



## **Robotics And Planning**

**Class: 4<sup>th</sup> year/ Branch: AI**

**Dr. Alia Karim Abdul Hassan**

# What is Robotics

Robotics develop man-made mechanical devices that can move by themselves, whose motion must be modelled, planned, sensed, actuated and controlled, and whose motion behaviour can be influenced by “programming”. Robots are called “intelligent” if they succeed in moving in safe interaction with an unstructured environment, while autonomously achieving their specified tasks.

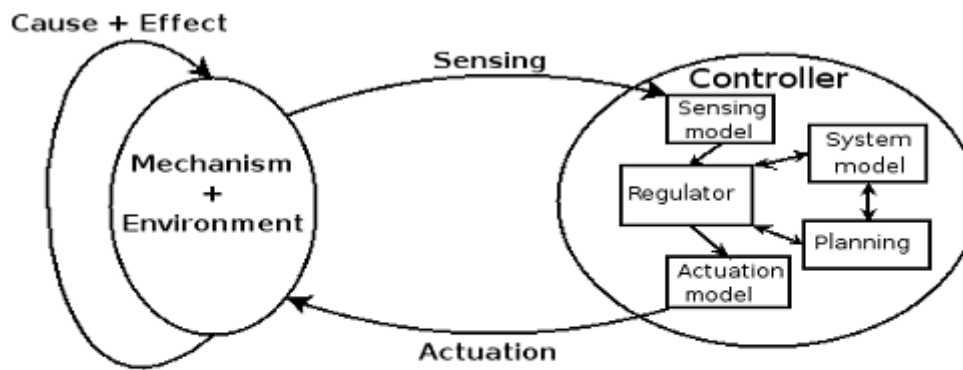
This definition implies that a device can only be called a “robot” if it contains a movable mechanism, influenced by sensing, planning, actuation and control components. It does not imply that a minimum number of these components must be implemented in software, or be changeable by the “consumer” who uses the device; for example, the motion behaviour can have been hard-wired into the device by the manufacturer.

**Robotics and Automation** includes “dumb” robots such as: metal and woodworking machines, “intelligent” washing machines, dish washers and pool cleaning robots, etc. These examples all have sensing, planning and control, but often not in individually separated components. For example, the sensing and planning behaviour of the pool cleaning robot have been integrated into the mechanical design of the device, by the intelligence of the human developer.

Robotics is, to a very large extent, all about system integration, achieving a task by an actuated mechanical device, via an “intelligent” integration of components, many of which it shares with other domains, such as systems and control, computer science, character animation, machine design, computer vision, artificial intelligence, cognitive science, biomechanics, etc. In addition, the boundaries of robotics cannot be clearly defined, since also its “core” ideas, concepts and algorithms are being applied in an ever increasing number of “external” applications, and, vice versa, core technology from other domains (vision, biology, cognitive science or biomechanics, for example) are becoming crucial components in more and more modern robotic systems.

## Components of robotic systems

The real robot is some mechanical device (“mechanism”) that moves around in the environment, and, in doing so, physically interacts with this environment. This interaction involves the exchange of physical energy, in some form or another. Both the robot mechanism and the environment can be the “cause” of the physical interaction through “**Actuation**”, or experience the “effect” of the interaction, which can be measured through “**Sensing**”.



**figure17:Robotics as an integrated system**

Robotics develop man-made mechanical devices that can move by themselves, whose motion must be modelled, planned, sensed, actuated and controlled, and whose motion behavior can be influenced by “programming”. Robots are called “intelligent” if they succeed in moving in safe interaction with an unstructured environment, while autonomously achieving their specified tasks.

This definition implies that a device can only be called a “robot” if it contains a movable mechanism, influenced by sensing, planning, actuation and control components. It does not imply that a minimum number of these components must be implemented in software, or be changeable by the “consumer” who uses the device; for example, the motion behaviour can have been hard-wired into the device by the manufacturer.

Sensing and actuation are the physical ports through which the “Controller” of the robot determines the interaction of its mechanical body with the physical world. As mentioned already before, the controller can, in one extreme, consist of software only, but in the other extreme everything can also be implemented in hardware.

Within the Controller component, several sub-activities are often identified:

**Modelling.** The input-output relationships of all control components can (but need not) be derived from information that is stored in a model. This model can have many forms: analytical formulas, empirical look-up tables, fuzzy rules, neural networks, etc.

“model” is to be understood with its minimal semantics: “any information that is used to determine or influence the input-output relationships of components in the Controller.”

The other components discussed below can all have models inside. A “System model” can be used to tie multiple components together, but it is clear that not all robots use a System model. The “Sensing model” and “Actuation model” contain

the information with which to transform raw physical data into task-dependent information for the controller, and vice versa.

**Planning.** This is the activity that predicts the outcome of potential actions, and selects the “best” one. Almost by definition, planning can only be done on the basis of some sort of model.

**Regulation.** This component processes the outputs of the sensing and planning components, to generate an actuation set point. Again, this regulation activity could or could not rely on some sort of (system) model.

The term “control” is often used instead of “regulation”, but it is impossible to clearly identify the domains that use one term or the other.

### **Scales in robotic systems**

The above-mentioned “components” description of a robotic system is to be complemented by a “scale” description, i.e., the following system scales have a large influence on the specific content of the planning, sensing, modeling and control components at one particular scale.

**Mechanical scale.** The physical volume of the robot determines to a large extent the limites of what can be done with it. Roughly speaking, a **large-scale** robot (such as an autonomous container crane or a space shuttle) has different capabilities and control problems than a **macro** robot (such as an industrial robot arm), a **desktop** robot (such as those “sumo” robots popular with hobbyists), or **milli micro** or **nano** robots.

**Spatial scale.** There are large differences between robots that act in 1D, 2D, 3D, or 6D (three positions and three orientations).

**Time scale.** There are large differences between robots that must react within hours, seconds, milliseconds, or microseconds.

**Power density scale.** A robot must be actuated in order to move, but actuators need space as well as energy, so the ratio between both determines some capabilities of the robot.

**System complexity scale.** The complexity of a robot system increases with the **number of interactions** between independent sub-systems, and the control components must adapt to this complexity.

**Computational complexity scale.** Robot controllers are inevitably running on real-world computing hardware, so they are constrained by the available **number of computations**, the available **communication bandwidth**, and the available **memory storage**.

Obviously, these scale parameters never apply completely independently to the same system. For example, a system that must react at microseconds time scale can not be of macro mechanical scale or involve a high number of communication interactions with subsystems.

### **Background sensitivity**

robotics has, roughly speaking, two faces:

- (i) the mathematical and engineering face, which is quite “standardized” in the sense that a large consensus exists about the tools and theories to use (“systems theory”), and
- (ii) the AI face, which is rather poorly standardized, not because of a lack of interest or research efforts, but because of the inherent complexity of “intelligent behaviour.”

Research in engineering robotics follows the bottom-up approach: existing and working systems are extended and made more versatile. Research in artificial intelligence robotics is top-down: assuming that a set of low-level primitives is available, how could one apply them in order to increase the “intelligence” of a system. The border between both approaches shifts continuously, as more and more “intelligence” is cast into algorithmic, system-theoretic form. For example, the response of a robot to sensor input was considered “intelligent behaviour” in the late seventies and even early eighties. Hence, it belonged to A.I. Later it was shown that many sensor-based tasks such as surface following or visual tracking could be formulated as control problems with algorithmic solutions. From then on, they did not belong to A.I. any more.

Most industrial robots have at least the following five parts:

Sensors, Effectors, Actuators, Controllers, and common effectors known as Arms.

Many other robots also have Artificial Intelligence and effectors that help it achieve Mobility.

**Sensors** Most robots of today are nearly deaf and blind. Sensors can provide some limited feedback to the robot so it can do its job. Compared to the senses and abilities of even the simplest living things, robots have a very long way to go.

The sensor sends information, in the form of electronic signals back to the controller. Sensors also give the robot controller information about its surroundings and lets it know the exact position of the arm, or the state of the world around it. Sight, sound, touch, taste, and smell are the kinds of information we get from our world. Robots can be designed and programmed to get specific information that is beyond what our 5 senses can tell us. For instance, a robot sensor might "see" in the dark, detect tiny amounts of invisible radiation or measure movement that is too small or fast for the human eye to see. Here are some things sensors are used for:

Physical Property	Technology
Contact	Bump, Switch
Distance	Ultrasound, Radar, Infra Red
Light Level	Photo Cells, Cameras
Sound Level	microphones
Strain	Strain Gauges

Rotation	Encoders
Magnetism	Compasses
Smell	Chemical
Temperature	Thermal, Infra Red
Inclination	Inclinometers, Gyroscope
Pressure	Pressure Gauges
Altitude	Altimeters

Sensors can be made simple and complex, depending on how much information needs to be stored. A switch is a simple on/off sensor used for turning the robot on and off. A human retina is a complex sensor that uses more than a hundred million photosensitive elements (rods and cones). Sensors provide information to the robots brain, which can be treated in various ways. For example, we can simply *react* to the sensor output: if the switch is open, if the switch is closed, go.

## **A robotic and artificial intelligence**

The term "artificial intelligence" is defined as systems that combine sophisticated hardware and software with elaborate databases and knowledge-based processing models to demonstrate characteristics of effective human decision making. The criteria for artificial systems include the following: 1) functional: the system must be capable of performing the function for which it has been designed; 2) able to manufacture: the system must be capable of being manufactured by existing manufacturing processes; 3) designable: the design of the system must be imaginable by designers working in their cultural context; and 4) marketable: the system must be perceived to serve some purpose well enough, when compared to competing approaches, to warrant its design and manufacture.

Robotics is one field within artificial intelligence. It involves mechanical, usually computer-controlled, devices to perform tasks that require extreme precision or tedious or hazardous work by people. Traditional Robotics uses Artificial Intelligence planning techniques to program robot behaviors and works toward robots as technical devices that have to be developed and controlled by a human engineer. The Autonomous Robotics approach suggests that robots could develop and control themselves autonomously. These robots are able to adapt to both uncertain and incomplete information in constantly changing environments. This is possible by imitating the learning process of a single natural organism or through Evolutionary Robotics, which is to apply selective reproduction on populations of robots. It lets a simulated evolution process develop adaptive robots.

The artificial intelligence concept of the "expert system" is highly developed. This

describes robot programmers ability to anticipate situations and provide the robot with a set of "if-then" rules. For example, if encountering a stairwell, stop and retreat. The more sophisticated concept is to give the robot the ability to "learn" from experience. A neural network brain equipped onto a robot will allow the robot to sample its world at random. Basically, the robot would be given some life-style goals, and, as it experimented, the actions resulting in success would be reinforced in the brain. This results in the robot devising its own rules. This is appealing to researchers and the community as it parallels human learning in lots of ways.

Artificial intelligence dramatically reduces or eliminates the risk to humans in many applications. Powerful artificial intelligence software helps to fully develop the high-precision machine capabilities of robots, often freeing them from direct human control and vastly improving their productivity. When a robot interacts with a richly populated and variable world, it uses its senses to gather data and then compare the sensed inputs with expectations that are imbedded in its world model. Therefore the effectiveness of the robot is limited by the accuracy to which its programming models the real world.

## **Definitions**

### **robot**

A versatile mechanical device equipped with actuators and sensors under the control of a computing system. Russell and Norvig define it as "an active, artificial agent whose environment is the physical world."

### **task planner**

A program that converts task-level specifications into manipulator-level specifications. The task planner must have a description of the objects being manipulated, the task environment, the robot, and the initial and desired final states of the environment. The output should be a robot program that converts the initial state into the desired final state.

### **effector**

The bits the robot does stuff with. That is, arms, legs, hands, feet. An *end-effector* is a functional device attached to the end of a robot arm (e.g., grippers).

### **actuator**

A device that converts software commands into physical motion, typically electric motors or hydraulic or pneumatic cylinders.

### **degree of freedom**

A dimension along which the robot can move itself or some part of itself. Free objects in 3-space have 6 degrees of freedom, three for position and three for orientation.

### **sensors**

Devices that monitor the environment. There are contact sensors (touch and force), and non-contact (e.g., sonar).

### **sonar**

Sensing system that works by measuring the time of flight of a sound pulse to be generated, reach an object, and be reflected back to the sensor. Wide angle but reasonably accurate in depth (the wide angle is the disadvantage).

### **infrared**

Very accurate angular resolution system but terrible in depth measurement.

### **recognizable set**

An envelope of possible configurations a robot may be in at present; recognizable sets are to continuous domains what multiple state sets are to discrete ones. A recognizable set with respect to a sensor reading is the set of all world states in which the robot might be upon receiving that sensor reading.

### **landmark**

An easily recognizable, unique element of the environment that the robot can use to get its bearings.

## **Task Planning**

By virtue of their versatility, robots can be difficult to program, especially for tasks requiring complex motions involving sensory feedback. In order to simplify programming, *task-level* languages exist that specify actions in terms of their effects on objects.

*Example: pin* A programmer should be able to specify that the robot should put a pin in a hole, without telling it what sequence of operators to use, or having to think about its sensory or motor operators.

Task planning is divided into three phases: modeling, task specification, and manipulator program synthesis.

There are three approaches to specifying the model state:

1. Using a CAD system to draw the positions of the objects in the desired configuration.
2. Using the robot itself to specify its configurations and to locate the object features.
3. Using symbolic spatial relationships between object features (such as (face1 against face2). This is the most common method, but must be converted into numerical form to be used.

One problem is that these configurations may *overconstrain* the state. Symmetry is an example; it does not matter what the orientation of a peg in a hole is. The final state may also not completely specify the operation; for example, it may not say how hard to tighten a bolt.



The three basic kinds of motions are free motion, guarded motion, and compliant motion.

An important part of robot program synthesis should be the inclusion of sensor tests for error detection.

## **Motion Planning**

The fundamental problem in robotics is deciding what motions the robot should perform in order to achieve a goal arrangement of physical objects. This turns out to be an extremely hard problem.

## **Motion Planning Definitions**

### **basic motion planning problem**

Let  $\mathbf{A}$  be a single rigid object (the robot) moving in a Euclidean space  $\mathbf{W}$ , called the *workspace*, represented as  $\mathbf{R}^n$  (where  $n = 2$  or  $3$ ). Let  $\mathbf{B}_1, \dots, \mathbf{B}_q$  be fixed rigid objects distributed in  $\mathbf{W}$ . These are called *obstacles*.

Assume that the geometry of  $\mathbf{A}$ , and the geometries and locations of the  $\mathbf{B}_i$ 's are accurately known. Assume also that no kinematic constraints limit the motions of  $\mathbf{A}$  (so that  $\mathbf{A}$  is a *free-flying object*).

Given an initial position and orientation and a goal position and orientation of  $\mathbf{A}$  in  $\mathbf{W}$ , the problem is to generate a path  $\mathbf{t}$  specifying a continuous sequence of positions and orientations of  $\mathbf{A}$  avoiding contact with the  $\mathbf{B}_i$ 's.

