Ch1. Intelligent Robot

Automation and Robots

Automation:

The age of automation started in the eighteenth century, when machines began to take over the job that had previously been performed by human beings. Since that time, new and new machines have been finding their way into factories as more and more new products have been manufactured and conceived by the human beings.

Definition:

"Automation can be defined as a technology that in concerned with the use of mechanical, electric, electrical and computer based systems in the operation and control of production."

"It can also be defined as a technology that is aimed at replacing human beings with automatic machines in a manufacturing process, i.e. capability to operate without human interaction."

Examples of Automation:

- Mechanized Assembly Lines
- Feedback control systems.
- NC and CNC machines
- Robots
- Automatic assembly process

Types of Automation

1. **Hard Automation(Fixed automation-hardware automation)**
2. **Soft Automation(software automation)**
    - **A) Programmable Automation**
    - **B) Flexible Automation**

Hard Automation:

- It is a type of fixed automation.
- As time elapses, new and new products will be released into market.
- As a result of which the old setup or the hardware has to be changed to suit or to manufacture the new product.
- Hence, the specialized machines has to be shut down and the hardware has to be retooled to suit for the next generation of models.
- Since periodic modification of the production, hardware is required; this type of automation is called as hard Automation.

- They have limited flexibility.

Fixed Automation:

- Sequencing of the operations is fixed by the equipment configuration itself.
- Used in places where the volume of production is very high and it is required to process and produce the same product at very production rates.

Disadvantage:

1. Initial investment cost is very high.
2. Equipment is specially designed to produce one product.

Soft Automation:

- Soft automation is used when the volume of production is less/more and when a variety of products has to be manufactured in a less time.
- In soft automation, automatic machines such as robots are used which can be programmed to do a variety of tasks.
  a) Programmable Automation:
     - ✓ The sequence of operations can be changed under the control of a program; because of this ability and programming feature, many different and new products can be made economically in batches and at less time.
     - ✓ Theses automatic programmable machines which do the required job through various programmed motions and without human interventions are called as robot.
  b) Flexible Automation:
     - ✓ It is an extension or evolution of programmable automation.
     - ✓ Its goal is to allow manufacturing of a number of batches of different products by minimizing the time lost for reprogramming the sequence of operations and the machines employed to pass on from one batch to the next.

Robot

Definition:

"A robot is a software-controllable machines device that uses sensors to guide one or more end effectors through programmed motion in a workspace in order to manipulate physical objects."

Robot classification:

1. Classification based on work Envelope(Geometry)
2. Classification based on drive technology/power supply
3. Classification based on motion control methods(Applications)

1) Classification based on work envelope(geometry)

The gross work envelope of a robot is defined as the locus of points in three dimensional spaces that can be reached by the wrist.

Accordingly we get five configurations of the robot based on work envelope geometry or coordinate geometry.

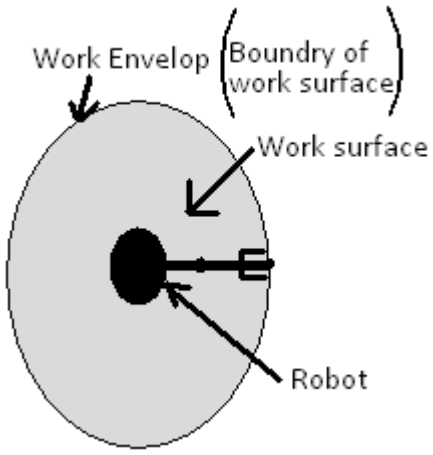
Note: There are two types of joints in this kind of robots.

1) Revolute /Rotary joint "R"

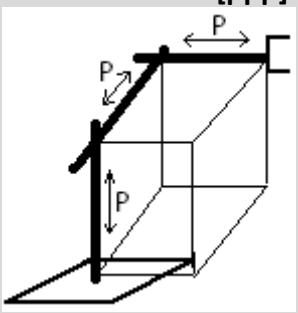
Rotation abo xis

2) Prismatic/sliding joint "P"

Translation along axis

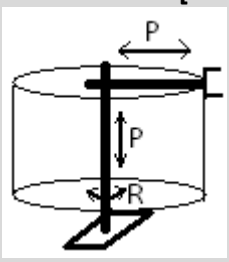


1) Cartesian Robot[ppp]



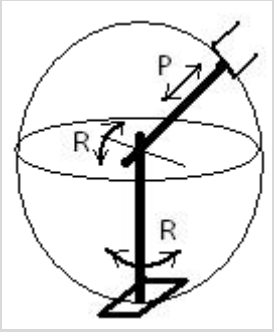
Work Envelope: Rectangle

2) Cylindrical Robot[RPP]



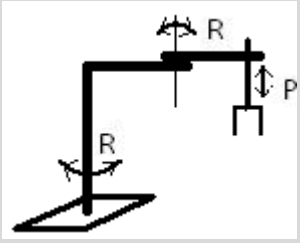
Work Envelope: Cylinder

3) Spherical Robot [RRP]

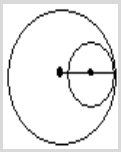


Work Envelope: Sphere

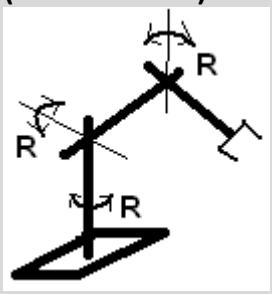
4) SCARA Robot [RRP]



