STACK(C,A)  
 ON(B,D)  
 ON(C,A) ^ ON(B,D) ^ ONTABLE(A)  
 ^  
 ONTABLE(D)

In this first route we can see three references to some block, x and these must refer to the same block, although in the search it is conceivable several blocks will become temporarily attached. Hence the binding of variables to blocks must be recorded. Investigating further we need to satisfy the first goal, and this requires stacking C on some block which is clear.

CLEAR(x)  
 HOLDING(C)  
 CLEAR(x) HOLDING(C)  
 STACK(C, x)  
 CLEAR(C)  
 ARMEMPTY

We now notice that one of the goals created is HOLDING(C) which was the goal we were trying to achieve by applying UNSTACK(C, *some block*) in this case and PICKUP(C) in the other approach. So it would appear that we have added new goals and not made progress and in terms of the A\* algorithm it seems best to try the other approach.

So looking at the second approach.

- We can see that the first goal is achieved block C is on the table.
- The second goal is also achieved block C is clear.
- Remember that HOLDING(B) is still true which means that the arm is not empty. This can be achieved by placing B on the table or placing it on block D if it is clear.
- Look ahead could be used here to compare the ADD lists of the competing operators with the goals in the goal stack and there is a match with ON(B,D) which is satisfied by STACK(B,D). This also binds some block to block D.
- Applying STACK(B,D) generates extra goals CLEAR(D) and HOLDING(B).

The new goal stack becomes;

CLEAR(D)

HOLDING(B)

CLEAR(D) ^ HOLDING(B)

STACK(B, D)

ONTABLE(C) ^ CLEAR(C) ^ ARMEMPTY

PICKUP(C)

At this point the top goal is true and the next and thus the combined goal leading to the application of STACK(B, D), which means that the world model becomes

ONTABLE(A) ONTABLE(C) ONTABLE(D) ON(B, D) ARMEMPTY. This means that we can perform PICKUP(C) and then STACK(C, A). Now coming to the goal ON(B,D) we realise that this has already been achieved and checking the final goal we derive the following plan:

- UNSTACK(B,A)
- STACK(B,D)
- PICKUP(C)
- STACK(C,A)

This method produces a plan using good Artificial Intelligence techniques such as heuristics to find matching goals and the A\* algorithm to detect unpromising paths which can be discarded.

## Checkpoint (1 of 2)



### Multiple choice questions:

1. BFS stands for \_\_\_\_\_.
  - a) Binary first search
  - b) Breadth first search
  - c) Block first statement
  - d) All of these above
  
2. The major components of an AI production system are \_\_\_\_\_.
  - a) A global database
  - b) A set of production rules
  - c) A control system
  - d) All of these above
  
3. The generate-and-test is a \_\_\_\_\_.
  - a) Depth first search
  - b) Exhaustive search
  - c) Binary first search
  - d) Both a) and b)

Figure 2-67. Checkpoint (1 of 2)

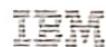
AIR011.0

### Notes:

Write your answers here:

- 1.
- 2.
- 3.

## **Checkpoint (2 of 2)**



IBM ICE (Innovation Centre for Education)

### **Fill in the blanks:**

1. A state can have a number of \_\_\_\_\_ states.
2. \_\_\_\_\_ describes a case where a leading expert on lymph-node pathology scoffs at a program's diagnosis of an especially difficult case.
3. \_\_\_\_\_ is helpful to think of the search process as building up a search tree of routes through the state space graph.
4. \_\_\_\_\_ expands the leaf node with the highest path cost so far, and keeps going until a goal node is generated.

### **True or False:**

1. AI is defined as the science and engineering of making machines intelligent with the help of intelligent agents' programs. True/ False
2. Computation of heuristic function can be done with infinite amount of computation. True/ False
3. Medical diagnosis programs based on probabilistic analysis have been able to perform at the level of an expert physician in several areas of medicine. True/ False

---

Figure 2-68. Checkpoint (2 of 2)

AIR011.0

### **Notes:**

Write your answers here:

### **Fill in the blanks:**

- 1.
- 2.
- 3.
- 4.

### **True or False:**

- 1.
- 2.
- 3.

## Question bank

### Two mark questions:

1. Define artificial intelligence and what are its applications?
2. Define problem and problem space. Illustrate.
3. Define knowledge representation. What are its characteristics?
4. Explain search paradigm and its importance in robotics.

### Four mark questions:

1. Discuss pegs and disks problem considering three disks.
2. Define BFS. Illustrate with an example.
3. Discuss any two heuristic search techniques with an example.
4. Discuss blocks of world problem with example.

### Eight mark questions:

1. Explain 8 queens' problem?
2. Write about goal stack planning algorithm with examples?

### Notes:

## **Unit summary**

**Having completed this unit, you should be able to:**



IBM ICE (Innovation Centre for Education)

- Gain knowledge on the basics of Artificial Intelligence (AI)
- Gain an insight into the AI problems and techniques
- Learn about the state space search and production systems
- Understand the concept of problem characteristics and search paradigm
- Learn about heuristic search techniques and knowledge representation

---

AIR0110

Figure 2-70. Unit summary

### **Notes:**

---

2-106 AIR01

Course materials may not be reproduced in whole or in part  
without the prior written permission of IBM.

© Copyright IBM Corp. 2019

## **Unit 3. Components of an Intelligent Robotic System**

### **What this unit is about**

This unit helps to gain knowledge on the basic concepts of robotics and its components. The role of machine learning in modern day robotics industry are discussed and hence the details on how machine learning concept is embedded in robotics is presented. This unit helps to gain an insight into the design and development of robotic components. This unit helps to learn about environment capturing sensors like CCD cameras and also helps to gain knowledge on the integration of these sensors with real time robotic system. The generic model of machine vision system along with functional components is also presented here. This unit provides an understanding on the role of neural networks for tool condition monitoring systems.

### **What you should be able to do**

After completing this unit, you should be able to:

- Gain knowledge on the basic concepts of robotics and its components
- Gain an insight into the role of machine learning in modern day robotics industry
- Learn about the kinematic and dynamic control concept with a focus on intelligent gripping systems
- Gain an insight into the design and development of robotic components
- Learn about environment capturing sensors like CCD cameras
- Gain knowledge on the integration of sensors with real time robotic system
- Learn about the fuzzy classification and uncertainties in tool condition monitoring system

### **How you will check your progress**

- Checkpoints

### **References**

IBM Knowledge Center

## **Unit objectives**

**After completing this unit, you should be able to:**



IBM ICE (Innovation Centre for Education)

- Gain knowledge on the basic concepts of robotics and its components
- Gain an insight into the role of machine learning in modern day robotics industry
- Learn about the kinematic and dynamic control concept with a focus on intelligent gripping systems
- Gain an insight into the design and development of robotic components
- Learn about environment capturing sensors like CCD cameras
- Gain knowledge on the integration of sensors with real time robotic system
- Learn about the fuzzy classification and uncertainties in tool condition monitoring system

---

Figure 3-1. Unit objectives

AIR011.0

### **Notes:**

Unit objectives are as stated above.

---

3-2 AIR01

Course materials may not be reproduced in whole or in part  
without the prior written permission of IBM.

© Copyright IBM Corp. 2019

## Introduction to robotics



IBM ICE (Innovation Centre for Education)

- Robotics is an inter disciplinary branch of engineering and science that deals with:
  - Design.
  - Construction.
  - Operation.
  - Use of robots.
  - Computer systems for their control.
  - Sensory feedback.
  - Information processing.
- Robotics is a multi-disciplinary domain it comprises of:
  - Mechanical Engineering - Deals with the machinery and structure of the robots.
  - Electrical Engineering - Deals with the controlling and sensing of robots.
  - Computer Engineering - Deals with the movement development and observation of robots.
- The robotics technologies are used to develop automated systems. It can be a replace for humans and imitate human actions.
- Uses of Robots:
  - Manufacturing processes.
  - Where humans cannot survive (e.g. in space).
  - An attempt to replicate walking, lifting, speech, cognition, and basically anything a human can do
  - Contribute to the field of bio-inspired robotics.

Figure 3-2. Introduction to robotics

AIR0110

### Notes:

Robotics is an interdisciplinary branch of engineering and science that deals with the design, construction, operation, and use of robots, as well as computer systems for their control, sensory feedback, and information processing. According to the "Robot Institute of America, a robot is defined as a reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks". Robotics is a reasonably new domain of engineering which is a multi-disciplinary domain comprises of mechanical engineering that deals with the machinery and structure of the robots, electrical engineering that deals with the controlling and sensing of robots and computer engineering that deals with the movement development and observation of robots.

The robotics technologies are used to develop automated systems that can be a replace for humans and imitate human actions. Robots can be used in many fields such as manufacturing processes, or where humans cannot survive (e.g. in space). Robots are made an attempt to replicate walking, lifting, speech, cognition, and basically anything a human can do. Many of today's robots are inspired by nature, contributing to the field of bio-inspired robotics.

"The concept of creating machines that can operate autonomously dates back to classical times, but research into the functionality and potential uses of robots did not grow substantially until the 20th century". Today, robotics is a rapidly growing field, as we have been witnessing the technological advances, researching, designing, and building new robots that serve various practical purposes, whether domestically, commercially, or militarily.

Many robots are built to do jobs that are hazardous to people such as finding survivors in unstable ruins and exploring mines and wreckages. As a teaching aid, Robotics is also explored in science, technology, engineering, and mathematics as a teaching aid.

The traces of the first robot can be found deep into the 18th century. These were simple mechanical dolls that imitated human and animal actions and their main aim is to demonstrate the art of mechanical design. These are called 'Generation-0' robots.

- The Generation-1 (second level) started in the 60s with the aid of programmable controllers. One of the main component is the Central Processing Unit (CPU) and is programmed to execute a sequence of moves, trained by a human. The program is stored in a paper tape and the overall utility of the robot is quite less.
- The Generation-2 robots came into existence during 1980s and their main feature is the use of a microprocessor. The mechanism is made cheaper by placing the microprocessor in a small chip and thus reduced the size of a robot. The robots under this generation support high level languages such as BAPS and VAL.
- The use of coordinate transformations, feedback control and PID, vision, networking and sensors made their application attractive to industry and thus usage of robots in manufacturing industries for machine-critical application has been raised.
- The third generation evolved over the last twenty years with main features like exploration of human-like sensors and Artificial Intelligence. In these days, the robots behave like human beings and their applications are abundant. For example, from autonomous driver vehicles to cleaning robots, and from medical applications to nano-robots. The applications of this generation of robots are rapidly increasing and it has been said that in the days ahead, robots will be a part of our daily activities in our life.

## Types of robots

- The types of robots are:
  - Outer space.
  - The intelligent home.
  - Exploration.
  - Military robots.
  - Farms robots.
  - The car industry.
  - Hospitals.
  - Disaster areas.
  - Entertainment.

Figure 3-3. Types of robots

AIR011.0

### Notes:

#### Types of robots

Robotics is an area of curiosity to human-beings for more than one hundred years. On the other hand, our opinion over robots is influenced by the media and international film industry. The types of robots are generally classified based on the robot's distinctiveness transforms and the atmosphere it works in. Below are some of the areas of robotics:

- **Outer Space:** Robotic arms that are under the control of a human being are employed to unload the docking cove of outer-space shuttles to launch satellites or to build a space station.
- **The Intelligent Home:** Robotic systems can now-a-days inspect home safety, ecological circumstances & energy consumption, locking and unlocking of doors and windows, automatic programming system to turn on lights and AC systems. This kind of robots are common which also explore IoT based solutions to achieve higher level of accuracy.
- **Exploration:** Robots can go into the environments that are harmful to human beings. An example is the observation of atmosphere within a volcano or investigating deep marine life. In fact, NASA has used robots for environmental study since the early 60's.
- **Military Robots:** In the defense sector, the flying robots called drones are used to have a close watch of the armed force.
- **Farms Robots:** We have seen the usage of pre-programmed robots that are used by farmers to cut and collect yields. Robotic milk farms are extensively used to nourish and milk their cattle distantly.

- **The Car Industry:** Robotic arms are used to execute several tasks in the car manufacturing and assembling process. They execute works such as sorting, cutting, welding, lifting, painting and bending. Now-a-days, even in food industries, robots are used to perform tasks such as trimming, cutting and processing different types of meats like- chicken, beef, fish, lamb, etc.
- **Hospitals:** The development of a robotic suit is under-way which will help nurses to raise patients without injuring their backbones. Scientists in Japan have crafted a power facilitated suit which provides additional power to nurses that they require to lift patients.
- **Disaster Areas:** These robots are built-in with superior sensing and imaging gears which works in hazardous environments like urban site spoiled by earthquakes by inspecting floors, walls, and roofs for structural reality.
- **Entertainment:** These robots are interactive in nature that shows behaviors and education capability. One such robot is owned by SONY which moves around freely, responds to all your commands, carries your luggage and even responds to your oral instructions.

## Classification of robots

- Applications:
  - Industrial.
  - Domestic or household robots.
  - Medical robots.
  - Service robots.
  - Military robots.
  - Entertainment robots.
  - Space robots.
- Robotics life:
  - Robots were invented by the humans.
  - Assist humans in various sectors.
  - Good for dreary, recurring tasks.
  - Make living more convenient.

Figure 3-4. Classification of robots

AIR0110

### Notes:

#### Classification of robots

The robots are also classified based on their applications, which are listed below:

- **Industrial robots:** These robots are explored in an industrialized manufacturing atmosphere, containing articulated arms created for applications such as material handling, painting, welding and many more. These robots may also contain of some automatically guided automobiles and other robots.
- **Domestic or household robots:** Robots which are used at home consists of numerous different gears for example- robotic pool cleaners, robotic sweepers, robotic vacuum cleaners, robotic sewer cleaners and other robots that can perform different household tasks.
- **Medical robots:** Robots employed in medicine and medicinal institutes, usually referred as surgical treatment robots.
- **Service robots:** Robots that could be used for various data collection, equipped to exhibit technologies, robots employed for research, etc.
- **Military robots:** This sort of robots consist of bomb discarding robots, various shipping robots, exploration drones. Often robots at the start produced for military and armed forces purposes can be employed in law enforcement, exploration and salvage and other associated fields.

- **Entertainment robots:** These types of robots are employed for entertainment and are extremely wide-ranging category such as Robosapiens or the running photo frames and concludes with real heavy weights like articulated robot arms employed as movement simulators.
- **Space robots:** This type of robots would consist of the robots employed in space shuttles, the International Space Station, together with Mars explorers and other robots employed in space exploration and other activities.

Robotics is a broad field and everyday there is a pioneering invention in the field. "Robots were invented by the humans just for fun but by now they are used for assisting humans in various sectors. Human beings are better suitable for multifaceted, imaginative, adaptive jobs, and robots are good for dreary, recurring tasks, permitting human beings to do the harder thinking jobs, whereas a robot is employed for substituting humans for various recurring tasks or entertainment to make living more convenient".

## Components of robot (1 of 4)

- Articulated arm or RRR robot - robot having 3 revolute joints and it mimics the human arm.

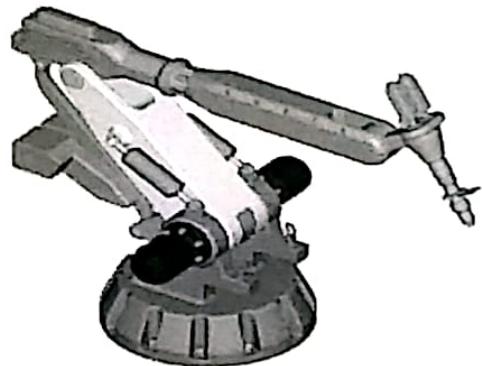
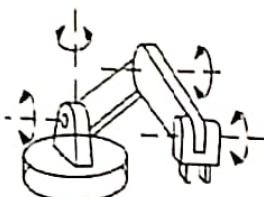
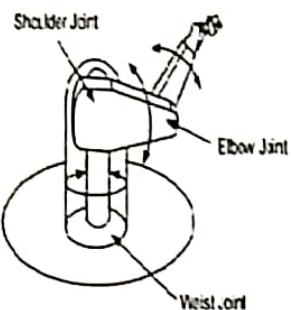


Figure: A schematic diagram of revolute robot

Figure: A revolute robot

Figure 3-5. Components of robot (1 of 4)

AIR011.0

### Notes:

The main components of a robot include:

- Vehicles
- Manipulator arms
- Wrists
- End effectors
- Actuators
- Transmission elements
- Sensors

The links are generally mechanically solid objects that connect two joints. For example, the elbow and the shoulder are joints that are linked by the upper arm. The joints give the name to the robot. Hence, if a robot has three prismatic joints, it is named as PPP. A robot arm usually has 3 moving parts/links and one immobile unit called as base. The first part simulate a human torso and is usually the part that is directly connected to the robot base by the waist joint/first joint. The second link, the upper arm, is connected to the torso by the shoulder joint/second joint. The third part is the forearm, which is connected to the upper arm link with elbow joint/third joint. A robot that has three revolute joints is called "articulated arm" or RRR robot and mimics the human arm.

## Components of robot (2 of 4)

IBM

IBM ICE (Innovation Centre for Education)

- Cartesian or PPP robot:
  - Simplest industrial manipulator.
  - Contains three prismatic joints.

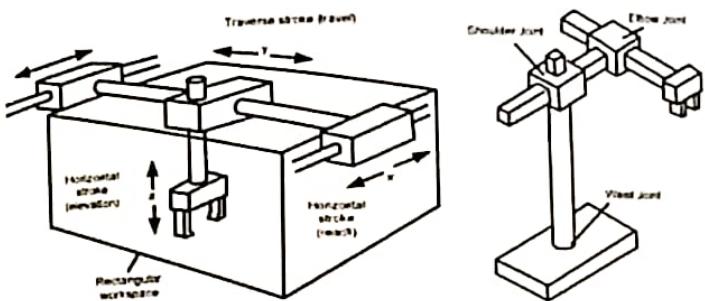


Figure: A schematic diagram of Cartesian (PPP) robot



Figure: A Cartesian robot

Figure 3-6. Components of robot (2 of 4)

AIR011.0

### Notes:

There are other types of simple robot geometries that substitute one or more revolute joints with prismatic ones. The simplest industrial manipulator is called as cartesian or PPP robot. It is very simple and contains three prismatic joints. It can reach any location/position in its rectangular-workspace by cartesian motions of the links.

## Components of robot (3 of 4)

- Cylindrical or PRP robot:
  - Replacing a joint of a Cartesian arm by a revolute joint.
  - A cylindrical geometry arm can be formed.
  - Has a cylindrical workspace.

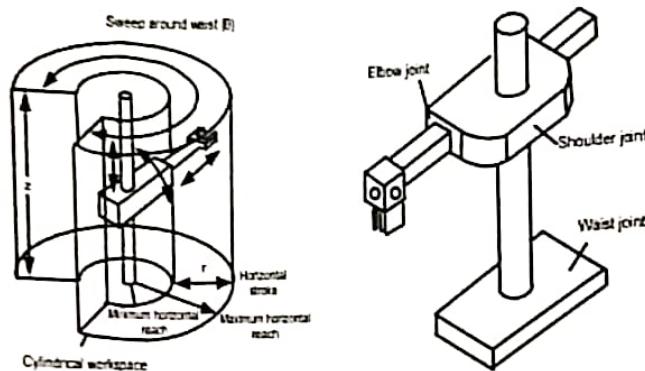


Figure: A schematic diagram of cylindrical robot

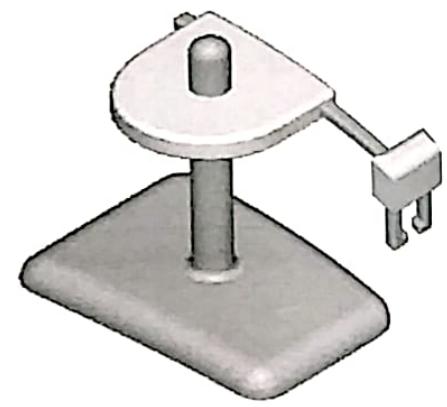


Figure: A cylindrical robot

Figure 3-7. Components of robot (3 of 4)

AIR0110

### Notes:

By replacing a joint of a cartesian arm by a revolute joint, a cylindrical geometry arm can be formed. This robot is known as cylindrical or PRP and it has a cylindrical workspace as shown in figures.

## Components of robot (4 of 4)



IBM ICE (Innovation Centre for Education)

- Polar or RRP robot:

- A revolute joint substitutes another prismatic joint then a polar geometry is formed.
- Has a spherical workspace.

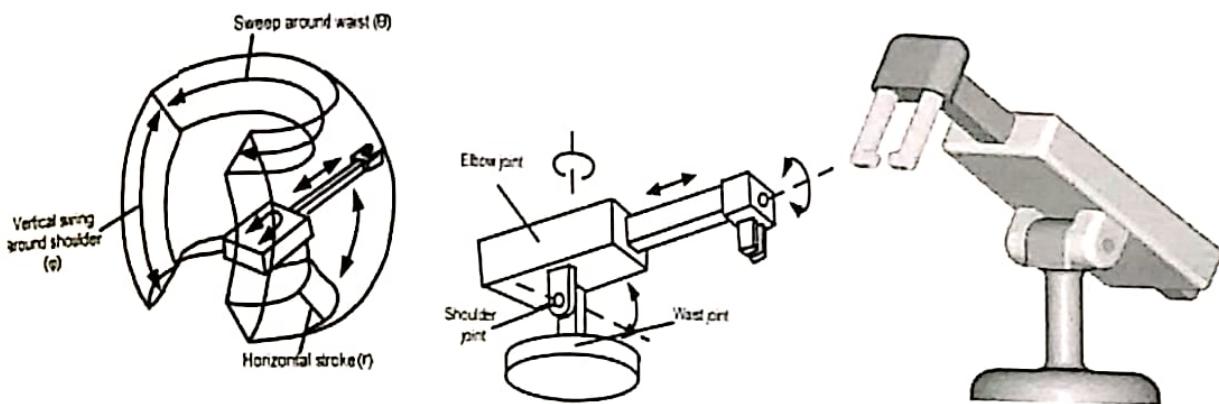


Figure: A schematic diagram of spherical robot

Figure: A spherical robot

Figure 3-8. Components of robot (4 of 4)

AIR011.0

### Notes:

If a revolute joint substitutes another prismatic joint, then a polar geometry is formed. This robot is known as Polar or RRP and has a spherical workspace as shown in figures.

## Manipulation arms

- Robot arm:
  - Consists of joints and links.
  - Joints connect different parts/links of the robot and they can slide/rotate.

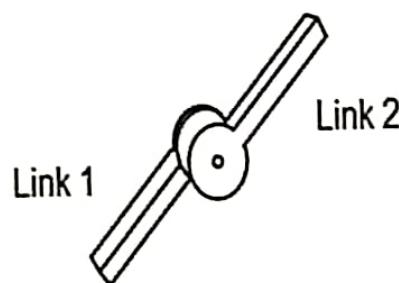


Figure: A revolute joint

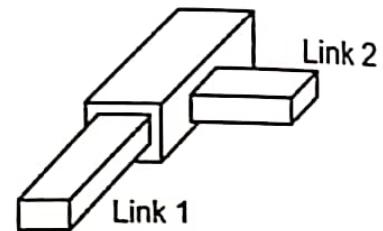


Figure: A prismatic joint

Figure 3-9. Manipulation arms

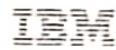
AIR011.0

### Notes:

#### Manipulation arms

A manipulator arm, also termed as robot arm, consists of joints and links. The joints connect different parts/links of the robot and they can slide/rotate. Hence they are called sliding/prismatic and revolute joints.

# Merits and demerits of robot types with different geometries



IBM ICE (Innovation Centre for Education)

Robot	Coordinates	Merits and Demerits
PPP or Cartesian	x	<ul style="list-style-type: none"> <li><input type="checkbox"/> Linear Motion in 3D Simple kinematics model</li> <li><input type="checkbox"/> Rigid structure</li> <li><input type="checkbox"/> Easy to visualize</li> <li><input type="checkbox"/> Can use inexpensive pneumatic drives for pick and place operation</li> <li><input type="checkbox"/> Requires large volume to operate in</li> <li><input type="checkbox"/> Workspace is smaller than robot volume</li> <li><input type="checkbox"/> Unable to reach areas under objects</li> <li><input type="checkbox"/> Guiding surfaces of prismatic joints must be covered to prevent ingress of dust</li> </ul>
	y	
	z	
PRP or Cylindrical	z	<ul style="list-style-type: none"> <li><input type="checkbox"/> Simple kinematic model</li> <li><input type="checkbox"/> Easy to visualize</li> <li><input type="checkbox"/> Good access into cavities and machine openings</li> <li><input type="checkbox"/> Very powerful when hydraulic drives are used</li> <li><input type="checkbox"/> Restricted workspace</li> <li><input type="checkbox"/> Prismatic guides difficult to seal from dust and liquids</li> <li><input type="checkbox"/> Back of robot can overlap work volume</li> </ul>
	$\theta$	
	r	
RRP or Spherical	$\theta$	<ul style="list-style-type: none"> <li><input type="checkbox"/> Covers a large area from a central support</li> <li><input type="checkbox"/> Can bend down to pick objects off the floor</li> <li><input type="checkbox"/> Complex kinematic model</li> <li><input type="checkbox"/> Difficult to visualize</li> </ul>
	$\phi$	
	r	
RRR or Articulated	$\theta_1$	<ul style="list-style-type: none"> <li><input type="checkbox"/> Maximum flexibility</li> <li><input type="checkbox"/> Covers large area of work relative to volume of robots</li> <li><input type="checkbox"/> Revolute joints are easy to seal</li> <li><input type="checkbox"/> Suits electrical motors</li> <li><input type="checkbox"/> Can reach over and under objects</li> <li><input type="checkbox"/> Complex kinematics</li> <li><input type="checkbox"/> Difficult to visualize</li> <li><input type="checkbox"/> Control of linear motions is difficult</li> <li><input type="checkbox"/> Structure not very rigid at full reach</li> </ul>
	$\theta_2$	
	$\theta_3$	

Figure 3-10. Merits and demerits of robot types with different geometries

AIR011.0

## Notes:

There are various robot types with different geometries, which can be used in different applications. Each model has its own merits and demerits, which are presented in table.

## Wrists

- A robot arm has a wrist to orient the end effectors.
- Wrist has three degrees of freedom:
  - roll (z axis rotation),
  - pitch (y axis rotation).
  - yaw (x axis rotation).

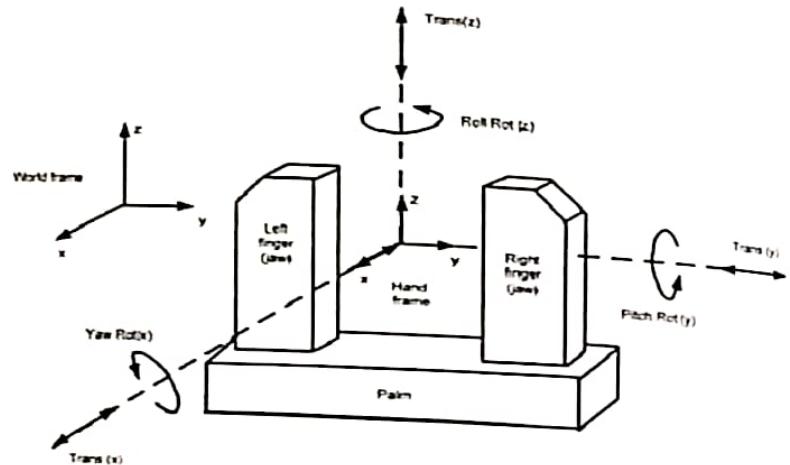


Figure: Wrists six motions for gripping

Figure 3-11. Wrists

AIR011.0

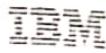
### Notes:

#### Wrists

A robot arm has three main joints namely waist, shoulder and elbow and three links namely torso, upper arm and forearm. The human body also has a wrist and end effectors/fingers. Similarly, a robot arm has a wrist to orient the end effectors. Usually the wrist has three degrees of freedom as shown in figure and the joints are revolute and not prismatic. The rotations of the wrists are commonly described as roll (z axis rotation), pitch (y axis rotation) and yaw (x axis rotation). There are some simplified wrists which may have two degrees of freedom (roll & pitch).

## Robot kinematics (1 of 3)

---



IBM ICE (Innovation Centre for Education)

- Robot kinematics refers the analytical study of the motion of a robot manipulator.
- Two different spaces used in kinematics modelling of manipulators are:
  - Cartesian space and quaternion space.
- The transformation between two Cartesian coordinate systems can be decomposed into a rotation and a translation.
- Some of the ways to represent rotation:
  - Euler angles.
  - Gibbs vector.
  - Pauli spin matrices.
  - Hamilton's quaternions.
- Homogenous transformations based on 4x4 real matrices (orthonormal matrices) have been used most often in robotics.

---

Figure 3-12. Robot kinematics (1 of 3)

AIR011.0

### Notes:

#### Robot kinematics

Kinematics studies the motion of bodies without consideration of the forces or moments that cause the motion. Robot kinematics refers the analytical study of the motion of a robot manipulator". Framing the appropriate kinematics models for a robot mechanism is very vital for analyzing the behavior of industrial manipulators. There are mainly two different spaces used in kinematics modelling of manipulators namely, Cartesian space and Quaternion space. The transformation between two Cartesian coordinate systems can be decomposed into a rotation and a translation.

There are many ways to represent rotation, including the following:

- Euler angles.
- Gibbs vector.
- Cayley-Klein parameters.
- Pauli spin matrices.
- Axis and angle.
- Orthonormal matrices.
- Hamilton's quaternions.

Of these representations, homogenous transformations based on 4x4 real matrices (orthonormal matrices) have been used most often in robotics. "It is proved that a general transformation between two joints requires four parameters. These parameters known as the Denavit-Hartenberg (DH) parameters have become the standard for describing robot kinematics".

Although quaternions constitute an elegant representation for rotation, they have not been used as much as homogenous transformations by the robotics community. Dual quaternion can present rotation and translation in a compact form of transformation vector, simultaneously. "While the orientation of a body is represented nine elements in homogenous transformations, the dual quaternions reduce the number of elements to four. It offers considerable advantage in terms of computational robustness and storage efficiency for dealing with the kinematics of robot chains.

## Robot kinematics (2 of 3)



IBM ICE (Innovation Centre for Education)

- It is proved that a general transformation between two joints requires four parameters. These parameters known as the Denavit-Hartenberg (DH) parameters have become the standard for describing robot kinematics.
- While the orientation of a body is represented nine elements in homogenous transformations, the dual quaternions reduce the number of elements to four. It offers considerable advantage in terms of computational robustness and storage efficiency for dealing with the kinematics of robot chains.
- The robot kinematics is studied as:
  - Forward kinematics and inverse kinematics.
- The forward kinematics problem is quite simple and there is no complexity in deriving the equations.
- Hence, there is always a forward kinematics solution of a manipulator.
- Inverse kinematics is a much more difficult problem than forward kinematics.
- The solution of the inverse kinematics problem is computationally expensive and generally consume more time in the real time control of manipulators.

Figure 3-13. Robot kinematics (2 of 3)

AIR011.0

### Notes:

The robot kinematics is studied as forward kinematics and inverse kinematics. The forward kinematics problem is quite simple and there is no complexity in deriving the equations. Hence, there is always a forward kinematics solution of a manipulator. On the other hand, Inverse kinematics is a much more difficult problem than forward kinematics. The solution of the inverse kinematics problem is computationally expensive and generally consume more time in the real time control of manipulators.

## Homogenous transformation modelling convention (1 of 2)

IBM

IBM ICE (Innovation Center for Education)

### Forward Kinematics:

- The coordinate frame assignment for a general manipulator.



- The general transformation matrix for a single link can be obtained as follows:

$$\begin{aligned} {}^i T &= R_z(\alpha_{i-1}) D_x(a_{i-1}) R_z(0) D_x(d_i) \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_{i-1} & -\sin\alpha_{i-1} & 0 \\ 0 & \sin\alpha_{i-1} & \cos\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 0 & 0 & 0 & 0 \\ 0 & \cos 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_i \\ \sin\theta_i \cos\alpha_{i-1} & \cos\theta_i \cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1} d_i \\ \sin\theta_i \sin\alpha_{i-1} & \cos\theta_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Figure 3-18: Homogenous transformation modelling convention (1 of 2)

AIR011.0

### Notes:

#### Forward Kinematics

"A manipulator is composed of serial links which are affixed to each other revolute or prismatic joints from the base frame through the end-effector". Computing the position and orientation of the end-effector in terms of the joint variables is called as forward kinematics. In order to have forward kinematics for a robot mechanism in a systematic manner, one should use a suitable kinematics model. "Denavit-Hartenberg method that uses four parameters is the most common method for describing the robot kinematics". These parameters are  $a_{i-1}$ ,  $\alpha_{i-1}$ ,  $d_i$ , and  $\theta_i$  and are respectively represent the link length, link twist, link offset and joint angle. A coordinate frame is attached to each joint to determine DH parameters, the Z<sub>i</sub> axis of the coordinate frame is pointing along the rotary or sliding direction of the joints.

As shown in figure:

- The distance from Z<sub>i-1</sub> to Z<sub>i</sub> measured along X<sub>i-1</sub> is assigned as a<sub>i-1</sub>.
- The angle between Z<sub>i-1</sub> and Z<sub>i</sub> measured along X<sub>i</sub> is assigned as  $\alpha_{i-1}$ .
- The distance from X<sub>i-1</sub> to X<sub>i</sub> measured along Z<sub>i</sub> is assigned as d<sub>i</sub>.
- The angle between X<sub>i-1</sub> to X<sub>i</sub> measured about Z<sub>i</sub> is assigned as  $\theta_i$ .

The general transformation matrix for a single link (forward kinematics) is presented here.

Where : Rx and Rz - rotation, Dx and Qi - translation, cθ and sθ - short hands of cosθ and sinθ

## Robot kinematics (3 of 3)

- The relationship between forward and inverse kinematics:

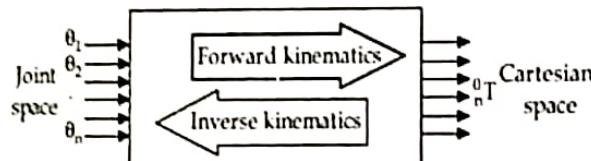


Figure: The schematic representation of forward and inverse kinematics.

- The two major solutions for the inverse kinematics problem are:
  - The *analytical* and *numerical* methods
- In analytical: The joint variables are solved analytically according to the given configuration-data.
- In numerical: The joint variables are obtained founded on the numerical techniques.
- Two solutions in analytical method:
  - Geometric and algebraic solutions.
- Geometric approach is applied to the simple robot structures (2-DOF planar manipulator or less DOF manipulator with parallel joint axes)
- Algebraic approach is explored for the inverse kinematics solution:
  - In the cases, For the manipulators with more links and whose arms extend into 3 dimensions or more.

Figure 3-14. Robot kinematics (3 of 3)

AIR0110

### Notes:

The analytical and numerical methods are the two major solutions for the inverse kinematics problem. In the first case, the joint variables are solved analytically according to the given configuration-data. In the second case, the joint variables are obtained founded on the numerical techniques. There are two solutions in analytical method: geometric and algebraic solutions. Geometric approach is applied to the simple robot structures, such as 2-DOF planar manipulator or less DOF manipulator with parallel joint axes. For the manipulators with more links and whose arms extend into 3 dimensions or more, the geometry gets much more tedious. In this case, algebraic approach is explored for the inverse kinematics solution.

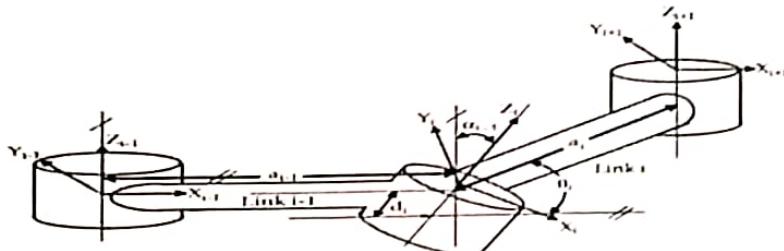
# Homogenous transformation modelling convention (1 of 2)

IBM

IBM ICE (Innovation Centre for Education)

## Forward Kinematics:

- The coordinate frame assignment for a general manipulator:



- The general transformation matrix for a single link can be obtained as follows:

$$\begin{aligned} {}^{i-1}T &= R_z(\alpha_{i-1})D_z(a_{i-1})R_x(\theta_i)Q_i(d_i) \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_{i-1} & -s\alpha_{i-1} & 0 \\ 0 & s\alpha_{i-1} & c\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Figure 3-15. Homogenous transformation modelling convention (1 of 2)

AIR011.0

## Notes:

### Forward Kinematics

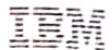
"A manipulator is composed of serial links which are affixed to each other revolute or prismatic joints from the base frame through the end-effector". Computing the position and orientation of the end-effector in terms of the joint variables is called as forward kinematics. In order to have forward kinematics for a robot mechanism in a systematic manner, one should use a suitable kinematics model. "Denavit-Hartenberg method that uses four parameters is the most common method for describing the robot kinematics". These parameters are  $a_{i-1}$ ,  $\alpha_{i-1}$ ,  $d_i$ , and  $\theta_i$ , and are respectively represent the link length, link twist, link offset and joint angle. A coordinate frame is attached to each joint to determine DH parameters, the  $Z_i$  axis of the coordinate frame is pointing along the rotary or sliding direction of the joints.

As shown in figure:

- The distance from  $Z_{i-1}$  to  $Z_i$  measured along  $X_{i-1}$  is assigned as  $a_{i-1}$ ,
- The angle between  $Z_{i-1}$  and  $Z_i$  measured along  $X_i$  is assigned as  $\alpha_{i-1}$ ,
- The distance from  $X_{i-1}$  to  $X_i$  measured along  $Z_i$  is assigned as  $d_i$ ,
- The angle between  $X_{i-1}$  to  $X_i$  measured about  $Z_i$  is assigned as  $\theta_i$ .

The general transformation matrix for a single link (forward kinematics) is presented here.

Where : Rx and Rz - rotation, Dx and Qi -translation,  $c\theta_i$  and  $s\theta_i$  - short hands of  $\cos\theta_i$  and  $\sin\theta_i$ ,



## Homogenous transformation modelling convention (2 of 2)

- The forward kinematics of the end-effector with respect to the base frame is determined by multiplying all of the matrices.

$$\text{end\_effector}^{\text{base}} T = {}^0_1 T {}^1_2 T \dots {}^{n-1}_n T$$

- An alternative representation of  $\text{end\_effector}^{\text{base}} T$  can be written as:

$$\text{end\_effector}^{\text{base}} T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Where
- $r_{kj}$ 's - rotational elements of transformation matrix (k and j=1, 2 and 3).
- $p_x$ ,  $p_y$  and  $p_z$  - elements of the position vector.

- For a six jointed manipulator, the position and orientation of the end-effector with respect to the base is given by:

$${}^0_6 T = {}^0_1 T(q_1) {}^1_2 T(q_2) {}^2_3 T(q_3) {}^3_4 T(q_4) {}^4_5 T(q_5) {}^5_6 T(q_6)$$

where  $q_i$  is the joint variable (revolute or prismatic joint) for joint i, (i=1, 2, ...6).

Figure 3-16. Homogenous transformation modelling convention (2 of 2)

AIR0110

### Notes:

The general transformation matrix for a single link (forward kinematics) is presented here.

## Example of forward kinematics (1 of 4)

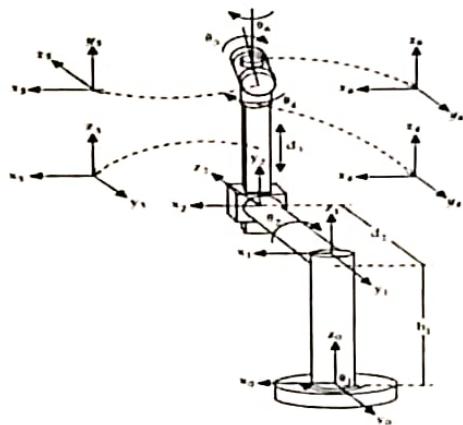


Figure: Rigid body and coordinate frame assignment for the Stanford Manipulator.

i	$q_i$	$a_{i-1}$	$a_{i-1}$	$d_i$
1	$q_1$	0	0	$h_1$
2	$q_2$	90	0	$d_2$
3	0	-90	0	$d_3$
4	$q_4$	0	0	0
5	$q_5$	90	0	0
6	$q_6$	-90	0	0

Table: DH parameters for the Stanford Manipulator.

Figure 3-17. Example of forward kinematics (1 of 4)

AIR011.0

### Notes:

Consider a 6-DOF manipulator (Stanford Manipulator) whose rigid body and coordinate frame assignment are illustrated in Figure. Note that the manipulator has an Euler wrist whose three axes intersect at a common point. The first (RRP) and last three (RRR) joints are spherical in shape. P and R denote prismatic and revolute joints, respectively. The DH parameters corresponding to this manipulator are shown in Table. It is straightforward to compute each of the link transformation matrices using equation 4.1.



## Example of forward kinematics (2 of 4)

IBM ICE (Innovation Centre for Education)

$${}^6T = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & h_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots (4.3)$$

$${}^4T = \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 & 0 \\ s\theta_4 & c\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots (4.4)$$

$${}^3T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots (4.5)$$

$${}^3T = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & 0 \\ 0 & 0 & -1 & -d_3 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots (4.6)$$

$${}^2T = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots (4.7)$$

$${}^2T = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots (4.8)$$

Figure 3-18 Example of forward kinematics (2 of 4)

AIR011.0

### Notes:

The forward kinematics of the Stanford Manipulator can be determined in the form of equation 4.2 multiplying all of the matrices, where  $i=1, 2, \dots, 6$ .