(Basically, given a robot, a bunch of objects, a start state and a goal state, find a path for the robot to reach the goal state.)

### **configuration of object A**

A specification of the position of every point in the object, relative to a fixed frame of reference. To specify the configuration of a rigid object  $\mathbf{A}$ , it is enough to specify the position and orientation of the frame  $\mathbf{F}_A$  with respect to  $\mathbf{F}_W$ . The subset of  $\mathbf{W}$  occupied by  $\mathbf{A}$  at configuration  $q$  is denoted by  $\mathbf{A}(q)$ .

### **configuration space of object A**

The space  $\mathbf{C}$  of all configurations of  $\mathbf{A}$ . The idea is to represent the robot as a point and thus reduce the motion planning problem to planning for a point.

### **dimension of C**

The dimension of a configuration space is the number of independent parameters required to represent it as  $\mathbf{R}^m$ . This is 3 for 2-D, and 6 for 3-D.

### **chart**

A representation of a local portion of the configuration space.  $\mathbf{C}$  can be decomposed into a finite union of slightly overlapping *patches* called *charts*, each represented as a copy of  $\mathbf{R}^m$ .

## Distance between configurations

The distance between configurations  $q$  and  $q'$  should decrease and tend to zero when the regions  $A(q)$  and  $A(q')$  get closer and tend to *coincide*.

## Path

A path from a configuration  $q_{init}$  to configuration  $q_{goal}$  is a continuous map  $t : [0,1] \rightarrow C$  with  $t(0) = q_{init}$  and  $t(1) = q_{goal}$ .

## free-flying object

An object for which, in the absence of any obstacles, any path is feasible.

## C-obstacle

An obstacle mapped into configuration space. Every obstacle  $B_i$  is mapped to the following region in the workspace called the C-obstacle:  $CB_i = \{ q \text{ in } C : A(q) \text{ intersected with } B_i \neq \text{empty set} \}$ .

## C-obstacle region

The union of all the C-obstacles.

## Free space

All of the configuration space less the C-obstacle region, called  $C_{free}$ . A configuration in  $C_{free}$  is called a *free configuration* and a *free path* is a path where  $t$  maps to  $C_{free}$  instead of to  $C$ . A *semi-free path* maps to the closure of  $C_{free}$ .

## dynamics

Motions of material bodies under the action of forces.

## force-compliant motion

Motions in which the robot may touch obstacle surfaces and slide along them. These are more accurate than position-controlled commands.

## Configuration Space

For a robot with  $k$  degrees of freedom, the state or configuration of the robot can be described by  $k$  real values. These values can be considered as a point  $p$  in a  $k$ -dimensional *configuration space* of the robot.

Configuration space can be used to determine if there is a path by which a robot can move from one place to another. Real obstacles in the world are mapped to *configuration space obstacles*, and the remaining *free space* is all of configuration space except the part occupied by those obstacles.

Having made this mapping, suppose there are two points in configuration space. The robot can move between the two corresponding real world points exactly when there is a continuous path between them that lies entirely in configuration free space.

## Generalized Configuration Space

The term *generalized configuration space* is used to describe systems in which other objects are included as part of the configuration. These may be movable, and their shapes may vary.

There are several ways of dealing with robot planning when there are several moving or movable objects. These are:

1. Partition the generalized configuration space into finitely many states. The planning problem then becomes a logical one, like the blocks world. No general method for partitioning space has yet been found.
2. Plan object motions first, and then motions for the robot.
3. Restrict object motions to simplify planning.

## Uncertainty

A robot may have little or no prior knowledge about its workspace. The more incomplete the knowledge, the less important the role of planning. A more typical situation is when there are errors in robot control and in the initial models, but these errors are contained within bounded regions.

<http://www.electronicsteacher.com/robotics/what-is-robotics.php>

# Mobile Robot Motion Planning

## Introduction

Motion planning, refers to the ability of a robot to plan its own motions. The basic motion planning problem is stated as follows :

*Given an initial configuration and a desired final configuration of the robot, find a path starting at the initial configuration and terminating at the final configuration, while avoiding collisions with obstacles. It is assumed that the geometry and location of the obstacles are completely known.*

A consistent model for individual and relational behaviors is required to provide a systematic methodology for behavior synthesis and analysis. Particularly convenient to model relational behaviors, where more than one teammate is involved .

## Mobile Robot

Robot manipulators (first and for most the popular stationary robot arms) work fine for instance in assembly applications in factories. However, mobile robots offer some very important advantages, for instance:

**Reach** Mobile robots are necessary if the problem the robot should solve is not restricted to some sufficiently small area.

**Flexibility** If the position of the problem to be solved is not static, the mobile robot has the ability to pursue it.

A mobile robot has to generate a navigational plan in a given environment between predefined starting and goal points. The robot environment includes many obstacles and thus finding the shortest path without touching the obstacles in many cases is an extremely complex problem. The complexity of the problem increases further, when the obstacles are dynamic.

The basic problem of a mobile robot is that of *navigation* moving from one place to another by a coordination of planning, sensing and control. In any navigation scheme the desire is to reach a destination without getting lost or crashing into anything. Simply the navigation problem is to find a path from start (S) to target (G) and traverse it without collision. Navigation may be decomposed into three sub-tasks:

**Subtask1:** mapping and modeling the environment; this concerns the representation of free space; the workspace through which a robot is to move amongst a number of obstacles.

*Subtask2:* path planning; this constitutes the core of the planner, it concerns the computation (i.e. searching) within predetermined criteria, of near optimal or even an optimal paths for a robot to navigate throughout its environment.

*Subtask3:* path following and collision avoidance; path following in the case of single robot motion. For multiple robot motion path following and coordination.

The relationship between these subtasks is shown in figure (1).

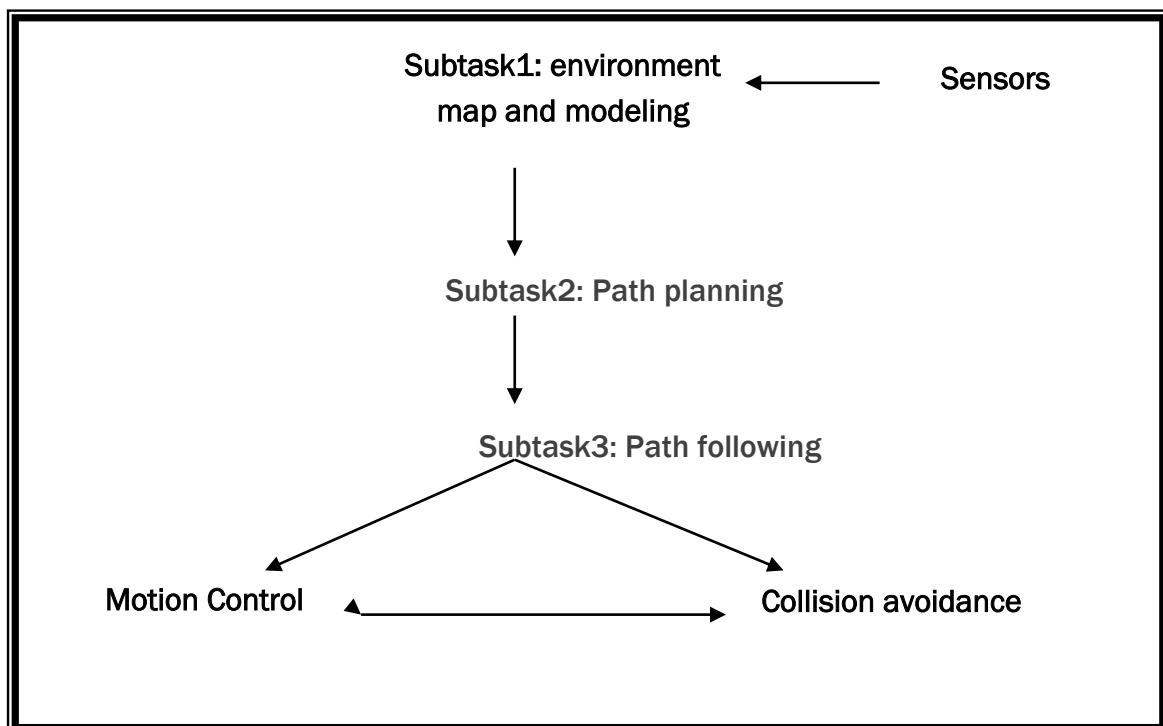


Figure1 Mobile robot motion planning basic problem

## Motion Planning

The process of generating a sequence of actions that has to be performed in order for a robot to move through its environment (also called workspace, see figure (2)) autonomously and without collisions is called motion planning.

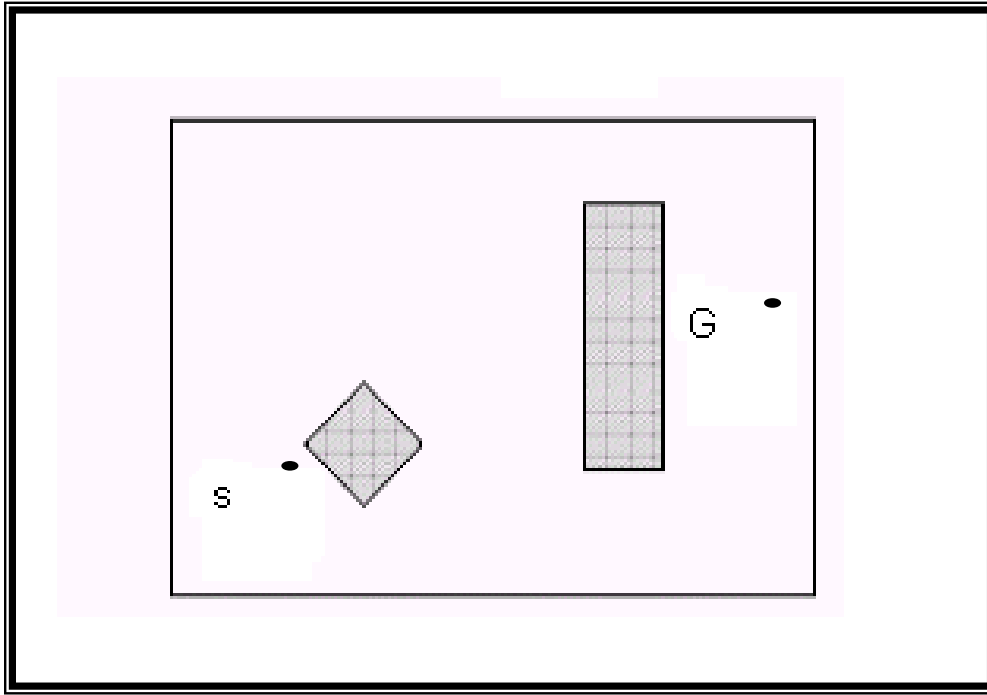


Figure 2: A simple workspace with two obstacles and an enclosing workspace boundary.

The space in which the motion problem “lives” will be defined as the workspace (world),  $W$ , for which there are two possible choices :

- 1) A two-dimensional (2D) workspace in which  $W = R^2$  ( $R$  denotes the set of real numbers and
- 2) Three-dimensional (3D) workspace, in which  $W = R^3$ .

Generally the workspace contains two entities:

- 1- Obstacles: Portions of the workspace that are “permanently “ occupied, for example, as in the walls of a building.
- 2- Robots: Geometric bodies that behave according to a motion strategy.

Basic motion planning problem (*single robot motion planning problem*) assumes that the robot is the only moving object in the workspace around stationary obstacles. This problem can be solved by merely constructing a geometric path.

If the case where several robots move independently in the same workspace among stationary obstacles the resulting problem is called the *multiple robot path-planning problem* .

In order to organize the various facets of motion planning in a coherent framework, the basic concepts to motion planning will be exposed in detail.

## Configuration Space and Point Robot

instead of handling a complex geometrical representation of a robot in the Euclidean representation of the workspace, the robot could be treated as a point in its Configuration space (C-space).

The configuration has as many dimensions as the robot has degrees of freedom (DoF). For example robot of three DoFs has two for translation and one for rotation. *Rotation invariant* robot is symmetric in its z-axis and C-space will be two dimensional, in fact, we say that an obstacle in the workspace “grows” with size of robot in the C- space of the robot.

The underlying concept is to represent the real-world robot as a point in an appropriate space, and to map obstacles into this same space. Then, the space contains a concise representation of the robot's geometrical constraints on motion, and a motion planner needs only to consider the path of the single point, which represents the robot. In figure (3) the configuration  $q$  of a rotation invariant robot  $A$  specifies the exact position and orientation of  $A$  relative to a fixed reference frame. Therefore, the C-space of  $A$  is the set of all possible configurations of  $A$ . Obstacles are mapped into C-space by determining which configurations of the robot produce collisions with an obstacle; these configurations are deemed forbidden. Let  $A(q)$  denote the location of  $A$ 's particles when  $A$  is in configuration  $q$ . A C-space obstacle (or “C-obstacle”) associated with a physical obstacle  $B$  is defined as

$$CB = \{q \in C \mid A(q) \cap B = \emptyset\} \text{-----} (2.1)$$

The complement of the C-obstacles is termed the “free space”:

$$C_{free} = C \setminus CB \text{-----} (2.2)$$

Motion plans are constructed in  $C_{free}$  (see figure (3)).

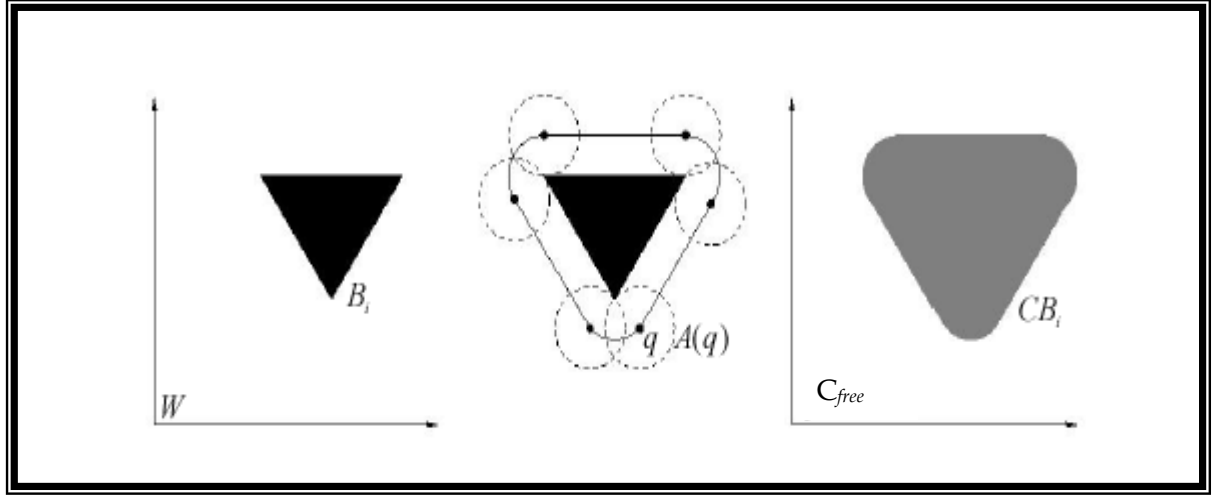


Figure 3: Configuration space for rotation invariant robot

## The Notion of Path

A path of  $A$  from the configuration  $q_{init}$  to the configuration  $q_{goal}$  is a continuous map:

$$\rho = [0,1] \rightarrow C_{free} \text{ ----- (2.3)}$$

with

$$\rho(0) = q_{init} \text{ and } \rho(1) = q_{goal}$$

$q_{init}$  and  $q_{goal}$  are the initial and goal configurations of the path, respectively. In order to match intuition of the path, the distance between two configurations  $q_{init}$  and  $q_{goal}$  should decrease and tend toward zero when the regions  $A(q_{init})$  and  $A(q_{goal})$  get closer and tend to coincide. A simple distance function that satisfies this condition is defined for two dimension workspace:

$$d(q_{init}, q_{goal}) = \max_{a \in A} \|a(q_{init}) - a(q_{goal})\| \text{ ----- (2.4)}$$



Where  $\|x - x'\|$  denotes the Euclidean distance between any two points  $x$  and  $x'$ . If the path does not touch the obstacles it is called **free path**. Paths that touch the obstacles are called **semi-free paths**.