Work Envelope:



5) Articulate Robot [RRR] (Orthromorhic)



Work Envelope:Complex

2) Classification based on drive technology/power supply

Commercially available industrial robots are powered by one of three types of drive systems.

a) Electric drive
b) Hydraulic drive
c) Pneumatic drive

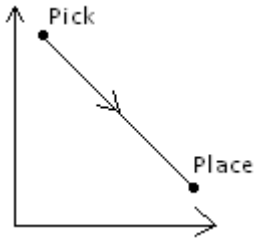
Hydraulic	Electric	Pneumatic
1. Require oil, fluid etc. 2. Good for large robots and heavy payload. 3. Highest power/weight ratio. 4. Stiff system, better response. 5. May leak, not fit for clean room applications. 6. Can be expensive and noisy, requires maintenance.	1. Require voltage. 2. Good for all size of robots. 3. Better control, good for high precision robots. 4. Higher compliance than hydraulic. 5. Does not leak, well for clean room. 6. Reliable, low maintenance.	1. Require air pressure, fillers etc. 2. Many components are usually off-the-shelf. 3. No leaks or sparks. 4. Good for on-off and for pick and place. 5. Noisy system.

3) Classification based on Motion control(Application)

a) Pick-n-Place Robot
b) Point-to-point robot
c) Continuous path

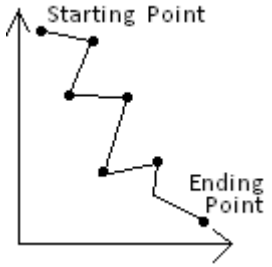
Pick-n-place robots (PNP)

A PNP robot just picks up objects from one place and moves it to another location.
Start and End point coordinates given by user.
E.g. use in machine loading and unloading.



Point-to-Point robots (PTP)

These types of robots allows the user to specify the set of points between which robot is to move.
The path for the movement of the robot between the two specified points or via a number of discrete points is to be specified.
E.g. Spot welding, drilling on automobile in an assembly plant.



Continuous Path robots (Control Path Motion-CP)

This type of robots allows the user to specify the path along which the robot should move.

E.g. Robots use for spray painting, arc welding.

Robot Specifications:

There are number of additional characteristics that allow the user to further specify robotic manipulators. Some of the common characteristics are shown in table below.

Characteristics	Units
Number of Axes	-
Load carrying capacity	Kg
Max-speed	mm/sec
Reach & stroke	mm
Tool orientation	deg
Repeatability,Accuracy,precision	mm
Operating environment	-

Number of axes [or Degree of Freedom (DOF)]:

"The number of independent movements that a robot can perform in a 3-D space is called as degree of freedom (DOF)", E.g. bus has got two DOF since it can move in X-Y.fan has got one degree of freedom.

Axes(DOF)	3-D space	2-D space
1. Major axes (Use for position)	1-3 (Base,Shoulder,Elbow)	1-2(x-y) (Base ,Shoulder)
2. Minor axes (Use for Orientation)	4-6 (Yaw,Pitch,Roll)	3 (Roll)(along Z-axis)
3. Kinematically redundant (Use to avoid obstacles)	7-n	4-n

Load carrying capacity and speed:

Load carrying capacity varies greatly between robots. E.g. The Minimover 5-mocrobot robot has load carrying capacity of 2.2 kg.

The maximum tool-tip speed can also vary substantially between manipulators.

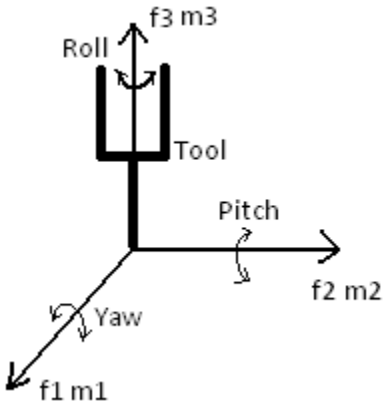
E.g. The Adapt one SCARA robot has a tool-tip speed of 9000 mm/sec.

Tool Orientation:

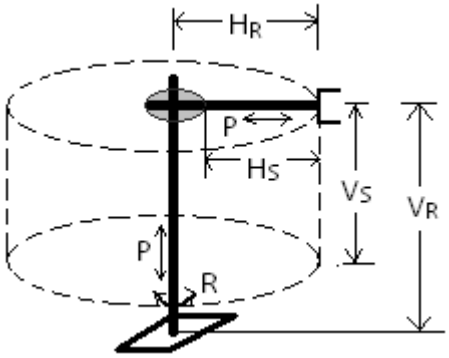
Three major axes determine shape of work envelope while remaining axes determine the kind of orientation that the tool or hand can assume.

The different minor axes are **Y**aw, **P**itch and **R**oll (YPR).

Yaw- Rotation about 1st axis.
Pitch- Rotation about 2nd axis.
Roll- Rotation about 3rd axis.

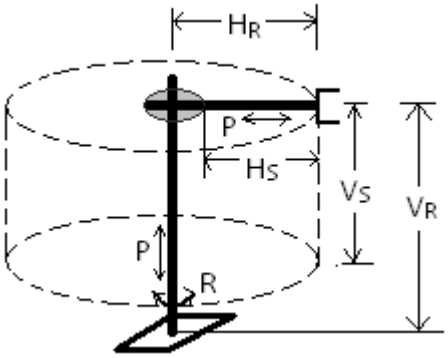


Reach and Stroke:
The *reach* and *stroke* of a robotic manipulator are rough measures of the size of the work envelope.