## Example of forward kinematics (3 of 4)

$$\overset{^n}{^e}\mathbf{T} = \begin{bmatrix} \mathbf{r}_{11} & \mathbf{r}_{12} & \mathbf{r}_{13} & \mathbf{p}_x \\ \mathbf{r}_{21} & \mathbf{r}_{22} & \mathbf{r}_{23} & \mathbf{p}_y \\ \mathbf{r}_{31} & \mathbf{r}_{32} & \mathbf{r}_{33} & \mathbf{p}_z \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 \end{bmatrix} \quad \dots(4.9)$$

where

$$\begin{aligned}
 \mathbf{r}_{11} &= -s\theta_6(c\theta_4s\theta_1 + c\theta_1c\theta_2s\theta_4) - c\theta_6(c\theta_1(s\theta_1s\theta_4 - c\theta_1c\theta_2c\theta_4) + c\theta_1s\theta_2s\theta_4) \\
 \mathbf{r}_{12} &= s\theta_6(c\theta_1(s\theta_1s\theta_4 - c\theta_1c\theta_2c\theta_4) + c\theta_1s\theta_2s\theta_4) - c\theta_6(c\theta_4s\theta_1 + c\theta_1c\theta_2s\theta_4) \\
 \mathbf{r}_{13} &= s\theta_6(s\theta_1s\theta_4 - c\theta_1c\theta_2c\theta_4) - c\theta_1c\theta_2s\theta_4 \\
 \mathbf{r}_{21} &= s\theta_6(c\theta_1c\theta_4 - c\theta_1s\theta_1s\theta_4) + c\theta_6(c\theta_1(c\theta_1s\theta_4 + c\theta_2c\theta_4s\theta_1) - s\theta_1s\theta_2s\theta_4) \\
 \mathbf{r}_{22} &= c\theta_6(c\theta_1c\theta_4 - c\theta_1s\theta_1s\theta_4) - s\theta_6(c\theta_1(c\theta_1s\theta_4 + c\theta_2c\theta_4s\theta_1) - s\theta_1s\theta_2s\theta_4) \\
 \mathbf{r}_{23} &= -s\theta_6(c\theta_1s\theta_4 + c\theta_2c\theta_4s\theta_1) - c\theta_6s\theta_1s\theta_2 \\
 \mathbf{r}_{31} &= c\theta_6(c\theta_2s\theta_4 + c\theta_4c\theta_1s\theta_2) - s\theta_6s\theta_1s\theta_4 \\
 \mathbf{r}_{32} &= -s\theta_6(c\theta_2s\theta_4 + c\theta_4c\theta_1s\theta_2) - c\theta_6s\theta_2s\theta_4 \\
 \mathbf{r}_{33} &= c\theta_2c\theta_4 - c\theta_4s\theta_2s\theta_4 \\
 \mathbf{p}_x &= d_2s\theta_1 - d_1c\theta_1s\theta_2 \\
 \mathbf{p}_y &= -d_2c\theta_1 - d_1s\theta_1s\theta_2 \\
 \mathbf{p}_z &= h_1 + d_1c\theta_2
 \end{aligned}$$

Figure 3-19. Example of forward kinematics (3 of 4)

AIR011.0

### Notes:

In order to check the accuracy of the mathematical model of the Stanford manipulator, the following steps should be taken. The general position vector in equation 4.9 should be compared with the zero-position vector in figure.

## Example of forward kinematics (4 of 4)

IBM ICE (Innovation Centre for Education)

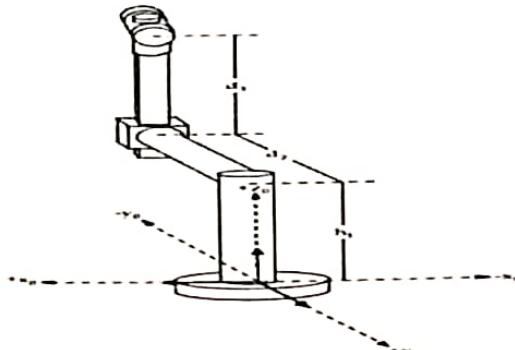


Figure: Zero position for the Stanford Manipulator.

- In order to obtain the zero position in terms of link parameters, set  $q_1=q_2=0^\circ$  in equation 4.10.

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} d_1 s(\theta_1) - d_2 c(\theta_1) s(\theta_2) \\ -d_1 c(\theta_1) - d_2 s(\theta_1) s(\theta_2) \\ h_1 + d_2 c(\theta_2) \end{bmatrix} \quad \dots(4.10)$$

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} d_1 s(0^\circ) - d_2 c(0^\circ) s(0^\circ) \\ -d_1 c(0^\circ) - d_2 s(0^\circ) s(0^\circ) \\ h_1 + d_2 c(0^\circ) \end{bmatrix} = \begin{bmatrix} 0 \\ -d_2 \\ h_1 + d_2 \end{bmatrix} \quad \dots(4.11)$$

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} 0 \\ -d_2 \\ h_1 + d_2 \end{bmatrix} \quad \dots(4.12)$$

Figure 3-20. Example of forward kinematics (4 of 4)

AIR011.0

### Notes:

#### Verification of mathematical model

The general position vector of the Stanford manipulator is given by:

- In order to obtain the zero position in terms of link parameters, set  $q_1=q_2=0^\circ$  in equation 4.10.
- All of the coordinate frames in figure are removed except the base which is the reference coordinate frame for determining the link parameters in zero position.
- Since there is not any link parameters observed in the direction of  $+x_0$  and  $-x_0$ ,  $p_x=0$ .
- There is only  $d_2$  parameter in  $-y_0$  direction, so  $p_y$  equals  $-d_2$ .
- The parameters  $h_1$  and  $d_3$  are the  $+z_0$  direction, so  $p_z$  equals  $h_1+d_3$ .
- It is explained above that the results of the position vector in equation 4.11 are identical to those obtained by equation 4.12.
- Hence, it can be said that the mathematical model of the stanford manipulator is driven correctly.

## Inverse kinematics (1 of 6)

IBM

IBM ICE (Innovation Centre for Education)

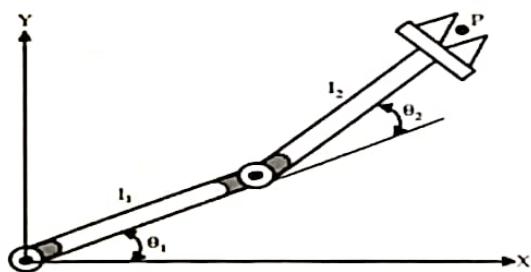


Figure (a): Planner manipulator

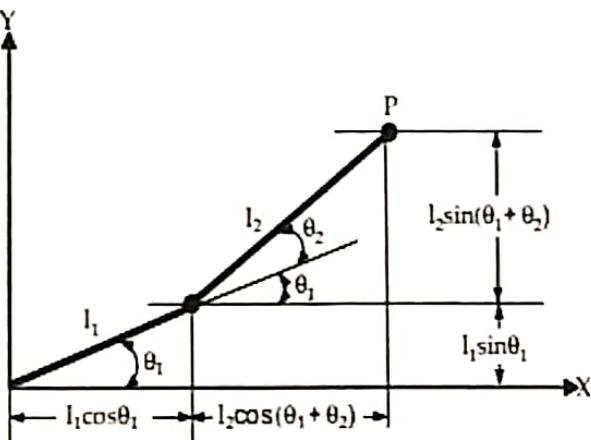


Figure (b): Solving the inverse kinematics based on trigonometry

Figure 3-21. Inverse kinematics (1 of 6)

AIR011.0

### Notes:

Inverse kinematics is the mathematical process of recovering the movements of an object in the world from some other data, such as a film of those movements, or a film of the world as seen by a camera which is itself making those movements. This is useful in robotics and in film animation. Solving the inverse kinematics is computationally expensive and generally takes a very longtime in the real time control of manipulators. The tasks are to be performed by a manipulator in the Cartesian space. Actuators work in joint space. Joint space is represented by joint angles. Cartesian space includes orientation matrix and position vector. The conversion of the position and orientation of a manipulator end-effector from Cartesian space to joint space is called as inverse kinematics problem. Two solutions used for deriving the inverse kinematics solution: Geometric and algebraic.

**Geometric solution approach:** Based on decomposing the spatial geometry of the manipulator into several plane geometry problems. It is applied to the simple robot structures, such as, 2 degrees-of-freedom planer manipulator whose joints are both revolute and link lengths are  $l_1$  and  $l_2$  shown in figure (a). Consider Figure (b) in order to derive the kinematics equations for the planer manipulator.



## Inverse kinematics (2 of 6)

IBM ICE (Innovation Centre for Education)

- The components of the point P ( $p_x$  and  $p_y$ ) are determined as follows:

$$p_x = l_1 c\theta_1 + l_2 c\theta_{12} \quad \dots(4.13)$$

$$p_y = l_1 s\theta_1 + l_2 s\theta_{12} \quad \dots(4.14)$$

where  $c\theta_{12} = c\theta_1 c\theta_2 - s\theta_1 s\theta_2$  and  $s\theta_{12} = s\theta_1 c\theta_2 + c\theta_1 s\theta_2$

$$p_x^2 = l_1^2 c^2 \theta_1 + l_2^2 c^2 \theta_{12} + 2l_1 l_2 c\theta_1 c\theta_{12} \quad \dots(4.15)$$

$$p_y^2 = l_1^2 s^2 \theta_1 + l_2^2 s^2 \theta_{12} + 2l_1 l_2 s\theta_1 s\theta_{12}$$

$$p_x^2 + p_y^2 = l_1^2 (c^2 \theta_1 + s^2 \theta_1) + l_2^2 (c^2 \theta_{12} + s^2 \theta_{12}) + 2l_1 l_2 (c\theta_1 c\theta_{12} + s\theta_1 s\theta_{12})$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2 (c\theta_1 [c\theta_2 - s\theta_1 s\theta_2] + s\theta_1 [s\theta_2 + c\theta_1 s\theta_2])$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2 (c^2 \theta_1 c\theta_2 - c\theta_1 s\theta_1 s\theta_2 + s^2 \theta_1 c\theta_2 + c\theta_1 s\theta_1 s\theta_2) \quad \dots(4.16)$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2 (c\theta_2 [c^2 \theta_1 + s^2 \theta_1])$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2 c\theta_2$$

Figure 3-22. Inverse kinematics (2 of 6)

AIR011.0

### Notes:

The solutions of  $\theta_2$  can be computed from summation of squaring equations (4.13) and (4.14). Since  $c^2 \theta_1 + s^2 \theta_1 = 1$ , the equation given above is simplified in equation (4.16).

## Inverse kinematics (3 of 6)



IBM ICE (Innovation Centre for Education)

- And hence,

$$c\theta_2 = \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1l_2} \quad \dots(4.17)$$

Since,  $c^2\theta_i + s^2\theta_i = 1$  ( $i = 1, 2, 3, \dots$ ),  $s\theta_2$  is obtained as

$$s\theta_2 = \pm \sqrt{1 - \left( \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1l_2} \right)^2}$$

$$\theta_2 = A \tan 2 \left( \pm \sqrt{1 - \left( \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1l_2} \right)^2}, \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1l_2} \right) \quad \dots(4.18)$$

$$c\theta_1 p_x = l_1 c^2 \theta_1 + l_2 c^2 \theta_1 c\theta_2 - l_2 c\theta_1 s\theta_1 s\theta_2$$

$$s\theta_1 p_y = l_1 s^2 \theta_1 + l_2 s^2 \theta_1 c\theta_2 + l_2 s\theta_1 c\theta_1 s\theta_2$$

$$c\theta_1 p_x + s\theta_1 p_y = l_1 (c^2 \theta_1 + s^2 \theta_1) + l_2 c\theta_2 (c^2 \theta_1 + s^2 \theta_1)$$

$$c\theta_1 p_x + s\theta_1 p_y = l_1 + l_2 c\theta_2 \quad \dots(4.20)$$

Figure 3-23. Inverse kinematics (3 of 6)

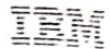
AIR011.0

### Notes:

Finally, two possible solutions for  $\theta_2$  can be written as in equation (4.18):

- Multiply each side of equation 4.13 by  $c\theta_1$  and
- Equation 4.14 by  $s\theta_1$ , and
- Add the resulting equations.

In order to find the solution of  $\theta_2$  in terms of link parameters and the known variable  $\theta_1$ , shown in equation (4.19). The simplified equation obtained as shown in equation (4.20).



IBM ICE (Innovation Centre for Education)

## Inverse kinematics (4 of 6)

- In this step, multiply both sides of equation 4.13 by  $-s\theta_1$ , and equation 4.14 by  $c\theta_1$ , and then adding the resulting equations produce.

$$\begin{aligned} -s\theta_1 p_x &= -l_1 s\theta_1 c\theta_1 - l_2 s\theta_1 c\theta_1 c\theta_2 + l_2 s^2 \theta_1 s\theta_2 \\ c\theta_1 p_y &= l_1 s\theta_1 c\theta_1 + l_2 c\theta_1 s\theta_1 c\theta_2 + l_2 c^2 \theta_1 s\theta_2 \\ -s\theta_1 p_x + c\theta_1 p_y &= l_2 s\theta_2 (c^2 \theta_1 + s^2 \theta_1) \\ -s\theta_1 p_x + c\theta_1 p_y &= l_2 s\theta_2 \end{aligned} \quad \dots(4.21)$$

$$\begin{aligned} c\theta_1 p_z^2 + s\theta_1 p_x p_y &= p_z (l_1 + l_2 c\theta_2) \\ -s\theta_1 p_x p_z + c\theta_1 p_y^2 &= p_z l_1 s\theta_1 \\ c\theta_1 (p_z^2 + p_y^2) &= p_z (l_1 + l_2 c\theta_2) + p_z l_2 s\theta_2 \end{aligned} \quad \dots(4.22)$$

$$s\theta_1 = \pm \sqrt{1 - \left( \frac{p_z (l_1 + l_2 c\theta_2) + p_z l_2 s\theta_2}{p_z^2 + p_y^2} \right)^2} \quad \dots(4.23)$$

$$\theta_1 = \text{Atan2} \left( \pm \sqrt{1 - \left( \frac{p_z (l_1 + l_2 c\theta_2) + p_z l_2 s\theta_2}{p_z^2 + p_y^2} \right)^2}, \frac{p_z (l_1 + l_2 c\theta_2) + p_z l_2 s\theta_2}{p_z^2 + p_y^2} \right) \quad \dots(4.24)$$

Figure 3-24. Inverse kinematics (4 of 6)

AIR011.0

### Notes:

Now, multiply each side of equation 4.20 by  $p_x$ , and equation 4.21 by  $p_y$ , and add the resulting equations in order to obtain  $c\theta_1$ .

- $s\theta_1$  is obtained as shown in equation (4.23)
- As a result, two possible solutions for  $\theta_1$  can be written as equation (4.24)

Although the planar manipulator has a very simple structure, as can be seen, its inverse kinematics solution based on geometric approach is very cumbersome.

## Inverse kinematics (5 of 6)

### Algebraic solution approach:

- Consider the equation 4.9 to find the inverse kinematics solution for a six-axis manipulator.

$${}^0T = \begin{bmatrix} {}^0r_1 & {}^0r_2 & {}^0r_3 & {}^0p_s \\ {}^1r_1 & {}^1r_2 & {}^1r_3 & {}^1p_s \\ {}^2r_1 & {}^2r_2 & {}^2r_3 & {}^2p_s \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^0T(q_1) {}^1T(q_2) {}^2T(q_3) {}^3T(q_4) {}^4T(q_5) {}^5T(q_6)$$

- To find inverse kinematics solution for first joint ( $q_1$ ) as a function of the known elements of ~~and different~~<sup>base</sup>  $T$

- The link transformation inverses are pre-multiplied as follows:

$$[{}^0T(q_1)]^{-1} {}^0T = [{}^0T(q_1)]^{-1} {}^0T(q_1) {}^1T(q_2) {}^2T(q_3) {}^3T(q_4) {}^4T(q_5) {}^5T(q_6)$$

$$[{}^0T(q_1)]^{-1} {}^0T(q_1) = I \quad \text{where } I \text{ is the identity matrix.}$$

- In this case the above equation is given by:

$$[{}^0T(q_1)]^{-1} {}^0T = {}^1T(q_2) {}^2T(q_3) {}^3T(q_4) {}^4T(q_5) {}^5T(q_6) \dots (4.25)$$

Figure 3-25. Inverse Kinematics (5 of 6)

AIR011.0

### Notes:

#### Algebraic Solution Approach

The geometry-based solution is more cumbersome if there are manipulators with more links and whose arm extends into 3 dimensions. Hence, algebraic approach is chosen for the inverse kinematics solution.



## Inverse kinematics (6 of 6)

- To find the other variables,

$$\left[ {}^0T(q_1) {}^1T(q_2) \right]^{-1} {}^0T = {}^3T(q_3) {}^4T(q_4) {}^5T(q_5) {}^6T(q_6) \quad \dots(4.26)$$

$$\left[ {}^0T(q_1) {}^1T(q_2) {}^3T(q_3) \right]^{-1} {}^0T = {}^4T(q_4) {}^5T(q_5) {}^6T(q_6) \quad \dots(4.27)$$

$$\left[ {}^0T(q_1) {}^1T(q_2) {}^3T(q_3) {}^4T(q_4) \right]^{-1} {}^0T = {}^5T(q_5) {}^6T(q_6) \quad \dots(4.28)$$

$$\left[ {}^0T(q_1) {}^1T(q_2) {}^3T(q_3) {}^4T(q_4) {}^5T(q_5) \right]^{-1} {}^0T = {}^6T(q_6) \quad \dots(4.29)$$

- Table for some trigonometric equations and solutions used in inverse kinematics.

Equations	Solutions
1 $a \sin \theta + b \cos \theta = c$	$\theta = A \tan 2(a, b) \mp A \tan 2(\sqrt{a^2 + b^2 - c^2}, c)$
2 $a \sin \theta + b \cos \theta = 0$	$\theta = A \tan 2(-b, a)$ or $\theta = A \tan 2(b, -a)$
3 $\cos \theta = a$ and $\sin \theta = b$	$\theta = A \tan 2(b, a)$
4 $\cos \theta = a$	$\theta = A \tan 2(\mp \sqrt{1-a^2}, a)$
5 $\sin \theta = a$	$\theta = A \tan 2(a, \mp \sqrt{1-a^2})$

Figure 3-26. Inverse kinematics (6 of 6)

AIR011.0

### Notes:

As we see, we have to solve twelve simultaneous set of nonlinear equations:

- The only unknown on the left-hand side of equation 4.17 is  $q_1$ .
- The twelve nonlinear matrix elements of right-hand side are either zero, constant or functions of  $q_2$  through  $q_6$ .
- If the elements on the left-hand side which are the function of  $q_i$  are equated with the elements on the right-hand side, then the joint variable  $q_i$  can be solved as functions of  $r_{11}, r_{12}, \dots, r_{33}, p_x, p_y, p_z$  and the fixed link parameters.
- Once  $q_1$  is found, then the other joint variables are solved by the same way as before.
- It is not necessary that the 1<sup>st</sup> equation will produce  $q_1$  and the 2<sup>nd</sup> produces  $q_2$  etc.
- To find suitable equation for the solution of the inverse kinematics problem, any equation defined above (equations 4.25-4.29) can be used arbitrarily.
- Some trigonometric equations used in the solution of inverse kinematics problem are given in Table.

## Algebraic solution approach: Example (1 of 6)

IBM

IBM ICE (Innovation Centre for Education)

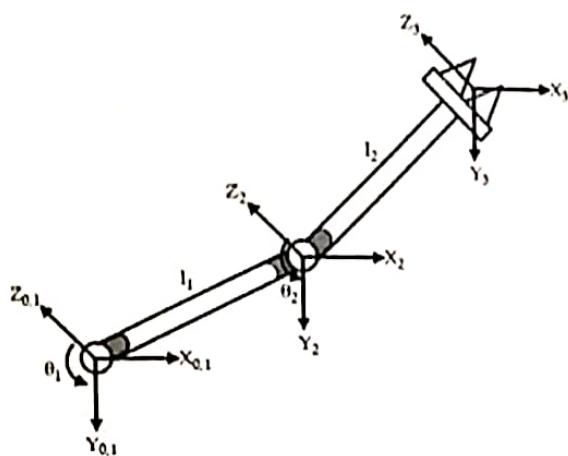


Figure: Coordinate frame assignment for the planar manipulator.

i	$\theta_i$	$a_{i-1}$	$d_{i-1}$
1	$\theta_1$	0	0
2	$\theta_2$	0	$l_1$
3	0	0	$l_2$

Table: DH parameters for the planar manipulator.

Figure 3-27. Algebraic solution approach: Example (1 of 6)

AIR011.0

### Notes:

As an example to describe the algebraic solution approach, let us consider the inverse kinematics for the planar manipulator. The coordinate frame assignment is shown in figure and DH parameters are provided in table.

## Algebraic solution approach: Example (2 of 6)



IBM ICE (Innovation Centre for Education)

- The link transformation matrices are given by:

$${}^0T = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^1T = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & l_1 \\ s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^2T = \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^3T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^0T {}^1T {}^2T$$

...(4.30)                    ... (4.31)                    ... (4.32)                    ... (4.33)

- Multiply each side of equation 4.33 by  ${}^0T^{-1}$ .

$${}^0T^{-1} {}^0T = {}^1T {}^2T \quad \dots (4.34) \qquad {}^0T^{-1} = \begin{bmatrix} {}^0R^T & -{}^0R^T {}^0P_1 \\ 0 & 1 \end{bmatrix} \quad \dots (4.33)$$

- In equation 4.35, and denote the transpose of rotation and position vector of respectively.
- Since,  ${}^0T^{-1} = I$ , equation 4.34 can be rewritten as follows:

$${}^0T^{-1} {}^0T = {}^1T {}^2T \quad \dots (4.36)$$

Figure 3-28. Algebraic solution approach: Example (2 of 6)

AIR011.0

### Notes:

Let us use the equation 4.9 to solve the inverse kinematics of the 2-DOF manipulator(4.33).

## Algebraic solution approach: Example (3 of 6)

IBM

IBM ICE (Innovation Centre for Education)

- Substituting the link transformation matrices into equation yields:

$$\begin{bmatrix} c\theta_1 & s\theta_1 & 0 & 0 \\ -s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & l_1 \\ s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \dots(4.37)$$

$$\begin{bmatrix} c\theta_1 p_x + s\theta_1 p_y \\ -s\theta_1 p_x + c\theta_1 p_y \\ p_z \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} l_2 c\theta_2 + l_1 \\ l_2 s\theta_2 \\ 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Squaring the (1, 4) and (2, 4) matrix elements of each side in equation 4.37.

$$c^2\theta_1 p_x^2 + s^2\theta_1 p_y^2 + 2p_x p_y c\theta_1 s\theta_1 = l_2^2 c^2\theta_2 + 2l_1 l_2 c\theta_2 + l_1^2$$

$$s^2\theta_1 p_x^2 + c^2\theta_1 p_y^2 - 2p_x p_y c\theta_1 s\theta_1 = l_2^2 s^2\theta_2$$

- And then adding the resulting equations above gives:

$$p_x^2(c^2\theta_1 + s^2\theta_1) + p_y^2(s^2\theta_1 + c^2\theta_1) = l_2^2(c^2\theta_2 + s^2\theta_2) + 2l_1 l_2 c\theta_2 + l_1^2$$

$$p_x^2 + p_y^2 = l_2^2 + 2l_1 l_2 c\theta_2 + l_1^2$$

$$c\theta_2 = \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1 l_2}$$

Figure 3-29. Algebraic solution approach: Example (3 of 6)

AIR011.0

### Notes:

Figure shows the link transformation matrices into equation yields.

## Algebraic solution approach: Example (4 of 6)



IBM ICE (Innovation Centre for Education)

- Finally, two possible solutions for  $\theta_2$  are computed as follows using the 4<sup>th</sup> trigonometric equation given in Table:

$$\theta_2 = A \tan 2 \left( \mp \sqrt{1 - \left[ \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1l_2} \right]^2}, \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1l_2} \right) \quad \dots(4.38)$$

$$c\theta_1 p_x + s\theta_1 p_y = l_2 c\theta_2 + l_1 \quad \dots(4.39)$$

- Using the 1<sup>st</sup> trigonometric equation in Table 4.2 produces two potential solutions:

$$\theta_1 = A \tan 2(p_y, p_x) \mp A \tan 2(\sqrt{p_y^2 + p_x^2 - (l_2 c\theta_2 + l_1)^2}, l_2 c\theta_2 + l_1) \quad \dots(4.40)$$

Figure 3-30. Algebraic solution approach: Example (4 of 6)

AIR011.0

### Notes:

Now the second joint variable  $\theta_2$  is known. The first joint variable  $\theta_1$  can be determined by equating the (1,4) elements of each side in equation 4.37.

## Algebraic solution approach: Example (5 of 6)



IBM ICE (Innovation Centre for Education)

- The inboard joint variables (first three joints) can be solved using the position vectors:

$$\begin{bmatrix} {}^0T_1 & {}^1T_2 \\ {}^1T_2 & {}^2T_3 \\ {}^2T_3 & {}^3T_4 \\ {}^3T_4 & {}^4T_5 \\ {}^4T_5 & {}^5T_6 \end{bmatrix}^{-1} \begin{bmatrix} {}^0T \\ {}^1T \\ {}^2T \\ {}^3T \\ {}^4T \\ {}^5T \\ {}^6T \end{bmatrix} = \dots \quad (4.41)$$

$$\begin{bmatrix} \cdot & \cdot & \cdot & c\theta_2(c\theta_1 p_x + s\theta_1 p_y) + s\theta_2(p_z - h_1) \\ \cdot & \cdot & \cdot & -s\theta_2(c\theta_1 p_x + s\theta_1 p_y) + c\theta_2(p_z - h_1) \\ \cdot & \cdot & \cdot & s\theta_1 p_x - c\theta_1 p_y - d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & d_3 \\ \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Using the 1<sup>st</sup>& 2<sup>nd</sup> trigonometric equations in Table 4.2, respectively:

$$\theta_1 = A \tan 2(p_x - p_y) \pm A \tan 2(\sqrt{p_x^2 + p_y^2 - d_2^2}, d_2) \quad \dots (4.42)$$

$$\theta_2 = \pm A \tan 2(c\theta_1 p_x + s\theta_1 p_y - p_z + h_1) \quad \dots (4.43)$$

- The prismatic joint variable  $d_3$  is extracted from the (2, 4) elements of each side in equation 4.41 as follows:

$$d_3 = -s\theta_2(c\theta_1 p_x + s\theta_1 p_y) + c\theta_2(p_z - h_1) \quad \dots (4.44)$$

Figure 3-31. Algebraic solution approach:Example (5 of 6)

AIR011.0

### Notes:

#### Algebraic solution approach

We have six-axis Stanford Manipulator. The link transformation matrices were previously developed. The equation 4.26 can be employed in order to develop equation 4.41. The inverse kinematics problem can be decoupled into inverse position and orientation kinematics. The inboard joint variables (first three joints) can be solved using the position vectors of both sides in equation 4.41. The revolute joint variables  $\theta_1$  and  $\theta_2$  are obtained by equating the (3,4) and (1,4)elements of each side in equation 4.41. The last three joint variables may be found using the elements of rotation matrices of each side.

## Algebraic solution approach: Example (6 of 6)

- The rotation matrices are given by:

$$\begin{bmatrix} \cdot & r_{11}s\theta_2 + r_{12}c\theta_1c\theta_2 + r_{13}c\theta_1s\theta_2 & \cdot \\ d & r_{11}c\theta_2 - r_{12}c\theta_1s\theta_2 - r_{13}s\theta_1s\theta_2 & \cdot \\ \cdot & r_{12}s\theta_1 - r_{13}c\theta_1 & \cdot \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & -c\theta_4s\theta_5 & \cdot \\ c\theta_6s\theta_5 & -s\theta_6s\theta_5 & c\theta_5 & \cdot \\ \cdot & \cdot & s\theta_4s\theta_5 & \cdot \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \dots(4.45)$$

where  $d = r_{11}c\theta_2 - r_{12}c\theta_1s\theta_2 - r_{13}s\theta_1s\theta_2$  and  $e = r_{12}s\theta_1 - r_{13}c\theta_1$ .

- The revolute joint variables  $\theta_5$  is determined by equating the (2,3) elements of both sides in equation 4.45 and using the 4th trigonometric equation in Table, as:

$$\theta_5 = A \tan 2 \left( \pm \sqrt{1 - (r_{11}c\theta_2 - r_{12}c\theta_1s\theta_2 - r_{13}s\theta_1s\theta_2)^2} \cdot r_{12}s\theta_1 - r_{13}c\theta_1s\theta_2 - r_{13}s\theta_1s\theta_2 \right)$$

$$\theta_4 = A \tan 2 \left( \frac{r_{12}s\theta_1 - r_{13}c\theta_1}{s\theta_5} \cdot - \frac{r_{11}s\theta_2 + r_{12}c\theta_1c\theta_2 + r_{13}c\theta_1s\theta_2}{s\theta_5} \right) \quad \dots(4.46)$$

$$\theta_6 = A \tan 2 \left( - \frac{r_{11}c\theta_2 - r_{12}c\theta_1s\theta_2 - r_{13}s\theta_1s\theta_2}{s\theta_5}, \frac{r_{11}s\theta_2 - r_{12}c\theta_1s\theta_2 - r_{13}s\theta_1s\theta_2}{s\theta_5} \right) \quad \dots(4.47)$$

Figure 3-32. Algebraic solution approach: Example (6 of 6)

AIR011.0

### Notes:

#### Algebraic solution approach

Extracting  $\cos\theta_4$  and  $\sin\theta_4$  from (1,3) and (3,3),  $\cos\theta_6$  and  $\sin\theta_6$  from (2,1)and (2,2) elements of each side in equation 4.45 and using the 3<sup>rd</sup>trigonometric equation in Table 4.42,  $\theta_4$  and  $\theta_6$  revolute joint variables can be computed. Although solution of the forward kinematics problem is steady forward, the solution of the inverse kinematics problem strictly depend on the robot structures. The inverse kinematics solution for a manipulator whose structure comprises of revolute joints generally produces multiple solutions. Each solution should be checked in order to determine whether or not they bring the end-effector to the desired position.

## Advanced robotics (1 of 5)



IBM ICE (Innovation Centre for Education)

- Due to low cost computing power and enhanced functionality of existing mechanisms, initiatives by European Community in advanced robotic activities.
- Extended the application domain into new and potentially gainful market sectors by exploring the innovations in robotics.
- An advanced robot was defined as a machine or system capable of accepting high level mission-oriented commands, navigation to a workplace and performing complex tasks in a semi-structured environment with a minimum of human intervention.
- The application areas include:
  - The nuclear industry, space, underwater, construction, health care and general service functions such as visual surveillance and cleaning.
- Advanced robotics is mainly dealing with the development of sensor-based control of mechanisms and automation and the development of appropriate system architectures to generate appropriate levels of functionality in order to execute the tasks.

Figure 3-33. Advanced robotics (1 of 5)

AIR011.0

### Notes:

#### Advanced robotics

Due to the increasing availability of low-cost computing power that enhance the functionality of existing mechanisms, initiatives have been taken within the European Community to foster the development of studies in and exploitation of advanced robotic activities. In addition, it is also planned to extend the application domain into new and potentially gainful market sectors by exploring the innovations in robotics. "An advanced robot was defined as a machine or system capable of accepting high level mission-oriented commands, navigation to a workplace and performing complex tasks in a semi-structured environment with a minimum of human intervention". The application areas which were planned to be considered for development include the nuclear industry, space, underwater, construction, health care and general service functions such as visual surveillance and cleaning.

"Advanced robotics is mainly dealing with the development of sensor-based control of mechanisms and automation and the development of appropriate system architectures to generate appropriate levels of functionality in order to execute the tasks". The human machinist will generally act in managerial role with most of the lower-level jobs being executed by sensor-based controllers embedded within an overall systems architecture which integrates real-time operation and components of artificial intelligence and machine intelligence. "The design of such architectures and the integration of the required enabling techniques of control, actuators, sensors and artificial intelligence provide intellectual challenges in current engineering research".

The efficient communication of a human operator with such potentially complex machines should generally involve processes which make best use of human skills and reduce cognitive loading, in particular in risky situations. There are several types of feedback that are commonly employed to attain these goals including visual and force feedback components. "If three-dimensional models of the robot and its environment are available then computer generated displays can create a virtual world in which a suitably instrumented operator can be immersed to explore the limits of both the machines and his own capabilities.

Changing scenarios can be developed by overlaying direct stereo video feedback and additional sensor/actuator combinations can be used to generate a range of tactile feedback signals (contact pressure, force, texture, slip, hardness and temperature) to enhance the sense of reality and provide additional information to the operator concerning the state and performance of the remote robotic device. The objective is not just to control remote mechanisms by slaving their operation to the natural movement of human joints (fingers, wrists, arms and head position) but to create the sensation of telepresence to allow the operator to master complexity by utilizing inherent human skills and natural reflex capabilities". Clearly the challenge here is not just to develop appropriate technology but also to comprehend and optimize human psychological performance in such a situation.

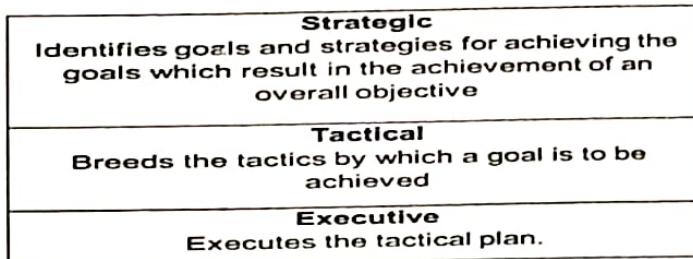
## Advanced robotics (2 of 5)

IBM

IBM ICE (Innovation Centre for Education)

### Implementation of a three-layer architecture:

- Three-layer systems architecture.



- A basic functional representation of a representative hierarchical architecture:

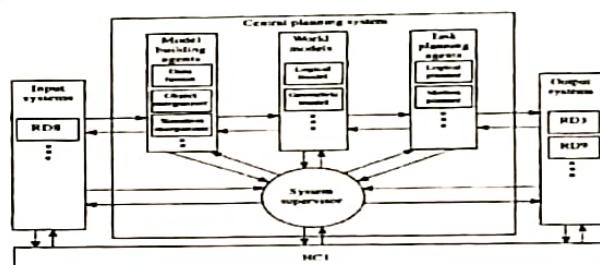


Figure: A hierarchy-based robot architecture

Figure 3-34. Advanced robotics (2 of 5)

AIR011.0

### Notes:

We have seen several methods that have been reformed to provide a comprehensive structure which will encompass aspects of artificial intelligence (as exemplified by task planning navigation/collision avoidance and situation analysis strategies), sensors and signal processing and human computer interface methodologies. "Traditionally, hierarchical deterministic procedures have found acceptance within the industrial environment because of the ready utilization of standard computing procedures and interfaces, accepted signal synchronization protocols and the basic transparency of the overall processing system provided by the use of finite state machines". A typical implementation of a three-layer architecture is shown in above figure.

A basic functional representation of a representative hierarchical architecture is given in above figure .The functions specified here generally require a more, or less, complex model of the real world (built up from either a priori knowledge or sensor data), task planning agents, appropriate advisory agents for the human operator and a communication link with the human computer interface.

## Advanced robotics (3 of 5)



IBM ICE (Innovation Centre for Education)

- Pre-specified priority of operation:
  - A layer with a high level of priority can inhibit the operation of a layer with a low level of priority.
  - The overall architecture consists of an assembly of such layers, as shown in figure.
  - The overall behavior of the robot being determined by how the various outputs are combined.
  - Each layer has its own relatively simple computing engine.
  - Parallel operation is possible.
  - The overall computing overheads are generally low.

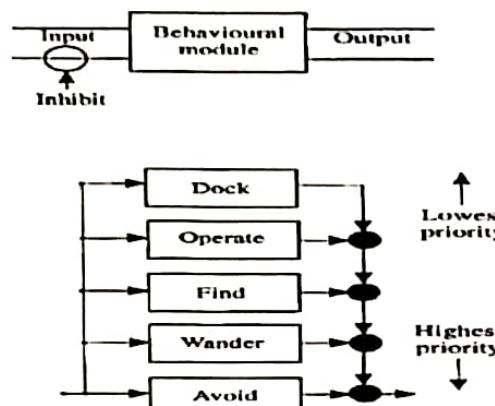


Figure: Behavior based robot architecture

Figure 3-35. Advanced robotics (3 of 5)

AIR011.0

### Notes:

On the other-hand, we have also seen another alternative architecture called behavioral control architectures that typically break-down the overall task into a set of behaviors/competences each of which selectively assists or assumes the control functions of other behaviors. The behaviors are usually organized into a layered architecture, with the lowest level being assigned the responsibility of, for example, obstacle avoidance. The priority of operation is pre-specified and accordingly a layer with a high level of priority can inhibit the operation of a layer with a low level of priority and the overall architecture consists of an assembly of such layers, as shown in figure with the overall behavior of the robot being determined by how the various outputs are combined.

There is no overall world-model to be shared between the various layers, intercommunication is of a simple protocol, narrow bandwidth type, each layer has its own relatively simple computing engine, parallel operation is possible and the overall computing overheads are generally low. "It shall be noted from the hierarchical approach that the emphasis is on logical functional decomposition and its distribution of data flows and control functions amongst the individual components whereas in behavioral control, operation is determined by active interaction with the physical world and the integration of a set of simple behaviors to achieve complex capabilities".

## Advanced robotics (4 of 5)

IBM

IBM ICE (Innovation Centre for Education)

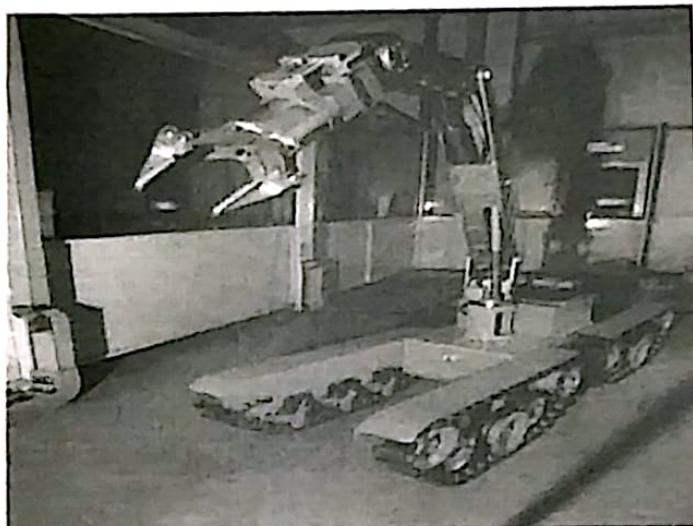


Figure: An advanced robot for the nuclear industry

---

Figure 3-36. Advanced robotics (4 of 5)

AIR011.0

### Notes:

Advanced robots possess the ability to operate in unstructured environments where hazards may be encountered. It is thus important that their design is inherently robust in every aspect that the device will cope with arena of difficult scenarios. In application areas such as health care, transportation, construction, surveillance, agriculture and the service industries, advanced robotic devices will come into close proximity to humans without the usual safety barriers normally associated with manufacturing robots. Over the last few years, we have witnessed a gradual expansion in the number of engineering applications of advanced robotic. The nuclear industry has always been an obvious application domain for robotics with an emphasis being placed on remote handling systems. Recent advances include the use of stereo vision and direct force feedback that employs six axis force sensitive joysticks based on configurations such as a "Bilateral Stewart Platform".

The Trajectory control, real-time collision avoidance and the extensive use of geometric modelling procedures for risk assessment and training are technical features now being introduced into material handling systems. A number of designs for relatively agile mobile robots is now being considered, such as that shown in above figure for intervention within unstructured/hazardous environments. Such devices must be mechanically robust, have predictable characteristics and operate in a totally stable/secure manner. One can find here much scope for the development of innovative human-computer interfaces to have a remote control of such devices and to ensure their certain detachment in any operational situation.

## Advanced robotics (5 of 5)



IBM ICE (Innovation Centre for Education)

- Medical field applications:
  - Robot controlled prostate surgery.
  - The precision robotic machining of bones for hip surgery utilizing geometrical models derived from precisely computed-tomography image-information.
  - Use of autonomous/machine guided vehicles to aid the disabled, and aspects of microsurgery that employ micromanipulators.
  - Virtual reality and force feedback concepts.
- Mobile automata
  - The ESPRIT Project PANORAMA:
    - Development of practical perception and navigation systems suitable for the control of vehicles.
    - Implications for modern high-density motorway utilization.
    - By enhancing safety and the overall throughput of the existing roadway systems.
- Applications in aerospace:
  - Advanced tele-operated devices.
  - Mobility for satellite maintenance.
  - Constructional purposes and planetary exploration.

Figure 3-37 Advanced robotics (5 of 5)

AIR0110

### Notes:

The medical field has embraced this advanced robotic technology to the maximum extent and certain applications include robot controlled prostate surgery, the precision robotic machining of bones for hip surgery utilizing geometrical models derived from precisely computed-tomography image-information, use of autonomous/machine guided vehicles to aid the disabled, and aspects of microsurgery that employ micromanipulators, virtual reality and force feedback concepts. The safety of such systems and concomitant legal aspects are key issues here and it is refreshing to see that these have been addressed in some detail by key workers in the field.

The study of mobile automata has been the focus of much robotic research over many years and this work is being utilized in practical applications. The ESPRIT Project PANORAMA has resulted in the development of practical perception and navigation systems suitable for the control of vehicles in general outdoor use with a particular application being the development of intelligent autonomous systems for cars which could have significant implications for modern high density motorway utilization, both by enhancing safety and the overall throughput of the existing roadway systems.

Applications in aerospace integrated advanced tele-operated devices and mobility for satellite maintenance, constructional purposes and planetary exploration with the latter encouraging the evolution of innovative mobility mechanisms such as legged robots to cope with difficult terrains. Mining is another fruitful area for the application of robotics as it combines a hazardous environment within a relatively structured topography and is certainly suitable for remote tele-operation.

# Machine intelligence: Architectures, controllers and applications

IBM

IBM ICE (Innovation Centre for Education)

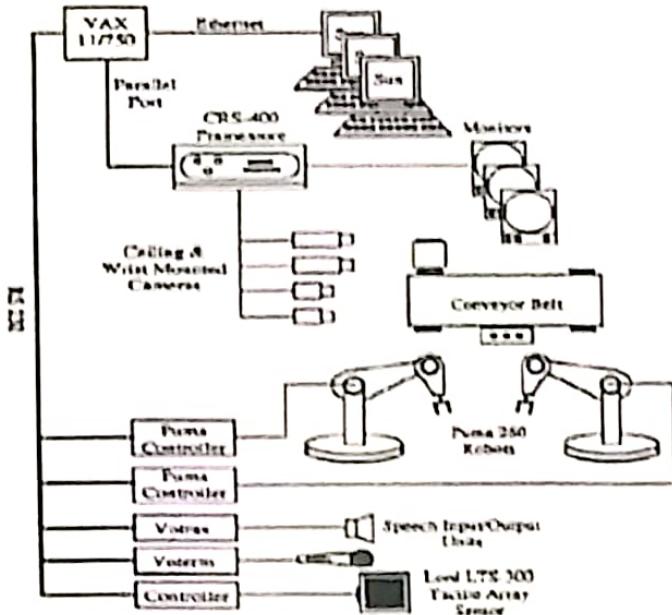


Figure: The Freddy 3 advanced robotic research test-bed.