## Methods for Motion Planning

There exist a large number of methods for solving motion planning problem for single and multiple mobile robots. Despite many external deference's, the methods are based on few different general approaches. These approaches will be described in the next subsections.

### Single robot motion planning

To date, motion planning approaches for single robot can be classified into three categories:

- 1) Skeleton (Roadmaps);
- 2) Cell decomposition;
- 3) Potential field;

1) In the **skeleton** approach, the free space is represented by a network of one-dimensional (1-D) paths called a Roadmap. There are many different roadmap methods, but one thing they all have in common is that they try to convert the free space of the workspace into a graph representation (a roadmap). A collision-free path can now be constructed (if one exists) by connecting the start position and destination to the roadmap.

The roadmap method called *visibility graph* constructs a shortest path, but it is only semi-free path. In the visibility graph method, all pairs of vertices of the obstacles in the workspace are connected. A connection between a pair of vertices is called an edge and all edges

form a possible path segment on an optimal path. In figure (4 a) the thick path segment and the dotted thick lines from  $S$  to  $G$  constitute a path.

The method called *voronoi diagram*, on the other hand, maximizes the distance between robot and obstacles. In the voronoi diagram method, a path is constructed by connecting  $S$  and  $G$  with the roadmap, which consists of positions that are on a maximum distance from the obstacles and workspace boundary (figure 4 b).

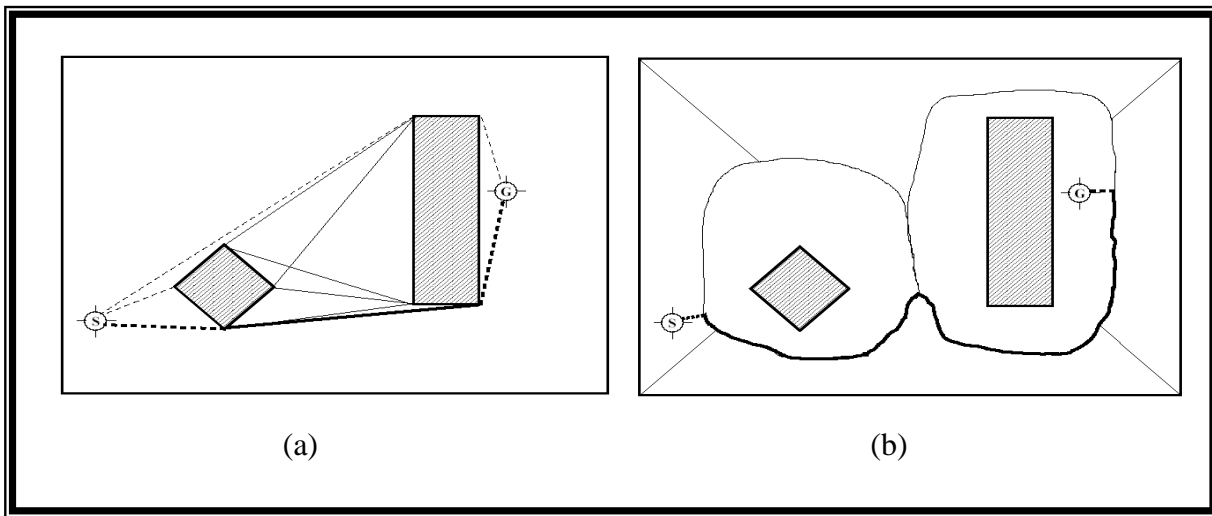


Figure 4: Roadmap method a) visibility graph method b) voronoi graph method

2) In the **cell-decomposition** methods the free space of the workspace is decomposed into a set of cells. The cells must be simple so that a path easily can be planned through each cell (a suitable cell is typically a convex polygon). A *channel* of free cells (i.e., a sequence of contiguous cells) is subsequently constructed starting with the cell, which contains the current position of the robot, and ending with the cell that contains its destination. Finally, a path can be planned from the start position through the channel to the destination .

Cell decomposition is further divided *into exact and approximate cell decompositions* figure (5).

*Exact cell decomposition methods* decompose the free space into cells whose union is exactly the free space. Exact methods are more mathematically involved and are complete i.e. they are guaranteed to find path whenever exists and return failure otherwise.

*Approximate cell decomposition* produces cells of predefined shape (e.g. rectangloids) whose union is strictly included in the free space. Approximate methods involve recursive simple computation, so they are much easier to implement than exact methods but are incomplete since they may fail to find a free path if one exists.

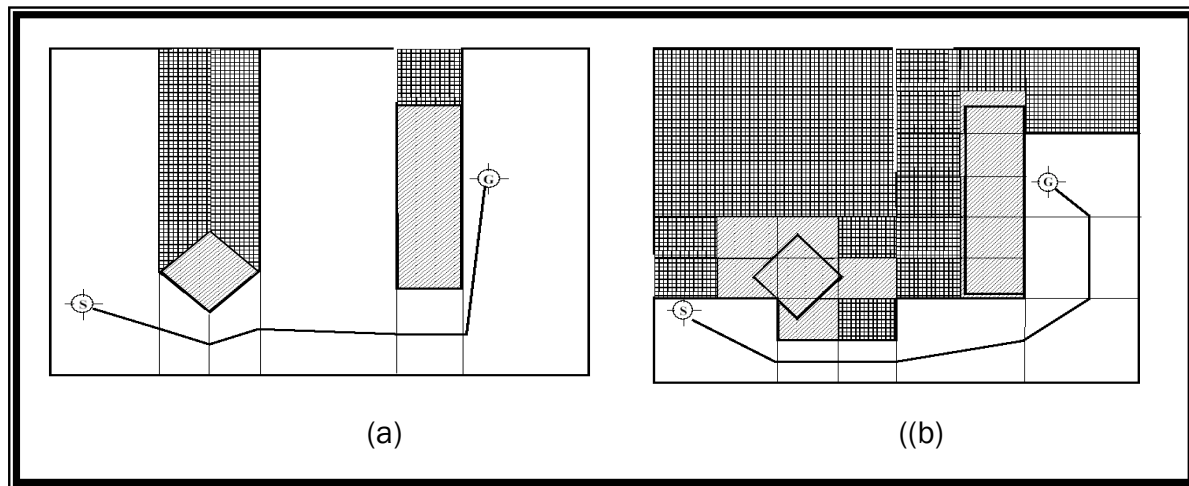


Figure 5: Cell decompositions methods (a)The Exact method cell decomposition method (b) The approximate cell decomposition method.

- 3) In the **potential-field** approach, a scalar potential function that has high values near obstacles and the global minimum at the goal is constructed. In this case the robot moves in the direction of the negative gradient of the potential.

Most methods for motion planning can be derived from these approaches or hybrids of these approaches .

The motion planning approach called cell decomposition method is quite attractive. It allows generation of collision-free paths (whereas, e.g., visibility graph only guarantees semi-free paths). Moreover it is practical (compared to, e.g., voronoi diagram which appears more difficult to implement) and it takes global knowledge into consideration (unlike potential field).

Cell decomposition methods have the following main steps :

- 1) Represent the free space as collection of cells.
- 2) Generate the connectivity graph representing the adjacency relation between cells.
- 3) Search the connectivity graph for a sequence of adjacent cells connecting the initial to the goal cell.
- 4) Transform the sequence of cells (if one has been produced) into a path. Each cell represents a connected region of free space.

## Multi Robot Motion Planning

All motion planning methods for single robot motion planning are applicable to multiple robot motion planning but with modification.

According to the way the multiple robots are treated the multi-robot motion planning approaches are often categorized as *centralized* and *decoupled*.

*Centralized* approaches treat the separate robots as one composite system, and typically perform the planning in a composite configuration space, formed by combining the configuration spaces of the individual robots.

*Decoupled* approaches first generate paths for the separate robots more or less independently, and then consider the interactions between the robots (with respect to the generated paths). Decoupled is more less computation complexity than the centralized approaches.

## Online and Off-line Motion Planning

An alternative way of classifying motion-planning methods is to say whether they are *on-line* or *off-line*. On-line planning is performed in real time, i.e., at the same time the robot is moving, and is exceptionally useful when the environment is not known. Off-line planning is performed before any robot motion and is not useful unless the workspace is known. Table (1) lists the differences between on-line and off line methods.

## Complete and Sound Methods

Almost all motion planning methods can be characterized along the following:

**Complete:** A method is said to be complete if it guaranteed to find a collision-free path if one exists; otherwise return failure.

**Sound:** if it guarantees that all its solutions are correct (i.e., collision free).

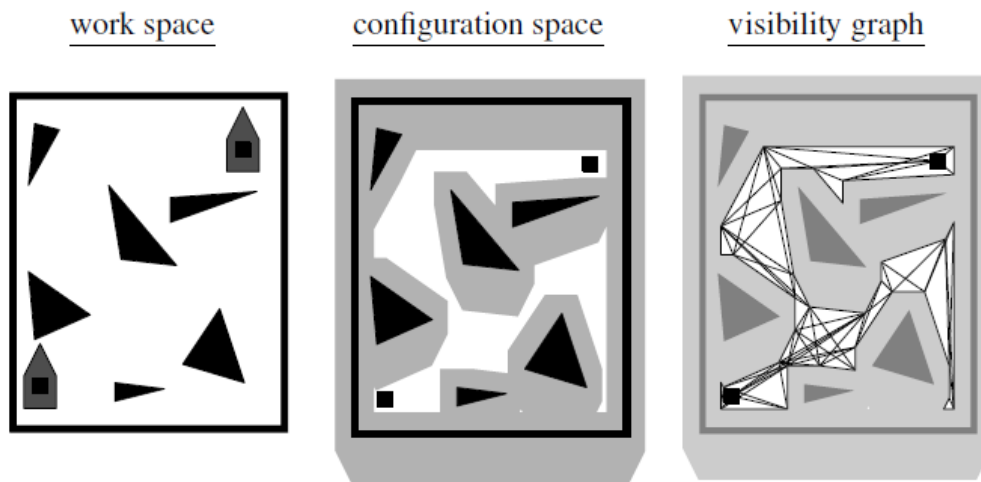
### visibility graph method

As described previously in the visibility graph each pairs of vertices of the obstacles in the workspace are connected. A connection between a pair of vertices is called an edge and all edges form a possible path segment on an optimal path. The following algorithm is used to construct visibility graph

**Algorithm** VISIBILITYGRAPH( $S$ )*Input.* A set  $S$  of disjoint polygonal obstacles.*Output.* The visibility graph  $\mathcal{G}_{\text{vis}}(S)$ .

1. Initialize a graph  $\mathcal{G} = (V, E)$  where  $V$  is the set of all vertices of the polygons in  $S$  and  $E = \emptyset$ .
2. **for** all vertices  $v \in V$
3.     **do**  $W \leftarrow \text{VISIBLEVERTICES}(v, S)$
4.         For every vertex  $w \in W$ , add the arc  $(v, w)$  to  $E$ .
5. **return**  $\mathcal{G}$

The procedure VISIBLEVERTICES has as input a set  $S$  of polygonal obstacles and a point  $p$  in the plane; in our case  $p$  is a vertex of  $S$ , but that is not required. It should return all obstacle vertices visible from  $p$ .

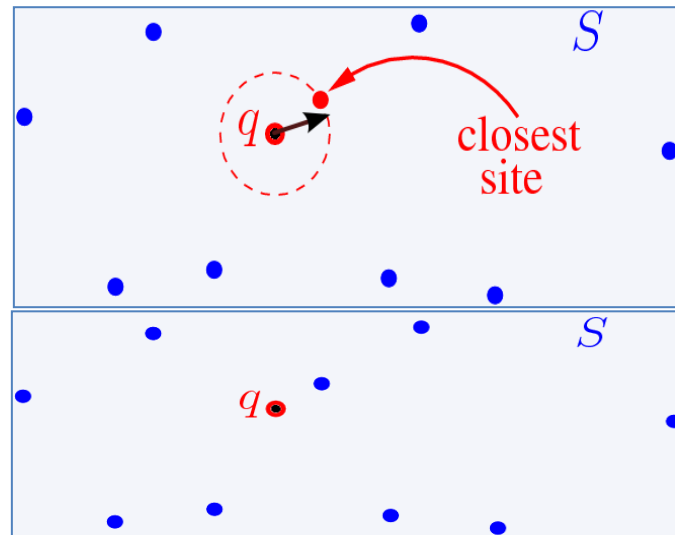
**Algorithm** SHORTESTPATH( $S, p_{\text{start}}, p_{\text{goal}}$ )*Input.* A set  $S$  of disjoint polygonal obstacles, and two points  $p_{\text{start}}$  and  $p_{\text{goal}}$  in the free space.*Output.* The shortest collision-free path connecting  $p_{\text{start}}$  and  $p_{\text{goal}}$ .

1.  $\mathcal{G}_{\text{vis}} \leftarrow \text{VISIBILITYGRAPH}(S \cup \{p_{\text{start}}, p_{\text{goal}}\})$
2. Assign each arc  $(v, w)$  in  $\mathcal{G}_{\text{vis}}$  a weight, which is the Euclidean length of the segment  $\overline{vw}$ .
3. Use Dijkstra's algorithm to compute a shortest path between  $p_{\text{start}}$  and  $p_{\text{goal}}$  in  $\mathcal{G}_{\text{vis}}$ .