Horizontal Reach (H_R): Maximum radial extension that the wrist can have from its base axis.
Horizontal Stroke (H_S): Total radial distance that the wrist can travel.
Vertical Reach (V_R): maximum elevation that the wrist can have from its work surface.
Vertical Stroke (V_S): Total vertical distance that the wrist can travel.
(Note: Stroke is a feature, gives rise to a need for programming safeguard, because an articulated robot can be programmed to collide with itself or work surface.)

Q) Suppose a cylindrical coordinate robot has a vertical reach of 480mm and a vertical stroke of 300mm as shown in figure.
How far off the floor do the parts have to be raise in order to be reachable by robot? Comment.



Repeatability, Precision, Accuracy:

Repeatability is defined as the measure of ability of the robot to position a tool tip in the same place repeatedly. E.g. Blind assembly; i.e. performing a picking and placing operation a hundred of times.

Precision is defines as the measure of the spatial resolution with which the tool can be positioned within the work space envelope.

Note: The precision of various types of robots as tabulated in following table.

Robot Type	Horizontal Precision	Vertical Precision
Cartesian	Uniform	Uniform
Cylindrical	Decreases radially	Uniform
Spherical	Decreases radially	Decreases radially
SCARA	Varies	Uniform
Articulated	Varies	Varies

Accuracy is defined as it is the measure of ability of robot to place tool-tip at an arbitrary prescribed location in the work envelope.

Repeatability is always less than accuracy. And smaller is the precision, better is the accuracy.

Operating Environment:

The operating environment for a robot depends on the type of task or application that is to be performed. E.g. Transport of radioactive material, painting zone, drilling etc., clean room robots are often used in semiconductor industry for handling of silicon wafers and photo masks.

Ch2.Direct Kinematics & Inverse Kinematics

2.1 Introduction:

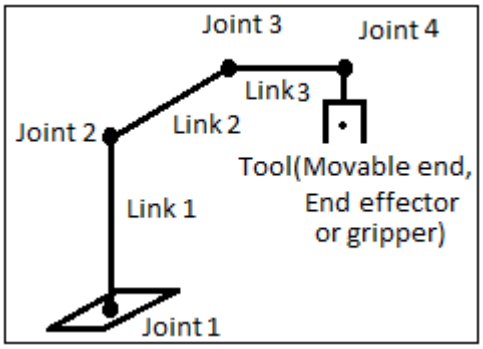
- In this chapter, we deal with the science of motion of objects regardless of the forces that generate the robot motion.
- We then apply our knowledge to the robot manipulators and the objects that are to be manipulated.
- By the definition, this treatment is called as kinematics of robot manipulation which includes the kinematics of robots and their associated objects, tool and their frames of references.

2.2 Definition of robot arm:

The robot arm is mechanical manipulator that can be modeled as an open-loop articulated chain of several rigid bodies called as links, inter connected in series either by revolute joints or prismatic joints and driven by actuators as shown in figure.at one end of chain there is supporting base, while the other end is free to move and is attached to the tool or end- effector which is very useful to manipulate objects or to do particular task.

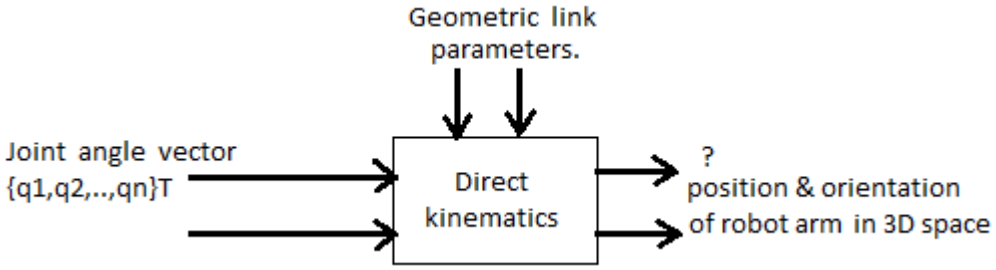
Links: Rigid member which are interconnected between two joints.

Joints: Points or planes around which the two links will rotate w.r.t. each other or translate w.r.t. each other.



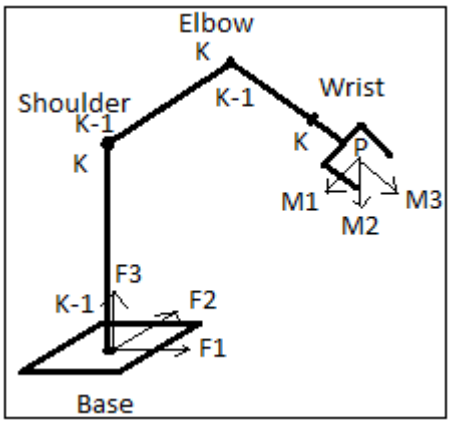
2.3 Direct Kinematics

Definition *" Given the vector of joint variables (joint angle vector q(t)={q1(t),q2(t),….,qn(t)}^{T} and geometric link parameter 'a' and 'd' where n is DOF) of a robotic manipulator, determine the position and orientation of the tool with respect to a coordinate frame attach to the robot base."*

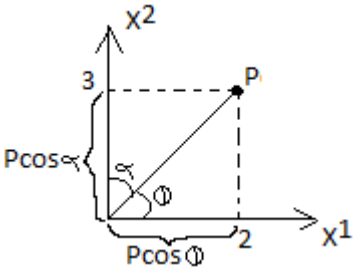


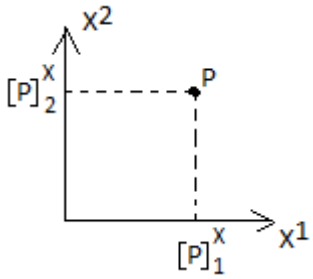
⇒ The solution to the direct kinematics problem requires that the position and orientation of the mobile tool be represented with respect to the coordinate frame attached to the frame base.