Figure 3-38. Machine Intelligence: Architectures, controllers and applications

AIR011.0

## Notes:

### Machine Intelligence

This deals with mechanisms and technologies for implementing machine intelligence in a robotic system that has an ability to learn and able to act autonomously in the presence of uncertainty. "The ability of a robot to adjust its actions based on sensed information could be seen as another prerequisite for intelligence. The actions taken by the machine would be considered to be intelligent if the same action would be taken by a human given the same conditions".

In advanced robotics systems, robots equipped with sensors such as vision, tactile, proximity, speech understanding and voice recognition, and components such as robot controllers, conveyors, vision processing equipment, and workstations, all networked together through a variety of high bandwidth (Ethernet) and low bandwidth (RS-232, IEE-488) communication channels. An advanced robotics systems that commonly incorporate the three major subsystems for machine intelligence namely: sensors, actuators and control as shown in figure.

## Architectures for intelligent control (1 of 2)

IBM

IBM ICE (Innovation Centre for Education)

- "The interconnection of physical systems, or the task undertaken by the system, does not make a machine intelligent.
- Intelligence comes from the manner in which the system is controlled or from the reasoning and decision making that the machine performs.
- Therefore intelligent-control is closely associated with machine intelligence" and is depicted in Figure.

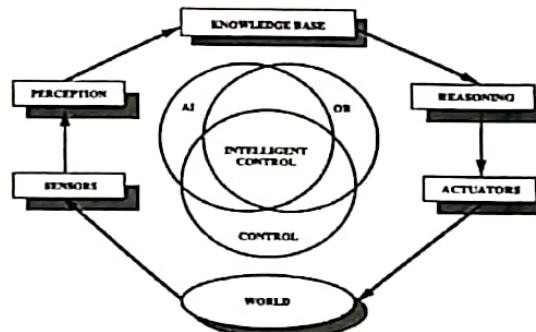


Figure: Architecture for intelligent control system.

Figure 3-39. Architectures for intelligent control (1 of 2)

AIR011.0

### Notes:

#### Architectures for intelligent control

The intelligent machines require the use of generalized control strategies to perform intelligent functions such as the simultaneous utilization of memory, learning or multi-level decision-making in response to fuzzy/qualitative commands and proposed that the intelligent functions should be implemented using intelligent control. An intelligent control is a blend of high-level decision making by using computers, and advanced mathematical modelling and synthesis techniques of systems theory together with linguistic methods that attempt to deal with imprecise or incomplete information from which appropriate control actions evolve. The control functions in an intelligent machine have been implemented as a hierarchy of processes. The upper layers concentrate on abstractions, decision-making and planning, while the lower levels concentrate as time-dependent sub-tasks, like processing data from sensors, or operating an actuator. Hierarchical decomposition is applied to complex control problems to reduce them to smaller sub-problems.

## Architectures for intelligent control (2 of 2)

IBM

IBM ICE (Innovation Centre for Education)

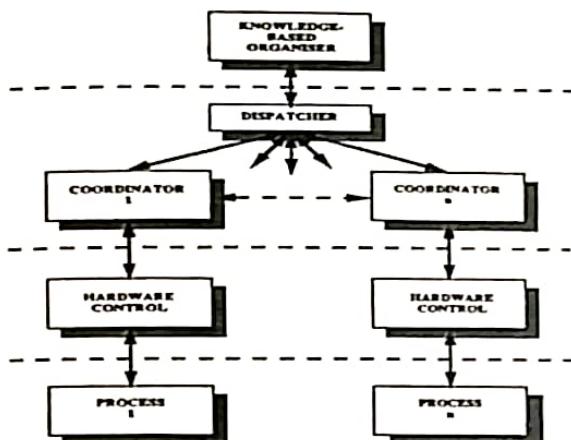


Figure: Architecture for intelligent control system

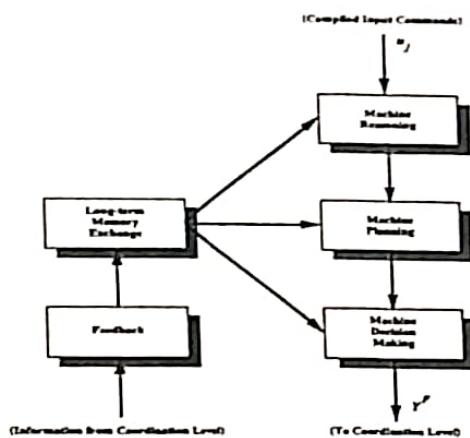


Figure: Saridis Processing Modules

Figure 3-40. Architectures for intelligent control (2 of 2)

AIR011.0

### Notes:

#### Saridis architecture

Saridis proposes a hierarchical structure of three basic levels of control namely organization level that deals with task planning, learning reasoning and information processing from long-term memory; co-ordination level that deals lower level information processing and short-term memory and control level that executes tasks through actuation and feedback control. This hierarchy is distributed according to his 'Principle of Precision with Decreasing Intelligence'. Each level consists of several layers of processing modules that communicate with each other with strings of symbols and the architecture is given in figures.

## Machine learning



IBM ICE (Innovation Centre for Education)

- Expert systems are the most successful implementation of AI.
- Applications include: Medical diagnosis, mineral analysis, control of complex processes, and fault diagnosis and monitoring; systems for fault diagnosis and monitoring account for half their applications.
- Machine learning is described as going from the specific to the general.
- Here, we will look at:
  - How the production rules for 'rule-based' control can be produced manually, and automatically, and
  - Discuss two approaches for achieving Machine Learned Control (MLC).
- First, we will describe a controller based on the machine learning algorithm BOXES.
  - An algorithm that uses a reinforcement learning approach.
- Second, we will discuss the implementation of neural networks for control.
  - Both reinforcement learning and competitive learning are considered.

Figure 3-41. Machine learning

AIR011.0

### Notes:

#### Machine learning

We have seen the exploration of artificial intelligence techniques that have been continuously developed and applied by industry, business, and commerce for various purposes. "Expert systems are the most successful implementation of AI. Applications include medical diagnosis, mineral analysis, control of complex processes, and fault diagnosis and monitoring; systems for fault diagnosis and monitoring account for half their applications". However, the difficulties surrounding the development of the production rules for expert systems, going from the general to the specific, including knowledge acquisition, descriptive knowledge representation and context handling, have led to the development of a sub-division of expert system technology known as 'machine learning'. Compared to expert systems, machine learning can be described as going from the specific to the general.

Here, we will look at how the production rules for 'rule-based' control can be produced manually, and automatically, and we will discuss two approaches for achieving machine learned control (MLC). Firstly, we will describe a controller based on the machine learning algorithm BOXES, an algorithm that uses a reinforcement learning approach. Secondly, we will discuss the implementation of neural networks for control; both reinforcement learning and competitive learning are considered.

## Machine learning: Rule-based control (1 of 3)

IBM

IBM ICE (Innovation Centre for Education)

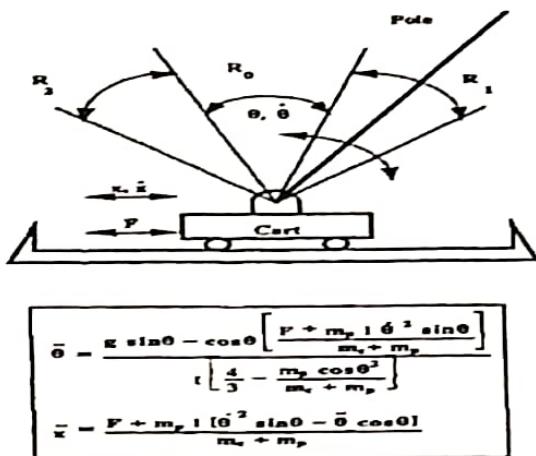


Figure: Pole and cart.

Figure 3-42. Machine learning: Rule-based control (1 of 3)

AIR011.0

### Notes:

#### Rule-based control

In many expert systems, the rule interpreter/inference engine uses an exhaustive backward-chaining strategy in which the rules are expressed in a highly focused 'IF-condition-THEN-result' form, the result is known or concluded, and, knowing the answer, the system searches backwards for the rule whose condition produces the result. This type of expert system is called as goal directed. The Goal-directed systems are therefore the result of going from the general to the particular since the search is, in fact, directed toward a goal i.e., the known result. An inference mechanism based on a backward chaining strategy is commonly equipped with an extensive explanation facility to reduce the amount of conflict resolution needed.

To be user friendly, an expert system must also possess a user interface that interrogates the human expert in a manner that is unambiguous. The information obtained from the user interface must then be stored as facts in a database and the relationship between individual facts must be stated. The rule structure determines which rule should be examined next, in a forward-chaining expert system, or it further questions the user in a backward chaining system. Finally, the representational structure must also include an uncertainty handling mechanism that is based on measurements of belief. A human can derive a set of control rules through the process of interpretation, Makarovic derived a rule by examining a system's differential equations of motion (follow the above figure).

## Machine learning: Rule-based control (2 of 3)

- if  $\theta_{dot} > \text{THRESHOLD}$  then push RIGHT
- if  $\theta_{dot} < -\text{THRESHOLD}$  then push LEFT
- if  $\theta > \text{THRESHOLD}$  then push RIGHT
- if  $\theta < -\text{THRESHOLD}$  then push LEFT
- if  $x_{dot} > \text{THRESHOLD}$  then push RIGHT
- if  $x_{dot} < -\text{THRESHOLD}$  then push LEFT
- if  $x > \text{THRESHOLD}$  then push RIGHT
- if  $x < -\text{THRESHOLD}$  then push LEFT

Figure 3-43. Machine learning: Rule-based control (2 of 3)

AIR011.0

### Notes:

#### Example of rule-based control

An example for rule-based machine learning system. It can be seen that the above Makarovic rule worked well where the system's specifications remain constant.

# Machine learning: Rule-based control (3 of 3)



IBM ICE (Innovation Centre for Education)

```

if (theta(k) > THRESHOLD)
then
if((theta(k)<theta(k-1))
and (ltheta(k)-theta(k-1)) > ltheta(k-1)-theta(k-2))
then
apply a RIGHT force
else
apply a LEFT force
if (theta(k) <-THRESHOLD)
then
if((theta(k)>theta(k-1))
and (ltheta(k)-theta(k-1)) > ltheta(k-1)-theta(k-2))
then
apply a RIGHT force
else
apply a LEFT force
if (ltheta(k)) <=THRESHOLD)
then
if(x(k)>=0)
then
if ((x(k)<x(k-1) and (lx(k)-x(k-1)-x(k-2)))|
then
apply a RIGHT force
else
apply a LEFT force
if(x(k)<0)
then
if ((x(k)>x(k-1) and (lx(k)-x(k-1))>lx(k-1)-x(k-2)))
then
apply a RIGHT force
else
apply a LEFT force

```

Figure 3-44. Machine learning: Rule-based control (3 of 3)

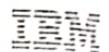
AIR011.0

## Notes:

### Example of rule-based control

When system parameters change, the Makarovic rule cannot guarantee success. Because of the arbitrary choice of threshold values by Makarovic. One set of threshold values is not ideal for a system configuration that alters. A rule derived from observing the systems performance is written without any threshold values being placed on observation. Here the condition part of the rules only deals with the sign of errors, and with the sign of variations of observed system state variables. It reflects the human control heuristics, and can adapt itself to varying system configurations and perform consistently well in all circumstances.

# Machine learning: Machine learned control



IBM ICE (Innovation Centre for Education)

- Machine learning is classified into two areas:
  - AI type learning on symbolic computation.
  - Neural networks.
- Expert systems based on artificial-intelligence learning:
  - Have exceeded the performances of human experts and,
  - Could communicate what they have learned to human experts for the purposes of verification.
- Artificial-intelligence type learning originated from an investigation into the possibility of using decision-trees/production-rules for concept representation.
- The field use decision-trees/production-rules in order to handle the most conventional data types, including those with noisy data, and as a knowledge acquisition tool.

Figure 3-45. Machine learning: Machine learned control

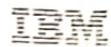
AIR011.0

## Notes:

### Machine learned control

Machine learning is classified into two areas namely artificial-intelligence type learning on symbolic computation, and neural networks. Expert systems based on artificial-intelligence learning have exceeded the performances of human experts and, could communicate what they have learned to human experts for the purposes of verification. Artificial-intelligence type learning originated from an investigation into the possibility of using decision-trees/production-rules for concept representation. Subsequently, the field is extended to use decision-trees/production-rules in order to handle the most conventional data types, including those with noisy data, and as a knowledge acquisition tool.

## Machine learning: Reinforcement learning



IBM ICE (Innovation Centre for Education)

- Reinforcement learning is similar to supervised learning.
- It uses feedback for adaptation.
- The feedback reinforcement learning gets is only an indication of the value of the systems action.
- Reinforcement is feedback on the correctness of an action.
- Reinforcement learning is useful in cases where:
  - Supervisory information is not available.
- Reinforcement learning falls into two categories:
  - Non-associative type.
  - Associative reinforcement learning.

Figure 3-46. Machine learning: Reinforcement learning

AIR011.0

### Notes:

#### Reinforcement learning

Reinforcement learning is similar to supervised learning which uses feedback for adaptation. However, unlike supervised learning, the feedback reinforcement learning gets is only an indication of the value of the systems action. Reinforcement is feedback on the correctness of an action, it is not information on what the correct action is. Reinforcement learning is useful in cases where supervisory information is not available. Also, reinforcement learning falls into two categories:

- Non-associative type, which only receives a reinforcement signal from the environment.
- Associative reinforcement learning, where the system receives both a reinforcement signal, and sensory information, on the state of the environment.

Here, sensors are used to discriminate between different situations.

## Advanced control systems for robotic arms



IBM ICE (Innovation Centre for Education)

- Robotic arm is a mechanical arm to perform the desired task.
- The links of the manipulator can be considered to form a kinematic chain.
- The business end of the kinematic chain of the schemer is called the end effector and it is analogous to the human hand.
- The end effector can be a gripper or designed to perform any desired task such as welding, painting, assembly, etc.
- Robot must:
  - Control the position of the tool with respect to the work as a function of time.
  - Control the operation of the tool. Actuators – move the joints by a particular form of drive system.
- Common drive systems used in robotics are:
  - Electric drive.
  - Hydraulic drive, and
  - Pneumatic drive.

Figure 3-47. Advanced control systems for robotic arms

AIR011.0

### Notes:

#### Advanced control systems for robotic arms

"Robotic arm is a mechanical arm to perform the desired task. In today's world there is an increasing need to create artificial arms for different inhuman situations where human interaction is difficult or impossible. Humans pick things up without thinking about the steps involved. So the robotic arm is controlled manually by using wired and wireless".

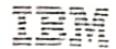
A robot manipulator consists of links connected by joints. The links of the manipulator can be considered to form a kinematic chain. The business end of the kinematic chain of the schemer is called the end effector and it is analogous to the human hand. The end effector can be a gripper or can be designed to perform any desired task such as welding, painting, assembly, etc. An end-effector is a tool or gripping mechanism attached to the end of a robot arm used to make intentional contact with an object or to produce the robot's final effect on its surroundings to accomplish some task. Tools are used in applications where the robot must perform some processing operation on the work part.

In each case the robot must not only control the corresponding position of the tool with respect to the work as a function of time, it must also control the operation of the tool. The actions of the individual joints must be controlled in order for the schemer to perform a desired motion. The robot's capacity to move its body, arm, and wrist is provided by the drive system used to power the robot. The joints are moved by actuators mechanized by a particular form of drive system. Common drive systems used in robotics are electric drive, hydraulic drive, and pneumatic drive.

The robot control problem can be divided into two main areas: kinematic control, the co-ordination of the links of the kinematic chain to produce desired motions of the robot, and dynamic/kinetic control, driving the actuators of the mechanism to follow the commanded positions/velocities. "For advanced robotic arms to operate with some increased degree of autonomy within non-structured, partially unknown, or hostile environments, systems will be required to enable the robots to identify potential problems in their environments and implement limited responses in real time". This requires that the kinematic control of the robot arm is oriented towards multiple task performance in real time in order to incorporate sensor information. In particular this has involved work in two areas, 'reactive' collision avoidance for robot arms, and the control of redundant robot arms.

The modern industrial robot is typically controlled, at the lowest level, by decoupled Proportional, Integral and Derivative (PID) controllers: a standard industrial control technique. The application of such simple linear controllers is made possible by using higher gear ratios between the drive motors and links, decoupling the non-linear payload from the actuator, and a relatively slow motion range. The current trend in industrial applications is towards lighter and faster robots where the applicability of these techniques is limited, and there is therefore a requirement for more sophisticated control algorithms.

## Kinematic and dynamic control



IBM ICE (Innovation Centre for Education)

- Most industrial robot arms are designed so that the principle of wrist partitioning can be used.
- In machine design, a mechanism is synthesized that functions with the minimum complexity of the structure.
- Generally claimed with six degrees-of-freedom non-redundant robot arm is a general purpose device.
- Welding robots, which usually do not need rotation about the welding torch, have five actuators.
- Kinematic redundancy: When a robot arm possesses more degrees of freedom than the minimum number required to execute a given task.
- A six degree-of-freedom non-redundant robot arm can 'freely' position and orient an object in the Cartesian workspace.

Figure 3-48. Kinematic and dynamic control

AIR011.0

### Notes:

#### Kinematic and dynamic control

Inverse kinematics include the computation needed to find joint angles from a given Cartesian position and orientation of the end-effector. This computation is central in the control of robot arms. It is, in general, a non-linear algebraic computation which has been shown for the general case of a 6 degree-of-freedom arm to require the solution of a sixteenth order polynomial equation. "Most industrial robot arms are designed so that the principle of wrist partitioning can be used, that is where the three wrist joint axes intersect at a point, which reduces the problem to that of a second order solution. It should be noted that even for many 6 DoF robot arms, multiple solutions exist relating to different configurations (e.g. elbow up/down for a PUMA) but these are of a finite number (e.g. eight for a PUMA) and in general the robot arm cannot move between configurations unless passing through a singularity".

#### Kinematic control design:

In machine design, a mechanism is synthesized that functions with the minimum complexity of the structure. Most industrial robots have been designed based on this commonsense rule. It is generally claimed that a six degrees-of-freedom non-redundant robot arm is a general-purpose device since it can 'freely' position and orient an object in the Cartesian workspace. Welding robots, which usually do not need rotation about the welding torch, have five actuators.

Kinematic redundancy occurs when a robot arm possesses more degrees of freedom than the minimum number required to execute a given task, and it is fundamental to understand why it becomes necessary to introduce kinematic redundancy into a robot with its commensurate increase in mechanical and control complexity.

Although nominally a six degree-of-freedom non-redundant robot arm can 'freely' position and orient an object in the Cartesian workspace, its geometry, in fact, has a number of kinematic flaws such as limited joint ranges, singularities, and workspace obstacles which prevent the arm from doing so. It is therefore desirable for a true general-purpose robot arm which is not constrained to structured, well designed environments, to dispose of additional degrees of freedom to overcome the above limitations. Dexterity in a robotic arm implies the mechanical ability to carry out a variety of non-trivial movement tasks. If a robot arm is to have high dexterity it should have seven or more degrees of freedom rather than six or less. The seven degree-of-freedom human arm constitutes an excellent model of a dexterous redundant arm.

## Intelligent gripping systems



IBM ICE (Innovation Centre for Education)

- Robotic grasping and restraint have evolved into an important field of robotics research, which is due partly to the evolution of industrial automation towards flexible automation, and partly to the progress in the study of fundamentals.
- Attention has been given to the mechanics of grasps and to the grasping of a broader class of objects, both in precise assembly and in hazardous environments.
- The need is to determine the graspability of objects, so as to plan a stable and optimal grasp, and to impart fine motion and force control.
- A versatile grasping system integrates sensors and grippers, together with appropriate software.
- At present, the gripping systems and mechanical hands are weakly related to the human hand, which raises the issue of the study of anthropomorphic hands.
- Anthropomorphic hands make use of human knowledge, relying upon a large database for gripper models and sensors for feedback.

Figure 3-49. Intelligent gripping systems

AIR011.0

### Notes:

#### Intelligent gripping systems

"In the recent days, robotic grasping and restraint have evolved into an important field of robotics research, which is due partly to the evolution of industrial automation towards flexible automation, and partly to the progress in the study of fundamentals. Attention has been given to the mechanics of grasps and to the grasping of a broader class of objects, both in precise assembly and in hazardous environments. This has pointed out the need for us to determine the graspability of objects, so as to plan a stable and optimal grasp, and to impart fine motion and force control".

The requirement is also heavily dependent on the need for a versatile grasping system with integration between sensors and grippers, together with appropriate software. At present, the gripping systems and mechanical hands are weakly related to the human hand, which raises the issue of the study of anthropomorphic hands. "Anthropomorphic hands make use of human knowledge, relying upon a large database for gripper models and sensors for feedback. However, the use of anthropomorphic hands is restricted in many specialist areas, and they lack general applicability".

Further, human hands focus attention on the issue of sensors in the grippers. "Although vision systems have been widely used, the successful execution of grasps still needs the assistance of tactile sensors, which not only sense the contact forces but also sense the geometry of an object, to establish feedback to plan and control the grasp". In addition, high expense and some special environments also hinder the use of vision systems. Meanwhile, although much work has gone into grasping research, the property of elasticity of the contacts has rarely been used.

Thus the real understanding of a grasp, and its force control have been hindered; in spite of *grasp-mode paradigms* and exhaustive heuristic methods being used, many published analyses of grasps have not been towards a general applicability. Thus, a good understanding of grasps including their planning and optimization are needed in building gripping systems.



## Overview of the Salford theories (1 of 2)

IBM ICE (Innovation Centre for Education)

- Salford is used to analyze and synthesize the grasp, with the application of:
  - Screw algebra, convex algebra and algebraic geometry, and a set of new approaches.
- A known grasp can be analyzed. A grasp of a known and unknown objects can be synthesized and planned.

### Theories on restraint and grasping:

- The minimum number of frictionless point contacts to complete restraint is needed to perform either translational restraint or complete restraint.
- Table Minimum number of contacts in restraint.

$$\sum_{i=1}^{n+1} f_i s_i = w$$

	Translational	Complete
Planar	3	4
Spatial	4	6

Figure 3-50. Overview of the Salford theories (1 of 2)

AIR011.0

### Notes:

#### Overview of the Salford theories

A new system of theories has been established at Salford to analyze and synthesize the grasp, with the application of screw algebra, convex algebra and algebraic geometry, and a set of new approaches has been created to facilitate the use of the theories. Thus a known grasp can be analyzed, a grasp of a known and unknown objects can be synthesized and planned.

#### Theories on restraint and grasping:

The minimum number of frictionless point contacts to complete restraint is shown in the table which is needed to perform either translational restraint or complete restraint. The restraint equation is thus formed to complete the restraint in matrix form.

Where in equation:

- $f_i$  is the magnitude of  $i^{\text{th}}$  contact force.
- $s_i$  is the screw of  $i^{\text{th}}$  contact force.
- $w$  is the external force.

*Student Notebook*

By incorporating screw theory, a set of theories of analyzing and synthesizing the grasp has been developed. They describes the state of instantaneous kinematic restraint of a rigid body in multiple point contact with its surroundings, it used for the synthesis of any required state of kinematic restraint or freedom. It based on a known set of contacts, their positions and screws, a further contact can be established to add to the existing set, whose line vector is reciprocal to all the desired freedoms.

3-60 AIR01

© Copyright IBM Corp. 2019

Course materials may not be reproduced in whole or in part  
without the prior written permission of IBM.



## Overview of the Salford theories (2 of 2)

IBM ICE (Innovation Centre for Education)

- In consideration of frictional restraint, equation (4.48) is in the form:

$$\sum_{i=1}^n f_i s_i = w$$

- Where  $n$  is the number of contact points, and  $v$  is 2/3 in planar or spatial cases respectively.
- A further equation of geometric compatibility of contact points has to be introduced in the form:

$$u = [J]^T [\Delta] D$$

- where  $[J]$  is a Jacobian matrix formed by contact screws  $s_i, i=1, \dots, v$ ,  
 $u$  is the displacement along  $v$  contact screws.  
 $[\Delta]$  is the matrix operator to exchange both parts of  $D$ , and  
 $D$  is the general twisting displacement of an object.

Figure 3-51. Overview of the Salford theories (2 of 2)

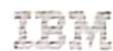
AIR011.0

### Notes:

#### Salford theories

As a necessary condition to form restraint either in the sense of partial restraint or complete restraint, the above restraint equation is over-constrained with more contacts than equations: A further equation of geometric compatibility of contact points has to be introduced in the form:  $u = [J]^T [\Delta] D$ , where  $[J]$  is a Jacobian matrix formed by contact screws  $s_i, i=1, \dots, v$ ,  $u$  is the displacement along  $v$  contact screws,  $[\Delta]$  is the matrix operator to exchange both parts of  $D$ , and  $D$  is the general twisting displacement of an object.

## Need and provision of fingertip sensor system



IBM ICE (Innovation Centre for Education)

- The implications of the above results suggest that for a grasp to be analyzed it is necessary to have a knowledge of the positions of the contact points, and the directions of the normal to the surface of the object at those contact points.
- If friction is involved, then it is also necessary to know the elastic properties of the contacts or fingers, and also the coefficients of limiting friction at the contacts.
- The new fingertip sensor system is:
  - Capable of allowing the detection of position of contact point.
  - Relative to the known position and orientation of the supporting frame or fingertip.
  - To a satisfactory level of accuracy, typically to 0.5 mm in any co-ordinate direction.

Figure 3-52. Need and provision of fingertip sensor system

AIR011.0

### Notes:

#### Need of fingertip sensor-system

The direction of the common normal between object and fingertip can similarly be assessed to a very high level of accuracy, typically to within  $1^\circ$ . The fingertip sensors can be used to discriminate between normal and tangential components of contact force so that the ratio between these components can be assessed, to an accuracy sufficient for most purposes of grasping. Given the levels of uncertainty normally associated with the phenomenon of surface friction, and provided that relative motion can be assured in the tangent direction on the object's surface, the coefficient of limiting friction can also be estimated. Typical errors are less than 1.0% in the use of the fingertip sensor systems. These were verified using random but known positions of fingertip and the surface of objects.

## Computer software package implementation (1 of 2)



IBM ICE (Innovation Centre for Education)

- A software has been developed:
  - To implement the new theories and approaches.
  - To effectively use the information from the fingertip system.
  - To facilitate the analysis and synthesis of grasps both in the analytical and experimental work.
- The software is written in C++:
  - Consists of a user-oriented package.
  - Two libraries of linear algebra and screw algebra, and
  - A set of independent programmes
- It is capable of:
  - Interrogating contact information, normal, position vectors etc. From the fingertip sensor system.
  - Giving detailed force distribution from the detection.
  - Assessing a grasp and giving the diagnosis of the grasp.
  - Planning a grasp on a set of contact information and further optimizing the grasp.
  - Analyzing a grasp with different properties such as stiffness.
  - Predicting preload and force distribution corresponding to any given external force.

Figure 3-53. Computer software package implementation (1 of 2)

AIR011.0

### Notes:

#### Computer software package implementation

To implement the new theories and approaches, and to effectively use the information from the fingertip system, a software has been developed to facilitate the analysis and synthesis of grasps both in the analytical and experimental work. The software is written in C++ consists of a user-oriented package, two libraries of linear algebra and screw algebra, and a set of independent programmes.

## Computer software package implementation (2 of 2)



IBM ICE (Innovation Centre for Education)

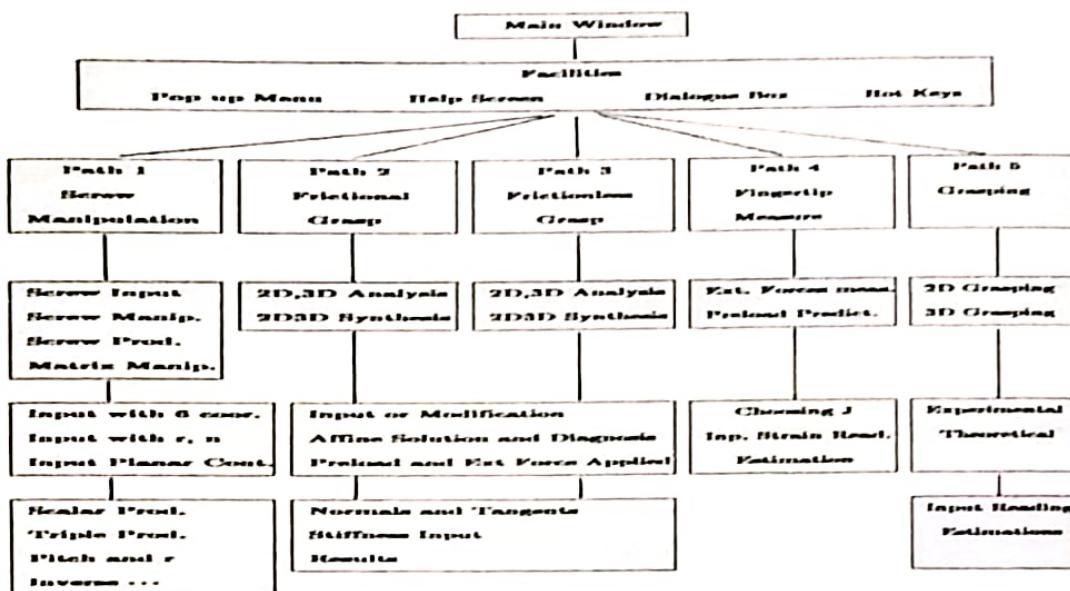


Figure: Flowchart of RASP package

Figure 3-54. Computer software package implementation (2 of 2)

AIR011.0

### Notes:

#### Computer software package implementation

The software package is composed of multiple user layers and paths, incorporated in a window interface with pop-up menus, hot keys, help menus and dialogue boxes. This above figure gives the flowchart of the RASP facility (Restraint Analysis and Synthesis Package).

#### The implementation of the gripping system consists of three parts:

- The detection of the object with the novel fingertip sensor system.
- The use of the software package which incorporates a new system of methodologies and algorithms.
- The execution of the grasp predicted from the software package.

One of the most demanding grasps, of a horizontal pipe, by implementing the gripping system. The tube is arranged so that its axis is horizontal, with three fingertips arranged in a horizontal plane which is located above the central axis of the tube. The location of this plane is indicated by an elevation angle  $\theta$ . The grouping of the three fingers is further arranged with its center either at or horizontally offset from the centroid of the tube. The horizontal offset of the group of the contacts is indicated by a centre distance  $p$ . The longitudinal spacing of the contacts is indicated by  $q$ . By arranging the case study with different grouping ratios  $p/q$  and different elevation angles  $\theta$ , a set of experiments was carried out. A comparison between the theoretical and experimental results gave excellent agreement. The errors were usually of the order of  $10^{-2}$  kgf in magnitude,  $10^{-1}$  degree in direction, and  $10^{-3}$  mm in pitch.

## **Force feedback control in robots and its application to decommissioning**



IBM ICE (Innovation Centre for Education)

- Force feedback control in robots and its application.
- Force control is a central requirement if robot arms are to use tools or interact with work-pieces in an unstructured environment.
- The paradigm considered is the use of tele-robots' - arms which are operated under human supervision while carrying out complex tasks.
- Robot force control - A technique to modify an arm's servo behavior when contacting its environment.
- Decommissioning is a task which requires the use of tools, such as drills and saws, in a radioactive contaminated environment. A number of systems have been, or are being, developed to accomplish this task.
- The experimental system at Oxford is based on a very high-performance parallel robot used as an input device coupled to a remote slave arm.

---

Figure 3-55. Force feedback control in robots and its application to decommissioning

AIR011.0

### **Notes:**

#### **Force feedback control**

"Force control is a central requirement if robot arms are to use tools or interact with work-pieces in an unstructured environment". We concentrate on a form of force control called active compliance control. The paradigm considered is the use of tele-robots' - arms which are operated under human supervision while carrying out complex tasks. The application area considered is nuclear decommissioning, but the technology is widely applicable to such diverse activities as bomb disposal and the Virtual reality' simulators of drug interactions. Robot force control is a generic term for a technique to modify an arm's servo behavior when contacting its environment. "Decommissioning is a task which requires the use of tools, such as drills and saws, in a radioactive contaminated environment.

A number of systems have been, or are being, developed to accomplish this task". The experimental system at Oxford is based on a very high-performance parallel robot used as an input device coupled to a remote slave arm. One of the possible control strategy that is being explored is use of force reflection or position-force tele-control. This strategy measures the position of the input device, or master arm, and transmits this to the position servoe of the remote, or slave arm. The slave arm moves in response to this command and a force sensor mounted on its wrist is used to drive the master arm motor torques. "It is well known that such a system suffers from stability problems on contact with a stiff environment, the most succinct explanation of this phenomenon may be found in Colgate and Hogan's work on the lack of passivity of this strategy". The main design point is that unless one is willing to pay for esoteric technology, slave arms should be based on standard industrial robots which brings the advantages of robustness and reliability.

Traditional master-slave systems rely on using common error as their control strategy, which links the dynamics, and more importantly, the drive friction of the slave, with the force sensed at the master. This may be acceptable for specialist designs using low friction drives and kinematically similar masters and slaves but prevents the use of standard robots with their associated commercial advantages. The objective is thus to control the behavior of the slave arm in a manner which permits the use of force reflection and yet still removes the effect of the slave drive friction from the force sensed at the master.

The design problem may be translated into that of modifying the perceived impedance at the master when the slave arm makes contact with the environment. This is achieved by a combination of force feedback around the slave and signal shaping between the force sensor and the torque fed to the master. The practical result is a high-performance master-slave system, which is able to reflect high frequency forces back to the operator with fidelity sufficient for remote tool use and typical operations such as saw blade replacement and insertion into a partially cut slot.

## Force feedback strategies

- One of the earliest force feedback strategies was active stiffness control.
- In this original formulation the joint torque to the motors is given by:

$$\tau = J^T K_p (q_d - q) + K_v (\dot{q}_d - \dot{q}) \quad \dots(4.50)$$

Where  $J$  is the arm Jacobian,

$K$  is the position gain chosen to achieve the requisite stiffness.

- This is a simple form of impedance control as developed by Hogan.
- The philosophy of impedance control is to specify the apparent robot end point dynamic impedance as seen by the environment.
- The term impedance is used within the context of the classical analogy of Bond graph theory, which maps force into effort (electrical voltage) and velocity into flow (electrical current).

Figure 3-56. Force feedback strategies

AIR011.0

### Notes:

#### Force feedback strategies

A number of early researchers addressed the problem of specifying end-point forces and/or compliance. It seemed self-evident that specific tasks exhibit specific degrees of freedom and types of constraint, i.e. inserting a peg into a hole might, at first glance, require position serving along the peg's axis and force serving orthogonal to it. A postulated solution to the associated control problem was hybrid control in which some Cartesian directions were position served, and their orthogonal complement force served directly (without any inner position loop). "The original hybrid control paper sparked a flurry of activity in the kinematics community who pointed out that the existence of such an orthogonal subspace as originally defined was not necessarily guaranteed. It was then shown that the original formulation also resulted in kinematic instability for revolute robots in certain configurations".

The kinematic instability arose because of interactions between the robot mass matrix and the robot Jacobian and is a simple consequence of not using an inner loop servo. Using a proper inner loop joint servo overcomes these kinematic instability problems, although one is still left with the problem of specifying a coordinate system for the 'Selection Matrix' used in such controllers for an unstructured environment.

There are more robust techniques for designing force controllers than those described above. One in particular is the class of controllers based on joint torque serving. As stated above, any practical controller requires a high gain inner loop servo to overcome the effects of drive friction, and this is usually provided by PD loops placed around the joints.

Joint torque servos should be placed downstream of the major joint non-linearities and can be tuned to reduce the effects of friction and cogging. They have the added advantage that the loop gains of a joint torque served system are not modulated by the environmental stiffness and in principal are more robust when the arm contacts a high stiffness environment.

However, the robustness of the joint torque servo itself, rather than the outer force loop, becomes an issue when the arm is in free space. Torque servos are non-collocated, and thus hard to stabilize, and cogging in the joint gears can introduce impulsive torques if the sensor is mounted directly on the output of the reducer and there is little inertia in the transmission; this limits the amount of phase advance or derivative action in the servo loop. These practical difficulties lead us to concentrate on traditional robot servo architectures, and in particular, active compliance control.



IBM ICE (Innovation Centre for Education)

## Introduction to mobile robots

---

- Mobile robots or autonomous guided vehicles (AGVs) follow fixed paths either by using an inductive sensor to locate a buried wire or a simple optical sensor to follow a white line.
- Various sensors can be used in robotic systems to achieve autonomous navigation:
  - Encoders.
  - Cameras.
  - Detectors.
  - Diversified range finders.
  - Radar.
  - Sonar.
  - Optoelectronic devices.

---

Figure 3-57. Introduction to mobile robots

AIR011.0

### Notes:

#### Introduction to mobile robots

Most mobile robots or autonomous guided vehicles (AGVs) follow fixed paths either by using an inductive sensor to locate a buried wire or a simple optical sensor to follow a white line. They are having very limited sensing capabilities. In the face of obstacles, they have no option but to stop. Even in a fixed environment, there are drawbacks, including erosion of painted lines, and the expense of altering the work area, especially if buried wires are used.

These limitations have seriously hampered commercial applications of AGVs. Many areas in which mobile robots could make an essential contribution, such as nuclear plant, oil rigs or surveillance demand greater flexibility and a degree of adaptation to the environment. In all of these applications, mobile robots may be required to explore and map environments and make changes to predetermined plans to cope with uncertain conditions. Such requirements demand that mobile robots have reliable onboard sensing to abstract suitable information from an unknown or dynamically changing environment.

Various sensors can be used in robotic systems to achieve autonomous navigation, for example shaft encoders, cameras, detectors and a diversified range finders based on technologies such as sonar and optoelectronic devices. The most obvious category is radar, which is being used, especially by the military, for off-road and outdoor vehicles, but its cost is still high compared with other technologies. Since every type of sensor has specific areas of operation and certain failure modes, a single sensor can only provide partial information. Therefore to handle general environments, multi-sensor systems become necessary. Many important issues are involved in multi-sensor robots such as establishing a common framework to include different sensors setting up communications between the various sensor systems handling noise, error, and conflict in sensory data, planning strategies for sensor integration.

The sensors which are being developed for robotics and discuss how they might be used together to provide a robust facility for real-time sensing, planning and control. The main requirement for robot control and planning is to determine the range of nearby objects; i.e., how far away they are from the robot. In the next section, we describe three categories of sensor for measuring range: those based on sonar, on optoelectronics, and on vision.

## Environment capturing with common sensors

- Sensors measure a physical quantity and convert it into a useable signal for your robot.
- We have a huge range of sensors that can be applied to a variety of projects which includes:
  - Environmental monitoring.
  - Distance.
  - Force.
  - Speed.
  - Pressure.
  - Temperature.
  - Magnetic flux.
  - Vibration.
  - Humidity.
  - Rotation.
  - Touch.
  - Imaging.
  - Light.
  - Biometrics.
  - Gas.
  - Acceleration.
  - Current.

Figure 3-58. Environment capturing with common sensors

AIR011.0

### Notes:

#### Environment capturing with common sensors

When designing your robot it is important to choose the correct sensors to enable it to be aware of its environment and perform the tasks required. Consider what it is about your environment you need your robot to know, and your needs in terms of the precision and accuracy required, this will dictate the type of sensors you will need. We have a huge range of sensors that can be applied to a variety of projects which includes: Environmental monitoring, distance, force, speed, pressure, temperature, magnetic flux, vibration, humidity, rotation, touch, imaging, light, biometrics, gas, acceleration, current, voltage, orientation, gravity, tilt sensing and speech recognition.

Be aware of the magnitude of input and output needed and choose a sensor with adjustable sensitivity or varying degrees of freedom if required. The type of environment which might be encountered, and its representation for planning, should direct the choice of sensors in robotic systems. As we see in this section, different sensors provide different types of information, and no sensor is completely accurate.

## CCD cameras (1 of 2)

IBM ICE (Innovation Centre for Education)



- A CCD camera was first used in the late 1960s.
- It captures and stores images in digital memory.
- CCDs are found in photocopiers, security surveillance cameras, fax machines.
- The beauty of CCD cameras is that it provides a low-noise, high quality image at a highly pixelated resolution.

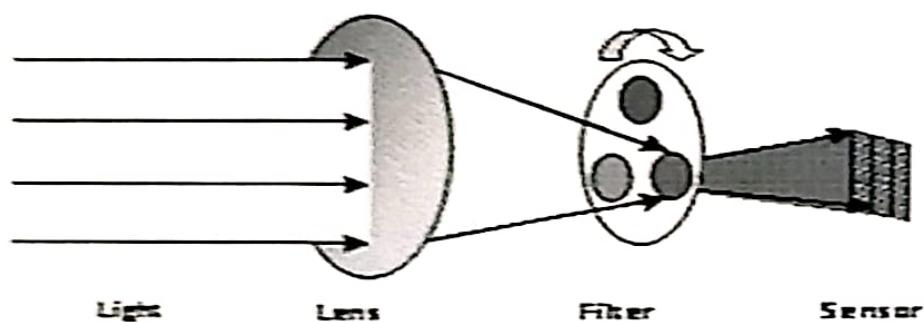


Figure: Image sensor setup to capture blue component.

Figure 3-52. CCD cameras (1 of 2)

AIR011.0

### Notes:

#### CCD cameras

People rely almost exclusively on vision for finding their way around. "CCD cameras are the most obvious electronic parallel and computer vision has been a major research topic for many years. Some impressive results have been obtained using specially built processing hardware, for example guiding vehicles on roadways". The image sensor employed by most digital cameras is a charge coupled device (CCD). A CCD camera was first used in the late 1960s; it captures and stores images in digital memory. The CCD cameras are utilized in both scientific and technology fields. CCDs are found in photocopiers, security surveillance cameras, fax machines and are used in mammography, dentistry X-rays and camcorders as well as handheld cameras. Digital cameras used today contain a CCD image sensor that captures and stores the digital prints. CCD cameras are used in astronomical research fields.