**The VORONOI DIAGRAM Method**

The Voronoi Diagram concept is a simple. Given a finite set of objects in a space, all locations in that space are associated with the closest member of the object set. The result is a partition of the space into a set of regions, Voronoi regions. The Generalized Voronoi Diagram is the frontier between these regions. Given its widespread use, it is not surprising that this concept has been discovered many times in many different places. [Example \(Voronoi Diagram in Plane\)](#)

- A dataset  $S$  of  $n$  points called *sites* in  $\mathbb{R}^2$ .  
Let  $S = \{s_1, s_2, \dots, s_n\}$ .
- (Query 1) For any given *query point* find the closest site to  $q$ .
- (Query 2) Find the *proximity region* of a site.
- (Query 3) Find the *proximity regions* of all sites.



What is nearest point to  $q$ ? Also, circle of shortest radius is empty of other sites.

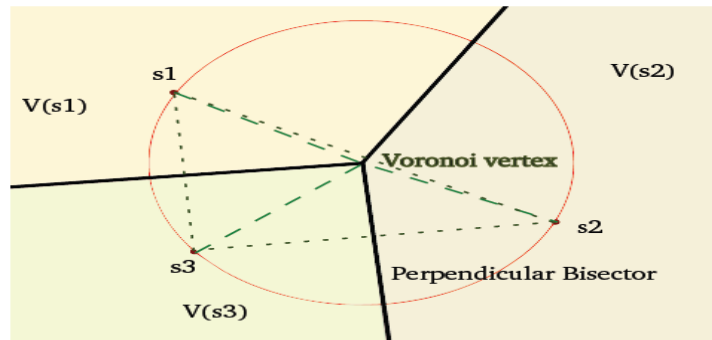
### Definition

- Let  $S$  be a set of  $n$  distinct points,  $s_i$ ,  $\forall i \in n$ , called *sites* in the plane
- The Voronoi diagram of  $S$  is the subdivision of the plane into  $n$  cells,  $V(s_i)$ , one for each site  $s_i$ ,
- A point  $q$  lies in  $V(s_i)$  iff  $\|q - s_i\| < \|q - s_j\|$ , for each  $s_j \in S, i \neq j$

Simply put,  $V(s_i)$  is the set of points whose nearest point is  $s_i$ .

## Properties of Voronoi Diagram

### Example (Voronoi Diagram of 3 points)



Bisectors intersect at circumcentre

Navigation

### Cell Decomposition Methods

In the **cell-decomposition** methods the free space of the workspace is decomposed into a set of cells. The cells must be simple so that a path easily can be planned through each cell (a suitable cell is typically a convex polygon). A *channel* of free cells (i.e., a sequence of contiguous cells) is subsequently constructed starting with the cell, which contains the current position of the robot, and ending with the cell that contains its destination. Finally, a path can be planned from the start position through the channel to the destination

Cell decomposition is further divided into *exact and approximate cell decompositions* figure (5).

*Exact cell decomposition methods* decompose the free space into cells whose union is exactly the free space. Exact methods are more mathematically involved and are complete i.e. they are guaranteed to find path whenever exists and return failure otherwise.

*Approximate cell decomposition* produces cells of predefined shape (e.g. rectangloids) whose union is strictly included in the free space. Approximate methods involve recursive simple computation, so they are much easier to

implement than exact methods but are incomplete since they may fail to find a free path if one exists.

Cell decomposition methods have the following main steps :

- 1) Represent the free space as collection of cells.
- 2) Generate the connectivity graph representing the adjacency relation between cells.



- 3) Search the connectivity graph for a sequence of adjacent cells connecting the initial to the goal cell.
- 4) Transform the sequence of cells (if one has been produced) into a path. Each cell represents a connected region of free space.

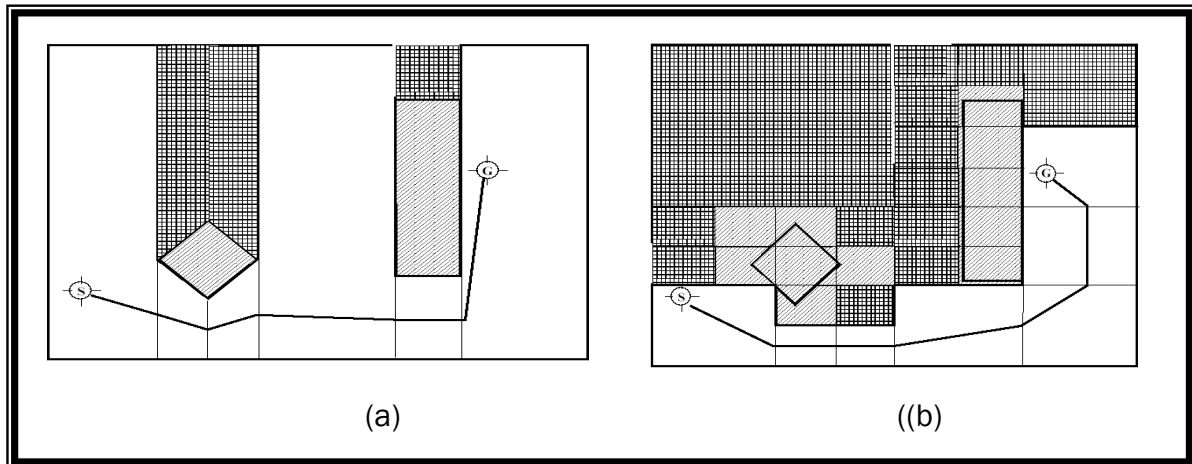


Figure 5: Cell decompositions methods (a)The Exact method cell decomposition method (b) The approximate cell decomposition method.

## Cell decomposition Methods are

- Exact Cell Decomposition
- Approximate Decomposition

### Exact cell decomposition

In this method the free space is decomposed into a collection of non overlapping regions cells whose union is exactly  $C_{free}$ . A connectivity graph which represents the adjacency relation among the cells constructed and searched. If successful the outcome of the search is a sequence of cells called a channel connecting the cell containing the initial configuration to the cell containing the goal configuration a path finally extracted from this sequence. The generated cells should have the following properties:

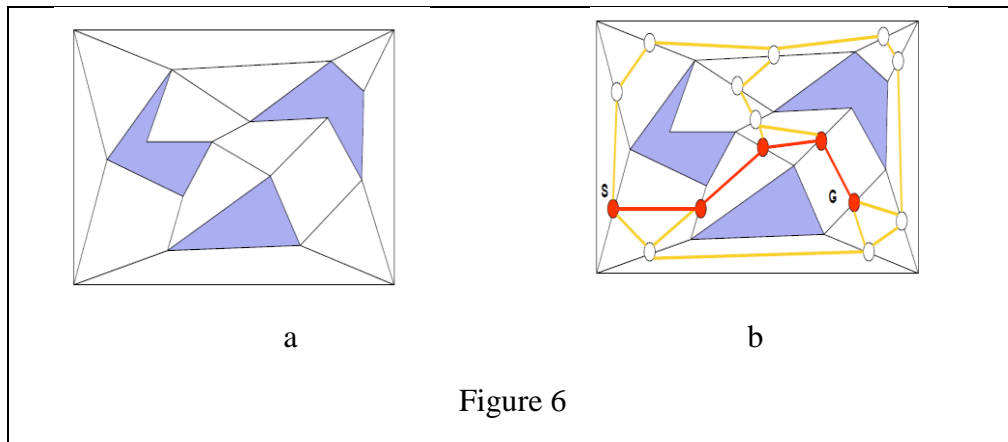
- a) The geometry of each cell should be simple to make it easy to compute a path between any two configurations.
- b) It should not be difficult to test the adjacency of any two cells and to find a path crossing the portion of boundary shared by two adjacent cells.

Three planning methods based on exact cell decomposition are:

- Trapezoidal cell decomposition
- Translation and rotation in plane cell decomposition
- Collins cell decomposition

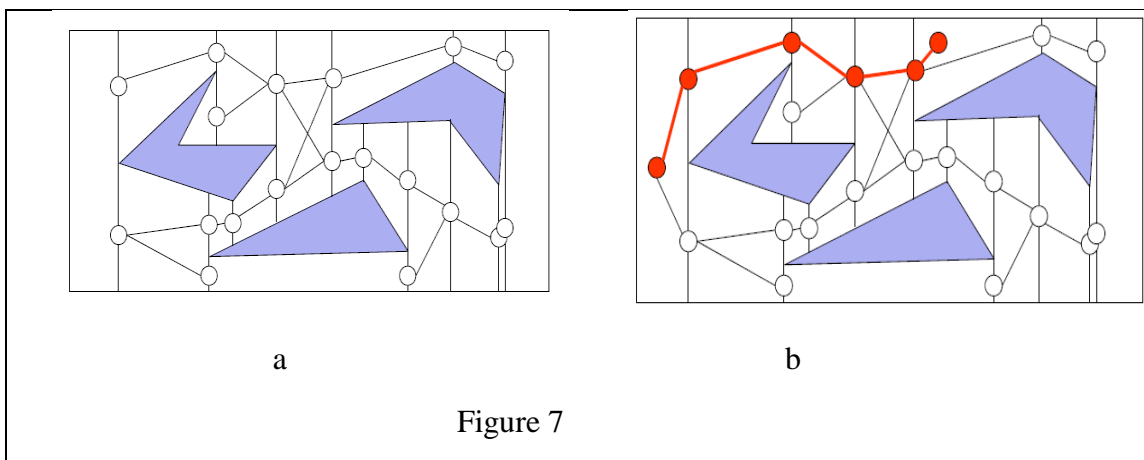


In figure 6 a collection of non-overlapping cells:  $\text{Union}(\text{Cells}) = \text{Free Space}$ . Finite set of convex polygons that cover the space (fig6 a), Midpoints of adjacent cells are “crossings”, then graph search(fig 6 b).



### Trapezoidal exact cell decomposition

Figure 7 describe trapezoidal exact cell decomposition a collection of non-overlapping cells:  $\text{Union}(\text{Cells}) = \text{Free Space}$ , then extend a bi-directional vertical line from each vertex until collision(fig 7a). Again a graph search(fig7 b).



### Exact cell decomposition analysis

- Complete? Yes
- Optimal? No
- Advantage? Efficiency!

## Approximate cell decomposition

Represent the robot free space as a collection of cells. The cells have a simple prespecified shape (eg. Rectangloid shape). Such that donot in general allow to represent free space exactly. The reason for standardization of the shape of cells are :

1. to achieve space decomposition iteratively by simple computation
2. easy to implement numerically
3. one can directly control the amount of the free space around a generated path by setting a minimal size for the cell
4. may fail to find a free path even if one exist

## Divide and label method (quad tree)

- The most usual approach: QUADTREE DECOMPOSITION

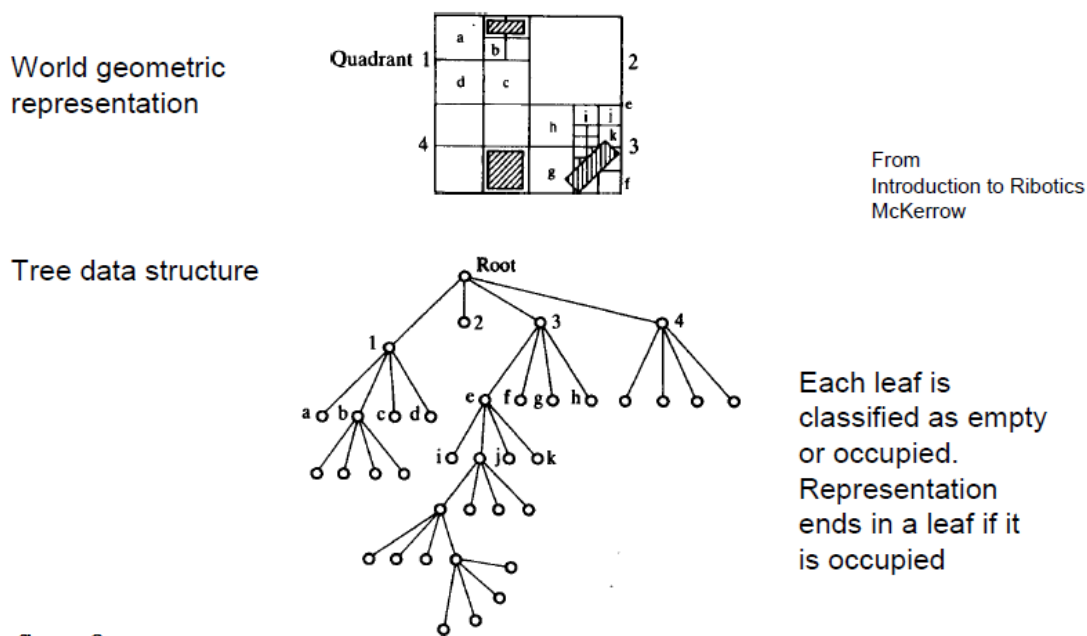
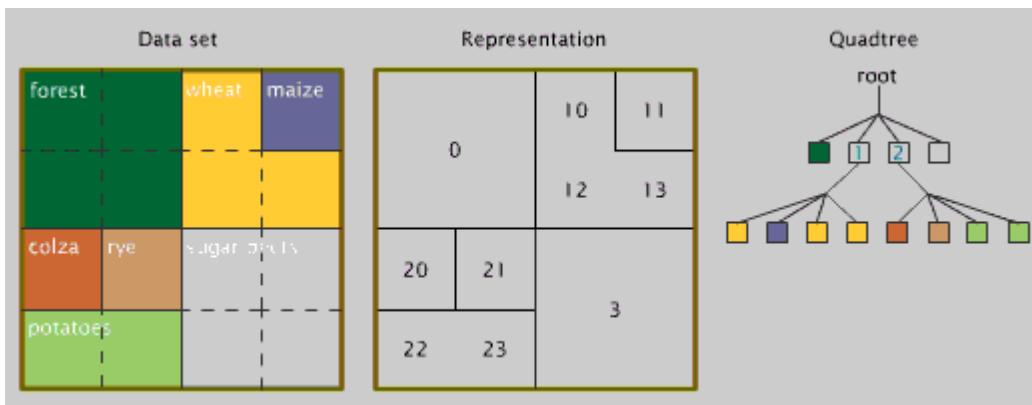


figure 8

## Quadtrees

Quadtrees are recursive grids. They are created by recursively subdividing each map square with non-uniform attributes into four equal-sized sub-squares. The division is repeated until a square is uniform or the highest resolution is reached. Quadtrees reduce memory requirements hereby allowing efficient partitioning of the environment. A single cell can be used to encode a large empty region.

Quadtrees represent a partition of space in two dimensions by decomposing the region into four equal quadrants, subquadrants and so on until the contents of the cells meet some criterion of data occupancy. The resolution (cell size) of the grid varies depending on the data density.



