UNIT-I

Introduction to AI:

What is artificial intelligence?

Artificial Intelligence is the branch of computer science concerned with making computers behave like humans.

Major AI textbooks define artificial intelligence as "the study and design of intelligent agents," where an **intelligent agent** is a system that **perceives** its **environment** and **takes actions** which maximize its chances of success. **John McCarthy**, who coined the term in 1956, defines it as "the science and engineering of making intelligent machines,especially intelligent computer programs."

The definitions of AI according to some text books are categorized into four approaches and are summarized in the table below :

Systems that think like humans “The exciting new effort to make computers think … machines with minds,in the full and literal sense.”(Haugeland,1985)	**Systems that think rationally** “The study of mental faculties through the use of computer models.” (Charniak and McDermont,1985)
Systems that act like humans The art of creating machines that perform functions that require intelligence when performed by people.”(Kurzweil,1990)	**Systems that act rationally** **“**Computational intelligence is the study of the design of intelligent agents.”(Poole et al.,1998)

The four approaches in more detail are as follows :

(a) Acting humanly : The Turing Test approach

- Test proposed by Alan Turing in 1950
- The computer is asked questions by a human interrogator.

The computer passes the test if a human interrogator,after posing some written questions,cannot tell whether the written responses come from a person or not. Programming a computer to pass ,the computer need to possess the following capabilities :

- **Natural language processing** to enable it to communicate successfully in English.
- **Knowledge representation** to store what it knows or hears
- **Automated reasoning** to use the stored information to answer questions and to draw new conclusions.
- **Machine learning** to adapt to new circumstances and to detect and extrapolate patterns

To pass the complete Turing Test,the computer will need

- **Computer vision** to perceive the objects,and
- **Robotics** to manipulate objects and move about.

(b)Thinking humanly : The cognitive modeling approach

We need to get inside actual working of the human mind :

(a) through introspection – trying to capture our own thoughts as they go by;

(b) through psychological experiments

Allen Newell and Herbert Simon,who developed **GPS**,the "**General Problem Solver**" tried to trace the reasoning steps to traces of human subjects solving the same problems. The interdisciplinary field of **cognitive science** brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind

(c) Thinking rationally : The "laws of thought approach"

The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking",that is irrefuatable reasoning processes. His **syllogism** provided patterns for argument structures that always yielded correct conclusions when given correct premises—for example,"Socrates is a man;all men are mortal;therefore Socrates is mortal.".

These laws of thought were supposed to govern the operation of the mind;their study initiated a field called **logic.**

(d) Acting rationally : The rational agent approach

An **agent** is something that acts. Computer agents are not mere programs ,but they are expected to have the following attributes also : (a) operating under autonomous control, (b) perceiving their environment, (c) persisting over a prolonged time period, (e) adapting to change.
A **rational agent** is one that acts so as to achieve the best outcome.

<u>The foundations of Artificial Intelligence</u>

The various disciplines that contributed ideas,viewpoints,and techniques to AI are given below :

Philosophy(428 B.C. – present)

Aristotle (384-322 B.C.) was the first to formulate a precise set of laws governing the rational part of the mind. He developed an informal system of syllogisms for proper reasoning,which allowed one to generate conclusions mechanically,given initial premises.

	Computer	Human Brain
Computational units Storage units	1 CPU,10^8 gates 10^{10} bits RAM	10^{11} neurons 10^{11} neurons

	10^{11} bits disk	10^{14} synapses
Cycle time	10^{-9} sec	10^{-3} sec
Bandwidth	10^{10} bits/sec	10^{14} bits/sec
Memory updates/sec	10^{9}	10^{14}

Table 1.1 A crude comparison of the raw computational resources available to computers(*circa* 2003) and brain. The computer's numbers have increased by at least by a factor of 10 every few years. The brain's numbers have not changed for the last 10,000 years.

Brains and digital computers perform quite different tasks and have different properties. Tablere 1.1 shows that there are 10000 times more neurons in the typical human brain than there are gates in the CPU of a typical high-end computer. Moore's Law predicts that the CPU's gate count will equal the brain's neuron count around 2020.

Psycology(1879 – present)

The origin of scientific psychology are traced back to the wok if German physiologist Hermann von Helmholtz(1821-1894) and his student Wilhelm Wundt(1832 – 1920)

In 1879,Wundt opened the first laboratory of experimental psychology at the university of Leipzig. In US,the development of computer modeling led to the creation of the field of **cognitive science**. The field can be said to have started at the workshop in September 1956 at MIT.

Computer engineering (1940-present)

For artificial intelligence to succeed, we need two things: intelligence and an artifact. The computer has been the artifact of choice.

A1 also owes a debt to the software side of computer science, which has supplied the operating systems, programming languages, and tools needed to write modern programs

Control theory and Cybernetics (1948-present)

Ktesibios of Alexandria (c. 250 B.c.) built the first self-controlling machine: a water clock with a regulator that kept the flow of water running through it at a constant, predictable pace. Modern control theory, especially the branch known as stochastic optimal control, has as its goal the design of systems that maximize an **objective function** over time.

Linguistics (1957-present)

Modem linguistics and AI, then, were "born" at about the same time, and grew up together, intersecting in a hybrid field called **computational linguistics** or **natural language processing.**

<u>The History of Artificial Intelligence:</u>

The gestation of artificial intelligence (1943-1955)

There were a number of early examples of work that can be characterized as AI, but it was Alan Turing who first articulated a complete vision of A1 in his 1950 article "Comput-ing Machinery and Intelligence." Therein, he introduced the Turing test, machine learning, genetic algorithms, and reinforcement learning.

The birth of artificial intelligence (1956)

McCarthy convinced Minsky, Claude Shannon, and Nathaniel Rochester to help him bring together U.S. researchers interested in automata theory, neural nets, and the study of intelligence. They organized a two-month workshop at Dartmouth in the summer of 1956. 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.**

Early enthusiasm, great expectations (1952-1969)

The early years of A1 were full of successes-in a limited way.

General Problem Solver (**GPS**) was a computer program created in 1957 by Herbert Simon and Allen Newell to build a universal problem solver machine. The order in which the program considered subgoals and possible actions was similar to that in which humans approached the same problems. Thus, GPS was probably the first program to embody the "thinking humanly" approach.