Bell Labs invented the charge couple device in 1969 and it was originally thought to be a new type of computer memory circuit. The CCD is charged by light and has a silicon finish; these devices are extremely sensitive to light. The inventors of the technology, George E. Smith and Willard Boyle determined that because of the CCD's light sensitivity that it would help cameras to capture more clear images because the more light that is collected, the more precise and clear the image will be. Digital cameras were the ideal outlet for CCD image sensors. Rather than using film, as the original cameras did, a digital camera houses a CCD image sensor to collect the light. The light inside the camera is captured and "placed" onto the silicon finish on the sensor. The image sensor converts the light into electrons which then convert the light into digital images.

The beauty of CCD cameras is that it provides a low-noise, high quality image at a highly pixelated resolution. Digital cameras capture and measure light in the hues of blue, green and red; the quality of the image depends on the quality of the camera purchased. Digital cameras are equipped with from one to three CCD arrays which decipher the hues. Generally, we will typically find a camera with one CCD array being deployed for security surveillance and in other uses when precision color images are not necessary.

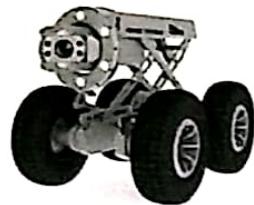
High-tech CCD cameras are utilized in astrophotography and life sciences and this technology is also used on the Hubble Telescope because CCD cameras allow for long exposure times. When a CCD camera is used in a telescope, the eyepiece is removed, and the camera is attached in its place. A CCD camera is capable of capturing up to 70 percent of the available light, as compared to the two percent of available light captured with lower quality cameras. CCD cameras are designed for specific use and are manufactured with C-mount or T-mounts and are also equipped with an adaptor to connect to a computer for ease in displaying the images.

Some cameras use complementary metal oxide semiconductor (CMOS) technology instead. Both CCD and CMOS image sensors convert light into electrons. A simplified way to think about these sensors is to think of a 2-D array of thousands or millions of tiny solar cells. Once the sensor converts the light into electrons, it reads the value (accumulated charge) of each cell in the image. A CCD transports the charge across the chip and reads it at one corner of the array. An analog-to-digital converter (ADC) then turns each pixel's value into a digital value by measuring the amount of charge at each photo-site and converting that measurement to binary form. On the other hand, CMOS devices use several transistors at each pixel to amplify and move the charge using more traditional wires.

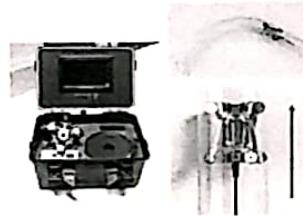
## CCD cameras (2 of 2)



IBM ICE (Innovation Centre for Education)



Figure(a): CCD sensor-based sewer pipe inspection robot camera



Figure(b): CCD Sensor based crawling pipe inspection camera



Figure(c): Outdoor wireless camera



Figure(d): Pipeline Inspection CAM robot camera

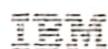
Figure 3-60. CCD cameras (2 of 2)

AIR011.0

### Notes:

#### CCD cameras

CCD cameras are designed for specific use and are manufactured with c-mount or t-mounts and are also equipped with an adaptor to connect to a computer for ease in displaying the images.



## **CCD Vs. CMOS**

- CCD sensors create high-quality, low-noise images. CMOS sensors are generally more susceptible to noise.
- Because each pixel on a CMOS sensor has several transistors located next to it, the light sensitivity of a CMOS chip is lower. Many of the photons hit the transistors instead of the photodiode.
- CMOS sensors traditionally consume little power. CCDs, on the other hand, use a process that consumes lots of power. CCDs consume as much as 100 times more power than an equivalent CMOS sensor.
- CCD sensors have been mass produced for a longer period of time, so they are more mature. They tend to have higher quality pixels, and more of them.

---

Figure 3-61. CCD Vs. CMOS

AIR011.0

### **Notes:**

#### **CCD Vs. CMOS**

CCDs and CMOS image sensors are increasingly used in robotics. Autonomous robots utilize cameras to track objects and identify movement characteristics. Unmanned vehicles that competed in the 2007 DARPA grand challenge used cameras in conjunction with LIDAR to identify objects in the environment in order to safely drive through the course. In the DARPA urban challenge, cameras were used in order to identify curbs and lane lines to keep the unmanned vehicles within lanes as they moved through an urban environment.

CCDs and CMOS image sensors use photo-active regions to read light characteristics in order to generate images. They provide the capability for environmental analysis, navigation, object identification and tracking. CCDs use a lens to direct light onto an array of capacitors which develop a charge that is proportional to the intensity of the incoming light. This charge is then transported across the exposed region and converted to voltage. These voltages can be used to recreate the image.

CMOS image sensors consist of an array of pixels built from transistors and a photodiode. The photodiode transforms the incoming light into a voltage. One transistor is used to reset the pixel for each image acquisition. Another serves as an amplifier so the signal is visible to the processing electronics. A third transistor is used as a switch so that a row of pixel voltages are sent to the processing electronics. Color information is recorded with one of two ways.

One method is to use three sensors, each dedicated to detecting the intensity of one of the three primary colors. The three images are added together to create the final image. The second is to use filters to isolate the primary colors of the incoming light. The sensor makes three acquisitions, one for each primary color, and adds them together for the final image. Robotics applications using both CCD and CMOS image sensors can be implemented with LabVIEW and national instruments products. Virginia tech's team victor tango's entry in the 2007 DARPA urban challenge used LabVIEW to control their vision systems and analysis of image data.

## Sonar sensors (1 of 2)

- The sonar sensors in various robotic applications:
  - Wall Following.
  - Obstacle Avoidance.
  - Positioning.
  - Room Mapping.
  - People Detection.
  - Collision Avoidance.

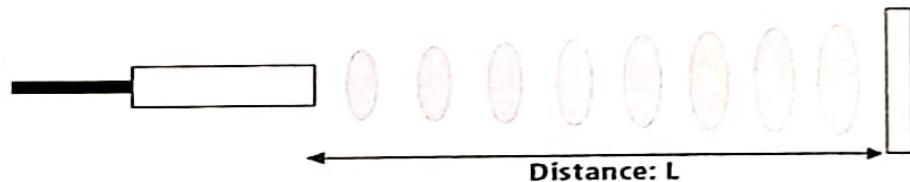


Figure: Sonar sensor technology

Figure 3-62. Sonar sensors (1 of 2)

AIR011.0

### Notes:

#### Sonar sensors

Sensors are the electronic eyes of a robot, and no robot is complete without at least one type of sensor. Ultrasonic sensors are key in robotics for autonomous obstacle avoidance, wall following, and distance sensing. The external sensor informs the robot of its surrounding environment to help establish the robot's position and aid in navigation. It is very much easier to detect range directly, usually at the expense of including mechanical scanning and building a bit of electronics.

One of the simplest ways to determine range is through measuring the time of flight of ultrasound signals, which are reflected by most common objects. Ultrasound signals can be induced through the piezoelectric effect or through electrostatic forces. Most sensors used in robotics are electrostatic since this mechanism is more efficient for coupling into air. Typical frequencies are between 40 and 100 kHz, the higher frequencies being easier to focus but suffering greater attenuation. The same transducer can be used for transmission and reception.

As the name indicates, ultrasonic sensors measure distance by using ultrasonic waves. The sensor head emits an ultrasonic wave and receives the wave reflected back from the target. Ultrasonic Sensors measure the distance to the target by measuring the time between the emission and reception. An optical sensor has a transmitter and receiver, whereas an ultrasonic sensor uses a single ultrasonic element for both emission and reception. In a reflective model ultrasonic sensor, a single oscillator emits and receives ultrasonic waves alternately. This enables miniaturization of the sensor head.

The distance can be calculated using the formula: Distance  $L = 1/2 \times T \times C$ , where  $L$  is the distance,  $T$  is the time between the emission and reception, and  $C$  is the sonic speed. (The value is multiplied by 1/2 because  $T$  is the time for go-and-return distance.)

## Sonar sensors (2 of 2)

IBM

IBM ICE (Innovation Centre for Education)

- The following list shows typical characteristics enabled by the detection system:
  - Transparent object detectable: Since ultrasonic waves can reflect off a glass or liquid surface and return to the sensor head, even transparent targets can be detected.
  - Resistant to mist and dirt: Detection is not affected by accumulation of dust or dirt.
  - Complex shaped objects detectable: Presence detection is stable even for targets such as mesh trays or springs.

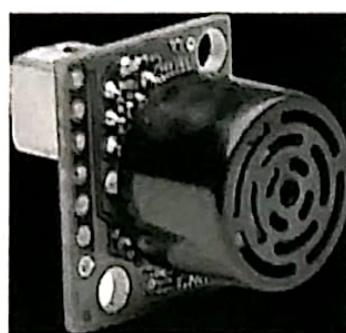


Figure: High performance ultrasonic sensor



Figure: Ultrasonic range finder

Figure 3-63. Sonar sensors (2 of 2)

AIR011.0

### Notes:

#### Sonar sensors

Ultrasonic sensors allow our robots to react to the world around them like we do with our eyes and ears. There are many types of ultrasonic sensors, and they provide your robot with new or improved senses. The ultrasonic sensor provides a great set of eyes for your robot's autonomous navigation. The ultrasonic sensors for robots interface over many communication protocols such as RS232, Pulse Width, Analog Voltage, TTL, and I2C. The high-Performance ultrasonic sensor provide short to long distance detection and ranging with a narrow beam pattern. The outdoor ultrasonic sensors have high power output, noise rejection, auto calibration, temperature compensation, and factory calibrated beam patterns.

## Optoelectronic sensors

- An optoelectronic sensor is a device that produces an electrical signal proportional to the amount of light incident on its active area.
- Integrated optoelectronic sensors are designed to respond to light so that they can recognize things such as patterns, images, motion, intensity, and color.
- Different types of Optoelectronic Sensors:

Optoelectronic Sensor Type	Description
Light-to-voltage converters	Produce a linear output voltage proportional to light intensity
Light-to-frequency converters	Convert light Intensity to digital format for direct connection to a microcontroller or DSP
Ambient light sensors	Measure what the human eye sees
Linear sensor arrays	Measure spatial relationships and light intensity
Color sensors	RGB (red/green/blue) filtered sensors for color discrimination, determination, and measurement
Reflective light sensors	Convert reflective light intensity to a voltage output

Figure 3-64. Optoelectronic sensors

AIR011.0

### Notes:

#### Optoelectronic sensors

The development of applications of new designs of intelligent robots in different environments still has to solve many scientific and technological problems. It concerns the development of appropriate measuring devices and systems to build effective sensory system of robot, the transmission of data for long distances in different environmental conditions, the transmission of high energy (e.g. laser radiation) and many other problems. Many needs of robotics at present and future stages of its development may be accomplished by the integration of optoelectronic systems (e.g. vision systems, lasers and fiber optics) into robotic systems. Some tendencies to solve those problems already seen as well as future prospects for the application of optoelectronics to robotic systems.

The wide beam width of sonar sensors is useful when they are being used as a bumper, or for gathering crude information on the environment. However they have such poor angular resolution that they cannot be used for accurate navigation alone without extensive processing. In addition, as we have seen, they are very noisy, mainly as there are a large number of ultrasound sources in the environment. The problems arising from the long wavelength of sonar are largely removed by using optoelectronic sources such as light emitting diodes and lasers. By definition, an optoelectronic sensor is a device that produces an electrical signal proportional to the amount of light incident on its active area. A number of devices meet this definition, but none is more prevalent than the semiconductor photodiode. Over the years, this two-terminal device has become the mainstay for light sensing.

Integrated optoelectronic sensors are designed to respond to light so that they can recognize things such as patterns, images, motion, intensity, and color. The sensor's ability to perform this recognition (and the complexity of the recognition possible) depends upon the type of integrated optoelectronic sensor. Some of the basic types of integrated optoelectronic sensors currently in use are outlined in table.

All of these types of sensors can be, and are, used in medical equipment applications. They help eliminate human error while providing more accurate readings and faster results. Rather than rely on human judgment to match colors or identify changes in light intensity, the sensors are designed to read or measure light - a real-world signal considered to be very stable and highly accurate - in a reliable, repeatable way. Data from the optoelectronic measurements are fed directly into the computer system, removing another possible source of error. The sensors are noncontact, able to perform their sensing or measurement functions without the need for physical contact with specimens such as blood, urine, or other bodily fluids. This is critical because if the specimens are tainted in any way, the resulting readings and measurements may not be accurate. Current medical applications that use optoelectronic sensors include pulse oximetry, measuring the amount of oxygen in the blood (see "Optical Sensors in Pulse Oximetry"); heart-rate monitors; blood diagnostics, such as blood glucose monitoring; urine analysis; and dental color matching.

Light-to-voltage and light-to-frequency converters serve as the platform on which the other integrated optoelectronic sensors are built, e.g., the IR optoelectronic sensors currently used in pulse oximetry systems and personal heart-rate monitors. These types of optoelectronic sensors integrate other functions such as current-to-voltage conversion, amplification, and A/D conversion, which results in smaller, less costly, and more reliable diagnostic systems. Integrated optoelectronic sensors have helped establish pulse oximetry as a viable medical procedure. Prior to pulse oximetry, medical professionals relied on blood samples to determine blood oxygen content, but these measurements could not give real-time results. Early pulse oximetry systems were large, bulky, and expensive, costing approximately \$10,000. With the advent of integrated optoelectronic sensors and better LEDs, modern pulse oximetry systems became possible, enabling real-time, accurate, and noninvasive measurements of blood oxygen content. The use of integrated optoelectronic sensors in personal heart-rate monitors has resulted in similar benefits.

Light-to-voltage and light-to-frequency converters also serve as the platforms for color sensors, which are currently used in blood glucose monitors designed for home use. Blood glucose monitors (see "Integrated color sensors in blood glucose meters") presently use one of two different methods:

- The optoelectronic sensor method, in which a drop of blood on a special test strip is read by an optoelectronic sensor, which measures a color change or the reflectance of a particular wavelength of light to determine blood glucose levels. The electrochemical method, in which a drop of blood on a special test strip provides an electro-resistive measurement to determine glucose levels.
- Of the two approaches, the optoelectronic sensor method offers several benefits: The test strips tend to be cheaper to manufacture; testing requires extremely low specimen volumes, leading to less pain for the patient; the method is less sensitive to the presence of prescription and over-the-counter drugs in the patient's system; and compared to the electrochemical method, its accuracy is less affected by the number of red blood cells present.

The accuracy and speedy results enabled by integrated color sensors also make them the optoelectronic sensors of choice for urine analysis and dental color matching applications. They allow medical professionals to automate analyses that used to rely on human judgments. Prior to optoelectronic sensors, urine analysis involved dipping a test strip into a specimen and trying to match the resulting color to a chart; dental color matching consisted basically of eyeballing color strips.

The combination of small size, sophisticated functionality, solid-state robustness, and low-power operation makes integrated optoelectronic sensors a natural choice in medical equipment design. The ability to create smaller, portable, noninvasive test equipment that provides quick, accurate results has led to the development of pulse oximeters, personal heart-rate monitors, and blood glucose meters. Integrated optoelectronic sensors have provided medical professionals with a quick, accurate procedure for analyzing urine specimens, as well as helping eliminate the need to look at color strips in dental color matching.

## Sensor integration

- The use of multiple, physically different sensors can help to overcome the shortcomings of each individual device.
- Integration of redundant data from multiple sensors can minimize the uncertainty.
- Typical methods for integrating data from multiple sensors can be categorized as follows:
  - Qualitative approach.
  - Quantitative approach.

Figure 3-65. Sensor integration

AIR011.0

### Notes:

#### Sensor integration

No matter how good a sensor is or how efficient the signal processing algorithms are, uncertainty in sensor data cannot be eliminated totally. The use of multiple, physically different sensors can help to overcome the shortcomings of each individual device. Further, the integration of redundant data from multiple sensors can reduce uncertainty especially if an individual sensor fails.

Typical methods for integrating data from multiple sensors can be categorized as follows:

- **Qualitative approach:** Symbolic descriptions of sensory information are used, such as rule bases.
- **Quantitative approach:** Using numerical methods which can incorporate uncertainty, for example through using probabilities.

Both methods are widely used in robotics research and each has areas of application. However, there are problems with each too: rule bases provide only qualitative information whereas statistical methods may rely on accurate models which are not always possible to formulate. There is a need to integrate both methods to achieve robustness in a changing real world.



IBM ICE (Innovation Centre for Education)

## Qualitative approaches (1 of 2)

- Qualitative approaches use powerful representation mechanisms to describe complex sensor information and heuristics to guide data interpretation.
- Rule bases are convenient when the statements are in the form 'if., then'.
- The rule base provided information to guide the robot as it followed a predetermined path in a factory with a number of allowable pathways.
- It provided an estimate of the following states of the robot's current path:
  - The path was clear.
  - It was blocked by a small obstacle (such that a side-step maneuver past it can be performed within the pathway).
  - It was blocked by a large obstacle (which entirely blocks the pathway, requiring the robot to backtrack and find a new pathway).

Figure 3-66. Qualitative approaches (1 of 2)

AIR011.0

### Notes:

#### Qualitative approaches

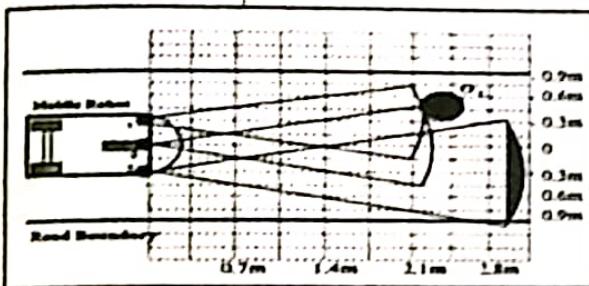
They are widely used when representation is important and quantification is difficult. They are typified by rule bases, such as expert systems. Rule bases are convenient when a problem can clearly be described in a few statements of the form 'if., then'. They allow us to build expert information into a system. However they are limited: they cannot handle numbers easily, they become complex very quickly and they do not incorporate uncertainty with any exactness. A simple example of a rule base was developed by Flynn to combine sonar and optical amplitude data to guide a mobile robot. This is a simple, typical example of exploiting the complementary characteristics of the two types of sensor. The sonar was used to provide range information and the near-infrared sensor localized doorways and edges. The integration of data from both sensors was conducted according to simple rules.

Although the rules allow a mobile robot to build a reasonably detailed map of the environment, there are limitations. For example specularity in the sonar sensor and sensitivity to different colors for the optical amplitude sensor are not taken into account. Therefore, there must be considerable risk associated with any robot actions based on this map. Simple rule bases can also be used to integrate data spatially from a single type of sensor. Hu, Brady and Probert developed rules to interpret data from a three sonar array. The array was to detect unexpected obstacles in the path of a robot in an industrial environment. Using the three sonars together provides more detailed information on the location of obstacles than using three separate readings.

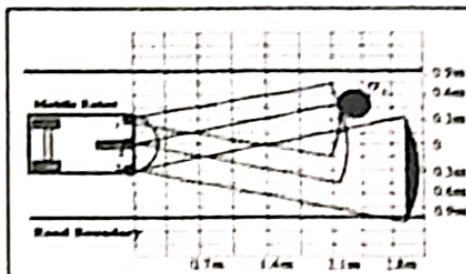
## Qualitative approaches (2 of 2)

IBM

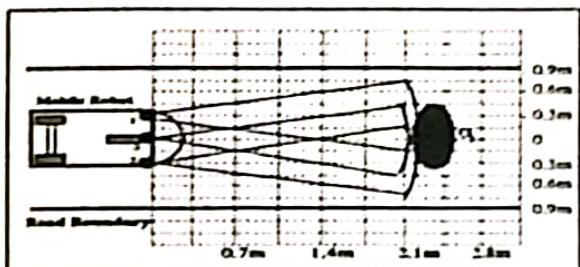
IBM ICE (Innovation Centre for Education)



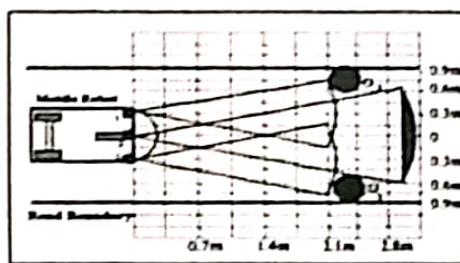
Figure(a): An obstacle appears at the left side



Figure(b): An obstacle appears at the right side



Figure(c): An obstacle appears in the middle



Figure(d): An obstacle appears on both sides

Figure 3-67. Qualitative approaches (2 of 2)

AIR011.0

### Notes:

#### Qualitative approaches

The rule base provided information to guide the robot as it followed a predetermined path in a factory with a number of allowable pathways. It provided an estimate of the following states of the robot's current path:

- The path was clear.
- It was blocked by a small obstacle (such that a side-step man oeuvre past it can be performed within the pathway).
- It was blocked by a large obstacle (which entirely blocks the pathway, requiring the robot to backtrack and find a new pathway).

The rules for interpretation of the data from the three sonars are described as follows:

- Rule 1: If all of three sonars see a clear path, then the predetermined path is clear, and no action should be taken.
- Rule 2: If the middle sonar sees nothing in the cone of its acoustic beam, then the predetermined path is clear even though the two side sonars may see obstacles, as shown in Figure(d).
- Rule 3: If any one of both side sonars sees clearance in the prediction zone and the road width Wr is at least twice the robot width, then even though the other two sonars may see an obstacle, there is room for a sidestep man oeuvre (i.e. blocked by a small obstacle) as shown in Figures (a) and (b).

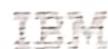
- Rule 4: If all three sonars see something, then the path is impassable (i.e. blocked by a large obstacle), as shown in Figure (c).

Qualitative approaches have been presented elsewhere. "A robotic system presented by Allen the integration of vision and touch for object recognition tasks. Vision and touch together provide geometric measures of the surfaces and features that are used in a matching phase to find model objects that are consistent with the sensory data". "Stansfield described a robotic perceptual system also utilizing vision and touch, but in this case using an active touch sensor. A two-stage exploration is implemented. Vision is first used in a feed-forward manner to segment the object and to obtain its position. Touch is then used in a feedback mode to further explore the object". "Shekar et al. described how to localize an object using a manipulator end effector that has been instrumented with both centroid and matrix touch sensors".

Qualitative approaches to the data fusion problem are gaining popularity in robotics because they offer the possibility of capturing the more human-like aspects of decision making. However, more work is needed to cope with the complexity and uncertainty inherent in the multi-sensor systems typical of robotics.

## **Quantitative approaches**

---



IBM ICE (Innovation Centre for Education)

- The quantitative approach to sensor integration requires numbers.
- The information obtained from sensors is quantified so that statistical and decision theoretic methods can be adopted.
- Through quantitative analysis and mathematical modelling, considerable success has been demonstrated with such approaches.
- We describe two approaches in this section:
  - Bayes statistics.
  - The Kalman filter.

---

Figure 3-68. Quantitative approaches

AIR011.0

### **Notes:**

#### **Quantitative approaches**

The problem with rule based approaches is that they do not handle quantitative information, such as numerical uncertainty on a sensor reading. Although bounds could be placed on uncertainty, only a few categories (e.g. 'small, medium, large') can be used, otherwise the rule base gets much too large for high bandwidth sensing. Since in a number of sensors, such as the optoelectronic sensors, the uncertainty of any measurement can be expressed numerically in terms of a variance, we would expect to improve the reliability of data interpretation through using this information.

## Bayes statistics

IBM

IBM ICE (Innovation Centre for Education)

- Bayes statistics, which use well established mathematical reasoning, can be used to integrate information from various sources (for example from multiple sensors or over time).
- Unlike the simple sonar rule base we introduced earlier, the Bayes formulation distinguishes between the measurement of a quantity and the true state of the quantity.
- To account for uncertainty, it reasons using probability density functions.
- The Bayes update rule, which updates an existing hypothesis on a quantity with new information:

$$P(O|Z) = \frac{P(Z|O)P(O)}{\sum P(Z|\bar{O})\bar{P}(O)}$$

- Where O is the true state of the system and
- Z is the possibly incorrect observation of the state.

Figure 3-69. Bayes statistics

AIR011.0

### Notes:

#### Bayes statistics

In more detail:

- $P(O|Z)$  is the probability density function of O following an observation Z (the posterior density function).
- $P(Z|O)$  has been made. It accounts for errors in the method of observation.
- $P(O)$  is the prior information about the state O.
- The summation on the bottom line simply normalizes the result.

Bayes statistics therefore allow us to combine measurement information with prior knowledge, the beliefs already held about a situation. Sometimes, however, we may have no prior information. For a single reading, this is a serious problem. However if we are taking a number of readings, then we can use Bayes theorem iteratively as follows:

- At time T, start with some arbitrary prior distribution over all states:  $P(O)$  (for example assuming that all are equally likely).
- Take a measurement Z. Then assuming we know the likelihood function  $P(Z|O)$  (we talk about this in a moment) use equation 2 to determine the new distribution  $P(O|Z)$ .
- Make this the new prior distribution and continue.

Convergence normally occurs over a few cycles even for a grossly incorrect choice of the prior. Of course if the environment changes suddenly, the new estimates of state will not keep up and there will always be a lag in the estimation.

Hu, Brady and Probert demonstrate the use of Bayes statistics to improve predictions made with the rule base described in the previous section for the three sonar array. Recall that the vehicle is travelling along a predetermined path and has to detect and plan a path in the presence of unexpected obstacles. For now, assume that the obstacles are static and that their distribution is random. As we describe above, if an obstacle is detected the path may consist of two states:

$O = (O_1, O_2) = (\text{passable}, \text{impassable})$  corresponding to small and large obstacles being present.

The Bayes update rule uses the likelihood function  $P(O/Z)$  to relate the measurement of state to its true value, and the value of this function determines the relative weightings given to the measurement and the prior information in the update rule. Incorrect values will either make a system unresponsive or over affected by noise. Generally, the sonar readings are corrupted by spiky noise and hence significant errors would be expected. Unless a clear physical model can be established, the only way to determine the likelihood function is through taking a large number of readings in known typical environments and analyzing the success and failure rates over the states of interest.

Measurements show that the probability of missing an object which is present increases as the robot approaches the obstacle closely. This is because there is often no reflection from the object surface itself as it is specular. However there is always a return from an edge, so, at longer distances when the edges are in sight more often the object is more likely to be detected. In contrast, the probability of reporting a non-existent object increases for distant obstacles. This results from the greater effect of environmental ultrasound sources when the signal is low. Unfortunately this means that as the robot approaches an object the sonar sensors are less and less likely to see it – an undesirable property for safe sensing.

Following an estimate of state, the robot must decide what to do. It has two possibilities once an obstacle has been detected: to sidestep or to backtrack. We define formally the robot action space:

$A = (a_1, a_2) = (\text{sidestep}, \text{backtrack})$

In the rule base described in the last section, action and measurement were linked implicitly. However this is incorrect as the action should depend on the state, not the measurement. Now that we have established a notion of the true state, we can set up the correct dependency structure. We also use our knowledge of the uncertainty of state to improve our chance of taking the best action. We define a loss function which quantifies the cost of the action in some common currency such as time or money. For example the loss of performing a side-step if the path is passable is zero, whereas the loss if the path is impassable may be very high, either because the robot collides with something or because it has lost time in trying to pursue an action which is doomed to failure. We define a loss function over all states and actions:

$L(O, a) \geq 0$ .

The Bayes risk then allows us to evaluate the expected loss of any action  $a_i$ . The Bayes risk of taking action  $a_i$  is the expected loss incurred through taking that action, summed over all states:

$$B(a_i) = \sum_i L(O_i, a_i) P(O_i)$$

where  $P(O_i)$  is the probability of  $O$  being in state  $i$  and  $L(O_i, a_i)$  the loss associated with action  $a_i$  in state  $O_i$ . The robot would normally take the action which minimized the Bayes risk.

The Bayes update rule allows the robot to build up a gradual picture of its environment and to build on past knowledge. The use of the prior knowledge and temporal integration prevented possible collision from individual incorrect readings. The Bayes risk function allows informed planning decisions to be made, which assess the costs or benefits of various options open to the robot under uncertainty in the state.

## Kalman filter



IBM ICE (Innovation Centre for Education)

- The filter uses two stages:
  - The previous state and a model of how it changes between iterations.
  - An observation of the new state.
- Therefore, like the Bayes update rule, it combines both prior expectation with measurements.
- Kalman filters are used too for spatial integration of data.
- The advantage of this scheme over Flynn's is that the map includes information on the variance as well as the expectation of each state variable, so the risk of plans based on this map can be quantified.

Figure 3-70. Kalman filter

AIR011.0

### Notes:

#### Kalman filter

The Kalman filter, or its non-linear form, the extended Kalman filter, offers an alternative approach to quantitative sensor integration for continuous, rather than discrete, measurement data. The filter uses two stages, each of which predicts an updated state: one based on the previous state and a model of how it changes between iterations, and one from an observation of the new state. Therefore, like the Bayes update rule, it combines both prior expectation with measurements (and indeed is an implementation of the same statistical models). In the linear case, the Kalman filter is not very fussy about how the model is set up, but in the non-linear case, the extended Kalman filter can be very sensitive both to the model and to good estimations of variance. For example, for sonar sensors, abstraction of data to a higher representation than range is needed to build a successful model and to eliminate systematic errors, such as specularity.

Kalman filters are used too for spatial integration of data, for example to determine line segments from individual range points. They are particularly well suited to optoelectronic sensors since the variance measured range point can be estimated through the signal amplitude. Kalman filters can be used to integrate data from several sensors. It is often difficult to use them meaningfully to integrate data from different types of sensor, as it is not possible to provide suitable models of each sensor or to find an appropriate common representation of the state. In a system which integrates sonar and optical amplitude detectors (almost identical sensors to those used in Flynn's rule base) Wen has shown how it is best to run Kalman filters independently for each sensor prior to sensor integration.

## Machine vision system

- Because humans are good at vision, the task of implementing vision artificially is often underestimated.
- Even without any financial constraints, it is still not possible to artificially implement vision comparable to that of humans.
- Fortunately, this is not required or even desirable for industrial sensors, since the sensing problem can invariably be greatly reduced in complexity by applying a priori knowledge.
- An area of research, commonly referred to as industrial machine vision, is concerned with the development of visual systems for use in constrained industrial environments.

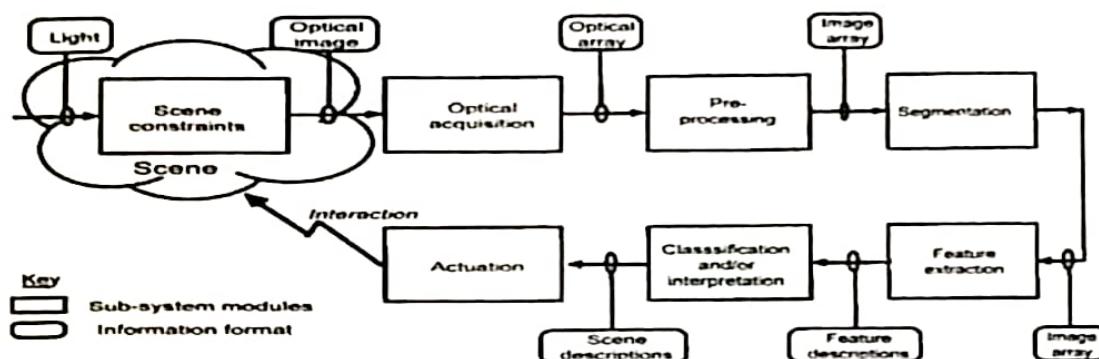


Figure: A generic model of a machine vision system

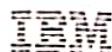
Figure 3-71. Machine vision system

AIR011.0

### Notes:

#### A generic model of a machine vision system

As with any system, a machine vision system can be decomposed into a number of functional modules. This approach has two main advantages. Firstly, each of these modules can be verified independently, simplifying system design and validation. Secondly, the data are refined from stage to stage, which reduces the computational load and memory requirements. A generic model of a machine vision sensing system has been proposed, and is presented pictorially in Figure. This model is very simplistic, since the complexity of the various modules in the model is dependent on the specific requirements of the application, and the boundaries between modules are often hard to define precisely. However, thorough analysis of each module's requirements during the design stage allows precise specification of the whole sensing system.



IBM ICE (Innovation Centre for Education)

## Phases of a machine vision system (1 of 2)

- The task of the machine vision system is to classify an object into a correct class based on the measurements about the object.
- The possible classes are usually well-defined already before the design of the machine vision system.
- Many machine vision systems can be thought to consist of five stages:
  - Sensing.
  - Pre-processing and segmentation.
  - Segmentation.
  - Classification.
  - Post-processing.

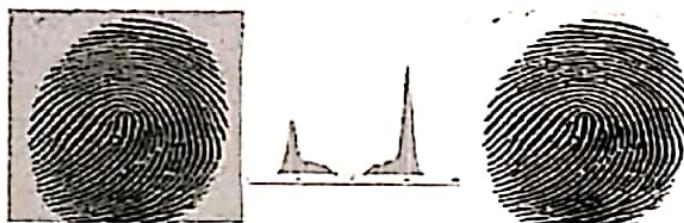


Figure: An image with bi-modal histogram

---

Figure 3-72. Phases of a machine vision system (1 of 2)

AIR011.0

### Notes:

#### Phases of a machine vision system

Many machine vision systems can be thought to consist of five stages:

##### Sensing:

Sensing refers to some measurement or observation about the object to be classified. For example, the data can consist of sounds or images and sensing equipment can be a microphone array or a camera. Often one measurement (e.g. image) includes information about several objects to be classified. For instance, assume that we want to recognize the address written on the envelope. We must then classify several characters to recognize the whole address. The data here is probably an image of the envelope including the sub-images of all the characters to be classified and some background that has nothing to do with the pattern recognition task.

##### Pre-processing:

Pre-processing refers to filtering the raw data for noise suppression and other operations performed on the raw data to improve its quality. Although the imposition of scene constraints at the image acquisition stage greatly simplifies the processing required, some pre-processing is often required to minimise the complexity of the segmentation and feature extraction stages.

In particular, pre-processing performs three basic functions:

- Increasing the contrast between homogenous regions in the image to aid segmentation.
- Compensation for any distortion in the image caused by the optics.
- Removal of noise from the image.

If the illumination and acquisition of the image have been carefully considered, then the first two of these will be fairly trivial. Algorithms for contrast stretching, image sharpening and inverse perspective transformation are widely accepted and documented. Random noise may be efficiently removed from images, without much degradation, using techniques such as median filtering and frame averaging.

**Segmentation:**

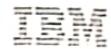
The process of feature extraction is greatly simplified if the image is segmented into meaningful regions. Segmentation is vital if a machine vision system is to be robust to noise and the presence of unexpected objects in the scene. There are two basic approaches to segmentation: region growing and edge based segmentation. Region growing techniques search the image and attempt to group together pixels that are in some way similar. Region growing is commonly used to segment complex images, such as natural scenes, but is rarely used in machine vision applications, due to its computational intensity.

Edge based segmentation techniques offer a more efficient means of segmenting images of constrained industrial scenes, which are characterised by more clearly defined regions. Edges in grey-scale images can be detected by applying a first or second order discrete difference filter. The position, strength and orientation of the individual edge points can then be used to generate a complete description of the boundaries between regions. The task of edge detection is greatly simplified if the number of grey levels in the image is reduced to two. Figure shows the distribution of grey-scale values in an image that has been acquired from the scene of a back-lit object. The histogram is essentially bimodal, and a simple threshold can be applied to reduce the image to two grey levels.

"A method for automatically selecting a suitable threshold from the distribution of grey scale values has been developed by Otsu. The use of a dynamically determined threshold, as opposed to a manually entered constant, allows the system to adapt to different levels of lighting, affording it extra robustness". Binary images require less storage space, and can be processed with much simpler algorithms. However, complete segmentation cannot be achieved by thresholding alone. In a simple case, an image may comprise a single object and a background, but in reality, this is rarely the case. If the sensing system is to be resilient to noise and tolerant to the presence of unexpected objects in the image, then the image must be further segmented into connected components before any feature extraction is attempted.

The image intensity histogram itself conveys information useful during the setup of the sensing system. For example, the lens aperture could be adjusted (either manually or automatically) until the histogram consists of two sufficiently distinct peaks. In segmentation, the measurement data is partitioned so that each part represents exactly one object to be classified. For example in address recognition, an image of the whole address needs to be divided to images representing just one character. The result of the segmentation can be represented as a vector that is called a pattern vector.

## Phases of a machine vision system (2 of 2)



IBM ICE (Innovation Centre for Education)

### Feature Extraction:

- Some of the well-known boundary descriptors where we describe region borders expressed in some mathematical form are as follows:
  - Chain codes.
  - Shape numbers
  - Geometric representations.
  - Boundary length.
  - Curvature
  - Bending energy.
  - Signature.
  - Chord distribution.
  - Fourier descriptors.
  - Segment sequence-based representations.
  - Polygonal representation.
  - B-spline representation.
  - Other representation.
  - Hough transforms.
  - Mathematical morphology.
  - Neural networks.

### Post-processing:

- The final task of the machine vision system is to decide upon an action based on the classification result.

Figure 3-73. Phases of a machine vision system (2 of 2)

AIR011.0

### Notes:

#### Feature extraction

Especially when dealing with pictorial information the amount of data per one object can be huge. A high-resolution facial photograph (for face recognition) can contain  $1024 \times 1024$  pixels. The pattern vectors have then over a 6 million components. The most part of this data is useless for classification. In feature extraction, we are searching for the features that best characterize the data for classification. The result of the feature extraction stage is called a feature vector. The space of all possible feature vectors is called the feature space.

There are two main ways of representing a region and thereby extracting features from such regions: - external characteristics and internal characteristics. In the case of external characteristics, a region may be represented by:

- Its boundary with the boundary described by features such as its length
- The orientation of the straight line joining the extreme points
- The number of concavities in the boundary.

An external representation is chosen when the primary focus is on shape characteristics. On the other hand, an internal representation is selected when the primary focus is on reflectivity properties, such as colour and texture. In either case, the features selected as descriptors should be as insensitive as possible to variations such as change in size, translation and rotation. Some of the well-known boundary descriptors where we describe region borders expressed in some mathematical form are as follows:

**Chain codes:**

Chain codes are used to represent a boundary. Chain code uses a logically connected sequence of straight-line segments. The line segments specify length and direction. The direction is coded using a number scheme based on 4-direction or 8-direction.

**Shape numbers:**

The boundary can be represented by the differences in the successive directions in the chain code instead of representing the boundary by relative directions. This can be computed by subtracting each element of the chain code from the previous one and taking the result modulo n, where n is the connectivity. After these operations, a rotationally invariant chain code is obtained by a cyclic permutation which produces the smallest number. Such a normalized differential chain code is called the shape number. Chain code derived in this way is not scale invariant. Although it is possible to scale two similar shapes into the same size, the resulted shape numbers can have a different number of digits, making it impractical to do matching between two shapes. In simple terms, the Shape number is another simple boundary descriptor and it is obtained by computing the chain code difference and we re-order this to create the minimum integer.

**Signature:**

A shape signature represents a shape by a one-dimensional function derived from shape boundary points, i.e., we represent 2-D object boundary in term of a 1-D function of radial distance with respect to centroid of a boundary. The procedure is as follows:

- Find the centroid of an object boundary.
- Divide the boundary of an object into 'm' number of intervals (angles).
- Find the distance from the centroid to the boundary at the specified angle.

Shape signatures are usually normalized into being translation and scale invariant. In order to compensate for orientation changes, shift matching is needed to find the best matching between two shapes. Most of the signature matching is normalized to shift matching in 1-D space. Shape signatures are sensitive to noise, and slight changes in the boundary can cause large errors in matching.

**Fourier descriptors:**

Spectral descriptors overcome the problem of noise sensitivity and boundary variations by analysing shape in spectral domain. Spectral descriptors include Fourier descriptor (FD) and wavelet descriptor (WD), they are derived from spectral transforms on 1-D shape signatures. One of the most widely used shape description methods is FD. Fourier descriptor is developed based on well-known and well-understood Fourier theory. The advantages of FD over many other shape descriptors are as follows:

- Simple to compute.
- Each descriptor has specific physical meaning.
- Simple to do normalization, making shape matching a simple task.
- Captures both global and local features.

With sufficient features for selection, FD overcomes the weak discrimination ability of those simple global descriptors. FD also overcomes the noise sensitivity and difficult normalization in the shape signature representations. The computation of FD is performed by considering coordinate pairs of points encountered in traversing an N-point boundary in the xy plane are recorded as a sequence of complex numbers. An N-point DFT is performed to the sequence and the complex coefficients obtained are called the Fourier descriptors of the boundary. In general, only the first few coefficients are of significant magnitude and are good enough to describe the general shape of the boundary. Fourier descriptors are not directly insensitive to geometrical changes such as translation, rotation and scale changes, but the changes can be related to simple transformations on the descriptors.

Some of the regional descriptors include area of the region and compactness. The area of a region is defined as the number of pixels contained within its boundary. The perimeter of a region is the length of its boundary. Descriptors providing measures of properties such as smoothness, coarseness and regularity are used to quantify the texture content of an object.

Statistical approaches yield characterizations of textures as smooth, coarse, grainy, and so on. In many statistical pattern recognition applications, we use standard statistical features to describe the image under processing. Some of the common statistical features include mean, variance, coefficient of variance, skewness, kurtosis, uniformity and entropy.

#### **Projection profiles:**

A useful region-based signature is the profile or projection. Projection profile of an image in a particular direction refers to the running sum of the pixels in that direction. The vertical profile is the number of pixels in the region in each column. The horizontal profile is the number of pixels in the region in each row. One can also define diagonal profiles, which count the number of pixels on each diagonal. Profiles have been used in character recognition applications, where an L and a T have very different horizontal profiles (but identical vertical profiles), or where A and H have different vertical profiles (but identical horizontal profiles). Boundary moments can be used to reduce the dimensions of the boundary representation. The advantage of boundary moment descriptors is that it is easy to implement. However, it is difficult to associate higher order moments with physical interpretation. Using nonlinear combinations of the lower order moments, a set of moment invariants (usually called geometric moment), which has the desirable properties of being invariant under translation, scaling and rotation, are derived.