### An example

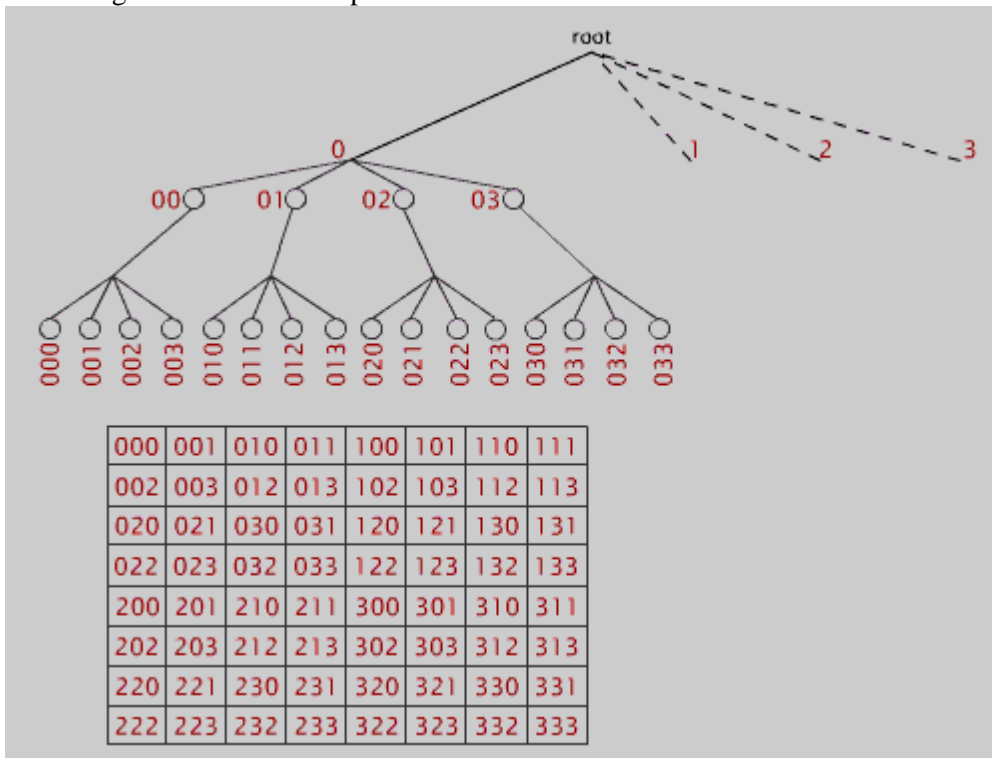
Considering a picture as a matrix  $A$  whose dimension is a power of 2, say  $2^n$ , this can be subdivided into four square matrices  $A_0, A_1, A_2, A_3$ , whose dimensions are half of  $A$ . This process can be repeated recursively  $n$  times, until the pixels within a quadrant are all of the same value. The levels can be numbered, starting with zero for the whole picture, down to  $n$  for the single pixel. A particular square may be labeled with one of the symbols 0, 1, 2, or 3, concatenated to the label of its predecessor square.

In this way, single pixels will have labels that are  $n$  characters long. We can express this arrangement as a tree, whose nodes correspond to the squares. Nodes are connected if one of the corresponding squares immediately contains the other. The root of the tree corresponds to the whole picture, the leaves to the single pixels, and all other nodes have down degree 4.

Since the  $k$ th-level contains  $4^k$  squares, the tree has a total of:

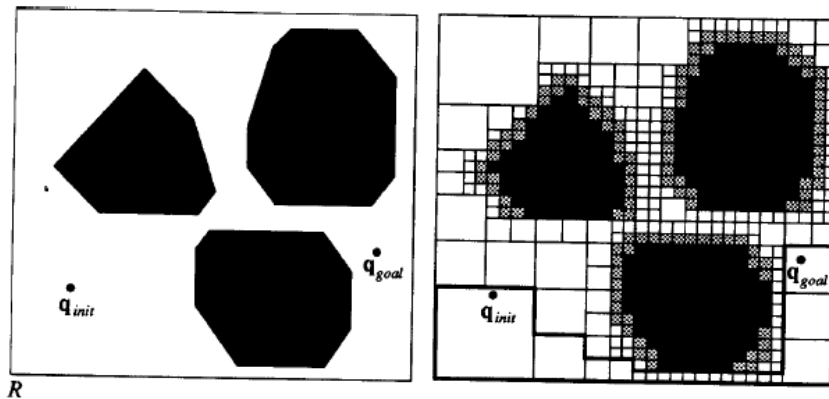
$$N = \sum_{k=0}^n 4^k = \frac{4^{n+1} - 1}{3} \approx \frac{4}{3} 4^n$$

nodes. Therefore, there are approximately 33% more nodes than pixels. The following figure shows the addressing notation for a  $8 \times 8$  picture:



## Searching Quadtrees

To search a linear quadtree index, in order to find stored data inside a search window, the window itself may be described in the form of a list of quadtree cells that cover it. It is not necessary for this search list to correspond exactly with the window, provided it covers it entirely. Once stored data cells are found that overlap the search cells, precise comparison can be performed with an exact (vector format) geometric definition of the search window.



- The rectangle  $R$  is recursively decomposed into smaller rectangles
- At a certain level of resolution, only the cells whose interiors lie entirely in the free space are used
- A search in this graph yields a collision free path

From  
Robot Motion Planning  
J.C. Latombe

figure 9

- Represent each free cell by its central point
- Do a graph search, minimizing the total path length
- Result: a set of spaced points

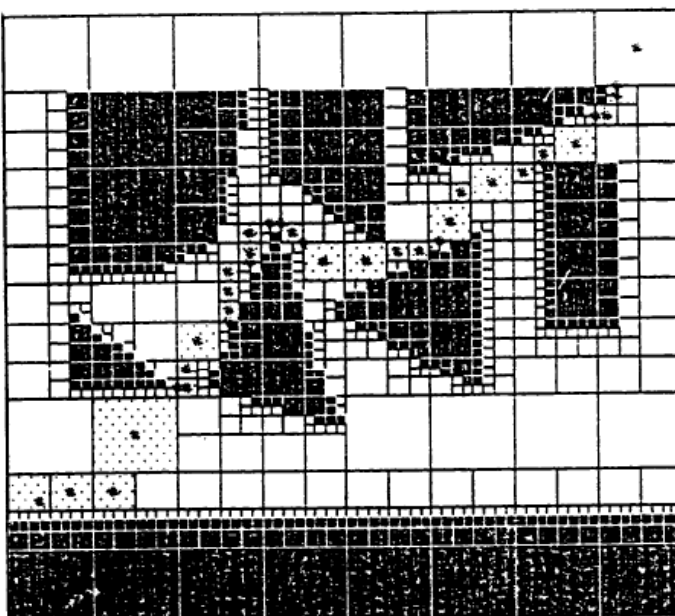
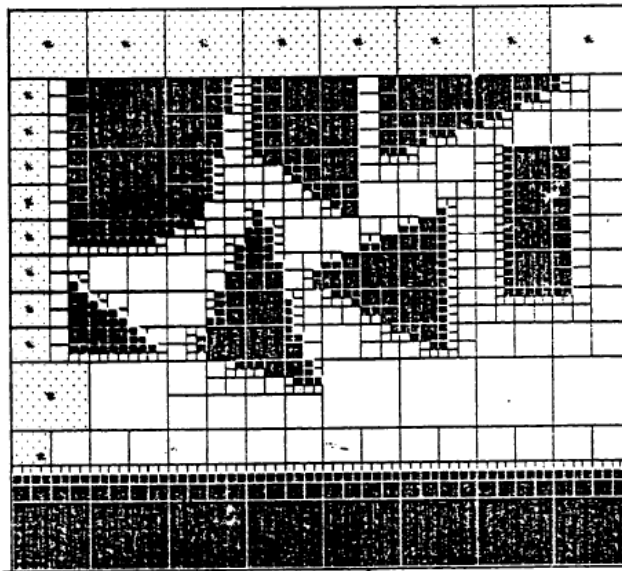


figure 10



Additional constraint:

Minimum distance to an obstacle is set

figure 11

## Potential Field

- The goal location generates an **attractive potential** – pulling the robot towards the goal
- The obstacles generate a **repulsive potential** – pushing the robot far away from the obstacles
- The **negative gradient of the total potential** is treated as an artificial force applied to the robot

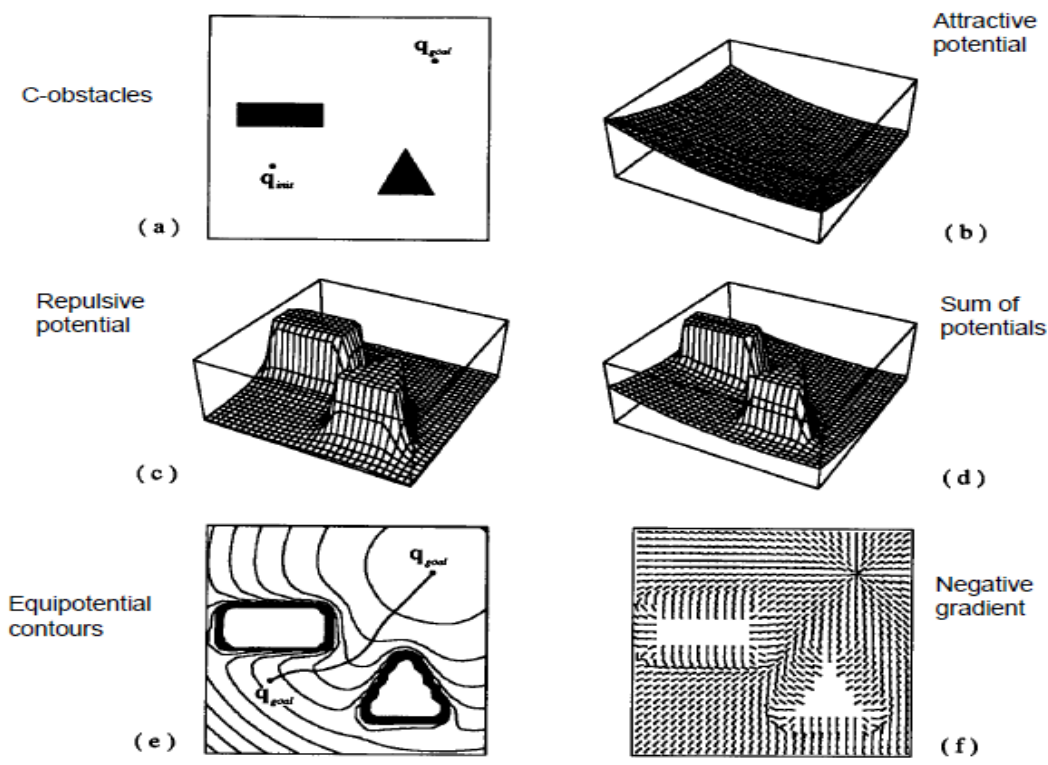


figure12

From  
Robot Motion Planning  
J.C. Latombe

# Comparative Study

Figure 8 show the comparative study between MotionPlanning methods

Comparison				
	Potential Fields	Approx Cell Decomp	Voronoi	Visibility
Practical above 2 or 3 D?	😊😊😊	😊😊😊		
Practical above 8 D?	😊😊😊			
Fast to Compute?	😊😊😊	😊😊😊		In 2-d
Usable Online?	😊😊😊	😊😊😊?	😊😊😊?	
Gives Optimal?				In 2-d
Easy to Implement?	😊😊😊		😊😊😊?	

fig18

## Kinematic Constraints

- In the basic path planning problem the only constraint of robot motion is due to obstacles
- There may occur other constraints – kinematic constraints (objects that cannot translate and rotate freely in the workspace)
- Two types of kinematic constraints
  - **Holonomic Constraints**
    - Do not fundamentally change the path planning problem
  - **Nonholonomic Constraints**
    - Much harder to deal in terms of path planning

## Holonomic constraints

A holonomic equality constraint is an equality relation among the parameters of the minimally-represented configuration space that can be solved for one of the parameters. Such a relation reduces the dimension of the actual configuration space of the robot by one. A set of  $k$  holonomic constraints reduces it by  $k$ . For example, a robot limited to rotating around a fixed axis has a configuration space of dimension 4 instead of 6 (since revolute joints impose 2 holonomic constraints).


### Holonomic Constraints

$$F(q, t) = 0$$

$F$  is a smooth function with non-zero derivative

$$F(q_1, q_2, \dots, q_m, t) = 0$$

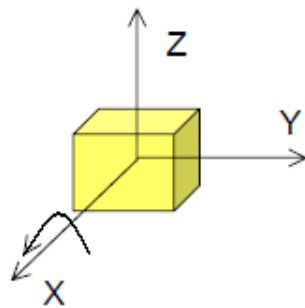
1 holonomic constraint = Relation (equality or inequality) among the parameters of  $C$  that can be solved for one of them as a function of the others

$k$  independent holonomic constraints   $\dim C = m - k$

**figure 13**

### Holonomic Constraints - Example

- A – tridimensional object that:
  - Can freely translate
  - Has a rotation along a fixed axis (relative to  $F_A$ )



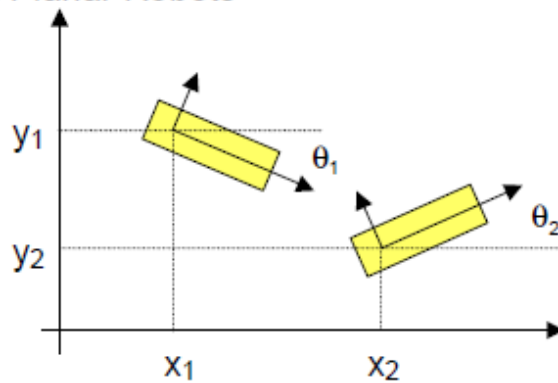
- Pitch angle = yaw angle = 0
- These two independent equations constraints the configuration

$$\begin{array}{lcl} \dim C = 6 & q = (x, y, z, \theta, \zeta, \psi) & \\ \downarrow & \zeta = 0, \psi = 0 & \text{holonomic constraints} \\ \dim = 4 & & \end{array}$$

fig14

### Articulated Robots

- 2 Planar Robots



$$q = (x_1, x_2, y_1, y_2, \theta_1, \theta_2)$$

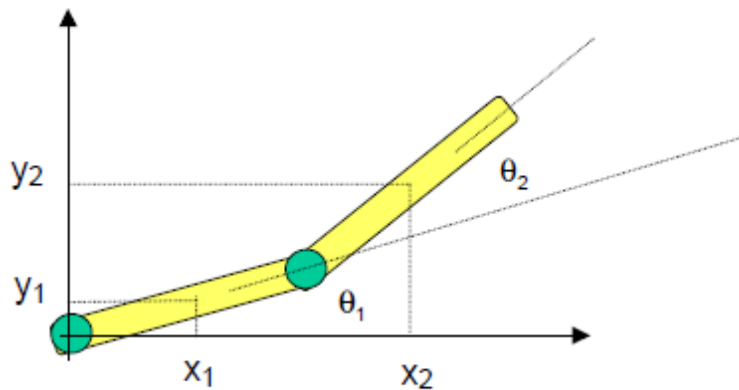
$$\dim C = 6$$

there are no holonomic constraints

fig15



- Planar Manipulator with two links



$$C' = (x_1, x_2, \theta_1, \theta_2)$$

there are two holonomic constraints

**fig16**  $\dim C' = \dim C - 2 = 4$

## Nonholonomic Constraints

A nonholonomic equality constraint is a non-integrable equation involving the configuration parameters and their derivatives (velocity parameters). Such a constraint does not reduce the dimension of the configuration space, but instead reduces the dimension of the space of possible differential motions.