⇒ This involves sequence of coordinate transformation from tool to wrist, wrist to elbow, elbow to shoulder and shoulder to base. Refer to figure.



2.4 Notations:





⇒ Consider a single axis robot consisting of only one revolute joint as shown in figure. This single joint connects a fixed link (body).

⇒ A fixed coordinate frame (reference coordinate frame) is given by F= {f1, f2, f3} which is attached to the fixed base.

⇒ A mobile coordinate frame [moving] is given by M= {m1, m2, m3} which attached to the tool or object that move with tool.

$[P]_1^X$: 1st coordinate of point P w.r.t. X

$[P]_2^X$: 2nd coordinate of point P w.r.t. X

Point P: Decomposed on to its orthogonal projections onto the one dimensional sub spaces generated by X^1 and X^2.

⇒ Point P is attached to the body (mobile part).

⇒ There are 2 sets of coordinates of point P.

1) Coordinate p of point P w.r.t. M

$$\mathbf{[P]^M = \{P.F^1,\ P.F^2,\ P.F^3\}^T}$$

2) Coordinates of point P w.r.t. F

$$\mathbf{[P]^F = \{P.M^1,\ P.M^2,\ P.M3\}^T}$$

Dot Products:

$u.v=||u||.||v||.\cos\theta$

$P.X^1=||P||.||X^1||.\cos\theta$

$P.X^2=||P||.||X^2||.\cos\alpha$

2.5 Coordinate Transformation matrix

In order to find the coordinates of point w.r.t. F i.e. $[P]^F$ or coordinates of point P w.r.t. M, i.e. $[P]^M$;

We have following equation,

$[P]^F=A.[P]^M$

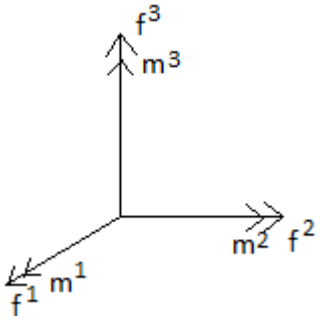
Where **A** is coordinate transformation matrix (CTM) which maps **M** coordinates into **F** coordinates.

$$A=\begin{bmatrix} f^1.m^1.\cos\otimes & f^1.m^2.\cos\otimes & f^1.m^3.\cos\otimes \\ f^2.m^1.\cos\otimes & f^2.m^2.\cos\otimes & f^2.m^3.\cos\otimes \\ f^3.m^1.\cos\otimes & f^3.m^2.\cos\otimes & f^3.m^3.\cos\otimes \end{bmatrix}$$

2.6 Fundamental Rotation matrices

If the mobile coordinate frame M is obtained from fixed coordinate frame F by rotating M about one of the unit vectors of F, then resulting transformation matrix is called as "Fundamental rotation matrix".

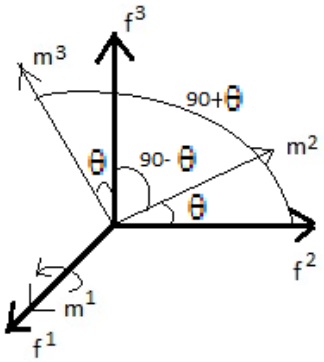
Initially both frames F and M are coincident.



Note: When F and M are coincident $A=I_{3x3}$

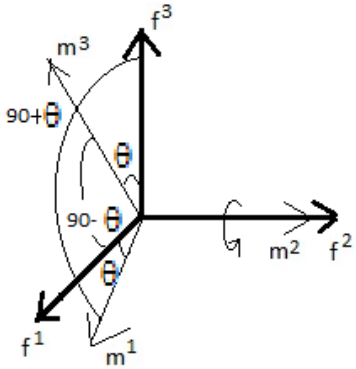
a) Rotation about first axis i.e. f^1 (Yaw)

$$A=R_1(\theta)=\begin{bmatrix} 1 & 0 & 0 \\ 0 & \text{Cos}\theta & -\text{Sin}\theta \\ 0 & \text{Sin}\theta & \text{Cos}\theta \end{bmatrix}$$



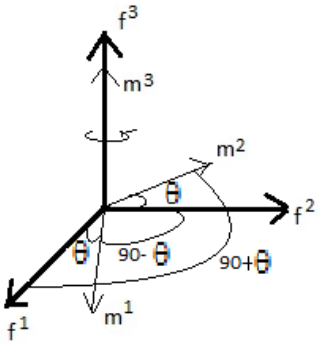
b) Rotation about second axis i.e. f^2 (pitch)

$$A=R_2(\theta)=\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$



c) Rotation about third axis i.e.f_3 (Roll)

$$A=R_3(\theta)=\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Problem based on Fundamental rotations:

1) Consider figure shown, M frame is rotated about f^1 by $\theta=\pi/3$ rads. If P is a point on M frame given by $[P]^M=[2, 0,3]^T$,find the coordinates of P in F frame.