At IBM, Nathaniel Rochester and his colleagues produced some of the first A1 pro-
grams. Herbert Gelernter (1959) constructed the Geometry Theorem Prover, which
was able to prove theorems that many students of mathematics would find quite tricky.

Lisp was invented by John McCarthy in 1958 while he was at the Massachusetts Institute of Technology (MIT). In 1963, McCarthy started the AI lab at Stanford.

Tom Evans's ANALOGY program (1968) solved geometric analogy problems that appear in IQ tests, such as the one in Figure 1.1

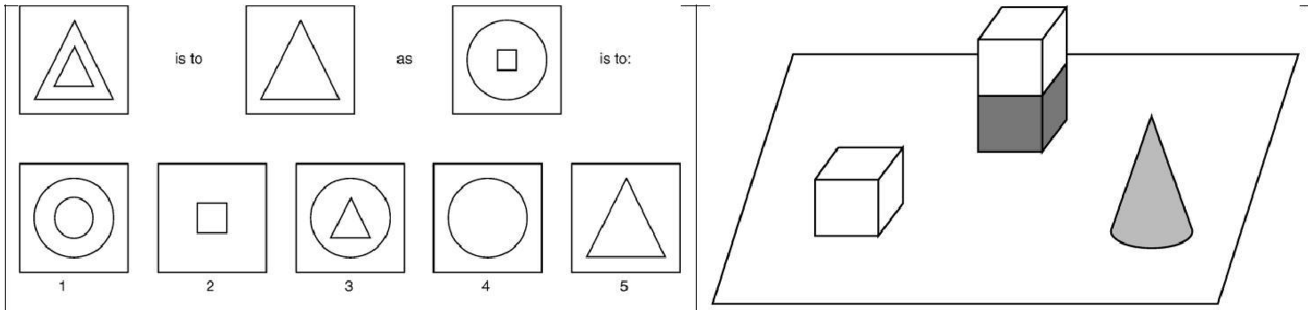


Figure 1.1 The Tom Evan’s ANALOGY program could solve geometric analogy problems as shown.

A dose of reality (1966-1973)

From the beginning, AI researchers were not shy about making predictions of their
coming successes. The following statement by Herbert Simon in 1957 is often quoted:

“It is not my aim to surprise or shock you-but the simplest way I can summarize is to say that there are now in the world machines that think, that learn and that create. Moreover, their ability to do these things is going to increase rapidly until-in a visible future-the range of problems they can handle will be coextensive with the range to which the human mind has been applied.

Knowledge-based systems: The key to power? (1969-1979)

Dendral was an influential pioneer project in artificial intelligence (AI) of the 1960s, and the computer software **expert system** that it produced. Its primary aim was to help organic chemists in identifying unknown organic molecules, by analyzing their mass spectra and using knowledge of chemistry. It was done at Stanford University by Edward Feigenbaum, Bruce Buchanan, Joshua Lederberg, and Carl Djerassi.

A1 becomes an industry (1980-present)

In 1981, the Japanese announced the "Fifth Generation" project, a 10-year plan to build intelligent computers running Prolog. Overall, the A1 industry boomed from a few million dollars in 1980 to billions of dollars in 1988.

The return of neural networks (1986-present)

Psychologists including David Rumelhart and Geoff Hinton continued the study of neural-net models of memory.

A1 becomes a science (1987-present)

In recent years, approaches based on **hidden Markov models** (HMMs) have come to dominate the area. Speech technology and the related field of handwritten character recognition are already making the transition to widespread industrial and consumer applications.

The **Bayesian network** formalism was invented to allow efficient representation of, and rigorous reasoning with, uncertain knowledge.

The emergence of intelligent agents (1995-present)

One of the most important environments for intelligent agents is the Internet.

<u>The state of art</u>

What can A1 do today?

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 (Jonsson *et* al., 2000). 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 (Goodman and Keene, 1997).

Autonomous control: The ALVINN 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.

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.

Logistics Planning: During the Persian Gulf crisis of 1991, U.S. forces deployed a Dynamic Analysis and Replanning Tool, DART (Cross and Walker, 1994), 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 (DiGioia *et* al., 1996) 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 a hip replacement prosthesis.

Language understanding and problem solving: PROVERB (Littman *et al.*, 1999) 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.

INTELLIGENT AGENTS:

1.2.1 Agents and environments

An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and SENSOR acting upon that environment through **actuators.** This simple idea is illustrated in Figure 1.2.

- A human agent has eyes, ears, and other organs for sensors and hands, legs, mouth, and other body parts for actuators.
- A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.
- A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

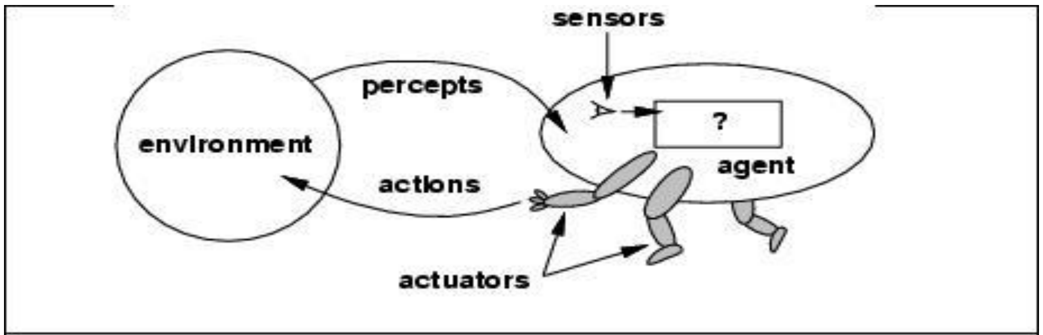


Figure 1.2 Agents interact with environments through sensors and actuators.

Percept

We use the term **percept** to refer to the agent's perceptual inputs at any given instant.

Percept Sequence

An agent's **percept sequence** is the complete history of everything the agent has ever perceived.

Agent function

Mathematically speaking, we say that an agent's behavior is described by the **agent function** that maps any given percept sequence to an action.