"Moment shape descriptors are usually concise, robust, easy to compute and match. The disadvantage of moment methods is that it is difficult to correlate high order moments with the shape's physical features. Among the many moment shape descriptors, Zernike moments are the most desirable for shape description. Due to the incorporation of a sinusoid function into the kernel, they have similar properties of spectral features". Moments is a method of estimation of population parameters such as mean, variance, median, etc.

#### **Principal component analysis:**

Some of the well known region-based representation technique use Principal Component Analysis (PCA) for image description. It is one of the powerful image description technique and used in many computer vision applications. PCA is a method to study the structure of the data, with emphasis on determining the patterns of covariance among variables. Thus, PCA is the study of the structure of the variance-covariance matrix. Given a set of p variables ( $X_1, \dots, X_p$ ), PCA calculates a set of p linear combinations of the variables ( $PC_1, \dots, PC_p$ ) such that:

- The total variation in the new set of variables or principal components is the same as in the original variables.
- The first PC contains the most variance possible, e.g. as much variance as can be captured in a single axis.
- The second PC is orthogonal to the first one (their correlation is 0) and contains as much of the remaining variance as possible.
- The third PC is orthogonal to all previous PC's and also contains the most variance possible.

The main idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The procedure is as follows:

- Mean center the data (optional).
- Compute the covariance matrix of the dimensions.
- Find eigenvectors of covariance matrix.
- Sort eigenvectors in decreasing order of eigenvalues.
- Project onto eigenvectors in order.

"Local binary pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. Due to its discriminative power and computational simplicity, LBP texture operator has become a popular approach in various applications. It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination

variations. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings".

The procedure to obtain LBP feature vector is as follows:

- Given an image, compute the descriptor of each pixel considering its 8-neighbours or more.
- For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise. Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).
- Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller, and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.
- Optionally normalize the histogram.
- Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.

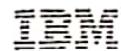
The feature vector can now be processed using the machine-learning algorithm to classify images. Such classifiers can be used for face recognition or texture analysis. Feature extraction is highly application specific although some general techniques for feature extraction and selection have been developed. In general, the line between feature extraction and classification is fuzzy. The task of the feature extractor is to produce a representation about the data that enables an easy classification. On the other hand, the task of the classifier is to produce the best classification accuracy given the extracted features. Clearly, these two stages are interdependent from the application point of view. Also, and perhaps more importantly, what is the best representation for the data depends on the classifier applied. There is no such thing as the universally optimal features.

The classifier takes as an input the feature vector extracted from the object to be classified. It places then the feature vector (i.e. the object) to class that is the most appropriate one. In address recognition, the classifier receives the features extracted from the sub-image containing just one character and places it to one of the following classes: 'A', 'B', 'C'..., '0', '1', ..., '9'. The classifier can be thought as a mapping from the feature space to the set of possible classes. Note that the classifier cannot distinguish between two objects with the same feature vector. The abstraction offered by the concept of the feature vector makes it possible to develop a general, application-independent theory for the design of classifiers. Therefore, it is possible to use the same underlying principles when designing classifiers for address recognition systems and bottle recycling machines.

#### **Post-processing:**

A machine vision system rarely exists in a vacuum. The final task of the machine vision system is to decide upon an action based on the classification result(s). A simple example is a bottle recycling machine, which places bottles and cans to correct boxes for further processing. Different actions can have also different costs associated with them. This information can be included to the classifier design. Also, we can have several interdependent classification results in our hands. For example in an address recognition task, we have the classification results for multiple characters and the task is to decide upon the address that these characters form.

In summary, while designing a machine vision based robotic system, a challenging sensing and control problem is to be addressed. Thorough analysis of the requirements and the relative merits of a number of sensing techniques has found machine vision to have the adaptivity necessary to overcome the difficulties encountered by traditional sensors. The application of a priori knowledge and adoption of the generic model of machine vision has allowed the development of sensors that are near optimal in terms of robustness and processing performance. Once a similar technique has been established to allow the integration of visual sensory data with data from traditional sensors, a complete sensing and control system can be developed and evaluated.



## Tool condition monitoring systems

- The following four essential components namely:
  - Sensing technique systems.
  - Feature extraction systems.
  - Decision making systems.
  - Knowledge learning systems.
- Any automated tool condition monitoring system has to emulate the human monitoring action.
- The major goals for tool condition monitoring are to develop self-adjusting and integrated monitoring systems able to function under various working conditions with minimum operator supervision.
- Basically, a monitoring process has two parts:
  - Sensing.
  - Monitoring.

Figure 3-74. Tool condition monitoring systems

AIR011.0

### Notes:

#### Tool condition monitoring systems

Condition monitoring and diagnosis systems capable of identifying machining system defects and their location are essential for unmanned machining. Unattended (or minimally manned) machining would result in increased capital equipment utilization, thus substantially reducing the manufacturing costs. Increased demands for even higher product quality, reliability, and manufacturing efficiency levels have imposed stringent requirements on automated product measurement and evaluation. Manufactured products of the modern day command ever-higher precision and accuracy, therefore automated process monitoring becomes crucial in successfully maintaining high quality production at low cost.

The automated tool condition monitoring processes imply the identification of cutting tool condition without interrupting the manufacturing process operation, under minimum human supervision. Unattended or minimally manned machining leads to increased capital equipment utilization, thus substantially reducing the manufacturing costs. Both these situations require intelligent sensor systems. An Intelligent Sensor System was defined as an integrated system consisting of sensing elements, signal conditioning devices, signal processing algorithms, and signal interpretation and decision-making procedures. In the absence of a human operator, the system should sense signals indicating the process status and its changes, interpret incoming sensed information, and decide on the appropriate control action. A system could be defined as Automated/Intelligent Monitoring System if sensing, analyzing, knowledge learning, and error correction abilities, essential to machining tool condition monitoring, are incorporated.

An automated/intelligent machining process and tool condition monitoring system should be able to emulate as closely as possible the abilities of human operators. Thus, the following four essential components namely sensing technique systems, feature extraction systems, decision making systems and knowledge learning systems have to be included in any automated tool condition monitoring system to emulate the human monitoring action. "The key concerns with both signal processing and decision-making algorithms, jointly known as monitoring methods, include reliable and fast identification or response to an abnormal event occurring at normal process conditions".

The major goals for tool condition monitoring are to develop self-adjusting and integrated monitoring systems able to function under various working conditions with minimum operator supervision. The purpose of automated tool condition monitoring in machining is to relate the process signals to the tool conditions and detect or predict tool failure. Automated tool condition monitoring implies identifying the characteristic changes of the machining process based on the evaluation of process signatures without interrupting normal operations. Basically, a monitoring process has two parts:

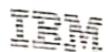
- **Sensing:** obtaining cutting process signals from sensors. Appropriate signals used for tool condition monitoring are force, torque, vibration, temperature, acoustic emission, electric current, etc.
- **Monitoring:** composed of signal processing and decision making, can be divided into model-based and feature-based methods. Both methods use sensor signals from the cutting process for the system input.

Any automated machining process and tool condition monitoring systems should include:

- **Multi-sensor system:** More than one sensor should be used for monitoring machining processes and tool conditions, yielding an extended survey of sensitive features.
- **Automated feature extracting systems:** Automatically generate monitoring features through learning. The signals sensed from multiple sensors are analyzed, compacted, and selected by the system to yield the most sensitive features to the monitoring subjects. The extracted features are also further refined or reselected by the monitoring system.
- **Learning and decision-making systems:** Build up flexible and comprehensive monitoring strategies and automatically generate control parameters. The concentrated information from the learning procedure is stored in the system for classification purposes and can be modified by knowledge updating procedures. With increasing experience, the system will become more and more reliable and promote the monitoring/control functions. These strategies should be robust and valid for a reasonable range of cutting conditions.

Significant work performed in this field focused on analytical forecast, dynamic structure identification, monitoring techniques, and adaptive control approaches. Thonshoff et.al, 1998, identified five monitoring tasks: machine, tool, process, tool condition, and work piece; the author described the monitored conditions in machining processes and classifies the monitored functions into two groups: time critical and non-time critical. The former requires a system response within a range of milliseconds while the later may take seconds or even minutes.

## Neural networks for tool condition monitoring systems



IBM ICE (Innovation Centre for Education)

- Neural network learning methods provide a robust approach to approximating real-valued, discrete-valued, and vector-valued target functions.
- Neural network learning methods provide a robust approach to approximating real-valued, discrete-valued, and vector-valued target functions.
- Appropriate problems for neural network learning.
- The training examples may contain errors:
  - Long training times are acceptable.
  - Fast evaluation of the learned target function may be required.
  - The ability of humans to understand the learned target function is not important.

Figure 3-75. Neural networks for tool condition monitoring systems

AIR0110

### Notes:

#### Neural networks for tool condition monitoring systems

Neural networks are computing systems made up of a number of simple, highly interconnected processing elements that provide the system with the capability of self-learning. Using neural networks, simple classification algorithms can be used, and the system parameters are easily modified. One major characteristic of building neural networks is the training time. Training times are typically longer when complex decision regions are required and when networks have more hidden layers. As with other classifiers, the training time is reduced and the performance improved if the size of a network is optimally tailored. The tasks of an automated tool condition monitoring system involve the ability to recognize the tool condition by analyzing measured cutting process parameters such as forces and vibrations.

This ability is based on the accumulation of useful information from related laws of physics and operators' experiences. In building automated/intelligent tool condition monitoring systems, some basic functions have to be considered:

- Fusion of multiple sensors.
- Learning or training strategies for the monitoring system.
- Knowledge updating techniques.
- Description of the imprecision in tool conditions for various cutting conditions.

With the increasing needs for effective and robust automated machining process and tool condition monitoring, a significant amount of research work has been performed to find decision making strategies. The principal constituents of soft computation include fuzzy logic for imprecision in the acquired data, neural networks for learning, and probability reasoning for uncertainty. These three components are usually overlapped. The "soft computation" is easily implemented by fuzzy neural networks.

#### **Appropriate problems for neural network learning**

ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones. It is also applicable to problems for which more symbolic representations are often used, such as the decision tree learning tasks. In these cases, ANN and decision tree learning often produce results of comparable accuracy. The back-propagation algorithm is the most commonly used ANN learning technique. It is appropriate for problems with the following characteristics:

**Instances are represented by many attribute-value pairs:** The target function to be learned is defined over instances that can be described by a vector of predefined features, such as the pixel values in the ALVINN example. These input attributes may be highly correlated or independent of one another. Input values can be any real values.

**The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes:** For example, in the ALVINN system the output is a vector of 30 attributes, each corresponding to a recommendation regarding the steering direction. The value of each output is some real number between 0 and 1, which in this case corresponds to the confidence in predicting the corresponding steering direction. We can also train a single network to output both the steering command and suggested acceleration, simply by concatenating the vectors that encode these two output predictions.

#### **The training examples may contain errors**

ANN learning methods are quite robust to noise in the training data.

**Long training times are acceptable:** Network training algorithms typically require longer training times than, say, decision tree learning algorithms. Training times can range from a few seconds to many hours, depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.

**Fast evaluation of the learned target function may be required:** Although ANN learning times are relatively long, evaluating the learned network, in order to apply it to a subsequent instance, is typically very fast. For example, ALVINN applies its neural network several times per second to continually update its steering command as the vehicle drives forward.

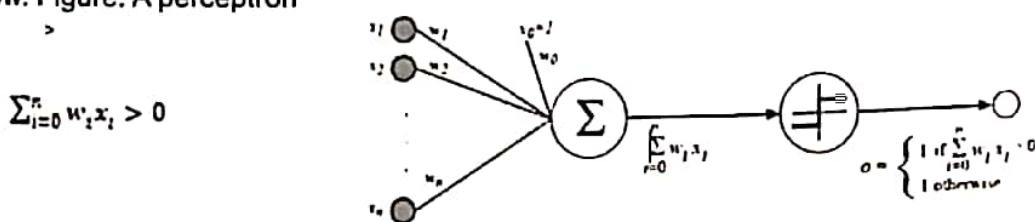
**The ability of humans to understand the learned target function is not important:** The weights learned by neural networks are often difficult for humans to interpret. Learned neural networks are less easily communicated to humans than learned rules.

IBM

## Basic understanding of neural networks

IBM ICE (Innovation Centre for Education)

- One type of ANN system is based on a unit called a perceptron, illustrated in figure given below. Figure: A perceptron



- Given inputs  $x_1$  through  $x_n$ , the output  $o(x_1, \dots, x_n)$  computed by the perceptron is:

$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

- To simplify notation, we imagine an additional constant input  $x_0=1$ , allowing us to write the above inequality as:

$$\sum_{i=0}^n w_i x_i > 0 \quad \vec{w} \cdot \vec{x} > 0$$

- For brevity, we will sometimes write the perceptron function as:

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

$$sgn(y) = \begin{cases} 1 & \text{if } y > 0 \\ -1 & \text{otherwise} \end{cases}$$

Figure 3-76. Basic understanding of neural networks

AIR011.0

### Notes:

#### Basic understanding of neural networks

A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise. More precisely, given inputs  $x_1$  through  $x_n$ , the output  $o(x_1, \dots, x_n)$  computed by the perceptron, where each  $w_i$  is a real-valued constant, or weight, that determines the contribution of input  $x_i$  to the perceptron output. Notice the quantity ( $-w_0$ ) is a threshold that the weighted combination of inputs  $w_1 x_1 + \dots + w_n x_n$  must surpass in order for the perceptron to output a 1. To simplify notation, we imagine an additional constant input  $x_0 = 1$ , allowing us to write the above inequality as

$$\sum_{i=0}^n w_i x_i > 0$$

$>0$  or in vector form as  $>0$ . For brevity, we will sometimes write the perceptron function.

## Representational power of perceptrons

IBM ISE Innovation Center for Education

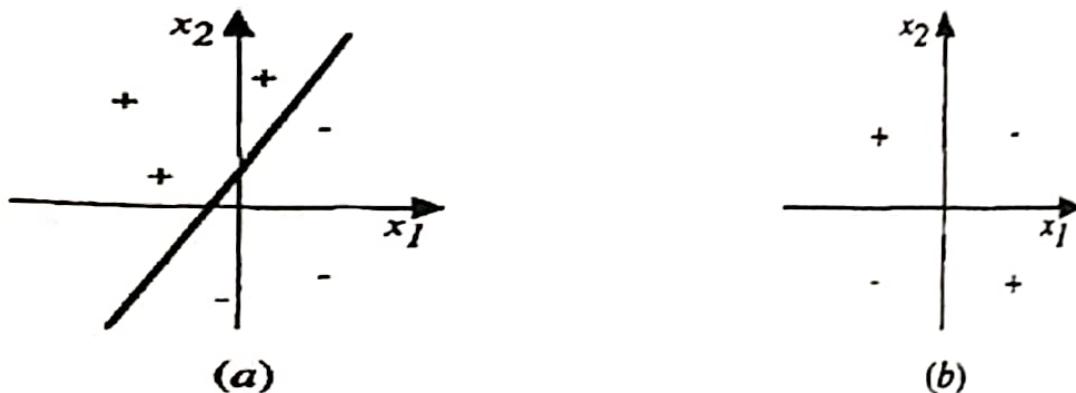


Figure: Perceptron representation.

- The decision surface represented by a two-input perceptron:
  - A set of training examples and the decision surface of a perceptron that classifies them correctly.
  - A set of training examples that is not linearly separable.

Figure 3-77. Representational power of perceptrons

AIR011.0

### Notes:

#### Representational power of perceptrons:

We can view the perceptron as representing a hyperplane decision surface in the  $n$ -dimensional space of instances (i.e., points). The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side.

- A set of training examples and the decision surface of a perceptron that classifies them correctly.
- A set of training examples that is not linearly separable (i.e., that cannot be correctly classified by any straight line).  $x_1$  and  $x_2$  are the Perceptron inputs. Positive examples are indicated by "+", negative by "-".

The equation for this decision hyperplane is  $\vec{w} \cdot \vec{x} = 0$ . Of course, some sets of positive and negative examples

$$\vec{w} \cdot \vec{x}$$

cannot be separated by any hyperplane. Those that can be separated are called linearly separable sets of examples.

- A single perceptron can be used to represent many Boolean functions. For example, if we assume Boolean values of 1 (true) and -1 (false), then one way to use a two-input perceptron to implement the AND function is to set the weights  $w_0 = -3$ , and  $w_1 = w_2 = .5$ . This perceptron can be made to represent the OR function instead by altering the threshold to  $w_0 = -3$ .
- Perceptrons can represent all of the primitive Boolean functions AND, OR, NAND ( $\neg$ AND), and NOR ( $\neg$ OR).

Unfortunately, however, some Boolean functions cannot be represented by a single perceptron, such as the XOR function whose value is 1 if and only if  $x_1 \neq x_2$ . Note the set of linearly non-separable training examples shown in above figure (right) correspond to this XOR function.

The ability of perceptrons to represent AND, OR, NAND, and NOR is important because every Boolean function can be represented by some network of interconnected units based on these primitives. In fact, every Boolean function can be represented by some network of perceptrons only two levels deep, in which the inputs are fed to multiple units, and the outputs of these units are then input to a second, final stage.

One way is to represent the Boolean function in disjunctive normal form (i.e., as the disjunction (OR) of a set of conjunctions (ANDs) of the inputs and their negations). Note that the input to an AND perceptron can be negated simply by changing the sign of the corresponding input weight. Because networks of threshold units can represent a rich variety of functions and because single units alone cannot, we will generally be interested in learning multilayer networks of threshold units.

# Architecture of neural networks

- An artificial neural network can be divided into three parts, named layers, which are known as:
  - Input layer.
  - Hidden, intermediate, or invisible layers.
  - Output layer.
- The layers of architecture are as follows:
  - Single-layer feed forward network.
  - Multilayer feed forward networks.
  - Recurrent networks.
  - Mesh networks.

Figure 3-78. Architecture of neural networks

AIR011.0

## Notes:

### Architecture of neural networks

In general, an artificial neural network can be divided into three parts, named layers, which are known as:

**Input layer:** This layer is responsible for receiving information (data), signals, features, or measurements from the external environment. These inputs (samples or patterns) are usually normalized within the limit values produced by activation functions. This normalization results in better numerical precision for the mathematical operations performed by the network.

**Hidden, intermediate, or invisible layers:** These layers are composed of neurons which are responsible for extracting patterns associated with the process or system being analyzed. These layers perform most of the internal processing from a network.

**Output layer:** This layer is also composed of neurons, and thus is responsible for producing and presenting the final network outputs, which result from the processing performed by the neurons in the previous layers. The main architecture of artificial neural networks, considering the neuron disposition, as well as how they are interconnected and how its layers are composed, can be divided as follows:

- Single-layer feedforward network.
- Multilayer feedforward networks.
- Recurrent networks.
- Mesh networks.

## Single-layer feed-forward architecture

IBM ICE (Innovation Centre for Education)

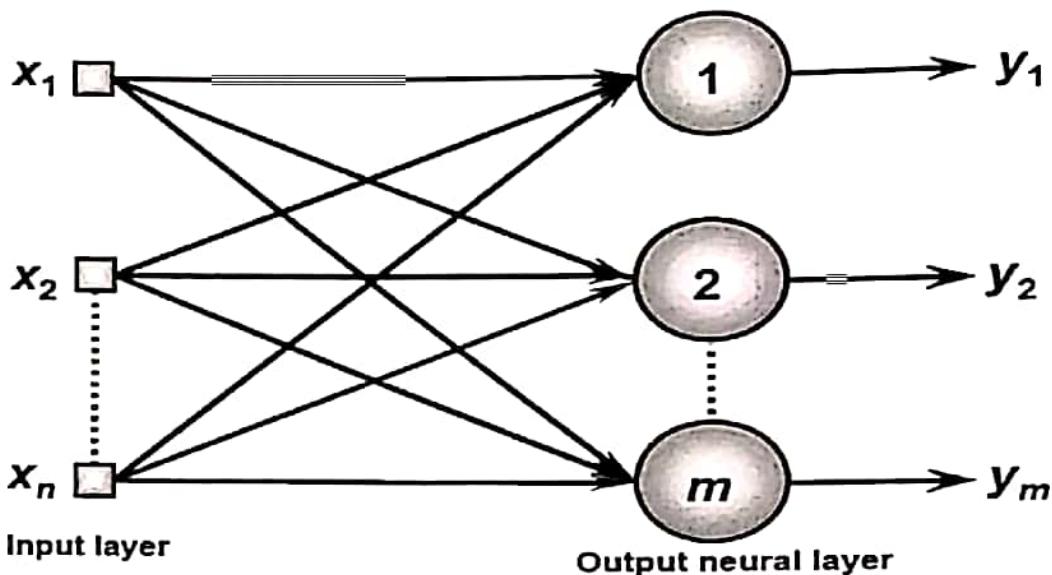


Figure: Example of single layer feed forward network

Figure 3-79. Single-layer feed-forward architecture

AIR011.0

### Notes:

#### Single-layer feed-forward architecture

This artificial neural network has just one input layer and a single neural layer, which is also the output layer. Figure illustrates a simple-layer feedforward network composed of  $n$  inputs and  $m$  outputs. The information always flows in a single direction (thus, unidirectional), which is from the input layer to the output layer. From figure, it is possible to see that in networks belonging to this architecture, the number of network outputs will always coincide with its amount of neurons. These networks are usually employed in pattern classification and linear filtering problems.

## Multiple-layer feed-forward architecture

IBM  
IEM ICE (Innovation Center for Education)

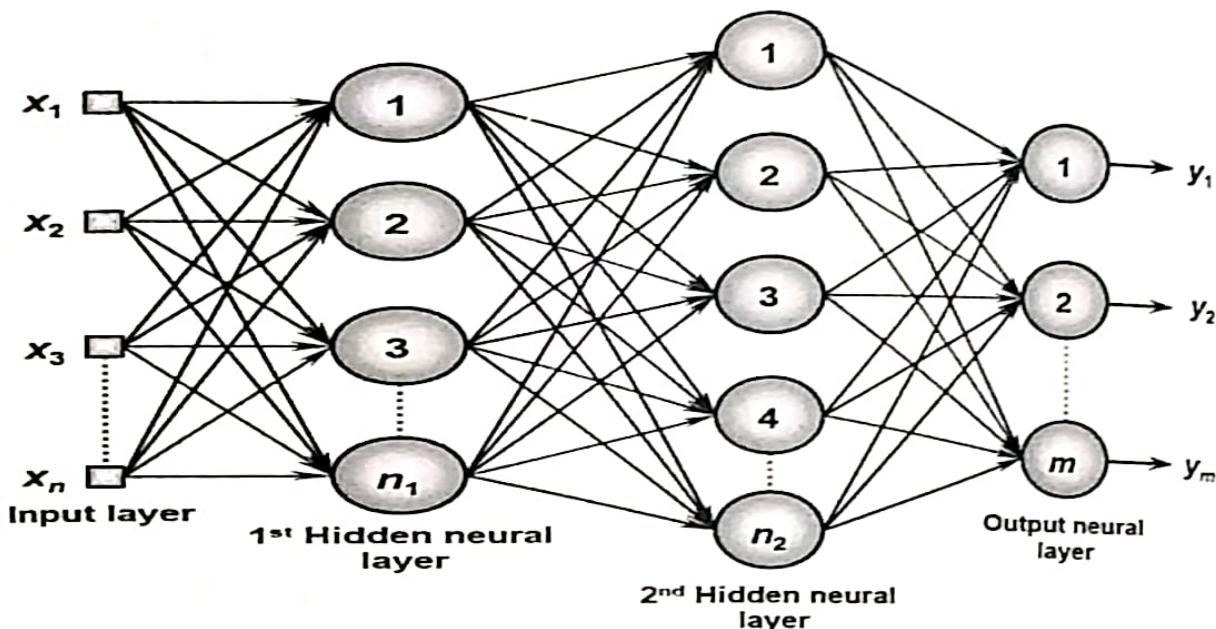


Figure: Example of a feedforward network with multiple layers

Figure 3-80. Multiple-layer feed-forward architecture

AIR0110

### Notes:

#### Multiple-layer feed-forward architecture

Differently from networks belonging to the previous architecture, feedforward networks with multiple layers are composed of one or more hidden neural layers. They are employed in the solution of diverse problems, like those related to function approximation, pattern classification, system identification, process control, optimization, robotics, and so on. A feedforward network with multiple layers composed of one input layer with  $n$  sample signals, two hidden neural layers consisting of  $n_1$  and  $n_2$  neurons respectively, and, finally, one output neural layer composed of  $m$  neurons representing the respective output values of the problem being analyzed. Among the main networks using multiple-layer feedforward architecture are the Multi Layer Perceptron (MLP) and the Radial Basis Function (RBF), whose learning algorithms used in their training processes are respectively based on the generalized delta rule and the competitive/delta rule.

The amount of neurons composing the first hidden layer is usually different from the number of signals composing the input layer of the network. In fact, the number of hidden layers and their respective amount of neurons depend on the nature and complexity of the problem being mapped by the network, as well as the quantity and quality of the available data about the problem. Nonetheless, likewise for simple-layer feed forward networks, the amount of output signals will always coincide with the number of neurons from that respective layer.

## Recurrent or feedback architecture

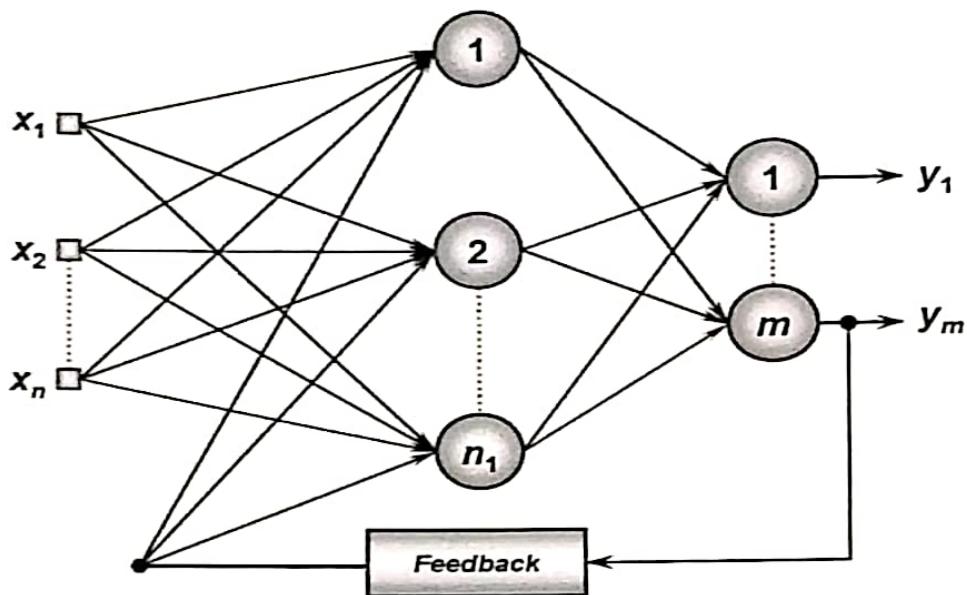


Figure: Example of a recurrent network

Figure 3-81. Recurrent or feedback architecture

AIR011.0

### Notes:

#### Recurrent or feedback architecture

In these networks, the outputs of the neurons are used as feedback inputs for other neurons. The feedback feature qualifies these networks for dynamic information processing, meaning that they can be employed on time-variant systems, such as time series prediction, system identification and optimization, process control, and so forth. Among the main feedback networks are the Hopfield and the perceptron with feedback between neurons from distinct layers, whose learning algorithms used in their training processes are respectively based on energy function minimization and generalized delta rule, as will be investigated in the next chapters. Figure given below illustrates an example of a perceptron network with feedback, where one of its output signals is fed back to the middle layer. Thus, using the feedback process, the networks with this architecture produce current outputs also taking into consideration the previous output values.

# Mesh architecture

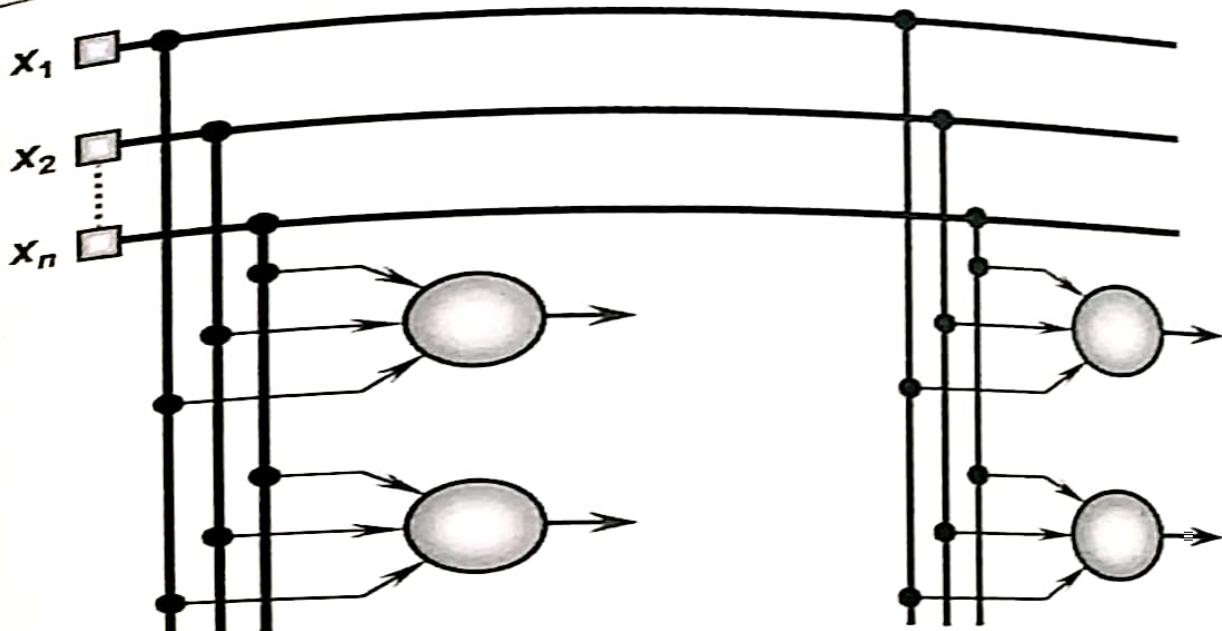


Figure: Structure of a mesh network

Figure 3-82. Mesh architecture

AP011.0

## Notes:

### Mesh architecture

The main features of networks with mesh structures reside in considering the spatial arrangement of neurons for pattern extraction purposes, that is, the spatial localization of the neurons is directly related to the process of adjusting their synaptic weights and thresholds. These networks serve a wide range of applications and are used in problems involving data clustering, pattern recognition, system optimization, graphs, and so forth. The Korhonen network is the main representative of mesh architecture, and its training is performed through a competitive process, as will be described in the following chapters. Figure given below illustrates an example of the Korhonen network where its neurons are arranged within a two-dimensional space.

1-108 AIR01

© Copyright IBM Corp. 2019

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



IBM ICE (Innovation Centre for Education)

## The perceptron training rule

- Precise learning problem is to determine a weight vector that causes the perceptron to produce the correct  $\pm 1$  output for each of the given training examples.
- Several algorithms are known to solve this learning problem. Here we consider two:
  - The perceptron rule.
  - The delta rule.
- One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training.

Figure 3-83. The perceptron training rule

AIR011.0

### Notes:

#### The perceptron training rule

Several algorithms are known to solve this learning problem. Here we consider two:

- The perceptron rule.
- The delta rule.

These two algorithms are guaranteed to converge to somewhat different acceptable hypotheses, under somewhat different conditions. They are important to ANNs because they provide the basis for learning networks of many units.

#### Let us understand perceptron rule:

One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly. Weights are modified at each step according to the perceptron training rule, which revises the weight  $w_i$  associated with input  $x_i$  according to the rule:

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

where

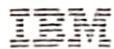
$$\Delta w_i = \eta(t - o)x_i$$

Here  $t$  is the target output for the current training example,  $o$  is the output generated by the perceptron, and  $\eta$  is a positive constant called the learning rate. The role of the learning rate is to moderate the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases. In fact, the above learning procedure can be proven to converge within a finite number of applications of the perceptron training rule to a weight vector that correctly classifies all training examples, provided the training examples are linearly separable and provided a sufficiently small  $\eta$  is used. If the data are not linearly separable, convergence is not assured.

3110 AIR01

© Copyright IBM Corp. 2019

Course materials may not be reproduced in whole or in part  
without permission of IBM.



IBM ICE (Innovation Centre for Education)

## Gradient descent and the delta rule

- The delta training rule is best understood by considering the task of training an unthresholded perceptron; that is, a linear unit for which the output  $o$  is given by:

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

- Although there are many ways to define this error, one common measure that will turn out to be especially convenient is training examples:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- Where  $D$  is the set of training examples,  $t_d$  is the target output for training example  $d$ .
- $O_d$  is the output of the linear unit for training example  $d$ .
- By this definition,  $E()$  is simply half the squared difference between the target output  $t_d$  and the hidden unit output  $o_d$ , summed over all.

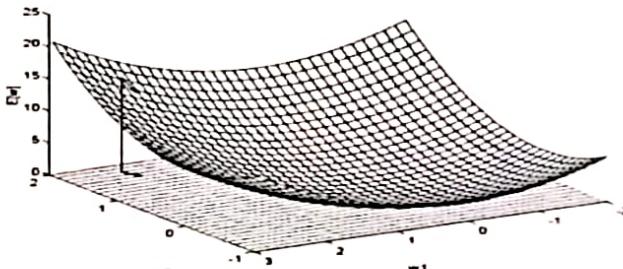


Figure: Error of different hypotheses

Figure 3-84. Gradient descent and the delta rule

AIR011.0

### Notes:

#### Gradient descent and the delta rule

Although the perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable. A second training rule, called the delta rule, is designed to overcome this difficulty. If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept. The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples. This rule is important because gradient descent provides the basis for the Back-propagation Algorithm, which can learn networks with many interconnected units. It is also important because gradient descent can serve as the basis for learning algorithms that must search through hypothesis spaces containing many different types of continuously parameterized hypotheses. Thus, a linear unit corresponds to the first stage of a perceptron, without the threshold.

In order to derive a weight learning rule for linear units, let us begin by specifying a measure for the training error of a hypothesis (weight vector), relative to the training examples. Although there are many ways to define this error, one common measure that will turn out to be especially convenient, where  $D$  is the set of training examples,  $t_d$  is the target output for training example  $d$ , and  $o_d$  is the output of the linear unit for training example  $d$ . By this definition,  $E()$  is simply half the squared difference between the target output  $t_d$  and the hidden unit output  $o_d$ , summed over all training examples.

**Visualizing Hypothesis space:** To understand the gradient descent algorithm, it is helpful to visualize the entire hypothesis space of possible weight vectors and their associated  $E$  values, as illustrated in Figure.

Here the axes  $w_0$  and  $w_1$  represent possible values for the two weights of a simple linear unit. The  $w_0, w_1$  plane therefore represents the entire hypothesis space. The vertical axis indicates the error  $E$  relative to some fixed set of training examples. The error surface shown in the figure thus summarizes the desirability of every weight vector in the hypothesis space (we desire a hypothesis with minimum error). Given the way in which we chose to define  $E$ , for linear units this error surface must always be parabolic with a single global minimum. The specific parabola will depend, of course, on the particular set of training examples.

Gradient descent search determines a weight vector that minimizes  $E$  by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps. At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface depicted in Figure This process continues until the global minimum error is reached. For a linear unit with two weights, the hypothesis space  $H$  is the  $w_0, w_1$  plane. The vertical axis indicates the error of the corresponding weight vector hypothesis, relative to a fixed set of training examples. The arrow shows the negated gradient at one particular point, indicating the direction in the  $w_0, w_1$  plane producing steepest descent along the error surface.



## Gradient descent algorithm

- Algorithm: Gradient Descent Algorithm for training a linear unit.
- GRADIENT-DESCENT(training examples,  $\eta$ ).
- Each training example is a pair of the form  $\langle x, t \rangle$ , where  $x$  is the vector of input values, and  $t$  is the target output value.  $\eta$  is the learning rate.

Figure 3-85. Gradient descent algorithm

AIR011.0

### Notes:

#### Method

- Initialize each  $w_i$  to some small random value
- Until the termination condition is met, Do
  - Initialize each  $\Delta w_i$  to zero
  - For each  $\langle x, t \rangle$  in training\_examples, Do
    - Input the instance  $x$  to the unit and compute the output  $o$
    - For each linear unit weight  $w_i$ , Do  $\Delta w_i = \Delta w_i + \eta(t - o)x_i$
  - For each linear unit weight  $w_i$ , Do  $w_i = w_i + \Delta w_i$

# Stochastic approximation to gradient descent (1 of 2)

IBM  
IBM ICE Innovation Center for Education

- The key practical difficulties in applying gradient descent are:
  - Converging to a local minimum can sometimes be quite slow (i.e., It can require many thousands of gradient descent steps).
  - If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.
- The modified training rule is like the training rule except that as we iterate through each training example, we update the weight according to:

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

- Where  $t$ ,  $o$ , and  $x_i$  are the target value, unit output, and  $i$ th input for the training example in question

- One way to view this stochastic gradient descent is to consider a distinct error function  $E_d(\vec{w})$  defined for each individual training example  $d$  as follows:

$$E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2$$

- Where  $t_d$  and  $o_d$  are the target value and the unit output value for training example  $d$ .
- Stochastic gradient descent iterates over the training examples  $d$  in  $D$ , at each iteration altering the weights according to the gradient with respect to  $E_d(\vec{w})$ .
- The sequence of these weight updates, when iterated over all training examples, provides a reasonable approximation to descending the gradient with respect to our original error function  $E(\vec{w})$ .
- By making the value of  $\eta$  (the gradient descent step size) sufficiently small, stochastic gradient descent can be made to approximate true gradient descent arbitrarily closely.

Figure 3-86. Stochastic approximation to gradient descent (1 of 2)

AIR0110

## Notes:

### Stochastic approximation to gradient descent

Gradient descent is an important general paradigm for learning. It is a strategy for searching through a large or infinite hypothesis space that can be applied whenever:

- The hypothesis space contains continuously parameterized hypotheses (e.g., The weights in a linear unit).
- The error can be differentiated with respect to these hypothesis parameters.

The key practical difficulties in applying gradient descent are:

- Converging to a local minimum can sometimes be quite slow (i.e., It can require many thousands of gradient descent steps).
- If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

One common variation on gradient descent intended to alleviate these difficulties is called incremental gradient descent, or alternatively stochastic gradient descent. Whereas the gradient descent training rule computes weight updates after summing over all the training examples in  $D$ , the idea behind stochastic gradient descent is to approximate this gradient descent search by updating weights incrementally.

## Stochastic approximation to gradient descent (2 of 2)



IBM ICE (Innovation Centre for Education)

- The key differences are listed below:

### Standard Gradient Descent

- Error is summed over all examples before updating weights
- Requires more computation per weight update step
- Converges to local minima

### Stochastic Gradient Descent

- Weights are updated upon examining each training example
- Require less computation
- Sometimes avoid falling into these local minima

- We have considered two similar algorithms for iteratively learning perceptron weights.
- The key difference between these algorithms are listed below:

### Perceptron training rule

Updates weights based on the error in the thresholded perceptron output  
 Converges after a finite number of iterations to a hypothesis that perfectly classifies the training data, provided the training examples are linearly separable.

### Delta rule

Updates weights based on the error in the unthresholded linear combination of inputs  
 Converges only asymptotically toward the minimum error hypothesis, possibly requiring unbounded time, but converges regardless of whether the training data are linearly separable.

Figure 3-87. Stochastic approximation to gradient descent (2 of 2)

AIR011.0

### Notes:

#### Stochastic approximation to gradient descent

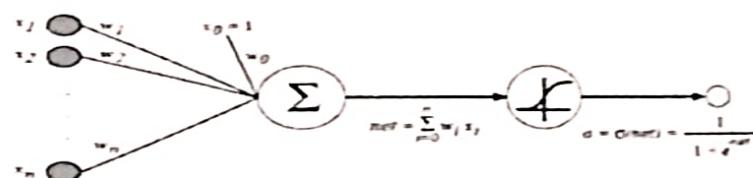
- Stochastic GRADIENT-DESCENT (training\_examples,  $\eta$ )
- Each training example is a pair of the form  $\langle x, t \rangle$ , where  $x$  is the vector of input values, and  $t$  is the target output value.  $\eta$  is the learning rate.

#### Method:

- Initialize each  $w_i$  to some small random value
- Until the termination condition is met, Do
  - Initialize each  $\Delta w_i$  to zero
  - For each  $\langle x, t \rangle$  in training\_examples, Do
    - Input the instance  $x$  to the unit and compute the output  $o$
    - For each linear unit weight  $w_i$ , Do  $w_i = w_i + \eta(t - o)x_i$

# Multilayer networks and back-propagation algorithm

- The kind of multilayer networks learned by the backpropagation algorithm are capable of expressing a rich variety of nonlinear decision surfaces.
- It is possible for the multilayer network to represent highly nonlinear decision surfaces that are much more expressive than the linear decision surfaces of single units.
- Figure: A sigmoid threshold unit



- The sigmoid unit computes its output  $\sigma$  as  $\sigma(y) = \frac{1}{1 + e^{-y}}$ 
  - Where  $\sigma = \sigma(\vec{w} \cdot \vec{x})$   $\sigma$  is often called the sigmoid function.
  - Note its output ranges between 0 and 1, increasing monotonically with its input.
  - Because it maps a very large input domain to a small range of outputs, it is often referred to as the squashing function of the unit.
  - The sigmoid function has the useful property that its derivative is easily expressed in terms of its output.  $\sigma'(y) = \sigma(y)(1 - \sigma(y))$

Figure 3-88. Multilayer networks and back-propagation algorithm

AIR0110

## Notes:

### Multilayer networks and back-propagation algorithm

Single perceptrons can only express linear decision surfaces. In contrast, the kind of multilayer networks learned by the back-propagation algorithm are capable of expressing a rich variety of nonlinear decision surfaces. A typical multilayer network and decision surface is depicted in Figure. Here the speech recognition task involves distinguishing among 10 possible vowels, all spoken in the context of "h-d" (i.e., "hid," "had," "head," "hood," etc.). The input speech signal is represented by two numerical parameters obtained from a spectral analysis of the sound, allowing us to easily visualize the decision surface over the two-dimensional instance space. As shown in the figure, it is possible for the multilayer network to represent highly nonlinear decision surfaces that are much more expressive than the linear decision surfaces of single units. This section discusses how to learn such multilayer networks using a gradient descent algorithm.