For example, a car-like robot has 3 dimensions: two for translation and one for rotation. However, the velocity of **R** is required to point along the main axis of **A**. (This is written as  $-\sin T \, dx + \cos T \, dy = 0$ .)

The instantaneous motion of the car is determined by two parameters: the linear velocity along the main axis, and the steering angle. However, when the steering angle is non-zero, the robot changes orientation, and its linear velocity with it, allowing the robot's configuration to span a three-dimensional space. Restricting the steering angle to  $\pi/2$  restricts the set of possible differential motions without changing its dimension.

Nonholonomic constraints restrict the geometry of the feasible free paths between two configurations. They are much harder to deal with in a planner than holonomic constraints.

In general, a robot with nonholonomic constraints has fewer *controllable* degrees of freedom than it has actual degrees of freedom; these are equal in a holonomic robot.

## Uncertainty

A robot may have little or no prior knowledge about its workspace. The more incomplete the knowledge, the less important the role of planning. A more typical situation is when there are errors in robot control and in the initial models, but these errors are contained within bounded regions.

## Grasp Planning

Many typical robot operations require the robot to grasp an object. The rest of the operation is strongly influenced by choices made during grasping.

Grasping requires positioning the gripper on the object, which requires generating a path to this position. The grasp position must be accessible, stable, and robust enough to resist some external force. Sometimes a satisfactory position can only be reached by grasping an object, putting it down, and re-grasping it. The grasp planner must choose configurations so that the grasped objects are stable in the gripper, and it should also choose operations that reduce or at least do not increase the level of uncertainty in the configuration of the object.

The object to be grasped is the *target object*. The *gripping surfaces* are the surfaces on the robot used for grasping.

There are three principal considerations in gripping an object. They are:

- **safety** -- the robot must be safe in the initial and final configurations
- **reachability** -- the robot must be able to reach the initial grasping configuration and, with the object in hand, reach the final configuration
- **stability** -- the grasp should be stable in the presence of forces exerted on the grasped object during transfer and parts-mating motions

*Example: peg placement.* By tilting a peg the robot can increase the likelihood that the initial approach conditions will have the peg part way in the hole. Other solutions are chamfers (a widening hole, producing the same effect as tilting), search along the edge, and biased search (introduce bias so that search can be done in at most one motion and not two, if the error direction is not known).

- Three different aspects in MOTION PLANNING
  - PATH PLANNING
  - MANOEUVRE PLANNING
  - TRAJECTORY GENERATION

### PATH

A PATH is a geometric locus of the points – in a given space – where the robot has to pass

### TRAJECTORY

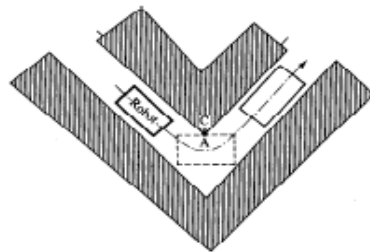
A TRAJECTORY is a path for which a temporal law is specified (e.g., acceleration and velocity in each point)

### MANOEUVERS

A mobile robot is not a point in the space

Piano-mover's problem

A path that the rectangular robot can negotiate only if it rotates around A as it turns the corner C



Parking a car in a narrow parking lot



**fig17**

## References

- 'Robot Motion Planning and Control', J.-P. Laumond (Ed.), Springer-Verlag London Limited 1998 .
- 'Introduction to Autonomous Mobile Robots Intelligent Robotics and Autonomous Agents ', Siegart, Roland.; Nourbakhsh, Illah Reza ,MIT Press ,2004.
- *Robot Motion Planning*. J.C. Latombe. Kluwer Academic Publishers, Boston, MA, 1991

# Basic Concepts of Robot control

## Robot Control System Task

The task of a robot control system is to execute the planned sequence of motions and forces in the presence of unforeseen errors.

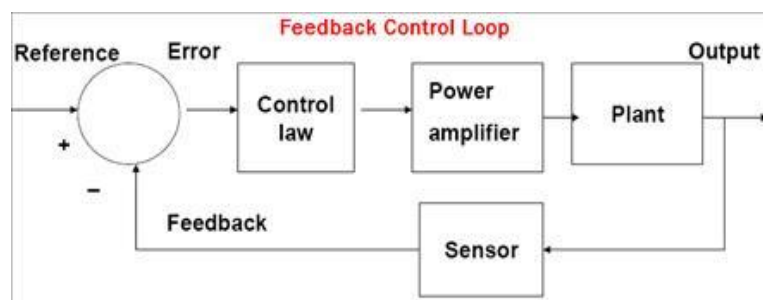
## Errors can arise from:

- inaccuracies in the model of the robot,
- tolerances in the workpiece,
- static friction in joints,
- mechanical compliance in linkages,
- electrical noise on transducer signals, and
- limitations in the precision of computation.

## Open Loop Control

No Feedback! Basic control suitable for systems with simple loads, Tight speed control is not required, no position or rate-of-change sensors, on each axis, there is a fixed mechanical stop to set the endpoint of the robot, its called “stop-to-stop” or “pick-and-place” systems.

The desired change in a parameter is calculated (joint angles), The actuator energy needed to achieve that change is determined, and the amount of energy is applied to the actuator. If the model is correct and there are no disturbances, the desired change is achieved.



## Feedback Control Loop

Determine rotor position and/or speed from one or more sensors. Position of robot arm is monitored by a position sensor, power to the actuator is altered so that the movement of the arm conforms to the desired path in terms of direction and/or velocity. Errors in positioning are corrected.

### **Feedforward Control**

It is a control, where a model is used to predict how much action to take, or the amount of energy to use. It is used to predict actuator settings for processes where feedback signals are delayed and in processes where the dynamic effects of disturbances must be reduced.

### **Adaptive Control**

This control uses feedback to update the model of the process based upon the results of previous actions. The measurements of the results of previous actions are used to adapt the process model to correct for changes in the process and errors in the model. This type of adaption corrects for errors in the model due to long-term variations in the environment but it cannot correct for dynamic changes caused by local disturbances.

### **Robot Arm Configurations:**

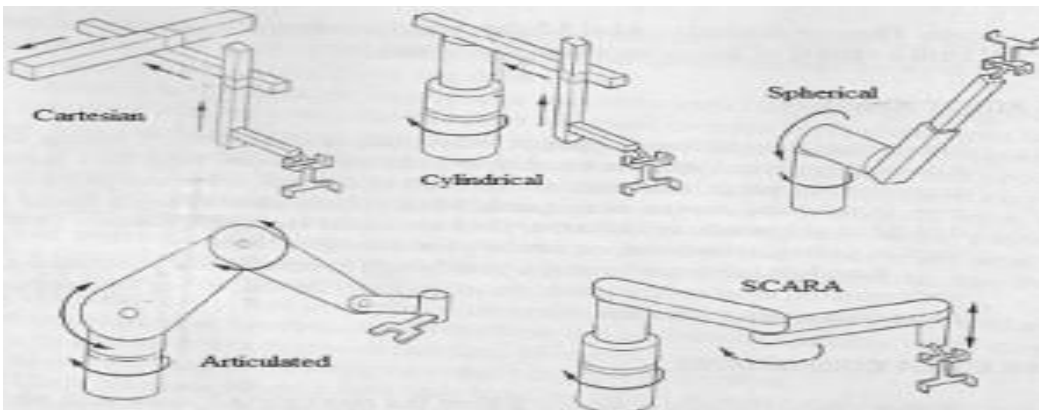
- Cartesian (3P)
- Cylindrical (R2P)
- Spherical (Polar) (2 RP)
- Articulated (3R)
- SCARA ( 2R in horizontal + 1P in vertical plane)

#### **Casrtesian (3P)**

- Due to their rigid structure they can manipulate high loads so they are commonly used for pick-and-place operations, machine tool loading, in fact any application that uses a lot of moves in the X,Y,Z planes.
- These robots occupy a large space, giving a low ratio of robot size to operating volume. They may require some form of protective covering.

#### **Cylindrical (R2P)**

- They have a rigid structure, giving them the capability to lift heavy loads through a large working envelope, but they are restricted to area close to the vertical base or the floor.
- This type of robot is relatively easy to program for loading and unloading of palletized stock, where only the minimum number of moves is required to be programmed.



### **Spherical (Polar) (2 RP)**

- These robots can generate a large working envelope.
- The robots can allow large loads to be lifted.
- The semi-spherical operating volume leaves a considerable space near to the base that cannot be reached.
- This design is used where a small number of vertical actions is adequate: the loading and unloading of a punch press is a typical application.

### **Articulated Arm (3R)**

- This is the most widely used arm configuration because of its flexibility in reaching any part of the working envelope.
- This configuration flexibility allows such complex applications as spray painting and welding to be implemented successfully.

### **SCARA**

- Although originally designed specifically for assembly work, these robots are now being used for welding, drilling and soldering operations because of their repeatability and compactness.
- They are intended for light to medium loads and the working volume tends to be restricted as there is limited vertical movement.

### **End Effector**

- Attached to the wrist a hand “end effector”.
- The end effector is not considered as part of the robot’s manipulator.
- 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**

- 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 relative position of the tool with respect to the work as a function of time, it must also control the operation of the tool.

### **Grippers**

Grippers are end effectors used to grasp and manipulate objects during the work

cycle. The objects are usually work-parts that are moved from one location to another in the cell.

### **Examples of Grippers**

- Mechanical grippers, in which the part is held between mechanical fingers and the fingers are mechanically actuated
- Vacuum grippers, in which suction cups are used to hold flat objects
- Magnetized devices, for holding ferrous parts
- Adhesive devices, where an adhesive substance is used to hold a flexible material such as fabrics.

A **sensor** is an electronic device that transfers a physical phenomenon (temperature, pressure, humidity, etc.) into an electrical signal. Sensors in Robotics are used for both internal feedback control and external interaction with the outside environment.

### Desirable Features of Sensors

- Accuracy.
- Precision.
- Operating range.
- Speed of response.
- Calibration.
- Reliability.
- Cost.
- Ease of operation.

### **Potentiometers**

The general idea is that the device consists of a movable tap along two fixed ends. As the tap is moved, the resistance changes. The resistance between the two ends is fixed, but the resistance between the movable part and either end varies as the part is moved. In robotics, pots are commonly used to sense and tune position for sliding and rotating mechanisms.

### **Switch Sensors**

Switches are the simplest sensors of all. They work without processing, at the electronics level. Switches measure physical contact. Their general underlying principle is that of an open vs. closed circuit. If a switch is open, no current can flow; if it is closed, current can flow and be detected.

### **Principle of Switch Sensors**

Contact sensors: detect when the sensor has contacted another object.

Limit sensors: detect when a mechanism has moved to the end of its range.

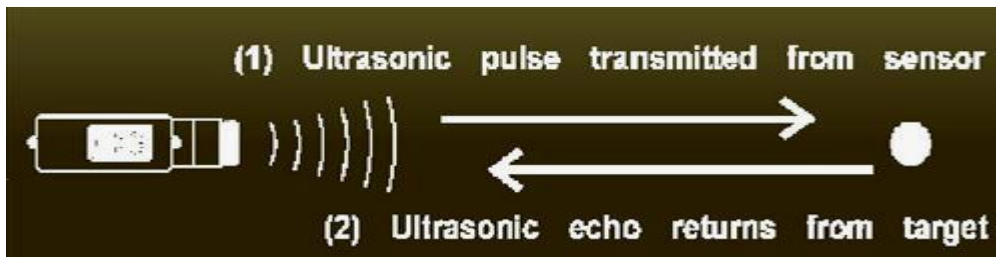
Shaft encoder sensors: detects how many times a shaft turns by having a switch click (open/close) every time the shaft turns.

### **Ultrasonic Sensors**

Ultrasonic sensors are used in wide range due to some considerations:

- very cheap in compare with other type of detectors.
- relatively have a good sensitivity
- available in different shapes.

Ultrasonic sensors measure the distance or presence of target objects by sending a pulsed ultrasound wave at the object and then measuring the time for the sound echo to return. Knowing the speed of sound, the sensor can determine the distance of the object.



### Ultrasonic Distance Sensing

Ultrasound sensing is based on the time-of-flight principle. The emitter produces a sonar of sound, which travels away from the source, and, if it encounters barriers, reflects from them and returns to the microphone. The amount of time it takes for the sound beam to come back is tracked and is used to compute the distance the sound traveled.

Sound wave travels with is a constant speed, which varies slightly based on ambient temperature. At room temperature, sound travels at 1.12 feet per millisecond.

### Ultrasonic Sensors Applications

\*Long sonar readings can be very inaccurate, as they may result from false rather than accurate reflections For example, a robot approaching a wall at a steep angle may not see the wall at all, and collide with it!

\*Sonar sensors have been successfully used for very sophisticated robotics applications, including terrain and indoor mapping, and remain a very popular sensor choice in mobile robotics.

One can find ultrasound used in a variety of other applications; the best known one is ranging in submarines. The sonars there have much more focused and have longer-range beams. Simpler and more mundane applications involve automated “tape measures”, height measures, burglar alarms, etc.

**Light sensors** measure the amount of light impacting a photocell, which is basically a resistive sensor. The resistance of a photocell is low when it is brightly illuminated, it is high when it is dark.

Light sensors can measure:

Light intensity (how light/dark it is)

Differential intensity(difference between photocells)

Break-beam (change/drop in intensity)



## **Optical Sensors**

Optical sensors consists of an emitter and a detector. Depending of the arrangement of emitter and detector relative to each other, we can get two types of sensors:

Reflective sensors (the emitter and the detector are next to each other, separated by a barrier; objects are detected when the light is reflected off them and back into the detector)

Break-beam sensors (the emitter and the detector face each other; objects are detected if they interrupt the beam of light between the emitter and the detector)

The emitter is usually made out of a light-emitting diode (an LED), and the detector is usually a photodiode/phototransistor in Reflective optical sensors. A light bulb in combination with a photocell can make a break-beam sensor.

## **Light Reflective Sensors**

Light reflectivity depends on the color (and other properties) of a surface. It may be harder (less reliable) to detect darker objects this way than lighter ones. In the case of object distance, lighter objects that are farther away will seem closer than darker objects that are not as far away.

What can be done with light reflectivity?