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

Agent program

Internally, The agent function for an artificial agent will be implemented by an **agent program.** It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running on the agent architecture.

To illustrate these ideas, we will use a very simple example-the vacuum-cleaner world shown in Figure 1.3. This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck, otherwise move to the other square. A partial tabulation of this agent function is shown in Figure 1.4.

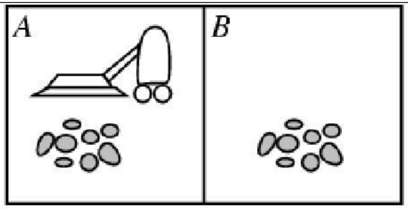


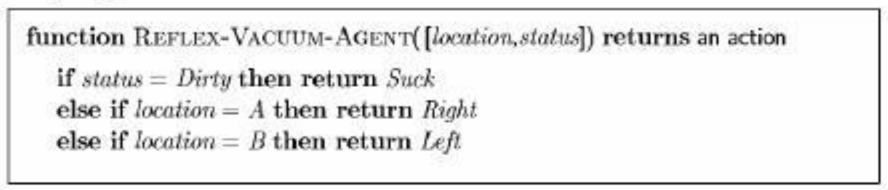
Figure 1.3 A vacuum-cleaner world with just two locations.

Agent function

Percept Sequence	**Action**
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
…	

Figure 1.4 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 1.3.

agent program



```
function REFLEX-VACUUM-AGENT([location,status]) returns an action
   if status = Dirty then return Suck
   else if location = A then return Right
   else if location = B then return Left
```

Rational Agent

A **rational agent** is one that does the right thing-conceptually speaking, every entry in the table for the agent function is filled out correctly. Obviously, doing the right thing is better than doing the wrong thing. The right action is the one that will cause the agent to be most successful.

Performance measures

A performance measure embodies the **criterion for success** of an agent's behavior. When an agent is plunked down in an environment, it generates a sequence of actions according to the percepts it receives. This sequence of actions causes the environment to go through a sequence of states. If the sequence is desirable, then the agent has performed well.

Rationality

What is rational at any given time depends on four things:

- The performance measure that defines the criterion of success.
- The agent's prior knowledge of the environment.
- The actions that the agent can perform.
- The agent's percept sequence to date.

This leads to a **definition of a rational agent:**

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.

Omniscience, learning, and autonomy

An **omniscient agent** knows the *actual* outcome of its actions and can act accordingly; but omniscience is impossible in reality.

Doing actions in order to modify future percepts-sometimes called **information gathering**-is an important part of rationality.

Our definition requires a rational agent not only to gather information, but also to **learn** as much as possible from what it perceives.

To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts, we say that the agent lacks autonomy. A rational agent should be **autonomous**-it should learn what it can to compensate for partial or incorrect prior knowledge.

Task environments

We must think about **task environments,** which are essentially the "**problems**" to which rational agents are the "**solutions**."

Specifying the task environment

The rationality of the simple vacuum-cleaner agent, needs specification of
- the performance measure
- the environment
- the agent's actuators and sensors.

PEAS

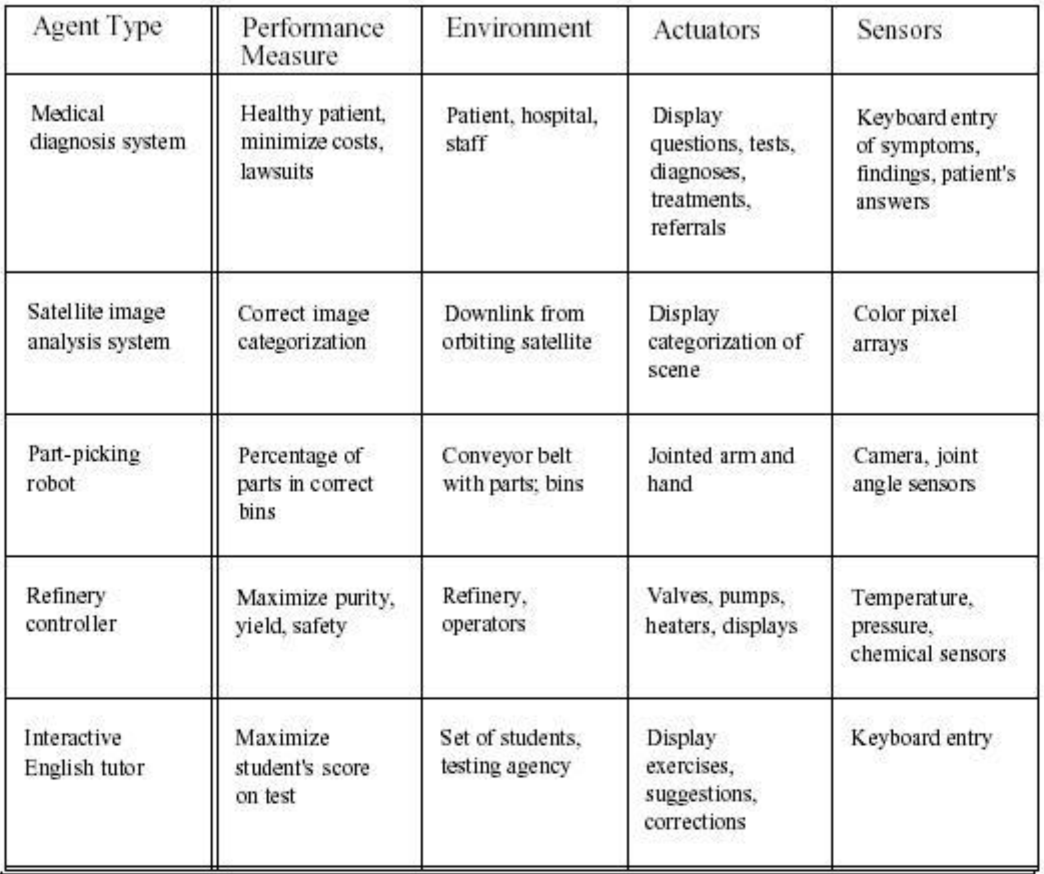
All these are grouped together under the heading of the **task environment.** We call this the **PEAS** (Performance, Environment, Actuators, Sensors) description.