$$[P]^F = A.\,[P]^M$$

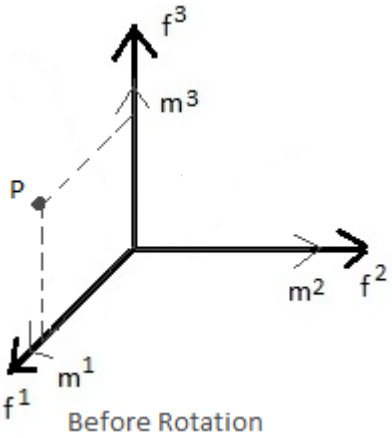
$$= R1(\theta).\,[P]^M$$

$$= R1(\pi/3).\,[P]^M$$

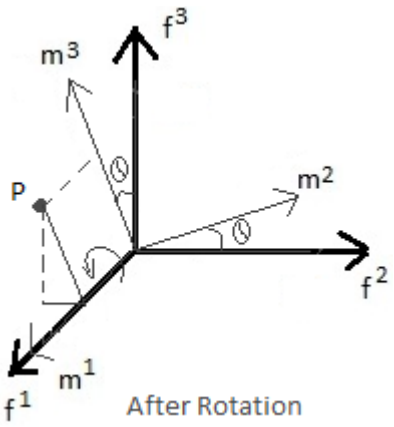
$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\pi/3) & -\sin(\pi/3) \\ 0 & \sin(\pi/3) & \cos(\pi/3) \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0 & -0.866 \\ 0 & 1 & 0 \\ 0.866 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix} = [2 \quad -2.598 \quad 1.5]^T$$



Before Rotation



After Rotation

2) Given that coordinate transformation matrix is a rotation matrix and represents fundamental rotation, what is the axis of rotation (1, 2, or 3) and what is the angle of rotation? i.e.K and θ

$$\mathbf{R_k(\theta)} = \begin{bmatrix} 0.5 & 0 & -0.866 \\ 0 & 1 & 0 \\ 0.866 & 0 & 0.5 \end{bmatrix}$$

Solution: This rotation matrix is similar to the second fundamental rotation matrix given by $R_2(\theta)$.

$$R_2(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \text{Cos}\theta \end{bmatrix}$$

Comparing, we get

$\cos\theta = 0.5$

$\theta = \cos^{-1}(0.5)$

$\theta = 60^0$

$\theta = \pi/3$ rads

Now, $-\sin\theta = \cos(90^0+\theta)$ (by trigonometry formula)

$-\sin\theta = -(-0.866)$ (from given value in matrix)

$= 0.866$

So, $\cos(90^0+\theta) = 0.866$

$(90^0+\theta) = \cos^{-1}(0.866)$

$(90^0+\theta) = 150^0$

$\theta = 150^0 - 90^0$

$\theta = 60^0$

$\theta = \pi/3$ rads

Axis of rotation=f^2

Axis of rotation =$\theta = 60^0$ or $\theta = \pi/3$ rads

3) For the following fundamental matrix, what is the axis of rotation and what is the angle of rotation?

$$\mathbf{R_k(\theta)} = \begin{pmatrix} -0.5 & 0 & 0.866 \\ 0 & 1 & 0 \\ 0.866 & 0 & -0.5 \end{pmatrix}$$

Solution: This rotation matrix is similar to the second fundamental rotation matrix given by $R_2(\theta)$.

$$R_2(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

By comparing, we get

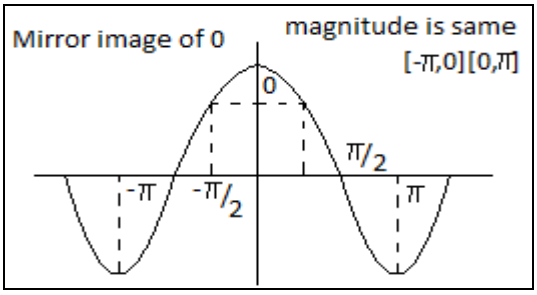
Axis of Rotation=f^2

And, $\cos\theta = -0.5$ $\theta = \cos^{-1}(-0.5) = +120$

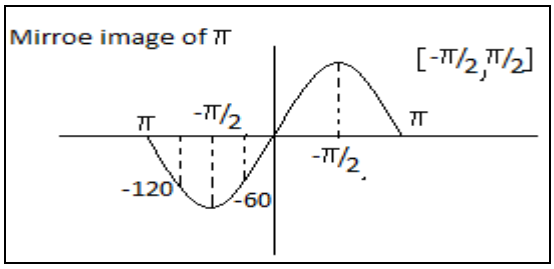
$\sin\theta = -0.866$ $\theta = \sin^{-1}(-o.866) = -60$

Here we are getting two different answers for θ,

Using sine and cosine waveform, we take polarity from sine answer (i.e.-60) and magnitude from cosine answer (i.e. 120).



COSINE Waveform



SINE Waveform

Sin(-60)=sin(-120)= -0.866

$\theta = -120^0$

Composite Rotation Matrix (C.R.M)

⇒ When numbers of fundamental rotation matrices are multiply together, then the product matrix is called a composite rotation matrix.

⇒ Advantage of using C.R.M. is, we can find the arbitrary orientation of the tool in 3 Dimensional Euclidean (3DE) space.

⇒ Each rotation of tool about unit vectors of F is represented by a fundamental rotation matrix given by $R_1(\theta)$, $R_2(\theta)$, $R_3(\theta)$.