- object presence detection

- object distance detection

- surface feature detection (finding/following markers/tape)

- wall/boundary tracking

- rotational shaft encoding (using encoder wheels with ridges or black & white color)

- bar code decoding

## **Light Sensors Calibration**

Source of noise in light sensors is ambient light. The best thing to do is subtract the ambient light level out of the sensor reading, in order to detect the actual change in the reflected light, not the ambient light. This done by taking two readings of the detector, one with the emitter on, and one with it off, and subtracting the two values from each other. The result is the ambient light level, which can then be subtracted from future readings. This process is called sensor calibration.

## **Beam-break Sensors**

Any pair of compatible emitter-detector devices can be used to produce such a sensors, for example: an incandescent flashlight bulb and a photocell, red LEDs and visible-light- sensitive photo-transistors or infra-red IR emitters and detectors

## **Infra Red Sensors**

Infra red sensors are a type of light sensors, which function in the infra red part of the frequency spectrum. IR sensors are active sensors: they consist of an emitter and a receiver. IR sensors are used in the same ways that visible light sensors: as break-beams and as reflectance sensors. IR is preferable to visible light in robotics

applications because it suffers a bit less from ambient interference, because it can be easily modulated, and simply because it is not visible.

### **Voice recognition**

This process involves determining what is said and taking an action based on the perceived information. Voice recognition systems generally work on the frequency content of the spoken words. Any signal may be decomposed into a series of sines & cosines of different frequencies at different amplitudes. It is assumed that every word (letter), when decomposed into the constituent frequencies, will have a unique signature composed of its major frequencies, which allow the system to recognize the word. The user must train the system by speaking the words a priori to allow the system to create a look up table of the major frequencies of the spoken words. When a word is spoken and its frequencies determined, the result is compared with the look up table. If a close match is found, the word is recognized. A universal system that recognizes all accents and variations in speaking may not be either possible or useful.

For better accuracy, it is necessary to train the system with more repetitions. The more accurate the frequencies, the narrower the allowable variations. This means that if the system tries to match many frequencies for better accuracy, in the presence of any noise or any variations in the spoken words, the system will not be able to recognize the word. On the other hand, if a limited number of frequencies is matched in order to allow for variations, then it may mix the words with other similar words.

Many robots have been equipped with voice recognition systems in order to communicate with the users. In most cases, the robot is trained by the user and it can recognize words that trigger a certain action in response. When the voice-recognition system recognizes the word, it will send a signal to the controller, which, in turn, will run the robot as desired.

### **Voice Synthesizers**

Voice synthesis is accomplished in two different ways:

One is to recreate each word by combining phonemes and vowels this can be accomplished with commercially available phonemes chip and a corresponding program. Although this type of system can reproduce any word, it sounds unnatural and machine like. The alternative is to record the words that the system may need to synthesize and to access them from memory or tape as needed. Although this system sounds very natural, it is limited. As long as all the words that the machine needs to say are known a priori, this system can be used.

### **Intelligent Control System Properties:**

- 1) Interact with its environment, Make decision when things go wrong during the work cycle,
- 2) Communicate with human beings,
- 3) Make computations.
- 4) Operate in response to advanced sensors.

## **Autonomous Robot Control**



The basic task of autonomous robot is to navigate from an initial position to a desired target position. To achieve the goals of autonomy, an intelligent control system must be designed to manage the robot's operation. Autonomy of robots can range from remote controlled means, through program controlled ones, to completely autonomous mobile robots. An aim of intelligent control research is to develop autonomous system that can dynamically interact with the real world.

**Reference: Robot basics online resource for robotics**

# Intelligent Planning

## Introduction

The word ‘planning’ informally refers to the generation of the sequence of actions to solve a complex problem. For instance, consider the problem of placing the furniture in your new-built house, so that you can fully utilize the available free space for common use and the rooms look beautiful.

An analysis of the problem reveals that there exist many possible alternative solutions to the problem. But finding even a single solution is not so easy.

## Planning with If-Add-Delete Operators

We consider the problem of blocks world, where a number of blocks are to be stacked to a desired order from a given initial order. The initial and the goal state of the problem is given similar to [fig. 2](#) and [3](#). To solve this type of problem, we have to define a few operators using the if-add-delete structures, to be presented shortly.

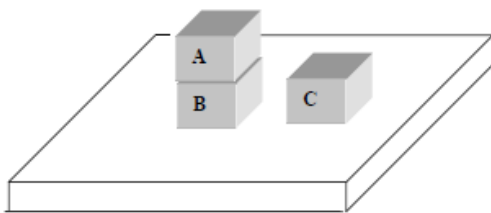


Fig. 2: The initial state of Blocks World problem.

The initial state:

On (A,B)  
On (B, Table)  
On (C, Table)  
Clear (A)  
Clear (C)

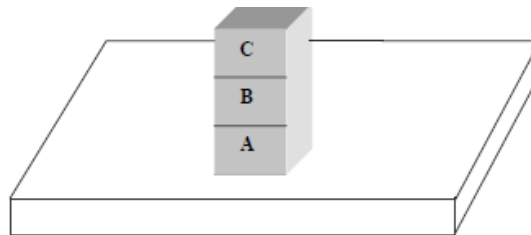


Fig. 3: The goal state of Blocks World problem.

The goal state:

On (B, A)  
On (C, B)  
On (A, Table)  
Clear (C)

We can try to solve the above problem by the following sequencing of operators. Rule 2 is applied to the initial problem state with an instantiation of  $X = A$  and  $Y = B$  to generate state S1 ([fig.4](#)). Then we apply Rule 3 with an instantiation of  $X = B$  and  $Z = A$  to generate state S2. Next Rule 3 is applied once again to state S2 with an instantiation of  $X = C$  and  $Z = B$  to yield the goal state. Generating the goal from the given initial state by application of a sequence of operators causes expansion of many intermediate states. So, forward reasoning is not appropriate for such problems. Let us try to explore the problem through backward reasoning.

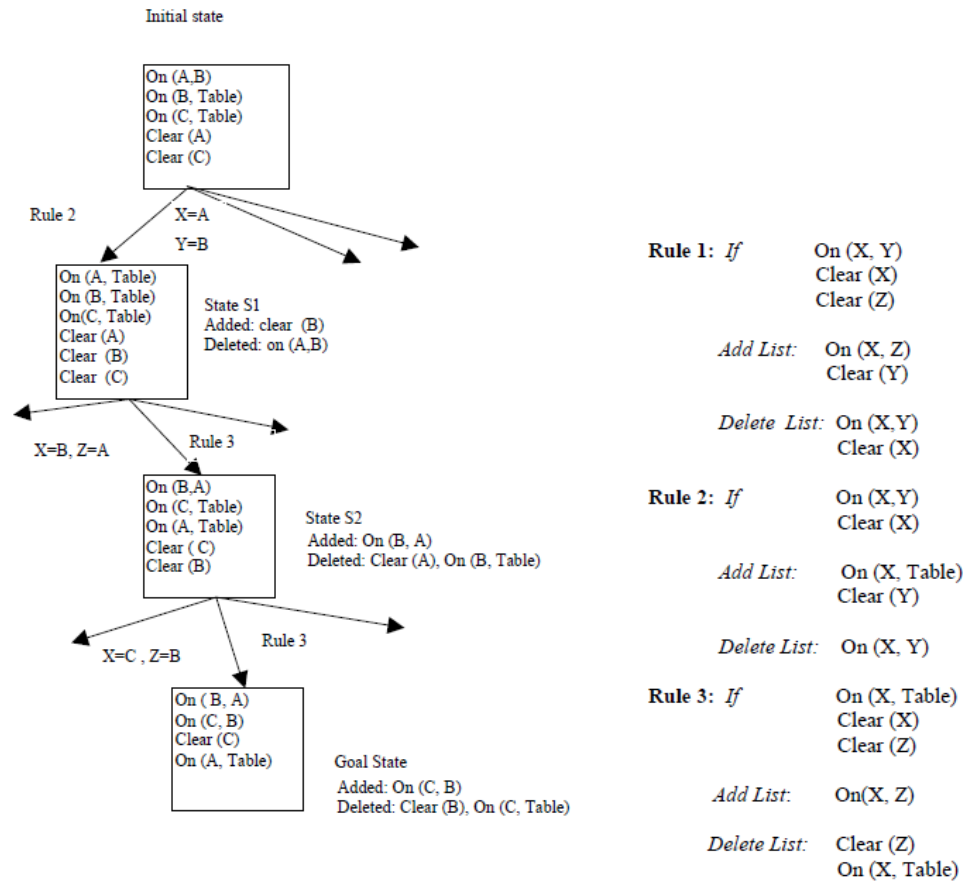


Fig. 4: The breadth first search of the goal state.

## Least Commitment Planning

The schemes of planning, described above, determine a list of sequence of operators, by a forward or backward reasoning in the state-space. When the number of blocks in the ‘Blocks world problem’ is large, determining the complete order of the sequence of operators is difficult by the proposed scheme.

An alternative approach for planning is to determine ‘approximate (partial) sequence of operators for each goal’ separately and defer the ordering of their steps later. Such planning is referred to as **least commitment planning**. In the literature of AI this is also called **non-linear planning**.

We now explain why it is called so. Since we delay in committing the order of operator in the partial plan of a sub-goal, it is called the leastcommitment planning. Further, the partial plan for each sub-goal is generated in parallel, unlike the previous state-space reasoning method for planning. It may be recollected that in the state-space approach, only after satisfying a sub-goal, the next sub-goal is considered for satisfaction. Thus in contrast to the state-space approach for linear planning, the current approach is termed non-linear planning.

### Operator Sequence in Partially Ordered Plans

Suppose realization of a goal requires 5 steps (sub-goals), denoted by operators, G1, G2, G3, G4 and G5 respectively. Let the order of the steps be represented by a graph like that in fig.10. Here the firm line ( — ) denotes exact ordering, while dotted line ( - - ) denotes the 'least committed' dependence relations (constraints) between two operators. Thus the above plan is an order of partially planned operators. The partially ordered plans for the problem of fig..10 are listed below:

{G1, G2, G3, G4, G5}

{G1, G3, G2, G4, G5} and

{G1, G3, G4, G2, G5}

We now have to select which of the above three partially ordered plans leads to a complete plan for the goal or the sub-goal. So, in the least commitment planning we first search in the space of partially ordered plans and then select the correct complete plan among those plans.

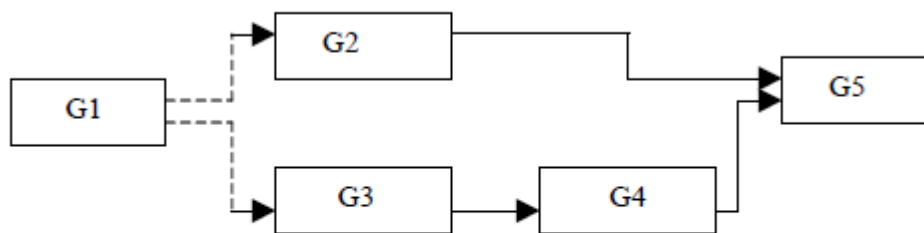


Fig. 10: Illustrating least commitment planning.

### Realizing Least Commitment Plans

For realizing a least commitment plan we require one or more of the following operations:

- Step Addition:** This stands for the generation of a partially ordered plan for one sub-goal.
- Promotion:** This constrains one step to come before another in a partially ordered plan.
- Declobbering:** Suppose state S1 negated (deleted) some precondition of state S3. So, add S2 such that S2 follows S1 and S3 follows S2, where S2 reasserts the negated pre-conditions of S3.
- Simple Assignment:** Instantiate a variable to ensure precondition of a step.
- Separation:** Instantiation of variables is sometimes not done intentionally to keep the size of the plan manageable.

The following example of the well-known 'blocks world' problem,

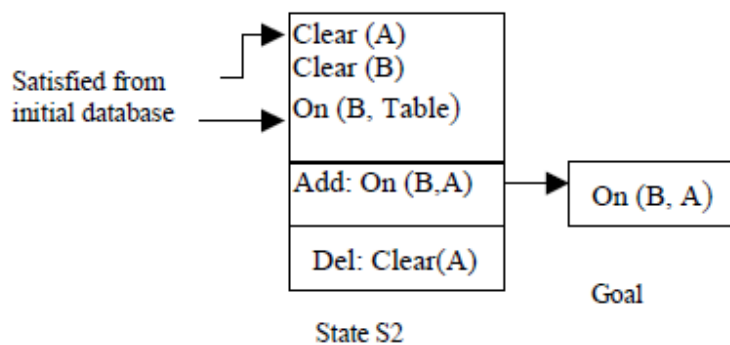
discussed earlier, will best illustrate the above definitions. Remember the problem was enlisted as follows:

**Given:**  $\text{On}(A,B) \wedge \text{Clear}(C) \wedge \text{Clear}(A) \wedge \text{On}(C, \text{Table}) \wedge \text{On}(B, \text{Table})$ .

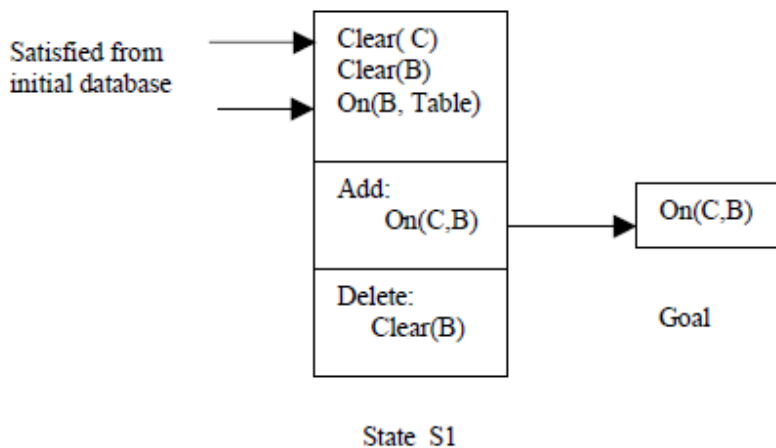
**Find a plan for:**  $\text{On}(B, A) \wedge \text{On}(C, B)$ .

To start solving the problem, we first generate partial plans to achieve  $\text{On}(B, A)$  and  $\text{On}(C, B)$  separately.

The goal  $\text{On}(A,B)$  may be generated by the following rule: If X is clear and Y is clear then put X on Y. Here the pre-conditions  $\text{Clear}(A)$  and  $\text{On}(B, \text{Table})$  are available in the in initial problem state. So, the partial plan for goal:  $\text{On}(B, A)$  can be constructed. The partial plan for this goal is presented in [fig11](#). To satisfy  $\text{On}(C, B)$  we need to generate its predecessor (see [fig. 12](#)).