In designing an agent, the first step must always be to specify the task environment as fully as possible.

Agent Type	**Performance Measure**	**Environments**	**Actuators**	**Sensors**
Taxi driver	Safe: fast, legal, comfortable trip,	Roads,other traffic,pedestrians,	Steering,accelerator, brake,	Cameras,sonar, Speedometer,GPS,

	maximize profits	customers	Signal,horn,display	Odometer,engine sensors,keyboards, accelerometer

Figure 1.5 PEAS description of the task environment for an automated taxi.



Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, minimize costs, lawsuits	Patient, hospital, staff	Display questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display categorization of scene	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Maximize purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Maximize student's score on test	Set of students, testing agency	Display exercises, suggestions, corrections	Keyboard entry

Figure 1.6 Examples of agent types and their PEAS descriptions.

Properties of task environments

- Fully observable vs. partially observable
- Deterministic vs. stochastic
- Episodic vs. sequential
- Static vs. dynamic
- Discrete vs. continuous
- Single agent vs. multi agent

Fully observable vs. **partially observable.**

If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable. A task environment is effectively fully observable if the sensors detect all aspects that are *relevant* to the choice of action;

An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.

Deterministic vs. **stochastic.**

If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic; other-wise, it is stochastic.

Episodic vs. **sequential**

In an **episodic task environment**, the agent's experience is divided into atomic episodes. Each episode consists of the agent perceiving and then performing a single action.Cru-cially, the next episode does not depend on the actions taken in previous episodes.

For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions;

In **sequential environments**, on the other hand, the current decision could affect all future decisions. Chess and taxi driving are sequential:

Discrete vs. **continuous.**

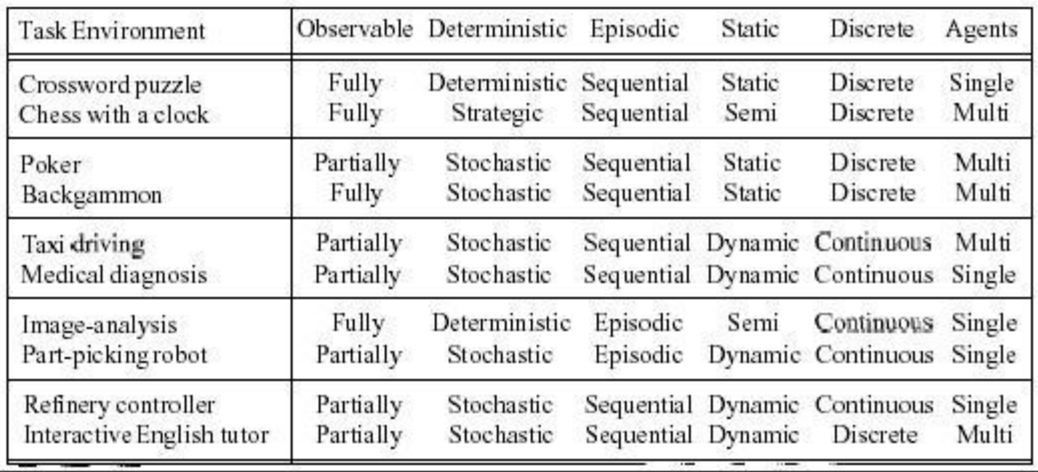
The discrete/continuous distinction can be applied to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent. For example, a discrete-state environment such as a chess game has a finite number of distinct states. Chess also has a discrete set of percepts and actions. Taxi driving is a continuous-state and continuous-time problem: the speed and location of the taxi and of the other vehicles sweep through a range of continuous values and do so smoothly over time. Taxi-driving actions are also continuous (steering angles, etc.).

Single agent vs. **multiagent.**

An agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two-agent environ-ment.

As one might expect, the hardest case is *partially observable, stochastic, sequential, dynamic, continuous*, and *multiagent.*

Figure 1.7 lists the properties of a number of familiar environments.



Task Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Crossword puzzle	Fully	Deterministic	Sequential	Static	Discrete	Single
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Poker	Partially	Stochastic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partially	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Image-analysis	Fully	Deterministic	Episodic	Semi	Continuous	Single
Part-picking robot	Partially	Stochastic	Episodic	Dynamic	Continuous	Single
Refinery controller	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Interactive English tutor	Partially	Stochastic	Sequential	Dynamic	Discrete	Multi

Figure 1.7 Examples of task environments and their characteristics.

Agent programs

The agent programs all have the same skeleton: they take the current percept as input from the sensors and return an action to the actuatom6 Notice the difference between the **agent program**, which takes the current percept as input, and the **agent function**, which takes the entire percept history. The agent program takes just the current percept as input because nothing more is available from the environment; if the agent's actions depend on the entire percept sequence, the agent will have to remember the percepts.

```
Function TABLE-DRIVEN_AGENT(percept) returns an action

    static: percepts, a sequence initially empty
            table, a table of actions, indexed by percept sequence

    append percept to the end of percepts
    action ← LOOKUP(percepts, table)
    return action
```

Figure 1.8 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns **an** action each time.

Drawbacks:

- **Table lookup** of percept-action pairs defining all possible condition-action rules necessary to interact in an environment
- **Problems**
  - Too big to generate and to store (Chess has about 10^120 states, for example)
  - No knowledge of non-perceptual parts of the current state
  - Not adaptive to changes in the environment; requires entire table to be updated if changes occur
  - Looping: Can't make actions conditional

- Take a long time to build the table
- No autonomy
- Even with learning, need a long time to learn the table entries

Some Agent Types

- **Table-driven agents**
  - use a percept sequence/action table in memory to find the next action. They are implemented by a (large) **lookup table**.
- **Simple reflex agents**
  - are based on **condition-action rules**, implemented with an appropriate production system. They are stateless devices which do not have memory of past world states.
- **Agents with memory**
  - have **internal state**, which is used to keep track of past states of the world.
- **Agents with goals**
  - are agents that, in addition to state information, have **goal information** that describes desirable situations. Agents of this kind take future events into consideration.
- **Utility-based agents**
  - base their decisions on **classic axiomatic utility theory** in order to act rationally.