### A differentiable threshold unit:

What type of unit shall we use as the basis for constructing multilayer networks? At first, we might be tempted to choose the linear units discussed in the previous section, for which we have already derived a gradient descent learning rule. However, multiple layers of cascaded linear units still produce only linear functions, and we prefer networks capable of representing highly nonlinear functions. The perceptron unit is another possible choice, but its discontinuous threshold makes it undifferentiable and hence unsuitable for gradient descent. What we need is a unit whose output is a nonlinear function of its inputs, but whose output is also a differentiable function of its inputs. One solution is the sigmoid unit—a unit very much like a perceptron, but based on a smoothed, differentiable threshold function.



IBM ICE (Innovation Centre for Education)

## The back-propagation algorithm

- Because we are considering networks with multiple output units rather than single units as before.
- We begin by redefining E to sum the errors over all of the network output units:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

- Where outputs are the set of output units in the network, and  $t_{kd}$  and  $o_{kd}$  are the target and output values associated with the kth output unit and training example d.
- The learning problem faced by Backpropagation search a large hypothesis space defined by all possible weight values for all the units in the network.
- The situation can be visualized in terms of an error surface similar to that shown for linear units in Figure.
- The error in that diagram is replaced by our new definition of E, and the other dimensions of the space correspond now to all of the weights associated with all of the units in the network.
- As in the case of training a single unit, gradient descent can be used to attempt to find a hypothesis to minimize E.

Figure 3-89. The back-propagation algorithm

AIR011.0

### Notes:

**Algorithm:** The stochastic gradient descent version of the Back-propagation algorithm for feed-forward networks containing two layers of sigmoid units.

**Backpropagation:** (training\_examples,  $\eta$ ,  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$ )

#### Input:

- Each training  $\eta$  example is a pair of the form  $(x, t)$ , where  $x$  is the vector of network input values, and  $t$  is the vector of target network output values.
- $\eta$  is the learning rate (e.g., .05).  $n_{in}$  is the number of network inputs,  $n_{hidden}$  the number of units in the hidden layer, and  $n_{out}$  the number of output units.
- The input from unit i into unit j is denoted  $x_{ji}$ , and the weight from unit i to unit j is denoted  $w_{ji}$ .

#### Method:

- Create a feed-forward network with  $n_{in}$  inputs,  $n_{hidden}$  hidden units, and  $n_{out}$  output units.
- Initialize all network weights to small random numbers (e.G., Between -.05 and .05).
- Until the termination condition is met, do

For each  $(x,t)$  in training examples, Do Propagate the input forward through the network:

- Input the instance x to the network and compute the output  $o_u$  of every unit u in the network.

Propagate the errors backward through the network:

- For each network output unit  $k$ , calculate its error term  $\delta_k$

$$\delta_k \leftarrow o_k(1 - o_k)$$

- For each hidden unit  $h$ , calculate its error term  $\delta_h$

$$\delta_h \leftarrow o_h(1 - o_h)(t_h - o_h)$$

- Update each network weight  $w_{ji}$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji} \text{ where } \Delta w_{ji} = \eta \delta_j x_i$$

One major difference in the case of multilayer networks is that the error surface can have multiple local minima, in contrast to the single-minimum parabolic error surface. Unfortunately, this means that gradient descent is guaranteed only to converge toward some local minimum, and not necessarily the global minimum error. Despite this obstacle, in practice Backpropagation Algorithm been found to produce excellent results in many real-world applications. The Backpropagation Algorithm is presented above table. The algorithm as described here applies to layered feedforward networks containing two layers of sigmoid units, with units at each layer connected to all units from the preceding layer. This is the incremental, or stochastic, gradient descent version of Backpropagation. The notation used here is the same as that used in earlier sections.

Notice the algorithm begins by constructing a network with the desired number of hidden and output units and initializing all network weights to small random values. Given this fixed network structure, the main loop of the algorithm then repeatedly iterates over the training examples. For each training example, it applies the network to the example, calculates the error of the network output for this example, computes the gradient with respect to the error on this example, and then updates all weights in the network. This gradient descent step is iterated (often thousands of times, using the same training examples multiple times) until the network performs acceptably well.

The gradient descent weight-update rule is similar to the delta training rule. Like the delta rule, it updates each weight in proportion to the learning rate  $\eta$ , the input value  $x_i$  to which the weight is applied, and the error in the output of the unit. The only difference is that the error ( $t - o$ ) in the delta rule is replaced by a more complex error term,  $\delta_j$ . The exact form of  $\delta_j$  follows from the derivation of the weight tuning rule. To understand it intuitively, first consider how  $\delta_k$  is computed for each network output unit  $k$ .  $\delta_k$  is simply the familiar  $(t_k - o_k)$  from the delta rule, multiplied by the factor  $o_k(1 - o_k)$ , which is the derivative of the sigmoid squashing function. The  $\delta_h$  value for each hidden unit  $h$  has a similar form.

The algorithm updates weights incrementally, following the presentation of each training example. This corresponds to a stochastic approximation to gradient descent. To obtain the true gradient of  $E$  one would sum the  $\delta_j x_i$  values over all training examples before altering weight values. The weight-update loop in Backpropagation may be iterated thousands of times in a typical application. A variety of termination conditions can be used to halt the procedure. One may choose to halt after a fixed number of iterations through the loop, or once the error on the training examples falls below some threshold, or once the error on a separate validation set of examples meets some criterion. The choice of termination criterion is an important one, because too few iterations can fail to reduce error sufficiently, and too many can lead to over-fitting the training data.

# Multiple principal component fuzzy neural networks

IBM

IBM ICE (Innovation Centre for Education)

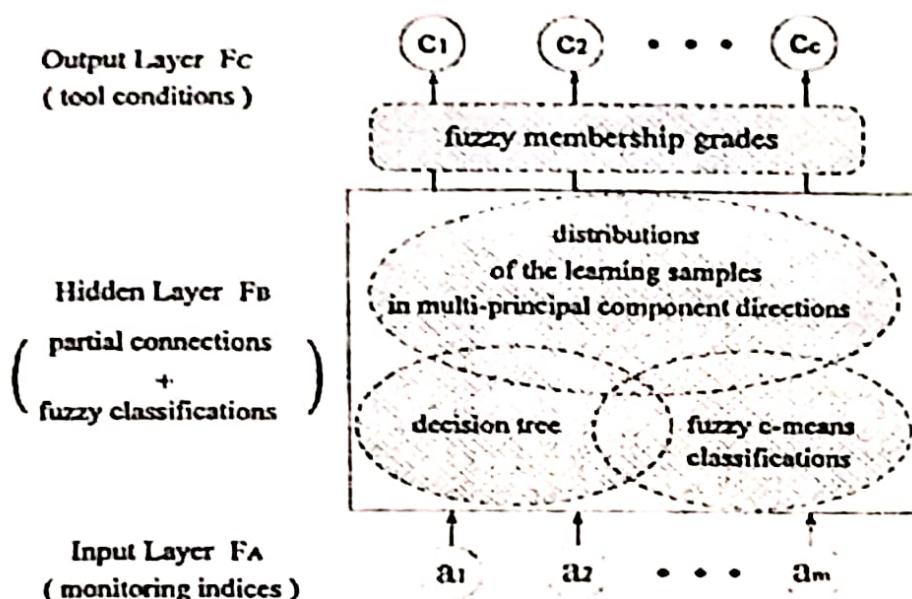


Figure: The Multiple Principal Component (MPC) fuzzy Neural Network

Figure 3-90. Multiple principal component fuzzy neural networks

AIR011.0

## Notes:

### Multiple Principal Component (MPC) fuzzy neural networks

The multiple principal component fuzzy neural networks are constructed based on the idea of "soft computation." Neural networks, fuzzy logic and statistical reasoning are employed. Simple classification procedures can be implemented at individual processing elements (neurons). The interconnections between neurons in the network communicate the information and make it possible to solve complex classification problems. Statistical reasoning is used in the learning procedure for the feature extraction and the partition strategies. For conventional neural networks, each of the processing elements (for input, output, and hidden layers) is always connected to every single processing element in the neighboring layers. Decision tree classifiers are hyper-plane classifiers that can be regarded as a type of partially connected neural networks since each node in the tree is connected to only its "father" and "sons", requiring comparatively less classification computations and can be implemented using parallelism from decision region by performing simple, easily understood operations on the neural network. In more sophisticated implementations, multi-layered neural networks, consisting of nonlinear connections between the inputs and the outputs are employed.

As an alternative to conventional neural networks, a partially connected, fuzzy neural network approach can be used for automated tool condition monitoring in machining. Different from matrix-type decision making, a tree structure is used for reducing unnecessary connections between elements in the input and the output layer. The fuzzy classifications are used in the neural networks to provide a comprehensive solution for certain complex problems. The neural network that utilizes fuzzy classification is shown in Figure.

The input layer,  $FA = (a_1, a_2, \dots, a_m)$ , has  $m$  processing elements, one for each of the  $m$  dimensions of the input pattern  $x^A$ . The hidden layer of the network,  $FB$ , consists of the neurons that use the fuzzy classification to separately address the subsets of the original data set while invoking necessary information from other neurons.

The probability distribution and the membership function are used for interconnections within the hidden layer and the connections to the output layer. The neurons of the output layer,  $FC$ , represent the degrees to which the input pattern  $x^k$  fits within the each class. There are two possible ways that the outputs of  $FC$  class nodes can be utilized. For a soft decision, outputs are defined with the fuzzy grades. For a hard decision, the  $FC$  nodes with the highest membership degree are located (a "winner-take-all" response). The connections within the hidden layer are not from one element to everyone in the neighboring layers. The structure depends on the training data and is created through the training process. These partial connections result in the simpler and faster training and classification.

A pattern classifier should possess several properties: on-line adaptation, nonlinear separability, dealing with overlapping classes, short training time, soft and hard decisions, verification and validation, tuning parameters, and nonparametric classification". The MPC fuzzy neural networks for automated tool condition monitoring has been developed in consideration of these requirements. The training and classification algorithms are based on the theories of neural networks, fuzzy logic, and probability reasoning.

With the application of fuzzy classification, the neural networks are effective in dealing with nonlinear separable and/or class-overlapping classification problems, which are common in the case of tool condition monitoring in machining, especially for the monitoring with varying cutting conditions. The partial interconnections within the fuzzy neural networks make the training time very short compared to that of fully connected networks such as the back-propagation neural networks. The calculations necessary for the classification are also significantly reduced since not all the neurons in the hidden layer are used while a sample is being processed. Soft and hard decisions are optional for different applications. The maximum partition algorithm is based on the distributions of the learning samples and no parameter estimations are needed. The linear fuzzy equation method uses a matrix to describe the relationship between the monitoring indices and the tool conditions. The MPC fuzzy neural networks use a tree structure similar to that in the fuzzy decision tree described by Li et.al., 1992.

Because the decision tree is more flexible than a matrix approach, it has better performance in the case of tool condition monitoring in machining. In constructing the fuzzy decision tree, the maximum partition generates nodes holding the samples from only one tool condition. The other samples are put into other nodes. This means each partition leads to a final decision at a leaf node of the tree. The maximum partition in the MPC fuzzy neural networks chooses a better partition so that a new-born neuron can hold samples from different tool conditions. A neuron can lead to other neurons in the hidden layer as well as neurons in the output layer. The consequence of this structure is simplicity in the interconnection and the short routines in the classification. Experimental tests by using the same set of data showed that the MPC fuzzy neural networks gave a better success rate than the fuzzy decision tree algorithm.

In the consideration of on-line adaptation (on-line learning) and the tuning parameters, a classifier should have as few parameters to tune in the system as possible. Both the back-propagation neural networks and the proposed MPC fuzzy neural networks have very few tuning parameters. The structure of the MPC fuzzy neural network is, however, easily modified with new learning samples. Unlike the back-propagation neural networks that require a complete retraining of the system with both the old and the new learning data, the MPC fuzzy neural networks need only to change partially their neurons and the connections when the new learning information is added.

Both supervised and unsupervised classification algorithms are easily implemented with the available learning samples. To insure a good distribution of the learning data, the training samples have to be representative of the whole span of the feature space. In tool condition monitoring all applicable tool and cutting conditions have to be considered during the training phase. On the other hand, if a poor distribution is encountered, then a modified feature extraction procedure has to be implemented. Information about new phenomena can be added to the monitoring system by knowledge updating.

# Fuzzy classification and uncertainties in tool condition monitoring



IBM ICE (Innovation Centre for Education)

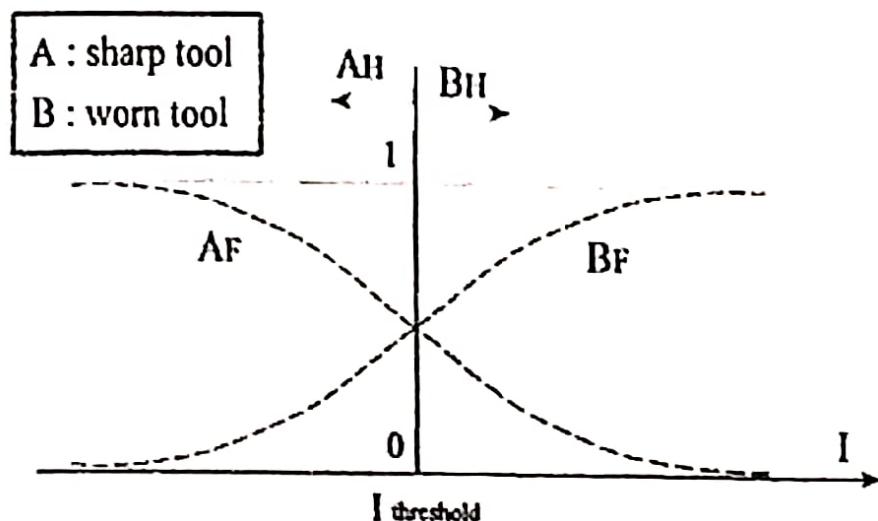


Figure: Soft boundaries in fuzzy classification

Figure 3-91. Fuzzy classification and uncertainties in tool condition monitoring

AIR011.0

## Notes:

### Fuzzy classification and uncertainties in tool condition monitoring

During machining, cutting conditions (e.g., cutting speed, feed, depth of cut) as well as tool conditions (e.g., tool wear) significantly affect the process parameters such as cutting forces and vibrations, which are usually used as the input signals to a monitoring system. Deterministic models which attempt to describe the relationship between the tool conditions and the various measured parameters are typically valid for a limited range of cutting conditions. The fuzzy classification can be used to describe the uncertainties and the overlapped relationship of the tool conditions and the monitoring indices. Briefly, the fuzzy expression of a tool condition.

where  $x$  is the value of A, and  $\mu_A(x)$  is a fuzzy measure, also known as the membership function. The  $\mu_A(x)$  is a monotonous function, and  $0 \leq \mu_A(x) \leq 1$ . The function increases with respect to the decrease of the uncertainty of A. If B is also a fuzzy set and is more uncertain than A, then:  $\mu_A(x) > \mu_B(x)$ . This might be interpreted as "the membership grade of small tool wear is greater than that of large tool wear". The fuzzy representation of the tool conditions in machining has its advantages. The concept of fuzzy decision making in machining tool condition monitoring is illustrated in Figure, where,  $A_H$  and  $B_H$  are categories classified by the hard decision, while  $A_F$  and  $B_F$  are classified by the fuzzy decision.

In summary, the combination of fuzzy logic with neural networks has a sound technical basis because these two techniques approach the design of intelligent machines from different angles. Fuzzy neural networks employ the advantages of both neural networks and fuzzy logic. Neural networks offer good performance in dealing with sensor information in parallel at a low computational level.

The high interconnection within the networks gives the capabilities of exchanging the information successfully, and managing nonlinearity. Fuzzy logic gives a means for representing, manipulating, and utilizing the data and the information that possess non-statistical uncertainties.

## Checkpoint (1 of 2)



IBM ICE (Innovation Centre for Education)

### Multiple choice questions:

1. Robotics is an inter disciplinary branch of engineering and science that deals with \_\_\_\_\_.
  - a) Construction
  - b) Operation
  - c) Use of robots
  - d) All the above
  
2. Wrist has three degrees of freedom \_\_\_\_\_.
  - a) Roll
  - b) Pitch
  - c) Both a) and b)
  - d) None of these
  
3. Machine learning is classified into two areas \_\_\_\_\_.
  - a) AI type learning on symbolic computation
  - b) Neural network
  - c) Both a) and b)
  - d) None of these

Figure 3-92. Checkpoint (1 of 2)

AIR011.0

### Notes:

Write your answers here:

- 1.
- 2.
- 3.

## Checkpoint (2 of 2)

### Fill in the blanks:

1. The traces of the first robot can be found deep into the \_\_\_\_\_ century.
2. \_\_\_\_\_ is a much more difficult problem than forward kinematics.
3. A software has been developed to effectively use the information from the \_\_\_\_\_.
4. The dominant lag in the force feedback path generates \_\_\_\_\_ around the force reflection path and so eases the operator's problems in stabilizing the system.

### True or False:

1. Force control is a central requirement if robot arms are to use tools or interact with work-pieces in an unstructured environment. True / False
2. The forward kinematics problem is quite complicated and there is complexity in deriving the equations. True / False
3. An objective of high-performance tele-operation is to give a human operator a sense of feel which can aid in the implementation of a remote task. True / False

Figure 3-93. Checkpoint (2 of 2)

AIR011.0

### Notes:

Write your answers here:

### Fill in the blanks:

- 1.
- 2.
- 3.
- 4.

### True or false:

- 1.
- 2.

1-124 AIR01

© Copyright IBM Corp. 2013

Course materials may not be reproduced in whole or in part  
without the prior written permission of IBM.

## Question bank



IBM ICE (Innovation Centre for Education)

### Two mark questions:

1. Define the robotics?
2. Define inverse kinematics?
3. Discuss the factors that affect the design of force controller.
4. What is Salford theories?

### Four mark questions:

1. Explain robot kinematics with examples?
2. Define inverse kinematics. Why it is complex.
3. Discuss different types of Saridis architecture?
4. Define machine learning. Explain rule-based machine learning system?

### Eight mark questions:

1. Explain the three layer architecture implementation in advanced robotics.
2. What is Force feedback strategy? Explain.

Figure 3-94. Question bank

AIR011.0

### Notes:

## Unit summary

**Having completed this unit, you should be able to:**

- Gain knowledge on the basic concepts of robotics and its components
- Gain an insight into the role of machine learning in modern day robotics industry
- Learn about the kinematic and dynamic control concept with a focus on intelligent gripping systems
- Gain an insight into the design and development of robotic components
- Learn about environment capturing sensors like CCD cameras
- Gain knowledge on the integration of sensors with real time robotic system
- Learn about the fuzzy classification and uncertainties in tool condition monitoring system

AIR0110

Figure 3-95. Unit summary

### Notes:

Unit summary is as stated above.

2-126 AIR01

Course materials may not be reproduced in whole or in part  
without the prior written permission of IBM.

© Copyright IBM Corp. 2019

# **Unit 4. Robot Operating System (ROS)**

## **What this unit is about**

This unit covers the concept of Robot Operating System (ROS). This unit also covers debugging and visualization under ROS. 3D modeling and simulation using ROS and some concepts will be discussed. The last section deals with computer vision applications.

## **What you should be able to do**

After completing this unit, you should be able to:

- Understand the operating system concepts for robotics
- Gain knowledge on debugging and visualization
- Understand the concept of 3D modeling and simulation
- Gain an insight into computer vision applications for robotics

## **How you will check your progress**

- Checkpoint

## **References**

IBM Knowledge Center

## Unit objectives

After completing this unit, you should be able to:

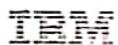
- Understand the operating system concepts for robotics
- Gain knowledge on debugging and visualization
- Understand the concept of 3D modeling and simulation
- Gain an insight into computer vision applications for robotics

Smart Notebook

Notes:

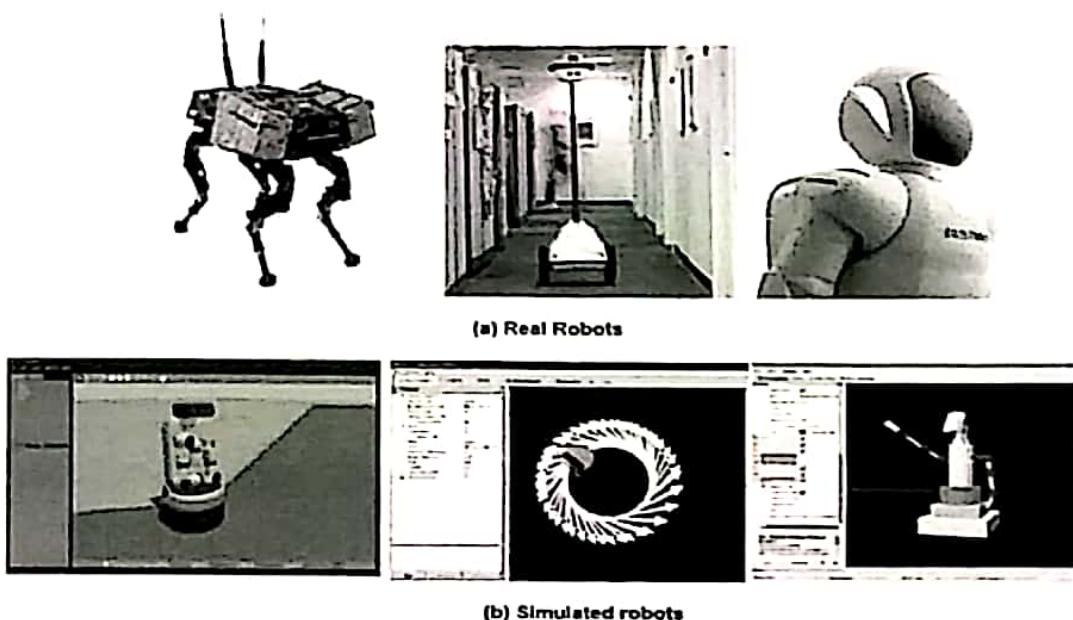
Unit objectives are as stated above.

ARQ110



IBM ICE (Innovation Centre for Education)

## Real and simulated robots



**Figure: Real and Simulated Robots**

Source: <https://robots.ree.org/learn/types-of-robots/> <https://docs.fetchrobotics.com/gazebo.html>, <https://www.pirobot.org/blog/0014/>

Figure 4-2. Real and simulated robots

AIR011.0

### Notes:

As shown in the figure (a), there are large varieties of hardware which are actual robots and ROS can be used with many of those hardware's. Figure (b) are various simulators for simulating a robot in ROS. The following are the three widely used simulators:

- Gazebo: This is a 3D physics simulator that was released before ROS and at present integrates with ROS.
- Stage: A 2D simulator for managing multiple robots with various sensors and scanners.
- ArbotiX: This is a python supported simulator which can simulate a differential drive robot without physics and sensor feedback.

There are various hardware robots available as shown in figure. Some of the famous ones that were generally used TurtleBot, Maxwell and Pi Robot.

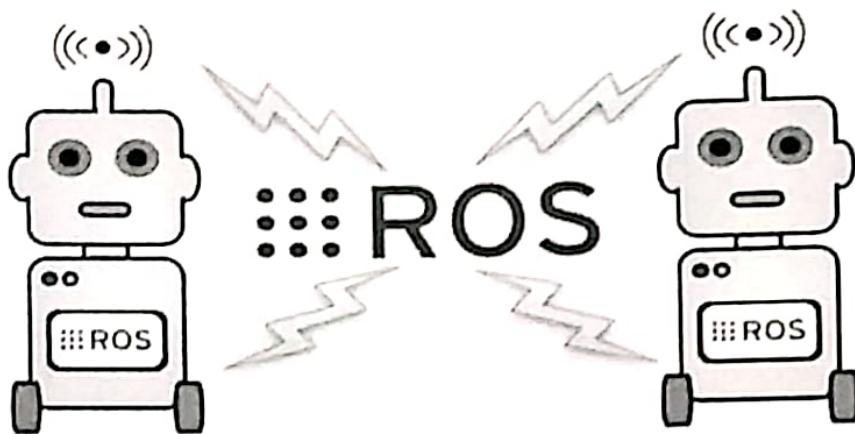
TurtleBot: This is an open-source compliant low-cost DIY robotic kit. This was created at Willow Garage in November 2010 by Melonee Wise along with Tully Foote. It can be programmed such that it can drive around the house, can be used for 3D application and can be used to create exciting and useful applications.

Maxwell: Is an inexpensive mobile manipulator, which provides 5-degree of freedom in arm, pan and tilt, read with depth sensor, and a base with differential drive. This can be fully integrated with ROS framework.

Pi Robot: The Pi Robot project is active since Dec 2005. The robot can support different body parts, micro-controllers, sensors, servos and drive motors. This also has a programming framework for producing autonomous behavior by coordinating sensory data and motor commands. The appearance of this robot has changed over time with different software and hardware configurations.

## Robot Operating System (ROS)

IBM ICE (Innovation Centre for Education)



**Figure: Robot Operating System (ROS)**

Source: <https://roboticsandautomationnews.com/2019/05/16/the-rise-of-the-robot-operating-system/22465/>

Figure 4-3. Robot Operating System (ROS)

AIR011.0

### Notes:

Robot operating system (ROS) is an application development platform that provides various features like message passing, code reuse and distributed computing. Many of the robotic companies are now using ROS extensively. ROS-Industrial extends the advanced capabilities of ROS software to manufacturing industries which is also an open source project. There are various other robotic platforms such as YARP, MRPT, Orocos and Player. ROS has various benefits as compared to these because of the below features:

**Lots of Tools:** ROS has got lots of tools for simulation, visualizing and debugging. Rviz, Gazebo and rqt\_gui.

**High-end sensors and actuators support:** ROS includes device drivers and interface packages of several sensors and actuators for robotics. Some of the high-end sensors which are included are Velodyne-LIDAR, Laser scanners, Kinect and so on. Actuators like Dynamixel servos are also supported.

**Ready to use high end capabilities:** ROS consists packages like Simultaneous Localization And Mapping (SLAM) and Adaptive Monte Carlo Localization (AMCL), which are required for autonomous navigation in mobile robots. There is a package named MoveIt for robot motion planning.

**Inter-platform operability:** The ROS message-passing middleware is used to communicate between various nodes. These nodes can be programmed in C, C++, Java or python provided the nodes have ROS client libraries.

## ROS basics and architecture

- **ROS Architecture:** is divided into three levels of concept or sections. These are
  - File system level.
  - Computation graph level.
  - Community level.

### Topic 4-4: ROS basics and architecture

AIR01

#### Notes:

ROS has been designed to run on different versions of Linux, Mac OS and on Windows (partly). The easiest and the most favored way is to use Ubuntu (A flavor of Linux). Open Source Robotics Foundation (OSRF) officially supports Ubuntu. ROS can be installed and used on a supercomputer as well as on the smallest beagle board. It is generally a requirement that the same ROS version should be run on all the machines connected to the same ROS network. This means the ROS version on the robot computer and the desktop work station/Laptop should be the same. This is because the ROS message structures that have some changes from one release to another.

It is also generally not a good idea to use an Ubuntu installation inside a virtual machine like VMWare or any other virtualization software. This is because the virtual machine will get bogged down when running the applications and also graphic intensive programs like Rviz might not run at all. For this course most of the work and work will take place at the command line along with a text editor. There are various text editors like pico, nano, vim, emacs etc.

© Copyright IBM Corp. 2019

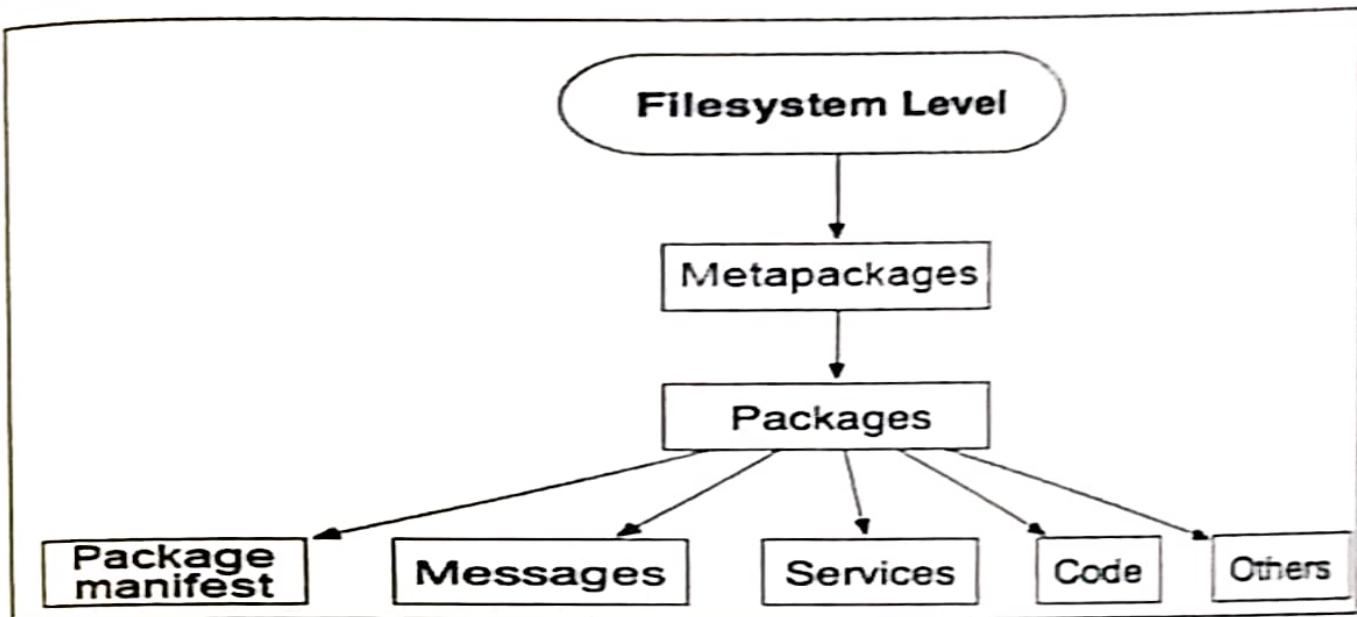
4 AIR01

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

ROS architecture conceptually is divided into:

- The file system level
- The computational graph level
- The community level
- File System Level is considered the first level. This level explains how the ROS is organized, its internal folder structure and the minimum set of files required to work.
- The second level which explains the communication between processes and systems is considered the Computation Graph level. This includes all the concepts and systems that ROS sets up, to handle various processes, to communicate with the computer (single or multiple) etc.
- Community level is the last level where sharing of knowledge among users happen about the tools and concepts, algorithms, and code. This level is very important because ROS being open source and being used by lot of developers, can grow quickly with their support.

## The File system level



**Figure: ROS File System Level**

Source. (Ref: [http://marte.aslab.upm.es/realmars/files/draft\\_p\\_dissertation\\_20141025\\_200\\_Botom\\_Effective\\_Robotics\\_Programming\\_with\\_ROS.pdf](http://marte.aslab.upm.es/realmars/files/draft_p_dissertation_20141025_200_Botom_Effective_Robotics_Programming_with_ROS.pdf))

### 4.5. The File system level

AP011.0

#### Notes:

ROS files are also organized in a particular fashion on the hard disk as is the case with many operating systems. Figure shows how the files are organized on the disk. Each of the block are explained below.

- **Meta Packages:** Meta packages are special purpose packages. These are grouped for carrying out a specific task. For example, ROS navigation stack is a meta package.
- **Packages:** Most basic unit of ROS package. These are atomic build and release items in ROS. They contain runtime process(nodes), libraries, configuration files, etc. and are organized as single unit.
- **Package manifest:** Manifest file is included inside a package and contains information such as author, license, dependencies, flags, compilation and so on. The package.xml file inside the ROS package is the manifest file of that package. Manifests are managed using a file named manifest.xml.
- **Meta packages manifest:** Similar to manifest of packages. One of the major differences is that it includes packages as run time dependencies which is declared using an export tag.
- **Messages (.msg):** ROS messages are information that are exchanged between one ROS process and the other. Custom message is defined in the msg folder inside a package (my\_package/msg/mymessagetype.msg). The extension of the message file is .msg.

- **Services (.srv):** ROS service is request/reply interaction between processes. Reply/request data are defined in the srv folder inside the package. (my\_package/srv/myservicetype.srv).
- **Repositories:** ROS packages are maintained using a VCS (Version control system) such as Git, mercurial etc.

## Files and folders in a sample package of ROS

```
mastering_ros_demo_pkg/
  action
    -- Demo_action.action
  CMakeLists.txt
  include
  msg
    -- demo_msg.msg
  package.xml
  src
    -- demo_action_client.cpp
    -- demo_action_server.cpp
    -- demo_msg_publisher.cpp
    -- demo_msg_subscriber.cpp
    -- demo_service_client.cpp
    -- demo_service_server.cpp
    -- demo_topic_publisher.cpp
    -- demo_topic_subscriber.cpp
  srv
    -- demo_srv.srv
```

Figure: Files and folders in a sample package of ROS

Source: Joseph, L. 2015. Mastering ROS for Robotics Programming. Birmingham: Packt Publishing Ltd.

Fig 4.6. Files and folders in a sample package of ROS

### Notes:

The screenshot in figure gives the idea or files and folders of a package in ROS. The packages show the various details like CMakeLists, Message files, Source files and srv files.

## ROS packages

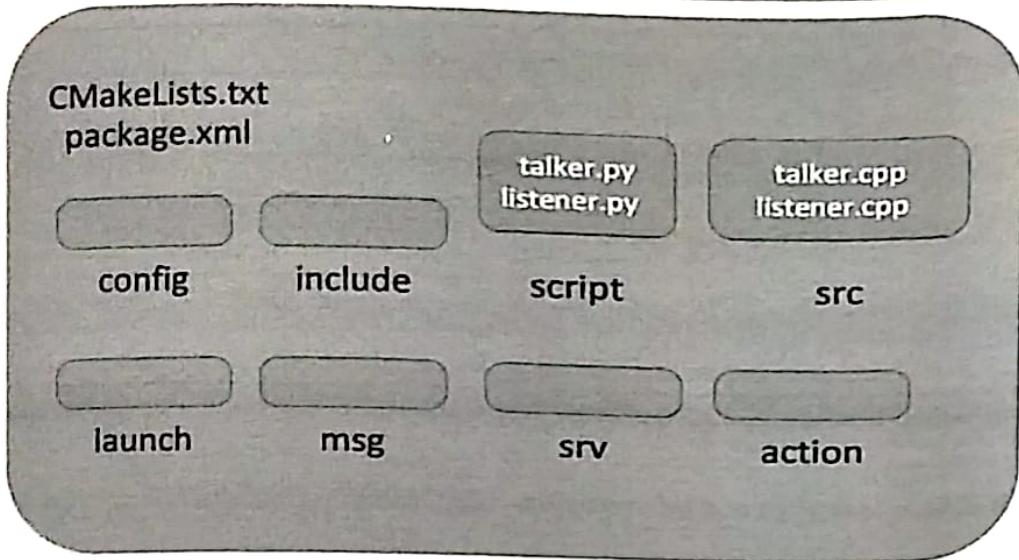


Figure: Structure of a typical ROS package

Source: Joseph, L. 2015. Mastering ROS for Robotics Programming. Birmingham: Packt Publishing Ltd.

Figure 4-7. ROS packages

AIR011.0

### Notes:

Various folders are as discussed below:

- config: All configuration files used in the ROS package are kept in this folder. This folder is created by the user and as a common practice this folder is named as config.
- include/package\_name: This folder consists of generally used headers and libraries along with the package.
- scripts: This folder is used to keep executable Python scripts. In the figure we can see two example scripts.
- src: This folder stores the source codes. Fig 4.4. shows two examples of the same.
- launch: This folder keeps the launch files used to launch the ROS nodes.
- msg: This folder contains custom message definitions.
- srv: This folder contains the service definitions.
- action: This folder contains the action definition. We will see more about actionlib in the upcoming sections.
- package.xml: This is the manifest file of this package.
- CMakeLists.txt: This is the CMake build file for this package.

## ROSbash

Some of the rosbash commands are:

- roscd
- roscp
- rosed
- rosrun



### Notes:

ROS provides a bash-like command called rosbash, which lets the user to work with packages and which can be used to navigate and manipulate the ROS package. Some of the rosbash commands are:

- roscd: This command is used to change the package folder. If we give the argument a package name, it will switch to that package folder.
- roscp: This command is used to copy a file from a package.
- rosed: This command is used to edit a file.
- rosrun: This command is used to run an executable inside a package.

## package.xml

IBM

IBM ICE (Innovation Centre for Education)

```

<package>
  .
  <name>hello_world</name>
  <version>0.0.0</version>
  <description>The hello_world package</description>

  <maintainer email="qboticslabs@gmail.com">Lentin Joseph</maintainer>
  <license>BSD</license>
  <url type="website">http://wiki.ros.org/hello_world</url>
  <author email="qboticslabs@gmail.com">Lentin Joseph</author>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend> rospy</build_depend>
  <build_depend> std_msgs</build_depend>

  <run_depend> roscpp</run_depend>
  <run_depend> rospy</run_depend>
  <run_depend> std_msgs</run_depend>

  <export>
    </export>
  </package>

```

Figure: Structure of a typical ROS package

Source: Joseph, L. 2015. Mastering ROS for Robotics Programming. Birmingham: Packt Publishing Ltd.

Figure 4-9. package.xml

AIR0110

### Notes:

The package.xml consists of the package name, version, description, author details, build dependencies and run time dependencies. The packages inside the `<build_depend></build_depend>` tag includes the packages that are necessary to build the source code of the package. Similarly, the packages inside the `<run_depend></run_depend>` tag are necessary during the runtime.

### ROS meta packages

Meta packages are specialized packages in ROS that only contain one file package.xml. It does not contain files and folders like a normal package. Meta packages group a set of multiple packages as a single logical package.

In the package.xml file, the meta package contains an export tag, as shown here:

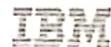
```

<export>
<metapackage/>
</export>

```

Also, in meta packages, there are no `<buildtool_depend>` dependencies for catkin, there are only `<run_depend>` dependencies, which are the packages grouped in the meta package.

## ROS messages



IBM ICE (Innovation Centre for Education)

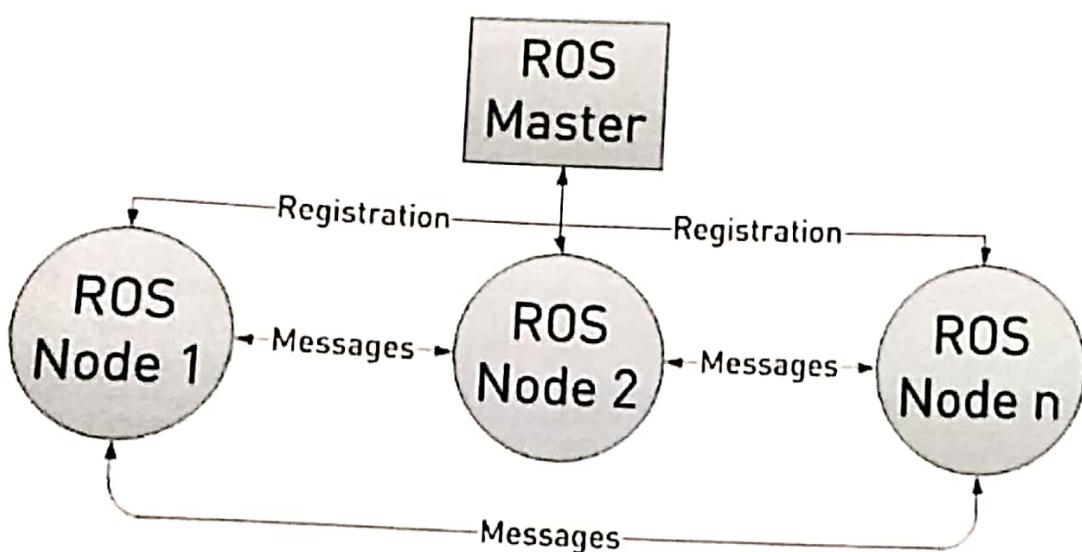


Figure: Ros messages

Source: [http://www2.ece.ohiostate.edu/~zhang/RoboticsClass/docs/ECE5463\\_ROSTutorialLecture1.pdf](http://www2.ece.ohiostate.edu/~zhang/RoboticsClass/docs/ECE5463_ROSTutorialLecture1.pdf)

Figure 4-10. ROS messages

AIR011.0

### Notes:

ROS nodes can publish data having a particular type. ROS messages describe this data using a simplified message description language. These data type descriptions are used to generate source code for the appropriate message type for various target languages. The data type description of ROS messages are stored in the msg subdirectory of a ROS package as .msg files.

The message definition consists of two parts: fields and constants. The field is split into field types and field name. Field types is the data type of the transmitting message and field name is the name of it. The constants define a constant value in the message file.

Here is an example of message definitions:

- int32 number
- string name
- float32 speed

Here, the first part is the field type and second is the field name. The field type is the data type and the field name can be used to access the value from the message.

4-14 AIR01

© Copyright IBM Corp. 2019

Course materials may not be reproduced in whole or in part  
without the prior written permission of IBM.