⇒ Combination of these fundamental rotation in any order gives rise to a composite rotation matrix (CRM) $R(\theta)$.

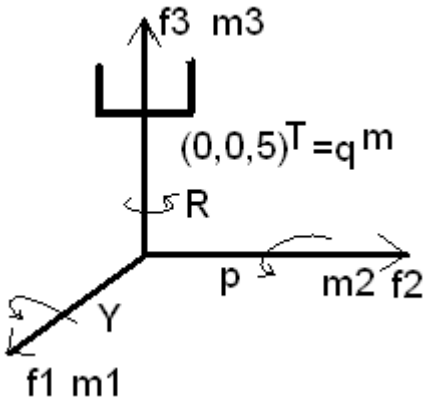
Methods /Algorithm of obtaining composite rotations.

1. CRM are build up in steps starting from (3x3) Identity matrix I which corresponds to no rotation at all, i.e. F and M coordinate frames being coincident.
2. **Case 1:** YPR- M about F
   Pre multiply **$(R_3(\theta).R_2(\theta).R_1(\theta).I_{3x3})$**
   R is basic rotation matrix ←
3. **Case 2:** RPY-M about its own unit vector
   Post multiply **$(I_{3x3} . R_1(\theta).R_2(\theta).R_3(\theta))$**
   R is basic rotation matrix →

Notes: In both the cases; the final orientation of tool is same; but, the route that is taken by the tool to reach the point P is different. i.e. YPR (θ) =RPY (θ)

Problem based on CRM

Q.1 Consider the robotic tool shown figure. Sketch the tool position after each intermediate position of the following YPR operation.
Yaw 90^0, Pitch -90^0, roll 90^0. Rotations are performed about the fixed axes of F frame. Find coordinates of q w.r.t. fixed axis. i.e. $[q]^F$.



Solution: Given yaw=$\pi/2$, pitch= $-\pi/2$, roll=$\pi/2$

$[q]^M= [0, 0, 5]^T$

$[q]^F=A. [q]^M$

$A(YPR)= R_3(\theta).R_2(\theta).R_1(\theta).I_{3x3}$

$A(YPR)= R_1(\pi/2).R_2(-\pi/2).R_3(\pi/2).I_{3x3}$

$$= \begin{pmatrix} \text{Cos}(\pi/2) & -\text{Sin}(-\pi/2) & 0 \\ \text{Sin}(\pi/2) & \text{Cos}(\pi/2) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \text{Cos}(-\pi/2) & 0 & \text{Sin}(-\pi/2) \\ 0 & 1 & 0 \\ -\text{Sin}(-\pi/2) & 0 & \text{Cos}(-\pi/2) \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \text{Cos}(\pi/2) & -\text{Sin}(\pi/2) \\ 0 & \text{Sin}(\pi/2) & \text{Cos}(\pi/2) \end{pmatrix}$$

$$= \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

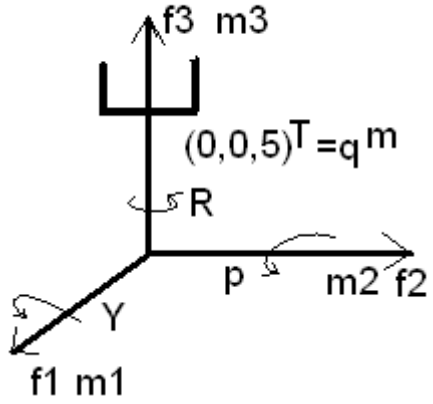
$$[q]^F = A.\,[q]^M$$

$$[q]^F = \begin{pmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 5 \end{pmatrix}$$

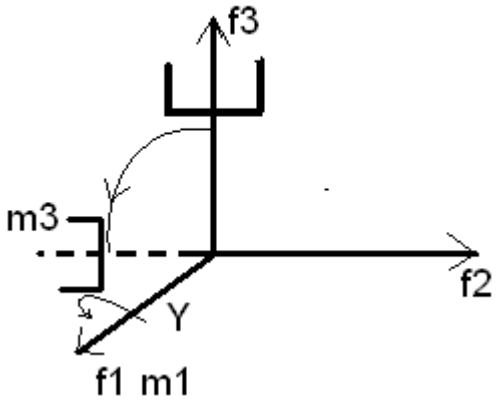
$$[q]^F = \begin{pmatrix} 5 \\ 0 \\ 0 \end{pmatrix} = [5, 0, 0]^T$$