Simple Reflex Agent

The simplest kind of agent is the **simple reflex agent.** These agents select actions on the basis of the *current* percept, ignoring the rest of the percept history. For example, the vacuum agent whose agent function is tabulated in Figure 1.10 is a simple reflex agent, because its decision is based only on the current location and on whether that contains dirt.

- Select action on the basis of *only the current* percept.
  E.g. the vacuum-agent
- Large reduction in possible percept/action situations(next page).
- Implemented through *condition-action rules*
  If dirty then suck

A Simple Reflex Agent: Schema

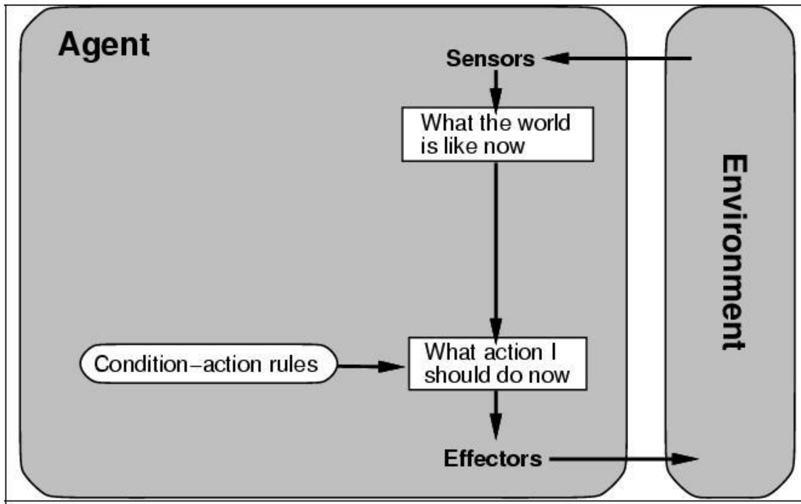


Figure 1.9 Schematic diagram of a simple reflex agent.

```
function SIMPLE-REFLEX-AGENT(percept) returns an action

static: rules, a set of condition-action rules
state ← INTERPRET-INPUT(percept) rule
← RULE-MATCH(state, rule) action ←
RULE-ACTION[rule]
return action
```

Figure 1.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

```
function REFLEX-VACUUM-AGENT ([location, status]) return an action
    if status == Dirty then return Suck
    else if location == A then return Right
    else if location == B then return Left
```

Figure 1.11 The agent program for a simple reflex agent in the two-state vacuum environment. This program implements the agent function tabulated in the figure 1.4.

❖ **Characteristics**

- Only works if the environment is fully observable.
- Lacking history, easily get stuck in infinite loops
- One solution is to randomize actions
-

Model-based reflex agents

The most effective way to handle partial observability is for the agent to *keep track of the part of the world it can't see now*. That is, the agent should maintain some sort of **internal state** that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.

Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program. First, we need some information about how the world evolves independently of the agent-for example, that an overtaking car generally will be closer behind than it was a moment ago. Second, we need some information about how the agent's own actions affect the world-for example, that when the agent turns the steering wheel clockwise, the car turns to the right or that after driving for five minutes northbound on the freeway one is usually about five miles north of where one was five minutes ago. This knowledge about "how the world working - whether implemented in simple Boolean circuits or in complete scientific theories-is called a **model** of the world. An agent that uses such a MODEL-BASED model is called a **model-based agent.**

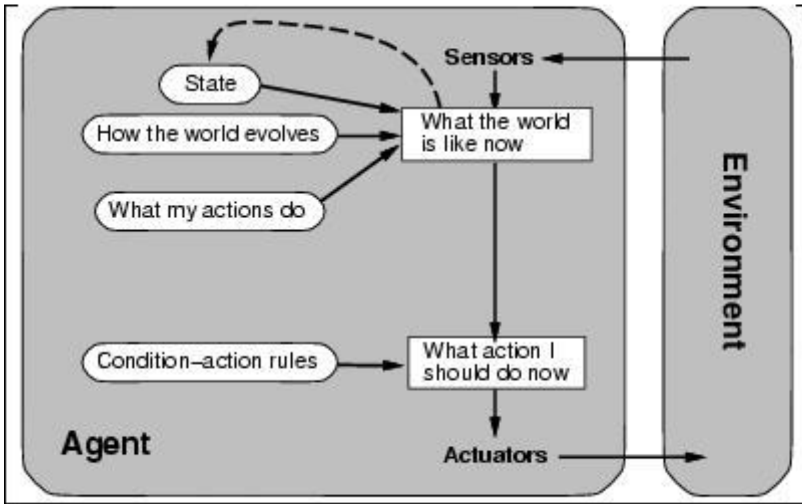


Figure 1.12 A model based reflex agent

```
function REFLEX-AGENT-WITH-STATE(percept) returns an action
static: rules, a set of condition-action rules
state, a description of the current world state
action, the most recent action.
state ← UPDATE-STATE(state, action, percept)
rule ← RULE-MATCH(state, rule)
```

```
action ← RULE-ACTION[rule]
return action
```

Figure 1.13 Model based reflex agent. It keeps track of the current state of the world using an internal model. It then chooses an action in the same way as the reflex agent.

Goal-based agents

Knowing about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to. In other words, as well as a current state description, the agent needs some sort of **goal** information that describes situations that are desirable-for example, being at the passenger's destination. The agent program can combine this with information about the results of possible actions (the same information as was used to update internal state in the reflex agent) in order to choose actions that achieve the goal. Figure 1.13 shows the goal-based agent's structure.