# ROS services

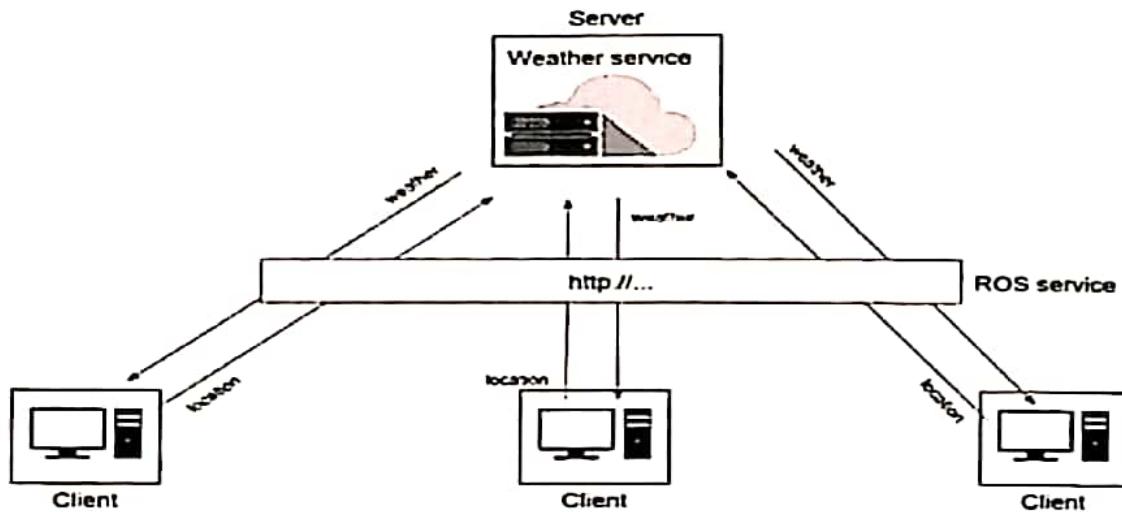


Figure: Ros Services

Source: <https://roboticsbackend.com/what-is-a-ros-service/>

Figure 4-11. ROS services

AIR01

## Notes:

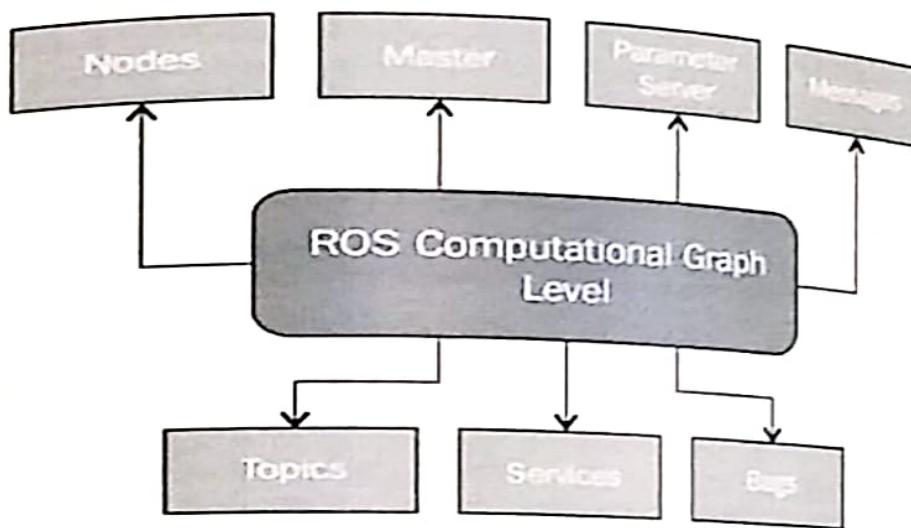
The ROS services are a type request/response communication between various ROS nodes. One node will send a request and will wait/listen until it gets a response from the receiver. The request/response communication is also done using the ROS message description. Similar to the message definitions which use the .msg file, we define the service definition in another file called .srv, which is kept inside the sub directory of the package. Similar to the message definition, a service description language is used to define the ROS service types.

An example service description format is as follows:

```
#Request message type
string str
#Response message type
string str
```

The first part is the message type of request that is separated by — and in the next part is the message type of response. For the above example, both Request and Response are strings.

# The computational graph level (1 of 2)



**Figure: ROS Computational graph level**

Source: Joseph, L. 2015. Mastering ROS for Robotics Programming. Birmingham: Packt Publishing Ltd.

Figure 4-12. The computational graph level (1 of 2)

## Notes:

The computation in ROS is carried out using ROS nodes. This computation network is represented and termed as the **computation graph**. The main components or concept in the computation graph are:

- ROS Nodes
- Master
- Parameter server
- Messages
- Topics
- Services and
- Bags.

Each of these contributes to the graph in different ways. The ROS communication related packages include core client libraries such as `roscpp` and `rospython`. The implementation of concepts such as topics, nodes, parameters, and services are included under a stack called `ros_comm`.

The stack also has a set of tools namely `rostopic`, `rosparam`, `rosservices` and `rosnode`. The `ros_comm` stack contains the **ROS communication middleware packages** which are collectively called as the **ROS graph layer**.

Let's now discuss briefly the various concepts in the figure.

**Nodes:** Nodes are the process that perform computation. ROS nodes are written using ROS client libraries (rospp and rospy). Different types of communication methods can be implemented in ROS nodes using client library APIs. There will be different nodes to perform different kinds of tasks in a robot. These nodes can communicate with each other and exchange data using ROS communication methods. One of the aims of ROS nodes is to build simple processes rather than a monolithic process with all functionality. ROS nodes are easy to debug because of a simple modular structure.

**Master:** Name registration and lookup facility for the nodes is provided by the ROS master. In the absence of the ROS Master, the nodes will not be able to locate each other, exchange messages, or invoke any services. In a distributed environment, we should run the master on one computer, and the remote nodes communicate among themselves using this master.

**Parameter Server:** This is a part of ROS Master. The parameter server is used to keep the data stored in a central location. These values can be accessed and modified by all the nodes.

**Messages:** Messages are used by nodes to communicate among themselves. Messages are simply a data structure containing the typed field, which can hold a set of data and that can be sent to another node. There are standard primitive types (integer, floating point, Boolean, and so on) and these are supported by ROS messages. We can also build our own message types using these standard types.

**Topics:** Named Buses called topics are used to transport messages in ROS. When a message is sent by the node through a topic, then it is meant that the node is publishing a topic. When a message is received by a node through a topic, then it is meant that the node is subscribing to a topic. The publisher node and subscriber node are not aware of each other's existence. We can even subscribe a topic that might not be published. In short, the production of information and consumption of it are decoupled. Each topic is uniquely named, and any node can have access to a topic and send data through it as long as the right message type is used.

**Services:** In some robot applications, a publish/subscribe model will not be enough if it needs a request/response interaction. The publish/subscribe model is a kind of one-way transport system and when we work with a distributed system, we might need a request/response kind of interaction. ROS Services are used in these cases. We can define a service definition that contains two parts; one is for requests and the other is for responses. Using ROS Services, we can write a server node and client node. The server node provides the service under a unique name, and when the client node sends a request message to this server, it will respond and send the result to the client. The client needs to wait until the server responds. The ROS service interaction is similar to an RPC (remote procedure call).

**Bags:** ROS message data can be saved and played back using formats called as Bags. Bags are an important mechanism for storing data, such as sensor data, which can be difficult to collect but is necessary for developing and testing robot algorithms. Bags are very useful features when we work with complex robot mechanisms.

## The computational graph level (2 of 2)

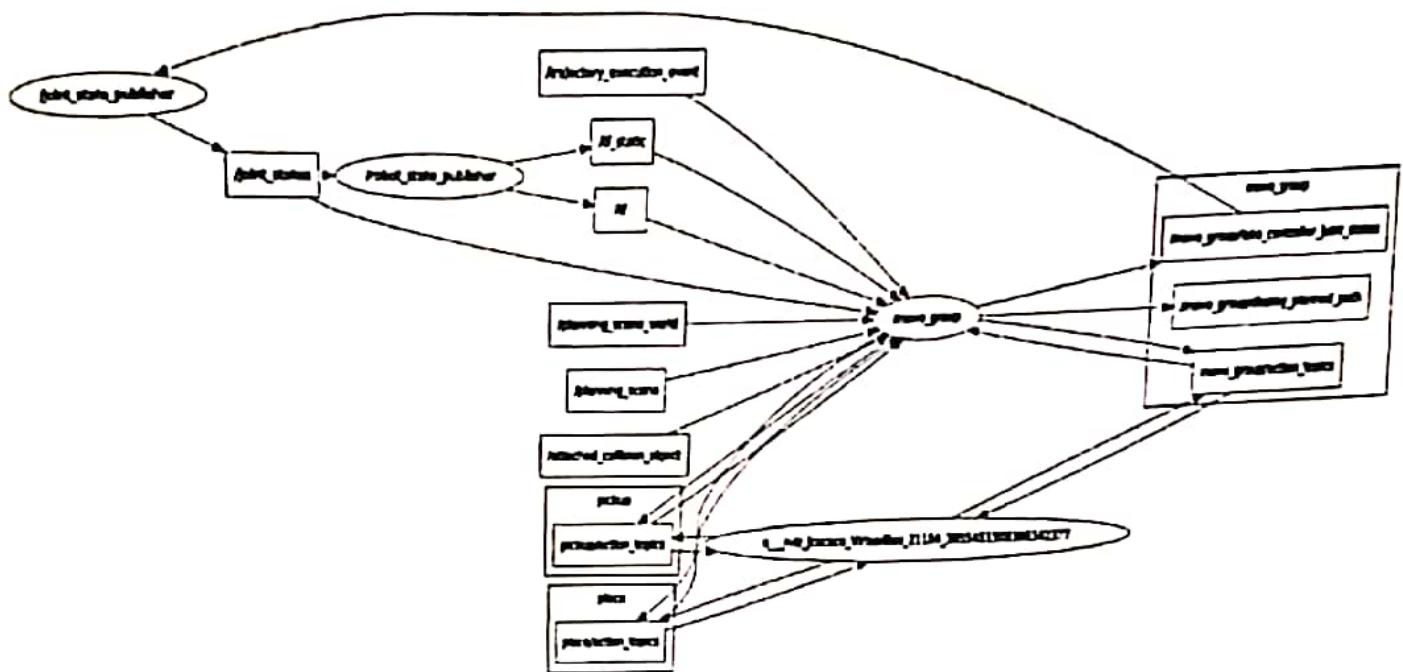


Figure: Graph of Communication using Topics

Source: Joseph, L. 2015. Mastering ROS for Robotics Programming. Birmingham: Packt Publishing Ltd.

Figure 4-13. The computational graph level (2 of 2)

AIR0110

### Notes:

#### ROS nodes

ROS nodes are a process that perform computation using ROS client libraries such as roscpp and rospy. One node can communicate with other nodes using ROS topics, services, and parameters. A robot might contain a number of nodes, for example, one node processes camera images, one node handles serial data from the robot, one node can be used to compute odometry, and so on.

Using nodes can make the system fault tolerant. Even if a node crashes, an entire robot system can still work. Nodes also reduce the complexity and increase debug-ability compared to monolithic codes because each node is handling only a single function. All running nodes should have a name assigned to identify them from the rest of the system. For example, /camera\_node could be a name of a node that is broadcasting camera images.

There is a rosbash tool to introspect ROS nodes. The rosnode command can be used to get information about a ROS node.

### ROS messages

ROS nodes communicate with each other by publishing messages to a topic. Messages as discussed earlier are simple data structures containing field types. The ROS message supports standard primitive datatypes and arrays of primitive types. Nodes can also exchange information using service calls. Services are also messages; the service message definitions are defined inside the srv file.

For ex. to access std\_msgs/msg/String.msg, we can use std\_msgs/String. If we are using the roscpp client, we have to include std\_msgs/String.h for the string message definition. ROS has inbuilt tools called rosmsg to get information about ROS messages

### ROS topics

ROS topics are named buses through which ROS nodes exchange messages. Topics can anonymously publish and subscribe, which means that the production of messages is decoupled from the consumption. The ROS nodes are not interested to know which node is publishing the topic or subscribing topics, it only looks for the topic name and whether the message types of publisher and subscriber are matching. The communication using topics are unidirectional, if we want to implement request/response such as communication, we have to switch to ROS services.

The ROS nodes communicate with topics using TCP/IP-based transport known as TCPROS. This method is the default transport method used in ROS. Another type of communication is UDPROS, which has low-latency, loose transport, and is only suited for tele-operation.

### ROS services

When we need a request/response kind of communication in ROS, we use the ROS services. ROS topics because it is unidirectional, can't do this kind of communication. ROS services are generally used in a distributed system. The ROS services are defined using a pair of messages. We define a request datatype and a response datatype in a srv file. The srv files are kept in a srv folder inside a package.

In ROS services, one node acts as a ROS server in which the service client requests the service from the server. If the server completes the service routine, it sends the results to the client. The ROS service definition can be accessed by the following method, for example, my\_package/srv/Image.srv can be accessed by my\_package/Image.

### ROS bags

A bag file in ROS stores ROS message data from topics and services. The .bag extension is used to represent a bag file. Bag files are created using the rosbag command, which subscribes one or more topics and store the message's data in a file as it's received. This file can play the same topics as they are recorded from or it can remap the existing topics also.

The main application of rosbag is data logging. The robot data can be logged and can be visualized and processed offline. The rosbag command is used to work with rosbag files.

## The community level

Various resources in these communities are as follows:

- Various resources in these communities are as follows:
  - Distributions
  - Repositories
  - ROS Wiki
  - Bug ticket system
  - Mailing lists
  - Blog

Figure 4-14. The community level

AIR011.0

### Notes:

These are the ROS resources that enables the community for ROS to exchange software packages and knowledge. The various resources in these communities are as follows:

**Distributions:** ROS distributions are similar to the Linux distribution, are a collection of versioned meta packages that we can install. The ROS distribution enables easier installation and collection of the ROS software. The ROS distributions maintain consistent versions across a set of software.

**Repositories:** These are federated network of code repositories, where different institutions develop and release their own software components related to robots, which ROS relies on.

**ROS Wiki:** ROS community Wiki is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and more.

**Bug ticket system:** Any user can register an bug in the existing software or any need for a new feature, can be logged [here](#).

**Mailing lists:** This is the primary communication channel about new updates to ROS, as well as a forum to ask questions about the ROS software.

**ROS Answers:** This website resource helps to ask questions related to ROS. If we post our doubts on this site, other ROS users can see this and give solutions.

**Blog:** The ROS blog updates with news, photos, and videos related to the ROS community (<http://www.ros.org/news>).

## Debugging and visualization (1 of 4)

IBM ICE (Innovation Centre for Education)

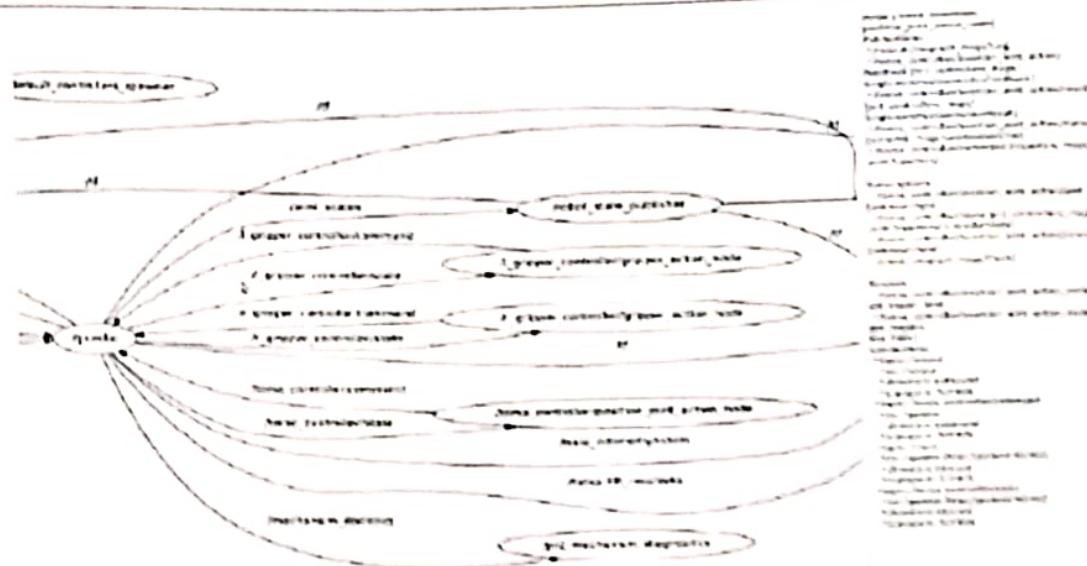


Figure: Debugging and visualization graph

Source: <http://www.ros.org/reps/rep-105-metapackage-structure> (Metapackage structure) - ROS 2015 Masterclass ROS for robotics programming Birmingham, part 1 publishing ROS

Figure 4-15 Debugging and visualization (1 of 4)

AIR0110

#### **Notes:**

ROS framework comes handy with a set of powerful tools which help the users and developers for debugging the code and for detecting problems while using both the hardware and software. The debugging facilities comprises of various log messages as well as visualization and inspection capabilities, which allows the users to visualize what is going on in the system easily.

ROS nodes may also use the GDB debugger. This is almost similar to debugging a regular C/C++ program, however a few aspects need be kept in mind. These aspects shall be the focus of this section. ROS provides an API for logging, which can be set for various logging levels which can be captured as an output or as print.

The user can be informed about various stages in the process of an algorithm with high-level informative messages, and at the same time warning the user about missed values, parameters, regular or fatal errors, which are considered as unrecoverable. Sometimes even if the program compiles and runs, it might still fail. This might be because of any both of the following reasons:

- An issue caused by the algorithm itself.
  - There exists a problem related to nodes, topics, services or any other ROS element

There are also a set of powerful tools with which we can inspect the state of the system, which include the node's graph, with all the connections (publishers and subscribers) among topics as discussed before. Both local and remote nodes can be addressed, with which the user can easily and rapidly detect a problem in a node that is not functional or a missed topic connection.

## Debugging and visualization (2 of 4)

IBM

IBM ICE (Innovation Centre for Education)

- Plotting Tools

- Time series plots
- Image visualization tools
- 3D visualization tools

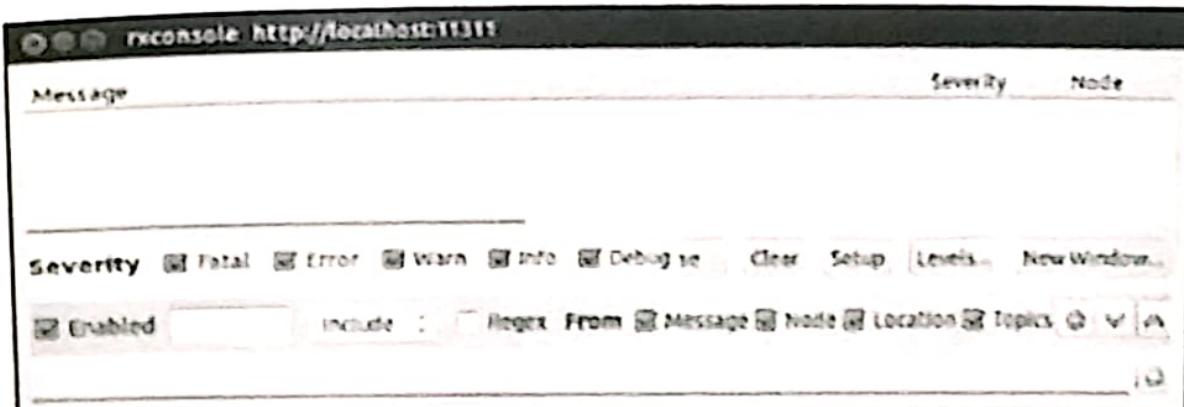


Figure: screenshot of rxconsole

Source: Joseph, L. 2015. Mastering ROS for Robotics Programming. Birmingham: Packt Publishing Ltd.

Figure 4-16 Debugging and visualization (2 of 4)

AIR011.0

### Notes:

ROS also provides plotting tools which can be used to analyze the output or results of the algorithms so that it becomes easy in case of detecting bugs. We have:

**Time series plots:** This is generally used for plotting the scalar values. For ex: fields of the messages transferred between nodes.

**Image visualization tools:** Tools to display images, including the support for stereo pairs.

**3D visualization tools:** These are used for rendering point clouds, laser scans, and so on. For ex: rviz.

All data belonging to a topic are placed on a frame so that the data is rendered in that frame. A robot operation generally consists of a collection of many frames including the transform frames. There are tools available to view the frame hierarchy which can help the user to see the relationships among them.

### GDB debugger with ROS nodes

With the GDB debugger the standard way of debugging C/C++ executables is generally followed. ROS also allows the use of GDB debugger with any C/C++ program. The standard practice is to use the programs executable file, which in the case of ROS would be a node implemented in C++. Therefore, we need to set the path such that it contains the node binary so that we can run it within GDB. Another way is to run our node directly without using rosrun <package> <node> syntax.

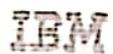
### Debugging messages

One of the best practices when writing a program is to include messages about what the program is doing. This has to be done without compromising the efficiency and the clearance of the programs output. ROS has an API that covers both of these features and is built on top of log4cxx which is a port of the well-known log4j logger library. Thus, we have support for several levels of messages, which might have a name depending on a condition or even throttle, without affecting the performance along with full integration with other tools in the ROS framework. They also are integrated seamlessly with the concurrent execution of nodes, meaning that that the messages do not get split, but they can be interleaved based on their timestamps.

### Graphical Tool – rxconsole

A series of tools can be integrated with logging messages to visualize and configure them. rosconsole is an API available in ROS framework comes that is generally used. ROS also provide a graphical tool, apart from the API to configure the logging messages in the nodes, which is a part of the rxtools package. This tool termed as rxconsole can be used to see the graphical interface that allows seeing, inspecting, configuring the logging subsystem of all the active running nodes. Fig shows the snapshot for rxconsole.

## Debugging and visualization (3 of 4)



IBM ICE (Innovation Centre for Education)

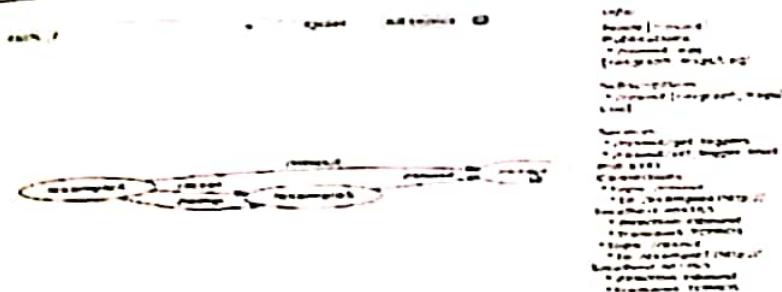


Figure: Node's graph online – rxgraph

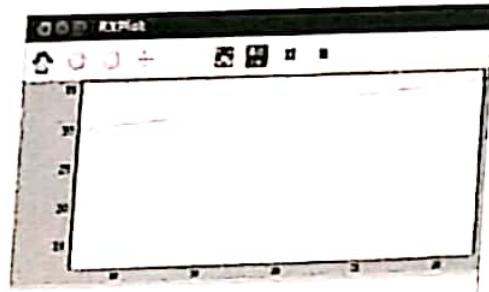


Figure: screenshot of rxplot

Source: Joseph, L. 2015. Mastering ROS for Robotics Programming. Birmingham: Packt Publishing Ltd.

Figure 4-17. Debugging and visualization (3 of 4)

AIR011.0

### Notes:

The current state of an ROS session in a simplest way can be illustrated using a directed graph that shows the nodes running on the system along with the publisher-subscriber connections among these nodes through the topics. Such node's graph can be generated using tools available in the ROS framework. rxgraph is such a tool which shows the node's graph during the system execution and allows the users to see how a node dynamically appears or disappears.

### Time series plot – rxplot

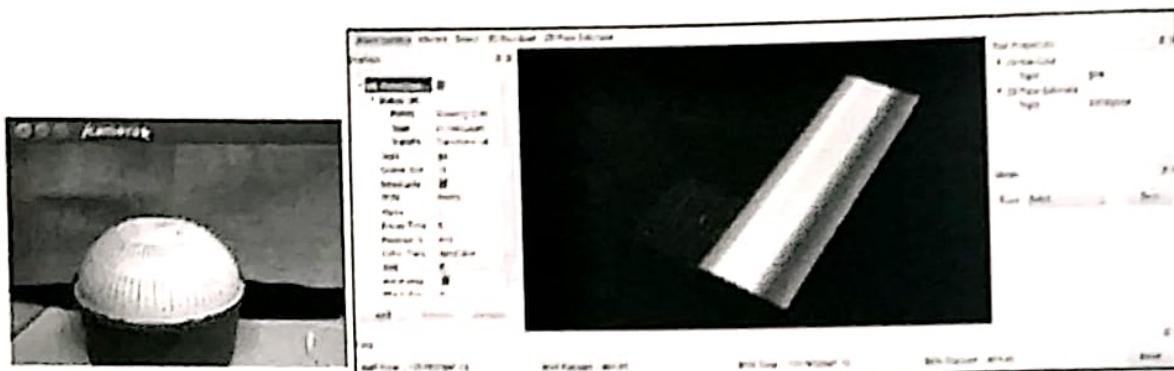
Scalar data can be plotted as a time series over the time based on the timestamps of the messages. This enables us to plot the scalar data on the y axis. rxplot is one such tool available in ROS framework. This also supports a powerful argument syntax that allows us to specify several fields of a structured message.



IBM ICE (Innovation Centre for Education)

## Debugging and visualization (4 of 4)

- Visualization of images



screenshot of output of a camera as a camera topic

screenshot of output of rviz

Source: Joseph, L. 2015. Mastering ROS for Robotics Programming. Birmingham: Packt Publishing Ltd.

Figure 4-18. Debugging and visualization (4 of 4)

AIR011.0

### Notes:

In ROS, we can configure a node that allows us to capture and display images coming from a camera on the fly. It is also possible to reproduce a video using a simple ROS node using any webcam/camera. A program can be implemented using a basic camera capture function/module using OpenCV. There are ROS bindings available that helps us to convert cv::Mat images into ROS image messages that can then be published in a topic. The node publishes the camera frames in the /camera topic.

#### 3D visualization – rviz

Stereo cameras, 3D laser, and the Kinect sensor are some devices that provide 3D data, usually in the form of point clouds which can be organized or unorganized. It is extremely useful to have tools that visualize this type of data. rviz, available in ROS, helps us to integrates an OpenGL interface with a 3D world representing the sensors' data in a modeled world. In such a situation which involves complex systems, the frame of each sensor and the transformations among them is of vital importance.

## Using sensors and actuators (1 of 3)

- Robots must sense the environment around them in order to react to variations in tasks. The sensors can range from very simple minimal setup designed for quick installation to highly complex and expensive sensor setups. For example:
  - Visual cameras.
  - Depth cameras.

Figure 4-10. Using sensors and actuators (1 of 3)

AIR0110

### Notes:

Now that we know the architecture of ROS and how to manage and debug the nodes, let's try to understand the basics and concepts of how to communicate with sensors and actuators, which can in turn interact with the real world. Many successful industrial deployments surprisingly do-little sensing. A large number of complex and sensitive industrial manipulation tasks can be performed through a combination mechanical engineering (limit switches) which operate an electrical circuit when a mechanical lever or plunger is pressed, in order to start execution of a preprogrammed robotic manipulation sequence.

Relatively simple sensors are a key part of modern industrial automation equipment, and their importance can never be overstated. There is another class of sensors that return scalar readings. For example, a pressure sensor can estimate the mechanical or barometric pressure and will typically output a scalar value in some range. Range sensors can be constructed from many physical phenomena (sound, light, etc.) and will also typically return a scalar value in some range.

Each sensor class has its own issues, constraints and problems that must be accommodated by sensor-processing algorithms. For example, a range sensor may have a "minimum distance" restriction: if an object is closer than that minimum distance, it will not be sensed. As a result of these constraints, it is often advantageous to combine several different types of sensors in a robotic system.

Any configuration of sensing hardware is possible (and has likely been tried), but for convenience, aesthetics, and to preserve line-of-sight with the center of the workspace, it is common for robots to have a sensor head on top of the platform that integrates several sensors in the same physical enclosure. Often, sensor heads sit atop a pan/tilt assembly, so that they can rotate to a bearing of interest and look up or down as needed. Let's now discuss sensors commonly found in robot sensor heads and on other parts of their bodies.

### Visual Cameras

Humans tends to rely on visual data to react to the world around them. If only robots were as smart as us! Unfortunately, using camera data intelligently is surprisingly not very easy. However, since the cameras are cheap and often useful for teleoperation, so it is common to see them on robot heads as sensors.

Interestingly, it is often more mathematically robust to describe robot tasks and environments in three dimensions (3D) than it is to work with 2D camera images. This is because the 3D data are invariant to changes in scene lighting, shadows, occlusions, and so on. In fact, in a surprising number of application domains, the visual data is largely ignored and the algorithms generally use 3D data.

When two cameras are simultaneously mounted to a common structure, they form a stereo camera. Each camera sees a slightly different view of the world, and these slight differences can be used to estimate the distances or various features in the image. However, this is not so trivial. The performance of a stereo camera depends on a large number of factors, such as the mechanical design, quality of the camera's, its lens type and quality, its resolution, and so on. Equally important are the qualities of the scene being imaged: a stereo camera can only estimate the distances to mathematically discernable features in the scene, such as sharp, high-contrast corners. A stereo camera cannot, for example, estimate the distance to a featureless wall, although it can most likely estimate the distance to the corners and edges of the wall, if they intersect a floor, ceiling, or other wall of a different color.

Many natural outdoor scenes possess sufficient information in form of texture that stereo vision can use to calculate depth estimation. Uncluttered indoor scenes, however, are not so easy to calculate. Several methods have emerged in the ROS community for handling cameras. Often, these ROS images need to be sent to and from OpenCV, a popular computer vision library. The cv\_bridge package is intended to simplify this operation.

### Depth cameras

As discussed, even though visual camera data is intuitively appealing, and seems like it is useful, many perception algorithms work much better with 3D data/information. In the past few years the cost of depth cameras has reduced and since depth cameras are active devices, they have become very useful for researchers. They illuminate the scene in various ways, which greatly improves the system performance. For example, a completely featureless indoor wall is not possible to detect using passive stereo vision.

Some common depth cameras, such as the Microsoft Kinect, project a structured light image. The device projects a precisely known pattern into the scene, its camera observes how this pattern is deformed as it lands on the various objects and surfaces of the scene, and finally a reconstruction algorithm estimates the 3D structure of the scene from this data. It's hard to overstate the impact that the Kinect has had on modern robotics! It was designed for the gaming market, which is orders of magnitude larger than the robotics sensor market, and could justify massive expenditures for the development and production of the sensor. Many robots are retrofitted to hold Kinects, and the sensor continues to be used across research and industry.

Although the Kinect is the most famous (and certainly the most widely used) depth camera in robotics, many other depth-sensing schemes are possible. One of the other examples is time-of-flight depth cameras. Just like visual cameras, depth cameras produce an enormous amount of data. This data is typically in the form of point clouds, which are the 3D points estimated to lie on the surfaces facing the camera. This allows for unstructured point cloud data, which is often advantageous, since depth cameras often cannot return valid depth estimates for each pixel in their images.

## Using sensors and actuators (2 of 3)

### Laser scanners

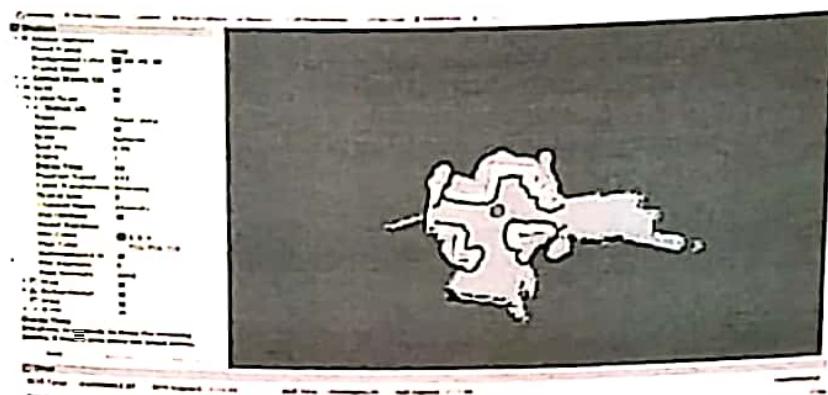


Figure: Screenshot of Laser Scanner in the Simulator

Source: [https://answers.ros.org/question/231560/smearing-ghosting-of-laser-scan-in-move\\_base-costmap/](https://answers.ros.org/question/231560/smearing-ghosting-of-laser-scan-in-move_base-costmap/)

Figure 4-20. Using sensors and actuators (2 of 3)

AIR0110

### Notes:

#### Laser scanners

Although depth cameras have greatly changed the depth-sensing market in the last few years due to their simplicity and low cost, there are still some applications in which laser scanners are widely used due to their superior accuracy and longer sensing range. There are many types of laser scanners, but one of the most common schemes used in robotics involves shining a laser beam on a rotating mirror spinning around 10 to 80 times per second (typically 600 to 4,800 RPM). As the mirror rotates, the laser light is pulsed rapidly, and the reflected waveforms are correlated with the outgoing waveform to estimate the time of flight of the laser pulse for a series of angles around the scanner.

Laser scanners used for autonomous vehicles are considerably different from those used for indoor or slow-moving robots. Vehicle laser scanners made by companies such as Velodyne must deal with the significant aerodynamic forces, vibrations, and temperature swings common to the automotive environment. Since vehicles typically move much faster than smaller robots, vehicle sensors must also have considerably longer range so that sufficient reaction time is possible. Additionally, many software tasks for autonomous driving, such as detecting vehicles and obstacles, work much better when multiple laser scanlines are received each time the device rotates, rather than just one. These extra scanlines can be extremely useful when distinguishing between classes of objects, such as between trees and pedestrians.

To produce multiple scanlines, automotive laser scanners often have multiple lasers mounted together in a rotating structure, rather than simply rotating a mirror. All of these additional features naturally add to the complexity, weight, size, and thus the cost of the laser scanner. The complex signal processing steps required to produce range estimates are virtually always handled by the firmware of the laser scanner itself. The devices typically output a vector of ranges several dozen times per second, along with the starting and stopping angles of each measurement vector. In ROS, laser scans are stored in sensor\_msgs/LaserScan messages, which map directly from the output of the laser scanner. Each manufacturer, of course, has their own raw message formats, but ROS drivers exist to translate between the raw output of many popular laser scanner manufacturers and the sensor\_msgs/LaserScan message format.

## Using sensors and actuators (3 of 3)

### Shaft Encoders



Figure: Rotary Encoder

Source: <http://www.robo-dyne.com/shop/rotary-encoder-illuminated-redgreen/?lang=en>

Figure 4-21. Using sensors and actuators (3 of 3)

AIR011.0

### Notes:

#### Shaft encoders

Estimating the motions of the robot is a critical component of virtually all robotic systems, with solutions ranging from low-level control schemes to high-level mapping, localization, and manipulation algorithms. Although estimates can be derived from many sources, the simplest and often most accurate estimates are produced simply by counting how many times the motors or wheels have turned.

Many different types of shaft encoders are designed expressly for this purpose. Shaft encoders are typically constructed by attaching a marker to the shaft and measuring its motion relative to another frame of reference, such as the chassis of the robot or the previous link on a manipulator arm. The implementation may be done with magnets, optical discs, variable resistors, or variable capacitors, among many other options, with trade-offs including size, cost, accuracy, maximum speed, and whether the measurement is absolute or relative to the position at power-up. Regardless, the principle remains the same: the angular position of a marker on a shaft is measured relative to an adjacent frame of reference.

Just like automobile speedometers and odometers, shaft encoders are used to count the precise number of rotations of the robot's wheels, and thereby estimate how far the vehicle has traveled and how much it has turned. Note that odometry is simply a count of how many times the drive wheels have turned, and is also known as dead reckoning in some domains. It is not a direct measurement of the vehicle position. Minute differences in wheel diameters, tire pressures, carpet weave direction (really!), axle misalignments, minor skidding, and countless other sources of error are cumulative over time.

As a result, the raw odometry estimates of any robot will drift; the longer the robot drives, the more error accumulates in the estimate. For example, a robot traveling down the middle of a long, straight corridor will always have odometry that is a gradual curve. Put another way, if both tires of a differential-drive robot are turned in the same direction at the exact same wheel velocity, the robot will never drive in a truly straight line. This is why mobile robots need additional sensors and clever algorithms to build maps and navigate.

Shaft encoders are also used extensively in robot manipulators. The vast majority of manipulator arms have at least one shaft encoder for every rotary joint, and the vector of shaft encoder readings is often called the manipulator configuration. When combined with a geometric model of each link of a manipulator arm, the shaft encoders allow higher-level collision-avoidance, planning, and trajectory-following algorithms to control the robot.

Because the mobility and manipulation use of shaft encoders are quite different, the ROS conventions for each use are also quite different. Although the raw encoder counts may also be reported by some mobile-base device drivers, odometry estimates are most useful when reported as a spatial transformation represented by a geometry\_msgs/Transform message. This concept will be discussed at great length throughout the book, but in general, a spatial transform describes one frame of reference relative to another frame of reference. In this case, the odometry transform typically describes the shaft encoder's odometric estimate relative to the position of the robot at power-up, or where its encoders were last reset.

In contrast, the encoder readings for manipulator arms are typically broadcast by ROS manipulator device drivers as sensor\_msgs/JointState messages. The JointState message contains vectors of angles in radians, and angular velocities in radians per second. Since typical shaft encoders have thousands of discrete states per revolution, the ROS device drivers for manipulator arms are required to scale the encoders as needed, accounting for transmissions and linkages, to produce a JointState vector with standard units. These messages are used extensively by ROS software packages, as they provide the minimal complete description of the state of a manipulator. That about covers it for the physical parts of a robot system. We now turn our attention to the "brains," where the robot interprets sensor data and determines how to move its body.

## 3D modeling and simulation (1 of 2)

IBM U.S. Innovation Center for Robotics



Stage

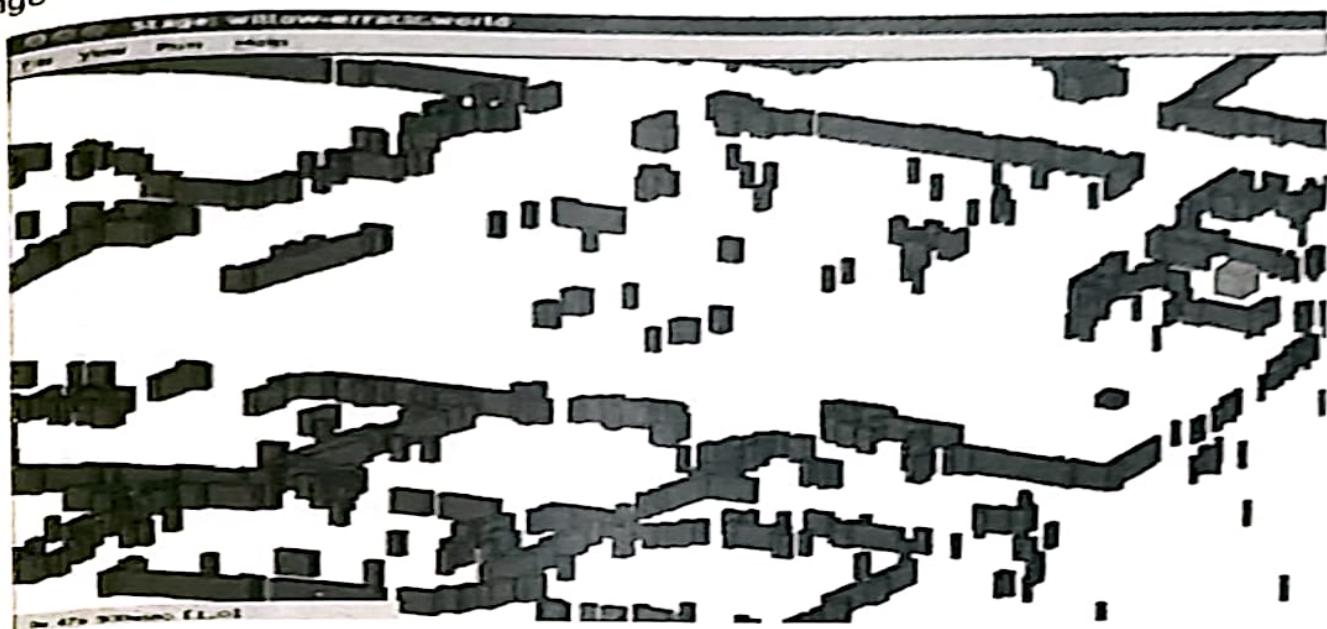


Figure: Screenshot of stage simulator

Source: Joseph, L. 2015. Mastering ROS for Robotics Programming. Birmingham: Pack Publishing Ltd.

Figure 4-22. 3D modeling and simulation (1 of 2)

AIR01.8

**Notes:**

Although the present-day robots include platforms that are considered to be remarkably low-cost compared to earlier robots of similar capabilities, they are still considered as significant investments. Additionally, real robots require logistics that include lab space, recharging of batteries, and operational quirks that often-become institutional knowledge of the organization operating the robot. Sadly, even the best robots break down periodically due to:

- operator error,
- environmental conditions,
- manufacturing or design defects, and so on.

Many of these headaches can be avoided by using simulated robots. At first glance, this seems to defeat the whole purpose of robotics; after all, the very definition of a robot involves perceiving and/or manipulating the environment. Software robots, however, are extremely useful. In simulation, we can model as much or as little of reality as we desire. Sensors and actuators can be modeled as ideal devices, or they can be incorporated with various levels of distortion, errors, and unexpected faults. Although data logs can be used in automated test suites to verify that sensing algorithms produce expected results, automated testing of control algorithms typically requires simulated robots, since the algorithms under test need to be able to experience the consequences of their actions.

Simulated robots are the ultimate low-cost platforms. They are free! They do not require complex operating procedures; you simply spawn a rosLaunch script and wait a few seconds, and a shiny new robot is created. At the end of the experimental run, a quick Ctrl-C and the robot vaporizes. For those of us who have spent many long nights with the pain and suffering caused by operating real robots, the benefits of simulated robots are simply magical.

Due to the isolation provided by the messaging interfaces of ROS, a vast majority of the robot's software graph can be run identically whether it is controlling a real robot or a simulated robot. At runtime, as the various nodes are launched, they simply find one another and connect. Simulation input and output streams connect to the graph in the place of the device drivers of the real robot. Although some parameter tuning is often required, ideally the structure of the software will be the same, and often the simulation can be modified to reduce the amount of parameter tweaks required when transitioning between simulation and reality.

As alluded to in the previous paragraphs, there are many use cases for simulated robots, ranging from algorithm development to automated software verification. This has led to the creation of a large number of robot simulators, many of which integrate nicely with ROS.

### **Stage**

For many years, the two-dimensional simultaneous localization and mapping (SLAM) problem was one of the most heavily researched topics in robotics. A number of 2D simulators were developed in response to the need for repeatable experiments, as well as the many practical annoyances of gathering long datasets of robots driving down endless office corridors. Canonical laser range-finders and differential-drive robots were modeled, often using simple kinematic models that enforce that, for example, the robot stays plastered to a 2D surface and its range sensors only interact with vertical walls, creating worlds that vaguely resemble that of Pac-Man. Although limited in scope, these 2D simulators were computationally fast, and were simple to interact with.