**Fig. . 11:** The partial plan for the goal  $\text{On}(B, A)$ .



**Fig. 12:** The goal  $\text{On}(C, B)$  and its predecessor.

It may be noted that  $\text{Clear}(B)$  is a pre-condition of both the goals  $\text{On}(C,B)$  and  $\text{On}(B,A)$ , but the process of generating  $\text{On}(C,B)$  deletes  $\text{Clear}(B)$ . This posts an additional constraint that state  $S2$  should follow state  $S1$ . We denoted it by a dotted line (constraint link) in [fig13](#). Now to satisfy the pre-conditions of  $S1$  and  $S2$ , we need to add new steps. Note that  $\text{Clear}(A)$  and  $\text{On}(B, \text{Table})$  in both the states  $S1$

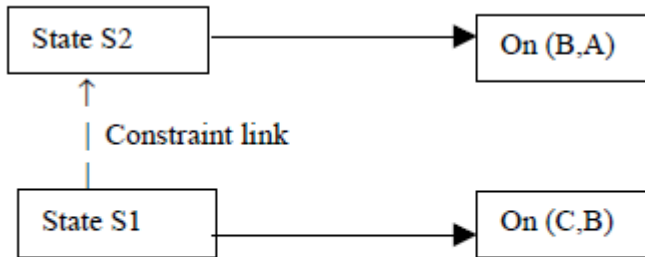
and S2 are satisfied. So, we need to satisfy Clear (B) only in state S2 and S1. To satisfy Clear (B) in S1 and S2, we employ the following rule:

*If* On (X,Y)  $\wedge$  Clear (X)

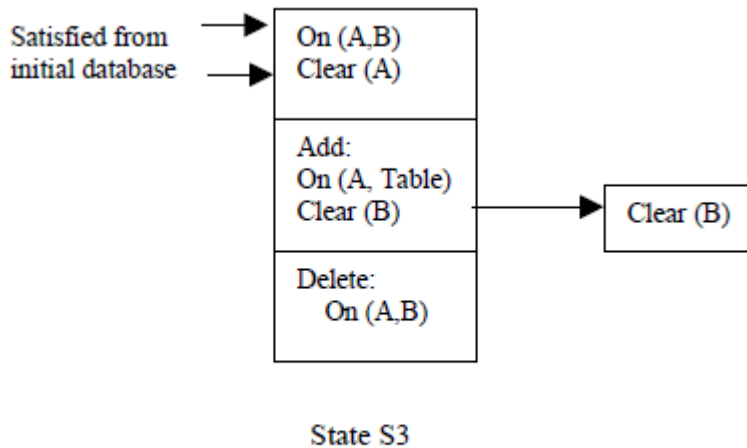
*Add:* On (X, Table)  $\wedge$  Clear (Y)

*Delete:* On(X,Y).

So, by backward reasoning, we generate the new state, vide [fig14](#).



**Fig. 13:** Precedence relationship of states by constraint (before) links.



**Fig. 14:** An approach to satisfy Clear (B).

We now have three partially ordered steps in our plan with one initial and one goal condition. These five partially ordered plans are presented below in a column structure.

**Plan 1:** *If* Clear ( C)  $\wedge$  Clear (B)  $\wedge$  On (B, Table)

*Add:* On (C,B)

*Delete:* Clear (B)

**Plan 2:** *If* Clear (A)  $\wedge$  Clear (B)  $\wedge$  On (B, Table)

*Add:* On (B,A)

*Delete:* Clear (A)

**Plan 3:** *If* On (A,B)  $\wedge$  Clear (A)



*Add:* On (A, Table)  $\wedge$  Clear (B)

*Delete:* On (A,B)

**Plan 4:** *If* Nil

*Add:* On (A,B)  $\wedge$  Clear (C)  $\wedge$  Clear (A)  $\wedge$  On (C, Table)  $\wedge$   
On (B, Table)

*Delete:* Nil

**Plan 5:** *If* On (B,A)  $\wedge$  On (C,B)

(goal) *Add:* Nil

*Delete:* Nil

The complete order of the plans that maintain satisfiability of the preconditions of each partial plan is given by

plan 4 < plan 3 < plan 2 < plan 1 < plan 5

where plan j < plan k means plan j is to be executed prior to plan k.

In the above scheme for ordering a list of partially ordered plans, we demonstrated only two steps: addition of steps and promotion by adding constraints. Let us now illustrate the principle of declobbering. Suppose, we choose the totally ordered plan as follows:

plan 4 < plan 3 < plan 1 < plan j < plan 2 < plan 5

where plan j will declobber the pre-condition (Clear (B)) of plan 2, which was clobbered by plan 1. The necessary steps in plan j are presented below:

**Plan j:** *If* On (C,B)  $\wedge$  Clear (C)

*Add:* On (C, Table), Clear (B)

*Delete:* On (C, B)

The incorporation of plan j between plan 1 and plan 2 serves the purpose of declobbering, but On (C,B) being deleted by plan j has to be executed later. Thus plan 1 has to be inserted again between plan 2 and plan 5.

The new total order of the plans thus becomes:

plan 4 < plan 3 < plan 1 < plan j < plan 2 < plan 1 < plan 5

This undoubtedly is bad planning and the reader may think that declobbering has no justification. But sometimes it is useful and the only approach to refine a plan.

The operations of least commitment planning we described so far include the first three. The operation of instantiating variables to ensure preconditions of a step is also clear from our previous examples. But the last operation of intentionally deferring a non-instantiation variable is useful in planning. For example assume that there are

two more blocks D and E on the table. In that case, instead of putting A on table in plan 3, we could put it on D and E as well; see our objective in plan 3 is to Clear (B). So, we employ the following rule to generate plan 3:

**Rule:** *If* On (X, Y)  $\wedge$  Clear (X)  $\wedge$  Clear (Z)

*Add:* On (X, Z)  $\wedge$  Clear (Y)

*Delete:* On (X, Y)

In the last rule Z could be a table or block D or E. We do not want to explicitly set the value of Z, because it is no longer required by other partial plans till now. Thus plan 3 could be:

**Plan 3:** *If* On (A,B)  $\wedge$  Clear (A)  $\wedge$  Clear (Z)

*Add:* On (A,Z)  $\wedge$  Clear (B)

*Delete:* On (A,B)

In this example the instantiation of Z is no longer required. However if Z is required to be instantiated later, we will then do it. It may be noted that the main benefit of deferring instantiation of variables is to keep the size of generated partial plans within limits.

## Hierarchical Task Network Planning

The hierarchical task network planning, also called **hierarchical planning**, is employed in complex decision making systems. It generates a relatively abstract ordering of steps to realize the goal and then each abstract step is realized with simpler plans. A hierarchical planning scheme looks somewhat like a tree structure, where the steps at the higher level of the tree represent more abstract and complex tasks. Let us, for example, consider the plan for ‘writing a book’.

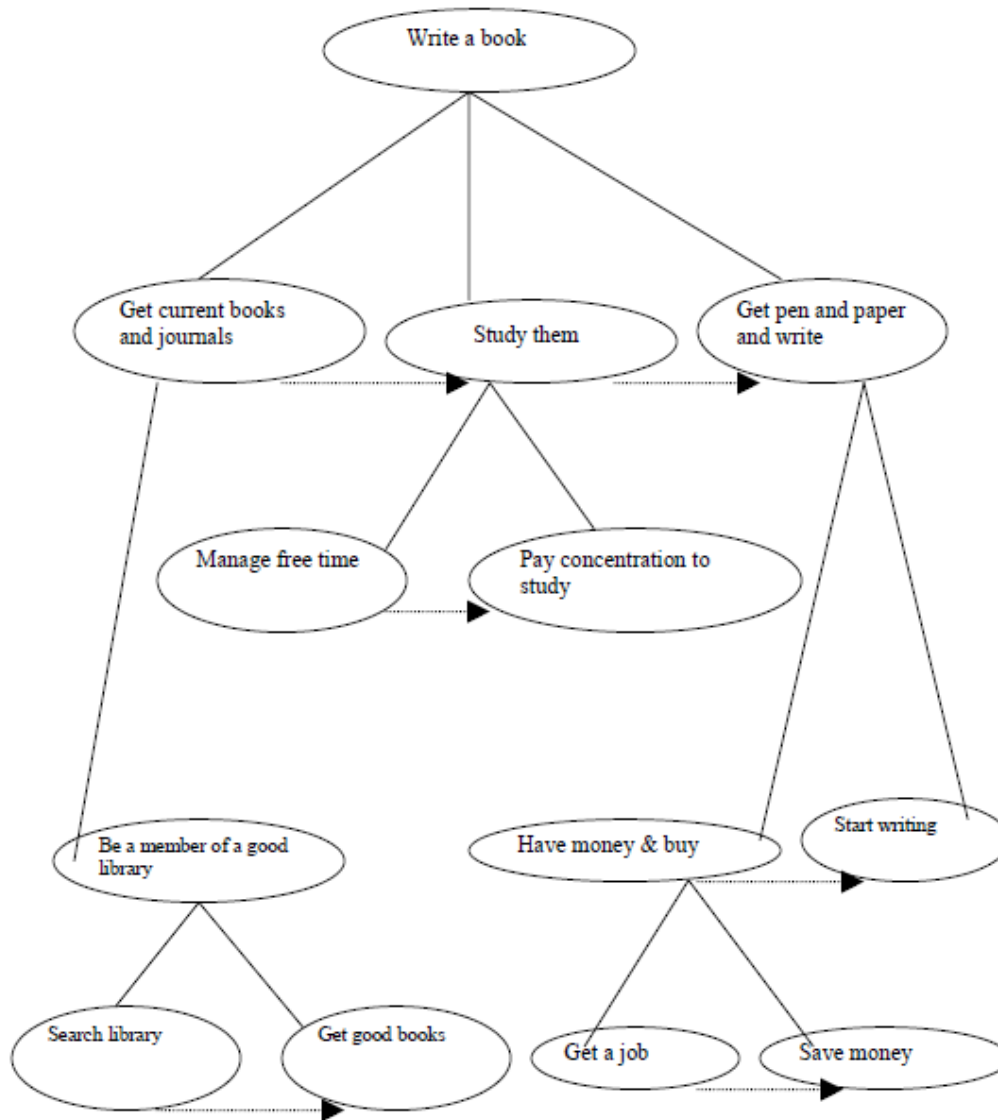
We, following the ABSTRIPS approach first break the plan into three linear abstract plans:

- i) get current books and journals,
- ii) study them and
- iii) get pen and paper and write. Each abstract plan is then realized by the children under it in a sequentially ordered fashion, denoted by the dotted arrow ( $- \dashrightarrow$ ) segment.

[Fig15](#) describes such a plan for ‘writing a book’. The steps in [fig.15](#) are simple and thus need no elaboration. The planning scheme in the present context takes care of the plan at a given level of the tree only before looking at the details in the next hierarchical level. Such a plan is often referred to as **length-first search**.

In the illustrative scheme of a hierarchical plan ([fig15](#)) we demonstrated only the feasible solution; but in situations we cannot guarantee the feasibility at the current

level, unless we explored at lower levels, So we may generate alternative abstract plans.



**Fig. 15:** A hierarchical plan of writing a book.

In [fig16](#), we describe such a plan, where the small dark rectangle denotes a primitive plan at a given level and the large rectangle (Y) denotes a sequential ordering of the primitive plans at a level. Let us assume that each level we select only one valid plan out of a possible number of  $b$  plans, i.e., the branching factor is  $b$ . Further, let the length of a selected plan at each layer be  $s$ . Thus, for executing such a plan, we need to consider a total of  $P$  plans [7], where

$$P = bs + bs^2 + bs^3 + \dots + bs^{d-1}$$

$$= \sum_{j=1}^d b(s)^j = O(b s^d).$$

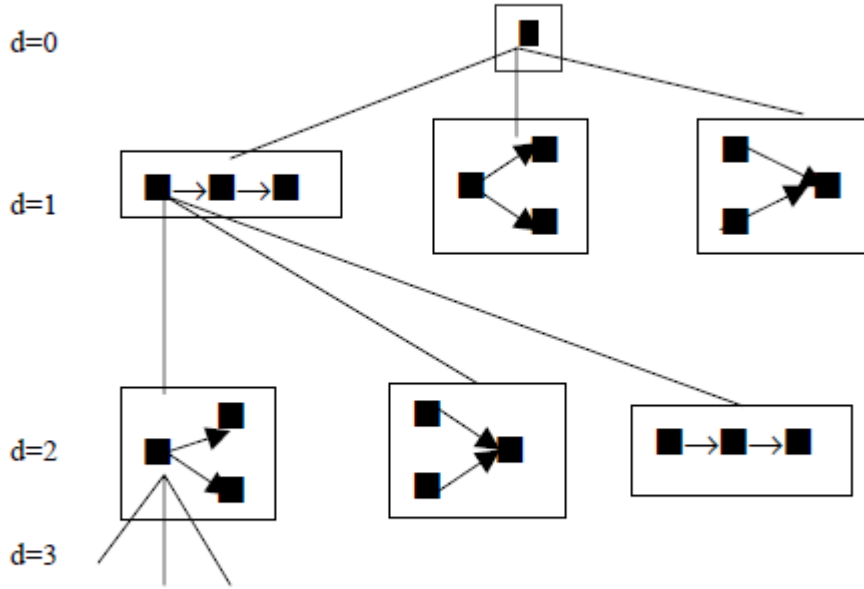


Fig. 16: A hierarchical plan with branching factor  $b=3$ , primitive steps  $s = 3$  in a plan and depth ( $d$ ) of the tree=3.

On the other hand, if we try to solve it by a linear planner it has to generate as many as

$$bs + (bs)^2 + (bs)^3 + \dots + (bs)^{d-1}$$

$$= O(bs)^d.$$

Further for linear ordering of these plans, we require a significant amount of search among these plans. The total search complexity for linear ordering will be  $O(bs)^{2d}$ . On the other hand, in a hierarchical plan, at each level, we select 1 out of  $b$  plans. So, the time required to eliminate inconsistent plans is  $O(b)$  and the time required to find a linear ordering at each level is  $O(s)$ . So, if there are  $d$  levels, the ordering time of plans is  $O(s \cdot d)$ . Now, we can compare the ordering time of a hierarchical planner with respect to a linear planner. The factor of improvement of a hierarchical planner with respect to a linear planner can be given by

$$\{(bs)^{2d} - (s \cdot d) / (s \cdot d)\} = (b^{2d} s^{2d-1} / d) - 1.$$

### **Exercises**

1. Given the following initial and the goal state for the Blocks world problem. Construct a set of operators (Rules) and hence generate a plan to reach the goal state from the initial state.

**Initial State:** On (C, A),  
Clear (C),  
On (B, Table) ,  
Clear (B).

**Goal State:** On (B, A),  
On (C, B).

2. Realize the above plan by the least commitment planning.  
3. Design a hierarchical plan for the construction of a house building. Clearly mark at least two sub-plans, which cannot be realized at the next level of the tree.

### **Reference**

Amit Konar , 'Artificial intelligence and Behavioral and Cognitive Modeling of the Human Brain',2000 by CRC Press LLC.