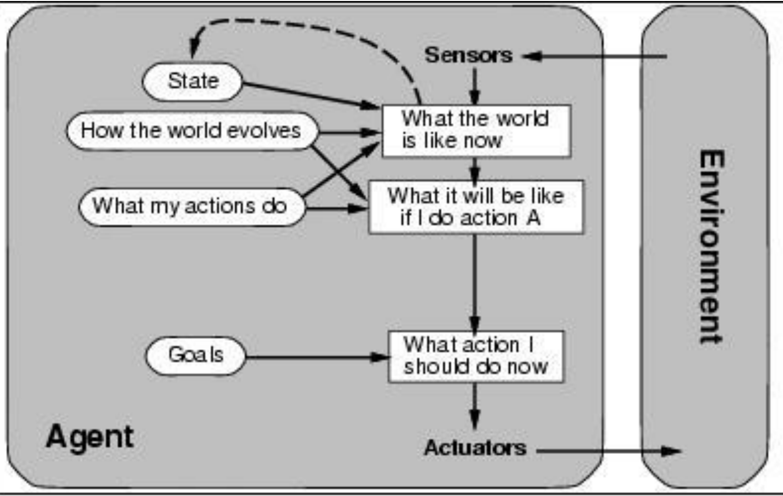

Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

Figure 1.14 A goal based agent

Utility-based agents

Goals alone are not really enough to generate high-quality behavior in most environments. For example, there are many action sequences that will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others. Goals just provide a crude binary distinction between "happy" and "unhappy" states, whereas a more general **performance measure** should allow a comparison of different world states according to exactly how happy they would make the agent if they could be achieved. Because "happy" does not sound very scientific, the customary terminology is to say that if one world state is preferred to another, then it has higher **utility** for the agent.

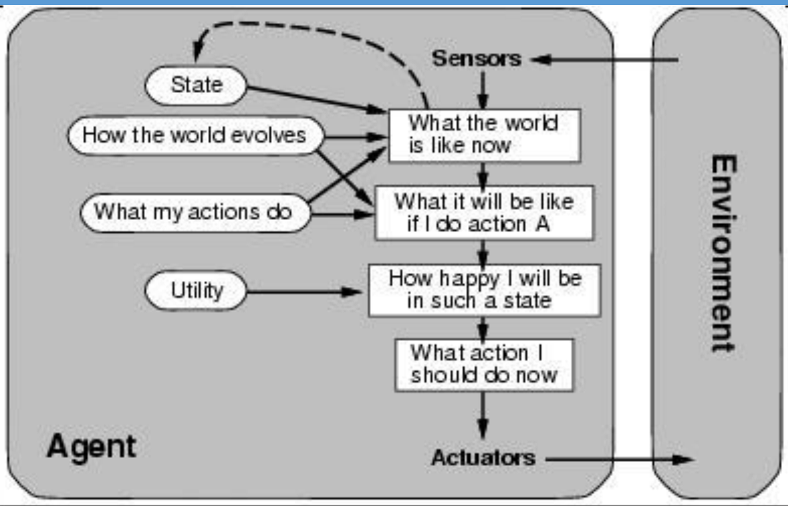

Sensors
State
What the world is like now
How the world evolves
What my actions do
What it will be like if I do action A
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Figure 1.15 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

- Certain goals can be reached in different ways.
  - Some are better, have a higher utility.
- Utility function maps a (sequence of) state(s) onto a real number.
- Improves on goals:
  - Selecting between conflicting goals
  - Select appropriately between several goals based on likelihood of success.

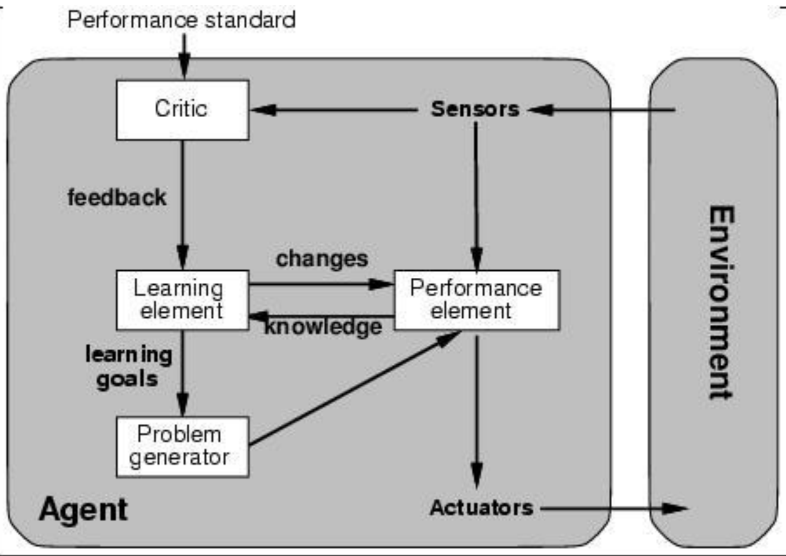


Figure 1.16 A general model of learning agents.

- All agents can improve their performance through **learning.**

A learning agent can be divided into four conceptual components, as shown in Figure 1.15 The most important distinction is between the **learning element**, which is responsible for making improvements, and the **performance element**, which is responsible for selecting external actions. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions. The learning element uses feedback from the **critic** on how the agent is doing and determines how the performance element should be modified to do better in the future.

The last component of the learning agent is the **problem generator**. It is responsible for suggesting actions that will lead to new and **informative experiences**. But if the agent is willing to explore a little, it might discover much better actions for the long run. The problem generator's job is to suggest these **exploratory actions**. This is what scientists do when they carry out experiments.

Summary: Intelligent Agents

- An **agent** perceives and acts in an environment, has an architecture, and is implemented by an agent program.
- Task environment – **PEAS (P**erformance**, E**nvironment, **A**ctuators, **S**ensors**)**
- The most challenging environments are inaccessible, nondeterministic, dynamic, and continuous.
- An **ideal agent** always chooses the action which maximizes its expected performance, given its percept sequence so far.
- An **agent program** maps from percept to action and updates internal state.
  - **Reflex agents** respond immediately to percepts.
    - simple reflex agents
    - model-based reflex agents
  - **Goal-based agents** act in order to achieve their goal(s).
  - **Utility-based agents** maximize their own utility function.
- All agents can improve their performance through **learning.**

Natural language processing:

Natural language processing (**NLP**) is a field of computer science, artificial intelligence, and linguistics concerned with the interactions between computers and human (natural) languages.

Many challenges in NLP involve natural language understanding, that is, enabling computers to derive meaning from human or natural language input, and others involve natural language generation.