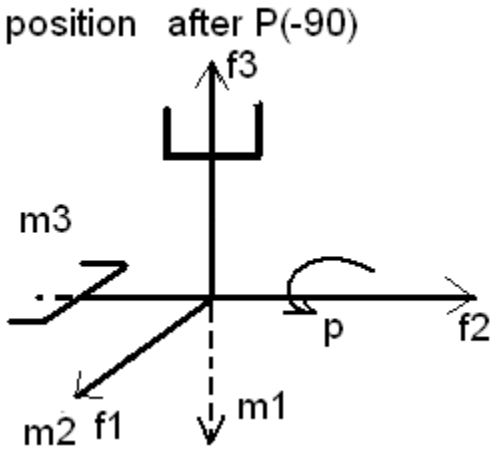
Sketch and verify

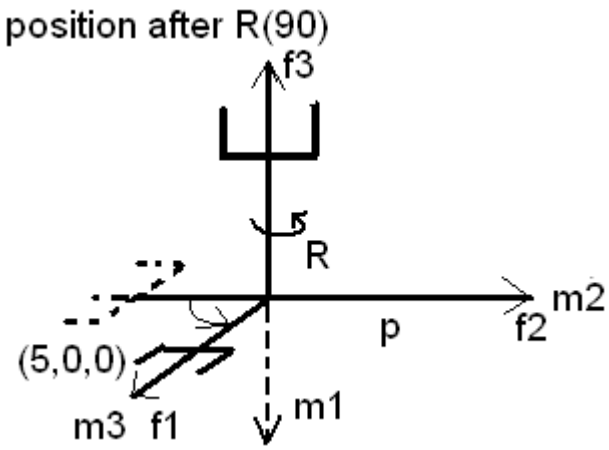
Initially



Position after Y(90)







Q.2 Yaw 90^0, Pitch -90^0, roll 90^0. Rotations are performed about the fixed axes of F frame. Find coordinates of q w.r.t. fixed axis. i.e. $[p]^F$.,[P]M={0,0,0.8}T.

Solution: Given yaw=π, pitch= $-\pi/2$, roll=$\pi/2$

$[p]^M = [0,0,0.8]^T$

$[p]^F = A.\ [p]^M$

$A(YPR) = R_3(\theta).R_2(\theta).R_1(\theta).I_{3x3}$

$A(YPR) = R_3(\pi/2).R_2(-\pi/2).R_1(\pi).I_{3x3}$

$$= \begin{pmatrix} \text{Cos}(\pi/2) & -\text{Sin}(-\pi/2) & 0 \\ \text{Sin}(\pi/2) & \text{Cos}(\pi/2) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \text{Cos}(-\pi/2) & 0 & \text{Sin}(-\pi/2) \\ 0 & 1 & 0 \\ -\text{Sin}(-\pi/2) & 0 & \text{Cos}(-\pi/2) \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \text{Cos}(\pi) & -\text{Sin}(\pi) \\ 0 & \text{Sin}(\pi) & \text{Cos}(\pi) \end{pmatrix}$$

$$= \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

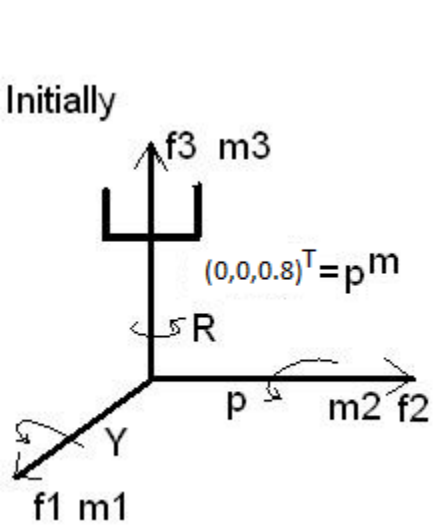
$$= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

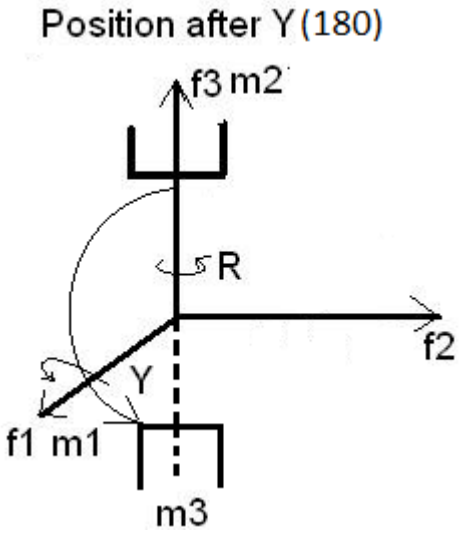
$$[p]^F = A.\,[p]^M$$

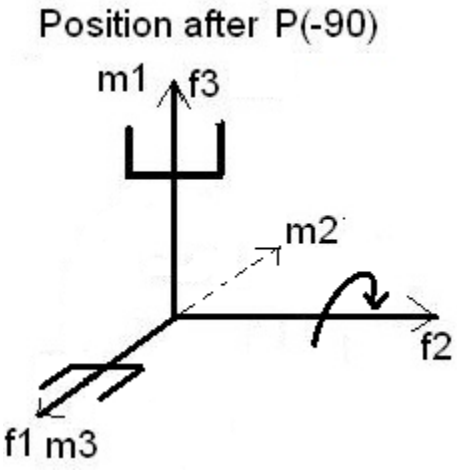
$$[p]^F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0.8 \end{bmatrix}$$

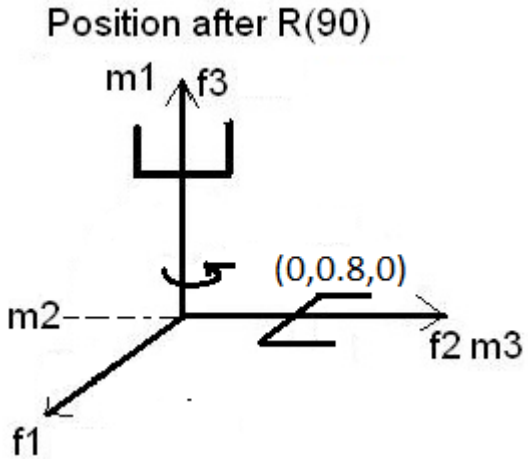
$$[p]^F = \begin{bmatrix} 0 \\ 0.8 \\ 0 \end{bmatrix} = [0,\ 0.8,\ 0]^T$$

Sketch and verify









Homogeneous Coordinate Transformation (HCT)

	Physical coordinates	Homogeneous coordinates
1	Three coordinates in R3	Four coordinates in R4
2	Point p in R3 w.r.t. F P={P1,P2,P3}T	Point p in R4 w.r.t. F P={P1,P2,P3,1}T Where 1 is scalar vector
3	3D space/system	4D space/system
4	Positional vector	Augmented vector

Definition of Homogeneous coordinates

- Representation of an 'n' component positional vector by a (n+1) component vector is called as Homogeneous coordinates representation (4 coordinates).
- A homogeneous coordinate is one which gives information about position as well as orientation.