### **Stage**

Stage is an excellent example of this type of 2D simulator. It has a relatively simple modeling language that allows the creation of planar worlds with simple types of objects. Stage was designed from the outset to support multiple robots simultaneously interacting with the same world. It has been wrapped with a ROS integration package that accepts velocity commands from ROS and outputs an odometric transformation as well as the simulated laser range-finders from the robot(s) in the simulation.

## 3D modeling and simulation (2 of 2)

IBM® IoT Innovation Center for Education



### Gazebo



**Figure:** Screenshot of Gazebo simulator

**Figure:** Joseph, L. 2015. Mastering ROS for Robotics Programming. Birmingham: Packt Publishing Ltd.

**Figure 4-23. 3D modeling and simulation (2 of 2)**

ABP11.0

### Notes:

Although Stage and other 2D simulators were computationally efficient and excelled at simulating planar navigation in office-like environments, it is important to note that planar navigation is one of the aspects in robotics. With regards to robot navigation, a vast array of environments requires nonplanar motion which range from outdoor ground vehicles to aerial, underwater, and space robotics. Such environments require three-dimensional simulation for software development.

Generally, robot motions can be classified into mobility and manipulation. 2D or 3D simulators can handle mobility aspects where the environment around the robot is static. Simulating manipulation, however, requires a significant increase in the complexity of the simulator to handle the dynamics of not just the robot, but also the dynamic models in the scene. For example, at the moment that a simulated household robot is picking up a handheld object, contact forces must be computed between the robot, the object, and the surface the object was previously resting upon. Simulators often use rigid-body dynamics in which all objects are assumed to be incompressible, as if the world were a giant pinball machine. This assumption drastically improves the computational performance of the simulator, but often requires clever tricks to remain stable and realistic, since many rigid-body interactions become point contacts that do not accurately model the true physical phenomena. The art and science of managing the tension between computational performance and physical realism are highly nontrivial. There are many approaches to this trade-off, with many well suited to some domains but ill-suited to others.

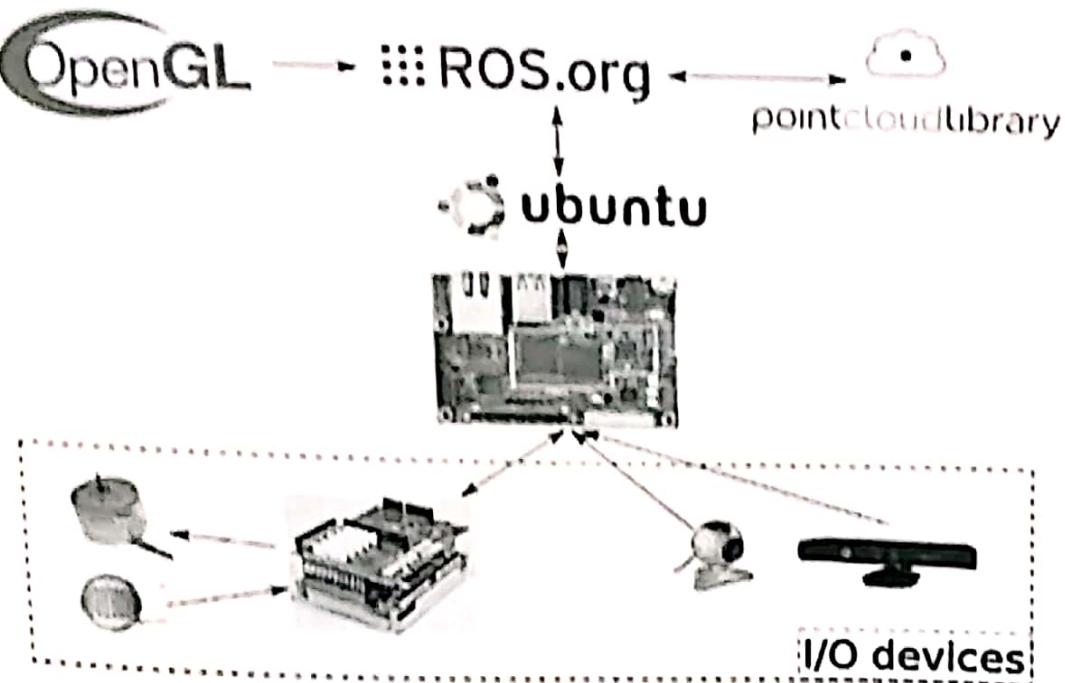
Like all simulators, Gazebo (Figure) is the product of a variety of trade-offs in its design and implementation. Historically, Gazebo has used the Open Dynamics Engine for rigid-body physics, but recently it has gained the ability to choose between physics engines at startup. For the purposes of this book, we will be using Gazebo with either the Open Dynamics Engine or with the Bullet Physics library, both of which are capable of real-time simulation with relatively simple worlds and robots and, with some care, can produce physically plausible behavior.

ROS integrates with Gazebo through the `gazebo_ROS` package. This package provides a Gazebo plug-in module that allows bidirectional communication between ROS and Gazebo. Simulated sensor and physics data can be streamed from Gazebo to ROS, and actuator commands can be streamed from ROS back to Gazebo. In fact, by choosing consistent names and data types for these data streams, it is possible for Gazebo to exactly match the ROS API of a robot. This in turn will help the developers to develop the robot software above the device-driver level which can be run identically both on the real robot, and (after parameter tuning) in the simulator. This is an enormously powerful concept which is of great help to the developers and advantage of using ROS.

# Computer vision

IBM

IBM ICE (Innovation Centre for Education)



**Figure: Computer Vision and ROS**

Source: [https://www.inforcecomputing.com/robots\\_hearts\\_are\\_beating\\_with\\_inforce\\_platforms/](https://www.inforcecomputing.com/robots_hearts_are_beating_with_inforce_platforms/))

Figure 4-24. Computer vision

AIR011.0

## Notes:

We are living in age where computer vision and artificial intelligence is the main pivot for application development in majority of the scientific and engineering domains. Webcams have become very affordable and depth cameras like the Kinect and Xtion have permitted researchers to work with 3D vision without investing on an expensive stereo camera. Acquiring the pixel and depth values into the computer is the first step. Using this data to extract meaningful information about the acquired world image/model is an interesting and challenging mathematical problem. We are fortunate enough to have decades of research which also has provided us with powerful vision algorithms starting from a simple color matching algorithm to people detectors that we can use.

The overall goal of machine vision is to recognize the structure of the world/image behind the changing pixel values. Individual pixels are continuously changing due to changes in lighting, viewing angle, object motion, occlusions and random noise. Computer vision algorithms extract stable features from these changing values. Some of the features which are generally used are corners, edges, blobs, colors, motion patches, etc. Once we are able to extract such features from an image or video stream, they can be tracked or grouped together to arrive at larger patterns supporting object detection and recognition.

### Support for OpenCV, OpenNI and PCL

ROS community has been working with OpenCV, OpenNI2 + OpenKinect, and PCL, which are considered the three pillars of computer vision. Application development which involve 2D image processing and machine learning extensively uses OpenCV. Appropriate drivers for various depth cameras like the Kinect by Microsoft and Xtion Pro by Asus are provided by OpenNI2 and OpenKinect. Point Cloud Library (PCL), is the library frequently used by many developers interested in processing 3D point clouds.

### Camera resolutions

Robots might be required to work with video cameras at various resolutions. Some of the resolutions used are 160x120, 320x240, 640x480 and 1280x960. It is true that setting the camera to the highest resolution helps us to capture more detail which is always better. However, the disadvantage is that more the number of pixels more computations are required to process a frame. For example, a 1280x960 video frame contains four times as many pixels as a 640x480 video frame. So for any kind of basic image processing operations we would have to compute four times as many operations per.

If the CPU and GPU are already being used at their maximum capacity at 20 frames per second (fps), processing a 640x480 video, we would end up getting 5 frames per second (fps) for a 1280x960 video which is too slow for any real-time vision processing on a mobile robot. Generally, all these extra pixels are not needed for any basic vision tasks. ROS framework use a 640x480 resolution for default camera launch files along with the standard package. If we are in need of lower or higher resolution, we need to change resolution modes in the camera's launch file or use rqt\_reconfigure.

### OpenCV: The open source computer vision library

Intel released the OpenCV code to general public in 2000. It was developed in 1999 to test CPU intensive image processing applications. Willow Garage in 2008, took charge of its maintenance and development. OpenCV has a learning curve compared to similar GUI-based vision packages (RoboRealm for Windows OS). OpenCV has got state-of-the-art vision algorithms and functions. It also provides methods for some machine learning algorithms (Support Vector Machines, artificial neural networks and Random Trees). OpenCV can be used with Linux, MacOS X, Windows and Android.

### ROS to OpenCV: The cv\_bridge Package

ROS video streams using OpenCV can be processed, if the drivers for the camera we are using are installed correctly. cv\_bridge package in the ROS framework acts as a bridge and helps us to convert between ROS and OpenCV and other image formats.

## Checkpoint (1 of 2)



IBM ICE (Innovation Centre for Education)

### Multiple choice questions:

1. Which of these is not a widely used simulator?

- a) Arbotix
- b) Stage
- c) Maxwell
- d) Gazebo

2. Robot operating system (ROS) is an \_\_\_\_\_ platform.

- a) Framework Development
- b) Application Development
- c) Interaction Design (ID)
- d) None of the above

3. ROS has got tools for

- a) Simulation
- b) Visualization
- c) Debugging
- d) All of the above

AIR0110

Figure 4-25. Checkpoint (1 of 2)

### Notes:

Write your answers here:

- 1.
- 2.
- 3.

## Checkpoint (2 of 2)



IBM ICE (Innovation Centre for Education)

Fill in the blanks:

1. ROS packages are maintained using \_\_\_\_\_.
2. \_\_\_\_\_ are the process that perform computation.
3. Scalar Values can be plotted using \_\_\_\_\_.
4. \_\_\_\_\_ is one of the 3D visualization tool.

True or False:

1. Actuators like Dynamixel servos are also supported in ROS. True/False
2. There is a package named MoveIt! for robot motion planning. True/False
3. Modeling in ROS is performed using URDF. True/False

Figure 4-26. Checkpoint (2 of 2)

AIR011.0

### Notes:

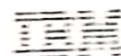
Write your answers here:

Fill in the blanks:

- 1.
- 2.
- 3.
- 4.

Fill in the blanks:

- 1.
- 2.
- 3.



## Question bank

### Two mark questions:

1. What is ROS?
2. Which are the various robotic platforms?
3. What is visualization graph?
4. Name the components of graph layer?

### Four mark questions:

1. Explain the file system level.
2. Describe the support for OpenCV in ROS.
3. Explain the concept of camera resolutions.
4. Explain gazebo simulator.

### Eight mark questions:

1. Explain the various debugging and visualization options in ROS.
2. Explain the different types sensors and actuators.

AR211.0

Figure 4-27. Question bank

### Notes:



IBM ICE (Innovation Centre for Education)

## Unit summary

Having completed this unit, you should be able to:

- Understand the concept of Robot Operating System
- Gain an insight into various levels of ROS package
- Gain knowledge on debugging and Visualization in ROS
- Understand the interaction of computer vision and ROS

---

Figure 4-28. Unit summary

AIR011.0

### Notes:

Unit summary is as stated above.

# Unit 5. Navigation, SLAM and Speech Recognition and Synthesis )

## What this unit is about

This unit covers robot navigation. Simultaneous localization and mapping (SLAM) is another problem during developing a fully functional mobile robot, which will also be covered in this unit. This unit will also cover the concept of Speech recognition and synthesis. Some insight into how ROS could be used for such application development will also be briefly discussed.

## What you should be able to do

After completing this unit, you should be able to:

- Understand the Simultaneous Localization And Mapping problem (SLAM)
- Gain knowledge on developing solution for the SLAM problem and implement it
- Understand the concept of Speech Recognition and Synthesis and implement it
- Gain an insight into how ROS could be used for such application development

## How you will check your progress

- Checkpoint

## References

IBM Knowledge Center

## Unit objectives

After completing this unit, you should be able to:

- Understand the Simultaneous Localization And Mapping problem (SLAM)
- Gain knowledge on developing solution for the SLAM problem and implement it
- Understand the concept of Speech Recognition and Synthesis and implement it
- Gain an insight into how ROS could be used for such application development



IBM ICE (Innovation Center for Education)

AIR0110

15.1 Unit objectives

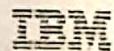
Notes:

Objectives are as stated above.

AIR01

© Copyright IBM Corp. 2019

Course materials may not be reproduced in whole or in part  
without the prior written permission of IBM.



## Navigation (1 of 3)

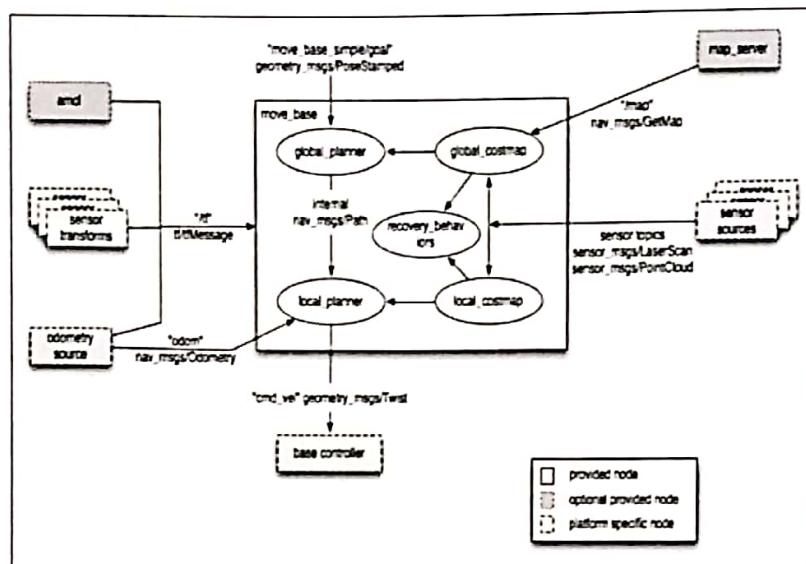


Figure: ROS Navigation Stack

Source: <https://robots.ieee.org/learn/types-of-robots/> <https://docs.fetchrobotics.com/gazebo.html>, <https://www.pirobot.org/blog/0014/>

Figure 5-2. Navigation (1 of 3)

AIR011.0

### Notes:

ROS navigation stack lets the robot autonomously drive from a given location in a map to a defined goal (final location). Navigation stack is a toolbox used for path planning and Simultaneous localization and mapping (SLAM). This section introduces the theory inside this stack and explains how to effectively implement SLAM using any type of robot. We begin with a brief introduction to the navigation stack along with some simple and straight-forward definitions and examples. Some other functionalities of the package will also be discussed with ROS concepts.

Moving around the world is the most basic thing which a robot can do. To achieve this successfully, the robot needs to know its present location and its destination. This is usually done by providing the robot with a map of the world, a starting location, and a goal location. The sensor data is generally used to build a map of the world. This data is also used to make robot navigate from one part of the world to another autonomously. The map and the ROS navigation packages together play an important role in achieving this.

## Student Notebook

The main components of the navigation stack are:

- The map\_server
- gmapping
- amcl
- global\_planner
- local\_planner
- move\_base

## Navigation (2 of 3)

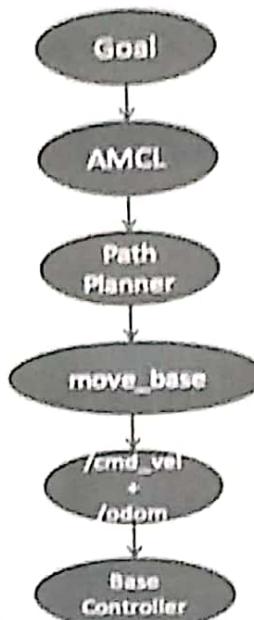


Figure: Navigation Main Steps

Source <https://robots.ieee.org/learn/types-of-robots/> <https://docs.fetchrobotics.com/gazebo.html>, <https://www.pirobot.org/blog/0014/>

Figure 5-3. Navigation (2 of 3)

AIR011.0

### Notes:

The navigation stack is capable of handling a differential drive and holonomic wheeled robots. It can also be used with biped robots for achieving localization, as long as the robot does not move sideways. To create the map and localization a planar laser must be mounted on the mobile base of the robot. Alternatively, Kinect can also be used to generate something equivalent of laser scans. The performance as per the literature can be best achieved with robots that are square or circular in design.

## Navigation (3 of 3)

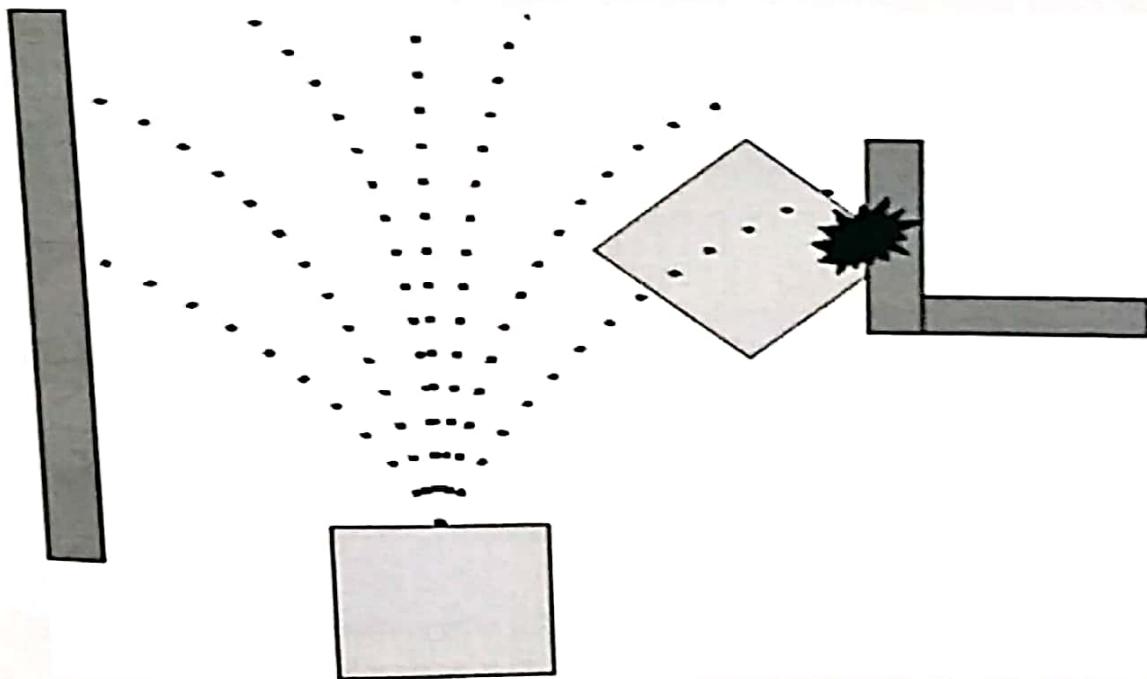


Figure: Trajectory roll out algorithm

Source: [http://library.isr.ist.utl.pt/docs/roswiki/base\\_local\\_planner.html](http://library.isr.ist.utl.pt/docs/roswiki/base_local_planner.html)

Figure 5-4. Navigation (3 of 3)

AIR011.0

### Notes:

Navigation planners:

- Global Planner
- Local Planner

The robot will be able to move through the map using two types of navigation - global and local. The global planner is used to create paths for a goal in the map or a far-off distance and the local planner is used to create paths in the nearby distances and avoid obstacles.

**Global Planner:** NavFn provides a fast interpolated navigation function that creates plans for a mobile base. Before the robot starts to move toward the next destination a global plan is computed. Dijkstra's algorithm is used by the planner which operates on a cost map to find a minimum cost plan from a start point to an end point in a grid. The global planner generates a series of waypoints which is followed by the local planner.

**Local Planner:** Chooses appropriate velocity commands for the robot to traverse the current segment of the global path. Combines sensory and odometry data with both global and local cost maps. It can recompute the robot's path on the fly to keep the robot from striking objects yet still allowing it to reach its destination.

Trajectory Rollout and Dynamic Window algorithm is usually used for implementing this.

Six steps of trajectory algorithm:

- Sampling
- Forward simulation
- Evaluation of the trajectory
- Discarding the unwanted
- Picking the best
- Repeat

There are six steps in the Trajectory Rollout algorithm, which are as given below:

- Discretely Sample the Robot's control space ( $dx, dy, d\theta$ ).
- For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
- Evaluate each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed.
- Discard illegal trajectories (those that collide with obstacles).
- Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
- Rinse and repeat.

The file `base_local_planner.yaml` contains a large number of ROS Parameters that can be set to customize the behavior of the base local planner. This is Grouped into several categories:

- Robot configuration
- Goal tolerance
- Forward simulation
- Trajectory scoring
- Oscillation prevention
- Global plan

#### **Costmap:**

A data structure that represents places that are safe for the robot to be in a grid of cells. It is based on the occupancy grid map of the environment and user specified inflation radius. There are two types of costmaps in ROS:

- Global costmap is used for global navigation.
- Local costmap is used for local navigation.

Each cell in the costmap has an integer value in the range [0 (FREE\_SPACE), 255 (UNKNOWN)].

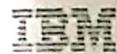
## Student Notebook

---

### Why Maps?

Building maps is one of the fundamental problems in mobile robotics. Maps allow robots to efficiently carry out their tasks, such as localization, path planning, activity planning, etc. There are different ways to create a map of the environment.

- Cellular Decomposition: Decompose free space for path planning. There are two possible ways:
  - Exact decomposition in which we cover the free space exactly. For example: trapezoidal decomposition, meadow map.
  - Approximate decomposition: Represent part of the free space, needed for navigation. For example: grid maps, quadtrees, Voronoi graphs.
- Occupancy Grid map: Maps the environment as a grid of cells. These Cell sizes typically range from 5 to 50 cm. Each cell holds a probability value that the cell is occupied in the range [0,100]. Unknown is indicated by -1. Usually unknown areas are areas that the robot sensors cannot detect (beyond obstacles).



IBM ICE (Innovation Centre for Education)

## Simultaneous localization and mapping

- Various SLAM techniques:
  - EKF SLAM.
  - Fast SLAM.
  - Graph-based SLAM.
  - Topological SLAM (mainly place recognition).
  - Scan Matching / Visual Odometry (only locally consistent maps).
  - Approximations for SLAM: Local submaps, Sparse extended information filters.
  - Sparse links, Thin junction tree filters, etc.

Figure 5-5. Simultaneous localization and mapping

AIR011.0

### Notes:

Localization is the problem of estimating the pose of the robot relative to a map. Localization is not terribly sensitive to the exact placement of objects so it can handle small changes to the locations of objects. ROS uses the amcl package for localization.

The simultaneous localization and mapping (SLAM) problem seeks to find a solution if it is possible for a mobile robot to be placed at an unknown location in an unknown environment so that the robot incrementally build a consistent map of this environment while simultaneously determining its location within this map. A solution to the SLAM problem has been seen as a means to make a robot truly autonomous.

The "solution" of the SLAM problem has been one of the notable successes of the robotics community over the past decade. SLAM has been formulated and solved as a theoretical problem in a number of different forms. SLAM is a process by which a mobile robot can build a map of an environment and at the same time use this map to deduce its location. In SLAM, both the trajectory of the platform and the location of all landmarks are estimated online without the need for any a priori knowledge of location.

There are various SLAM techniques, namely:

- EKF SLAM
- Fast SLAM
- Graph-based SLAM
- Topological SLAM (mainly place recognition)
- Scan Matching / Visual Odometry (only locally consistent maps)

Approximations for SLAM: Local submaps, Sparse extended information filters, Sparse links, Thin junction tree filters, etc. Let's try to briefly discuss one of the Particle filter techniques which is widely used the Fast SLAM: Represent probability distribution as a set of discrete particles which occupy the state space.

- Main steps of the algorithm:
  - Start with a random distribution of particles.
  - Compare particle's prediction of measurements with actual measurements.
  - Assign each particle a weight depending on how well its estimate of the state agrees with the measurements.
  - Randomly draw particles from previous distribution based on weights creating a new distribution.
  - Efficient: scales logarithmically with the number of landmarks in the map.



## Setting up rviz for navigation stack

- 2D pose estimation.
- 2D nav goal.
- Static map.
- Particle cloud.
- Obstacles.
- Global plan.
- Local plan.
- Planner plan.
- Current goal.

Figure 5-6. Setting up rviz for navigation stack

AIR011.0

### Notes:

It is good practice to visualize the possible data and to know what the navigation stack does. We will discuss the visualization topic that we must add to rviz to see the correct data sent by the navigation stack.

**2D pose estimation:** The 2D pose estimate allows us to initialize the localization system used by the navigation stack by setting the pose of the robot in the world. The navigation stack waits for the new pose of a new topic with the name initial pose. This topic is sent using the rviz windows where we previously changed the name of the topic.

**2D nav goal:** The 2D nav goal (G shortcut) allows the user to send a goal to the navigation by setting a desired pose for the robot to achieve. The navigation stack waits for a new goal with the new topic's name.

**Static Map:** This displays the static map that is being served by the map\_server, if one exists. When we add this visualization, we will see the map we captured.

**Particle Cloud:** It displays the particle cloud used by the robot's localization system. The spread of the cloud represents the localization system's uncertainty about the robot's pose. A cloud that spreads out a lot reflects high uncertainty, while a condensed cloud represents low certainty.

**Obstacles:** It shows the obstacles that the navigation stack sees in its costmap. For the robot to avoid collision, the robot's footprint should never intersect with a cell that contains an obstacle.

**Global Plan:** It shows the portion of the global plan that the local planner is currently pursuing. During the movement the robots may find obstacles and the navigation stack will recalculate a new path to avoid collisions and try to follow the global plan.

**Local Plan:** It shows the trajectory associated with the velocity commands currently being commanded to the robot by the local planner. We can know whether the robot is moving and the approximate velocity.

**Planner Plan:** It displays the full plan for the robot computed by the global planner. We will see that it is similar to the global plan.

**Current Goal:** It shows the goal pose that the navigation stack is attempting to achieve. We can use it to know the final position of the robot.

# Adaptive Monte Carlo Localization

IBM ICE (Innovation Centre for Education)



- Monte Carlo localization (MCL), also known as particle filter localization.
- It is an algorithm for robots to localize using a particle filter.



Figure: (AMCL) Adaptive Monte Carlo Localization

Source: <https://answers.ros.org/question/216813/amcl-acceptable-robot-speed/>

Figure 5-7. Adaptive Monte Carlo Localization

AIR0110

## Notes:

Monte Carlo localization (MCL), also known as particle filter localization, is an algorithm for robots to localize using a particle filter. Given a map of the environment, the algorithm estimates the position and orientation of a robot as it moves and senses the environment. The algorithm uses a particle filter to represent the distribution of likely states, with each particle representing a possible state, that represents the hypothesis of where the robot is. The algorithm generally starts with a uniform random distribution of particles over the configuration space, meaning the robot has no information about where it is and assumes it is equally likely to be at any point in space.

Whenever the robot moves, it shifts the particles to predict its new state after the movement. Whenever the robot senses something, the particles are resampled based on recursive Bayesian estimation, i.e., how well the actual sensed data correlate with the predicted state. Ultimately, the particles should converge towards the actual position of the robot.

# Avoiding obstacles

- Avoiding obstacles during the movement.

IBM

IBM T.J. Watson Research Center for Robotics

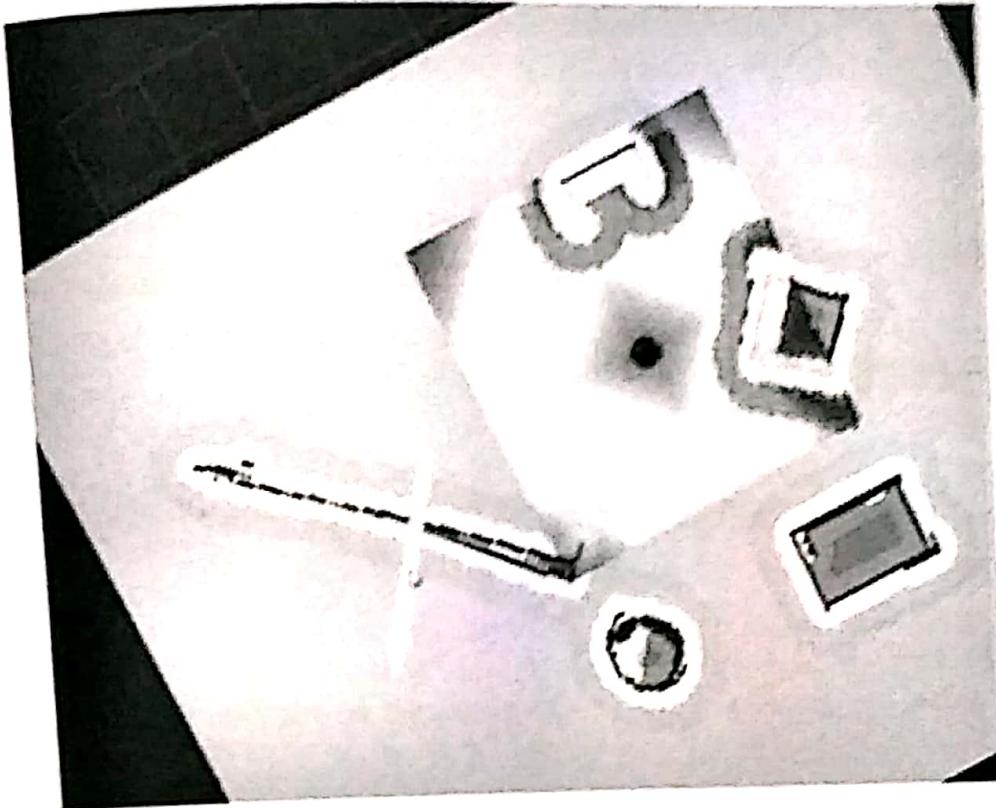


Figure: Avoiding obstacles

Source: <https://learn.turtlebot.com/2015/02/03/10/>

AIR011.0

Figure 5-8. Avoiding obstacles

## Notes:

A great functionality of navigation stack is the recalculation of the path if it finds obstacles during the movement. We can add this feature in the simulator. The navigation stack detects the new obstacle and automatically creates an alternative path.

# Speech recognition and synthesis

- Speech: Perception is that the speech is built with words and each word consists of phones.
  - Acoustic properties - phones.
  - Diphones.
  - Three states in a phone.
  - Tri phones.
  - Phones → subwords → words.
- Recognition process:
  - Take a waveform.
  - Split it by utterances by silences.
  - Then try to recognize what's being said in each utterance.
  - Take all possible combinations of word and try to match them with audio.

Figure 5-9. Speech recognition and synthesis

AIR011.0

## Notes:

Speech is perceived to be built with words and each word consists of phones .

- Speech is a audio stream which is continuous where stable states mix with dynamically changes states.
- Acoustic properties of a waveform corresponding to a phone depends on factors like phone context, speakers style of speech etc.
- Transitions between the words are more informative than stable regions.
- Diphones - parts of phones between two consecutive phones.
- Three states in a phone – first part depends on the preceding phone – the middle part is stable and next part depends on the subsequent phone.
- Sometimes the phones are considered in context – these are called triphones or even quinphones.
- For computational purpose it is helpful to detect parts of triphones instead of triphones as a whole, for example if you want to create a detector for the beginning of a triphone and share it across many triphones. The whole variety of sound detectors can be represented by a small amount of distinct short sound detectors. Usually we use 4000 distinct short sound detectors to compose detectors for triphones. We call those detectors senones. A senone's dependence on context can be more complex than just the left and right context. It can be a rather complex function defined by a decision tree, or in some other ways.

## Student Notebook

- Subwords form words. Words are important in speech recognition because they reduce computation of phones significantly. If there are 40 phones and an average word has 7 phones, there must be 40<sup>7</sup> words. Luckily, even people with a rich vocabulary rarely use more than 20k words in general, which makes recognition way more feasible.

### Matching Process:

- Features
- Feature vectors
- Concept of models
- How well does the model describe reality?
- Can the model be made better of its internal model problems?
- How adaptive is the model if conditions change?

System is tested on a test database that is meant to represent the target task correctly. Following characteristics are used:

- Word error rate
- Accuracy

### ROC curves

Word error rate. Let's assume we have an original text and a recognition text with a length of  $N$  words.  $I$  is the number of inserted words,  $D$  is the number of deleted words and  $S$  represent the number of substituted words. With this, the word error rate can be calculated as:

$$WER = (I + D + S) / N$$

The WER is usually measured in percent.

Accuracy: It is almost the same as the word error rate, but it doesn't take insertions into account.

$$Accuracy = (N - D - S) / N$$

For most tasks, the accuracy is a worse measure than the WER, since insertions are also important in the final results. However, for some tasks, the accuracy is a reasonable measure of the decoder performance.

Speed: Suppose an audio file has a recording time (RT) of 2 hours and the decoding took 6 hours. Then the speed is counted as  $3 \times RT$ .

ROC curves: When we talk about detection tasks, there are false alarms and hits/misses. To illustrate these, ROC curves are used. Such a curve is a diagram that describes the number of false alarms versus the number of hits. It tries to find the optimal point where the number of false alarms is small and the number of hits matches 100%.



IBM ICE (Innovation Centre for Education)

## Checkpoint (1 of 2)

### Multiple choice questions:

1. Which of these is not a part of the ROS navigation Stack.
  - a) Global\_planner
  - b) Local\_planner
  - c) AMCL
  - d) Gazebo
  
2. gmapping provides \_\_\_\_\_.
  - a) Map data as a ROS Service
  - b) Laser based SLAM
  - c) Probabilistic localization system
  - d) None of the above
  
3. Performance as per literature can be best achieved with robots that are \_\_\_\_\_.
  - a) Square
  - b) Circular
  - c) All of the above
  - d) None of the above

Figure 5-10. Checkpoint (1 of 2)

AIR011.0

### Notes:

Write your answers here:

- 1.
- 2.
- 3.

## Checkpoint (2 of 2)

Fill in the blanks:



IBM ICE (Innovation Centres for Education)

1. \_\_\_\_\_ provides a fast interpolated navigation function.
2. Local planner uses algorithm named \_\_\_\_\_.
3. Robot configuration is a parameter defined for \_\_\_\_\_.
4. \_\_\_\_\_ is one of the method of creating a map of the environment.

True or False:

1. EKF SLAM is one of the SLAM techniques. True/False
2. Monte Carlo localization is also known as random filter localization. True/False
3. There are three states in a phone. True/False

Figure 6-11. Checkpoint (2 of 2)

AIR011.0

### Notes:

Write your answers here:

Fill in the blanks:

- 1.
- 2.
- 3.
- 4.

True or False:

- 1.
- 2.
- 3.

## Question bank

### Two mark questions:

1. Why is ROS Navigation stack used for?
2. What is AMCL.
3. What is the use of map\_server?
4. What does gmapping used for?

### Four mark questions:

1. Which are the various packages of navigation stack.
2. Describe the two types of navigation.
3. Explain the concept speech recognition and synthesis.
4. Explain the need of Maps and how do you create them?

### Eight mark questions:

1. Explain the various steps of the trajectory rollout algorithm.
2. Explain SLAM in detail.

Figure 5-12. Question bank

AIR011.0

### Notes:

## Unit summary

Having completed this unit, you should be able to:

- Understand the Simultaneous Localization And Mapping problem (SLAM)
- Gain knowledge on developing solution for the SLAM problem and implement it
- Understand the concept of Speech Recognition and Synthesis and implement it
- Gain an insight into how ROS could be used for such application development

AIR011.0

Figure 5-13. Unit summary

### Notes:

Unit summary is as stated above.

## Appendix A. Checkpoint solutions

### Unit 1, "System Modeling"

Solutions for Figure 1-30, "Checkpoint (1 of 2)," on page 1-49



IBM ICE (Innovation Centre for Education)

### Checkpoint solutions (1 of 2)

#### Multiple choice questions:

1. Which of these can be called a system .

- a) Human body
- b) Pick and place robot
- c) Graph database
- d) All of these

2. Who coined the term robot .

- a) Karel Capek
- b) Romi K
- c) Isac Asimo
- d) None of the above

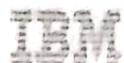
3. NHS means

- a) Nested Home Controller
- b) Nested Hierarchical Controller
- c) New Home controller
- d) None of the above

*solutions*

## Solutions for Figure 1-31, "Checkpoint (2 of 2)," on page 1-50

## Checkpoint solutions (2 of 2)



IBM ICE (Innovation Centre for Education)

Fill in the blanks:

1. NHC and RCS architecture are best suited for semi-autonomous controlled robots
2. Device attached to the robot's wrist to perform a specific task are called grippers.
3. A robot, which has two parallel rotary joints to provide compliance in a plane is called SCARA Robot.
4. Repetitability is the ability of a robot to repeatedly position itself when asked to perform a task multiple times.

True or False:

1. Reactive paradigm did not have planning or any reasoning functions. True
2. The Subconscious Thought module receives the information from the Sensory System and conveys commands to the Muscular System. True
3. Kinematics means the study of motion without regards to the forces or torques. True

## Unit 2, "Artificial intelligence for robotics engineering"

Solutions for Figure 2-67, "Checkpoint (1 of 2)," on page 2-103



IBM ICE (Innovation Centre for Education)

### Checkpoint solutions (1 of 2)

Multiple choice questions:

1. BFS stands for \_\_\_\_\_.

- a) Binary first search
- b) Breadth first search
- c) Block first statement
- d) All of these above

2. The major components of an AI production system are \_\_\_\_\_.

- a) A global database
- b) A set of production rules
- c) A control system
- d) All of these above

3. The generate-and-test is a: \_\_\_\_\_.

- a) Depth first search
- b) Exhaustive search
- c) Binary first search
- d) Both a) and b)

## solutions for Figure 2-68, "Checkpoint (2 of 2)," on page 2-104

### Checkpoint solutions (2 of 2)



IBM ICE (Innovation Centre for Education)

Fill in the blanks:

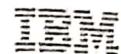
1. A state can have a number of successor states.
2. Heckerman describes a case where a leading expert on lymph-node pathology scoffs at a program's diagnosis of an especially difficult case.
3. Search tree is helpful to think of the search process as building up a search tree of routes through the state space graph.
4. DFS expands the leaf node with the highest path cost so far, and keeps going until a goal node is generated.

True or False:

1. AI is defined as the science and engineering of making machines intelligent with the help of intelligent agents' programs. True
2. Computation of heuristic function can be done with infinite amount of computation. False
3. Medical diagnosis programs based on probabilistic analysis have been able to perform at the level of an expert physician in several areas of medicine. True

## **Unit 3, "Components of an Intelligent Robotic System"**

**Solutions for Figure 3-92, "Checkpoint (1 of 2)," on page 3-123**



IBM ICE (Innovation Centre for Education)

### **Checkpoint solutions (1 of 2)**

#### **Multiple choice questions:**

1. Robotics is an inter disciplinary branch of engineering and science that deals with \_\_\_\_\_.

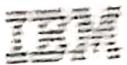
- a) Construction
- b) Operation
- c) Use of robots
- d) All the above

2. Wrist has three degrees of freedom \_\_\_\_\_.

- a) Roll
- b) Pitch
- c) Both a) and b)
- d) None of these

3. Machine learning is classified into two areas \_\_\_\_\_.

- a) AI type learning on symbolic computation
- b) Neural network
- c) Both a) and b)
- d) None of these



IBM ICE (Innovation Centre for Education)

## Checkpoint solutions (2 of 2)

Fill in the blanks:

1. The traces of the first robot can be found deep into the 18th century.
2. Inverse kinematics is a much more difficult problem than forward kinematics.
3. A software has been developed to effectively use the information from the fingertip system.
4. The dominant lag in the force feedback path generates a phase lead around the force reflection path and so eases the operator's problems in stabilizing the system.

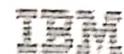
True or False:

1. Force control is a central requirement if robot arms are to use tools or interact with work-pieces in an unstructured environment. True
2. The forward kinematics problem is quite complicated and there is complexity in deriving the equations. False
3. An objective of high performance tele-operation is to give a human operator a sense of feel which can aid in the implementation of a remote task. True

## **Unit 4, "Robot Operating System (ROS)"**

**Solutions for Figure 4-25, "Checkpoint (1 of 2)," on page 4-38**

### **Checkpoint solutions(1 of 2)**



IBM ICE (Innovation Centre for Education)

#### **Multiple choice questions:**

1. Which of these is not a widely used simulator?
  - a) Arbotix
  - b) Stage
  - c) Maxwell
  - d) Gazebo
  
2. Robot operating system (ROS) is an \_\_\_\_\_ platform.
  - a) Framework Development
  - b) Application Development
  - c) Interaction Design (ID)
  - d) None of the above
  
3. ROS has got tools for
  - a) Simulation
  - b) Visualization
  - c) Debugging
  - d) All of the above

## Solutions for Figure 4-26, "Checkpoint (2 of 2)," on page 4-39



IBM ICE (Innovation Centre for Education)

### Checkpoint solutions (2 of 2)

Fill in the blanks:

1. ROS packages are maintained using VCS
2. Nodes are the process that perform computation
3. Scalar Values can be plotted using Time series plot
4. rviz is one of the 3D visualization tool.

True or False:

1. Actuators like Dynamixel servos are also supported in ROS . True
2. There is a package named MoveIt! for robot motion planning. False
3. Modeling in ROS is performed using URDF. True

## **Unit 5, "Navigation, SLAM and Speech Recognition and Synthesis )"**

**Solutions for Figure 5-10, "Checkpoint (1 of 2)," on page 5-17**



IBM ICE (Innovation Centre for Education)

### **Checkpoint solutions (1 of 2)**

**Multiple choice questions:**

1. Which of these is not a part of the ROS navigation Stack.

- a) Global\_planner
- b) Local\_planner
- c) AMCL
- d) Gazebo

2. gmapping provides .

- a) Map data as a ROS Service
- b) Laser based SLAM
- c) Probabilistic localization system
- d) None of the above

3. Performance as per literature can be best achieved with robots that are

- a) Square
- b) Circular
- c) All of the above
- d) None of the above

## Solutions for Figure 5-11, "Checkpoint (2 of 2)," on page 5-18

### Checkpoint solutions (2 of 2)



IBM ICE (Innovation Centre for Education)

Fill in the blanks:

1. NavFn provides a fast interpolated navigation function
2. Local planner uses algorithm named Trajectory rollout and dynamic window algorithm
3. Robot configuration is a parameter defined for local planner
4. Cellular decomposition is one of the method of creating a map of the environment.

True or False:

1. EKF SLAM is one of the SLAM techniques. True
2. Monte Carlo localization is also know as random filter localization. False
3. There are three states in a phone. True