NLP using machine learning

Modern NLP algorithms are based on machine learning, especially statistical machine learning.

The machine-learning paradigm calls instead for using general learning algorithms — often, although not always, grounded in statistical inference — to automatically learn such rules through the analysis of large *corpora* of typical real-world examples.

Many different classes of machine learning algorithms have been applied to NLP tasks.

Some of the earliest-used algorithms, such as decision trees, produced systems of hard if-then rules similar to the systems of hand-written rules that were then common.

Major tasks in NLP

A list of some of the most commonly researched tasks in NLP

Automatic summarization

Produce a readable summary of a chunk of text. Often used to provide summaries of text of a known type, such as articles in the financial section of a newspaper.

Machine translation

Automatically translate text from one human language to another.

Morphological segmentation

Separate words into individual morphemes and identify the class of the morphemes.

Named entity recognition (NER)

Given a stream of text, determine which items in the text map to proper names, such as people or places, and what the type of each such name is

Natural language generation

Convert information from computer databases into readable human language.

Natural language understanding

Convert chunks of text into more formal representations such as first-order logic structures that are easier for computer programs to manipulate. Natural language understanding involves the identification of the intended semantic from the multiple possible semantics which can be derived from a natural language expression which usually takes the form of organized notations of natural languages concepts.

Optical character recognition (OCR)

Given an image representing printed text, determine the corresponding text.

Part-of-speech tagging

Given a sentence, determine the part of speech for each word. Many words, especially common ones, can serve as multiple parts of speech.

Parsing

Determine the parse tree (grammatical analysis) of a given sentence.The grammar for natural lnguages is ambiguous and typical sentences have multiple possible analyses.

Question answering

Given a human-language question, determine its answer.

Relationship extraction

Given a chunk of text, identify the relationships among named entities.

Speech recognition

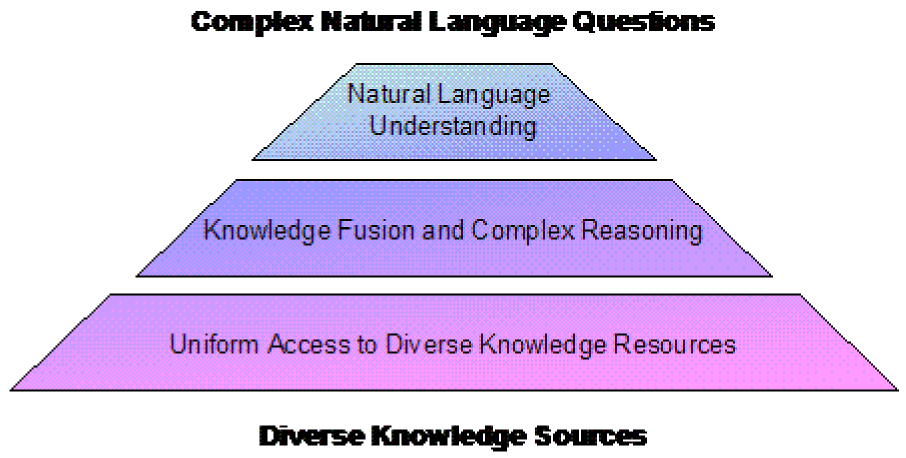
Given a sound clip of a person or people speaking, determine the textual representation of the speech. This is the opposite of text to speech and is one of the extremely difficult problems colloquially termed "AI-complete"

Information retrieval (IR)

This is concerned with storing, searching and retrieving information. It is a separate field within computer science (closer to databases), but IR relies on some NLP methods (for example, stemming).

Information extraction (IE)

This is concerned in general with the extraction of semantic information from text. This covers tasks such as named entity recognition, Coreference resolution, relationship extraction, etc.


Complex Natural Language Questions
Natural Language Understanding
Knowledge Fusion and Complex Reasoning
Uniform Access to Diverse Knowledge Resources
Diverse Knowledge Sources

Computer vision:

Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions.

A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image.

Computer vision has also been described as the enterprise of automating and integrating a wide range of processes and representations for vision perception.

As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems. Examples of applications of computer vision include systems for:

Controlling processes, *e.g.*, an industrial robot;

Navigation, *e.g.*, by an autonomous vehicle or mobile robot;

Detecting events, *e.g.*, for visual surveillance or people counting;

Organizing information, *e.g.*, for indexing databases of images and image sequences;

Modeling objects or environments, *e.g.*, medical image analysis or topographical modeling;

Computer vision system methods

The organization of a computer vision system is highly application dependent. Some systems are stand-alone applications which solve a specific measurement or detection problem, while others constitute a sub-system of a larger design which, for example, also contains sub-systems for control of mechanical actuators, planning, information databases, man-machine interfaces, etc. The specific implementation of a computer vision system also depends on if its functionality is pre-specified or if some part of it can be learned or modified during operation. Many functions are unique to the application. There are, however, typical functions which are found in many computer vision systems.

- **Image acquisition** – A digital image is produced by one or several image sensors, which, besides various types of light-sensitive cameras, include range sensors, tomography devices, radar, ultra-sonic cameras, etc. Depending on the type of sensor, the resulting image data is an ordinary 2D image, a 3D volume, or an image sequence. The pixel values typically correspond to light intensity in one or several spectral bands (gray images or colour images), but can also be related to various physical measures, such as depth, absorption or reflectance of sonic or electromagnetic waves, or nuclear magnetic resonance.
- **Pre-processing** – Before a computer vision method can be applied to image data in order to extract some specific piece of information, it is usually necessary to process the data in order to assure that it satisfies certain assumptions implied by the method. Examples are
  - Re-sampling in order to assure that the image coordinate system is correct.
  - Noise reduction in order to assure that sensor noise does not introduce false information.
  - Contrast enhancement to assure that relevant information can be detected.
  - Scale space representation to enhance image structures at locally appropriate scales.
- **Feature extraction** – Image features at various levels of complexity are extracted from the image data. Typical examples of such features are
  - Lines, edges and ridges.
  - Localized interest points such as corners, blobs or points.

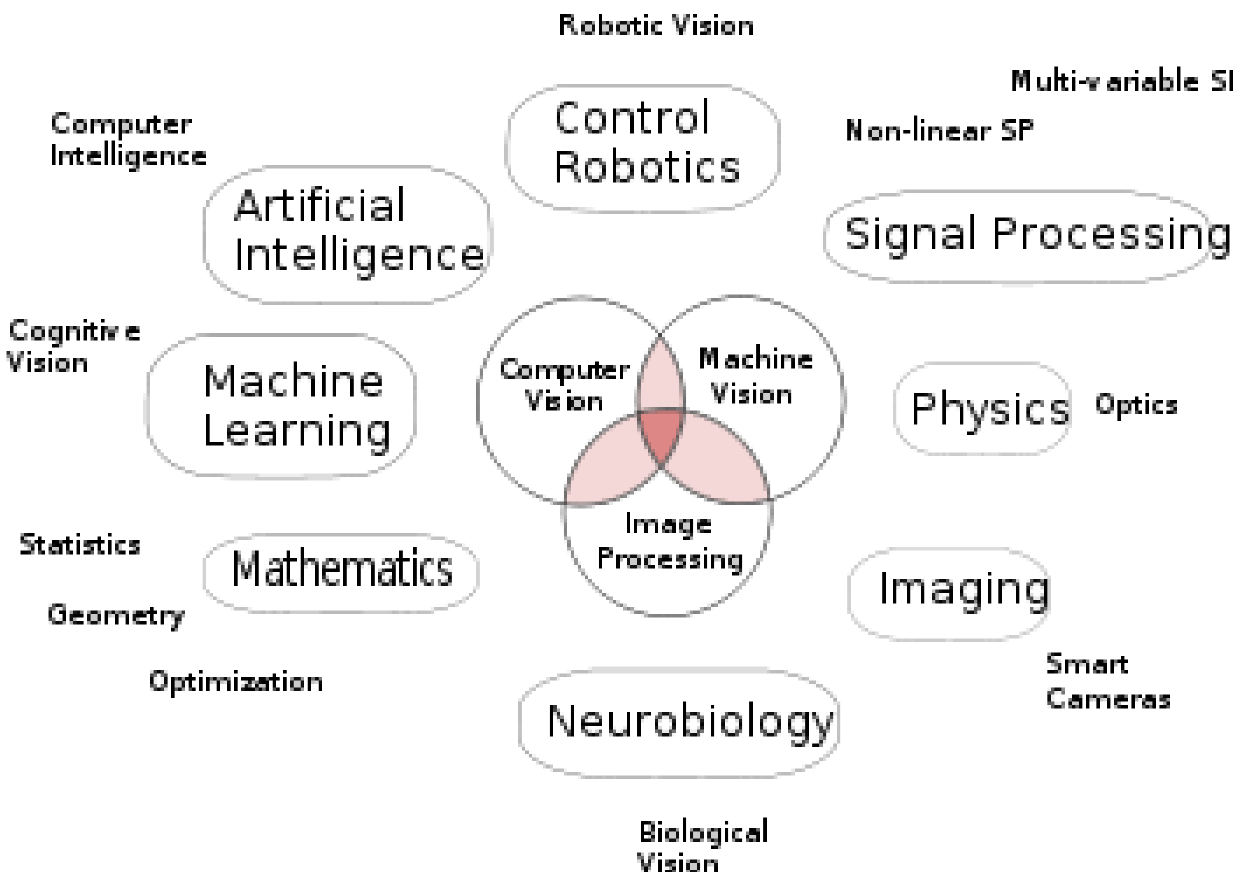
  More complex features may be related to texture, shape or motion.

  - **Detection/segmentation** – At some point in the processing a decision is made about which image points or regions of the image are relevant for further processing.[Examples are
    - Selection of a specific set of interest points
    - Segmentation of one or multiple image regions which contain a specific object of interest.
  - Segmentation of image into nested scene architecture comprised foreground, object groups, single objects or salient object parts (also referred to as spatial-taxon scene hierarchy)[**High-level processing** – At this step the input is typically a small set of data, for example a set of

points or an image region which is assumed to contain a specific objectThe remaining processing deals with, for example:

- Verification that the data satisfy model-based and application specific assumptions.
- Estimation of application specific parameters, such as object pose or object size.
- Image recognition – classifying a detected object into different categories.
- Image registration – comparing and combining two different views of the same object.

- **Decision making** Making the final decision required for the application,[12] for example:
  - Pass/fail on automatic inspection applications
  - Match / no-match in recognition applications



Applications for computer vision

Applications range from tasks such as industrial machine vision systems which, say, inspect bottles speeding by on a production line, to research into artificial intelligence and computers or robots that can comprehend the world around them. The computer vision and machine vision fields have significant overlap. Computer vision covers the core technology of automated image analysis which is used in many fields. Machine vision usually refers to a process of combining automated image analysis with other methods and technologies to provide automated inspection and robot guidance in industrial applications. In many computer vision applications, the computers are pre-programmed to solve a particular task, but methods based on learning are now becoming increasingly common. Examples of applications of computer vision include systems for:

- Controlling processes, *e.g.*, an industrial robot;
- Navigation, *e.g.*, by an autonomous vehicle or mobile robot;
- Detecting events, *e.g.*, for visual surveillance or people counting;
- Organizing information, *e.g.*, for indexing databases of images and image sequences;
- Modeling objects or environments, *e.g.*, medical image analysis or topographical modeling;
- Interaction, *e.g.*, as the input to a device for computer-human interaction, and
- Automatic inspection, *e.g.*, in manufacturing applications.

One of the most prominent application fields is medical computer vision or medical image processing. This area is characterized by the extraction of information from image data for the purpose of making a medical diagnosis of a patient. Generally, image data is in the form of microscopy images, X-ray images, angiography images, ultrasonic images, and tomography images. An example of information which can be extracted from such image data is detection of tumours, arteriosclerosis or other malign changes. It can also be measurements of organ dimensions, blood flow, etc. This application area also supports medical research by providing new information, *e.g.*, about the structure of the brain, or about the quality of medical treatments. Applications of computer vision in the medical area also includes enhancement of images that are interpreted by humans, for example ultrasonic images or X-ray images, to reduce the influence of noise