Uses of homogeneous coordinate representation

- Useful in developing matrix transformation that include rotation, translation, scaling and perspective transformation.
- Gives position and orientation of robot arm in R4 space.
- To find the arm matrix.

Homogeneous Coordinates Transformation Matrix (HCTM)

A HCTM is a (4x4) homogeneous coordinates transformation matrix which maps or transform a positional vector expressed in homogeneous coordinate from one coordinates frames to another.

HCTM is denoted by H or T.

A HCTM consists of four sub-matrices given by

1. 3x3 Rotation matrix R(θ)
2. 3x1 Transformation matrix P
3. 1x3 perspective vector ηT
4. 1x1 Scaling matrix σ

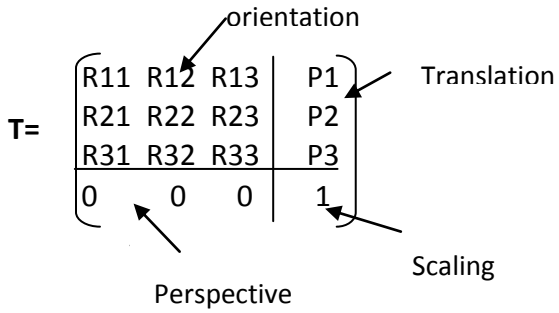
$$\mathbf{T=}\left[\begin{array}{c|c} \text{Rotation matrix} & \text{Positional vector} \\ \hline \text{Perspective transformation} & \text{Scaling factor} \end{array}\right] = \left[\begin{array}{c|c} R_{3x3} & P_{3x3} \\ \hline \eta T_{1x3} & \sigma_{1x1} \end{array}\right]$$

R=> (3x3) sub matrix of T which represents rotation matrix (orientation of M w.r.t. F).

P=>(3x1) column vector P of T which represent position of origin of M w.r.t. F.

ηT=>(1x3) sub matrix ηT of T which gives perspective vector ,specifies point for visual sensing (using a camera).

σ => (1x1) global scaling factor=1 (standard scaling is used).



Pure Rotation

$$\begin{bmatrix} R & \begin{matrix}0\\0\\0\end{matrix} \\ \hline 0\ 0\ 0 & 1 \end{bmatrix}$$

Pure Translation

$$\begin{bmatrix} 1 & 0 & 0 & P1 \\ 0 & 1 & 0 & P2 \\ 0 & 0 & 1 & P3 \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

No Translation & Rotation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

Problems based on HCTM

Q.1 Given coordinates of a q w.r.t M, i.e. $q^M=\{0,0,10,1\}^T$. Find qF ,i.e. coordinates of q w.r.t. F. Initially M and F are coincident M is rotated by 45 ($\pi/4$ rads) about 1st axis of F, i.e.f1 .find the resulting HCTM. Sketch the diagram before and after rotation and verify.

Solution: As per given data rotation is about f1 and there is no translation.so it is example of pure rotation.

$$\text{HCTM}= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

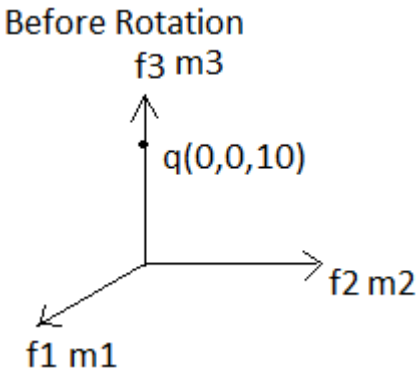
$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.707 & -0.707 & 0 \\ 0 & 0.707 & 0.707 & 0 \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

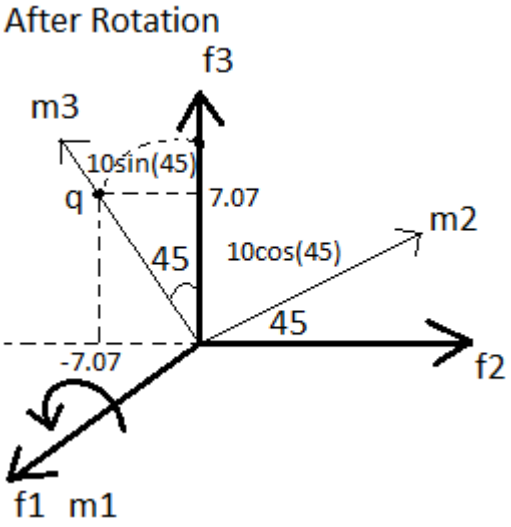
$$[q]^F = A_{(HCTM)}.[q]^M$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.707 & -0.707 & 0 \\ 0 & 0.707 & 0.707 & 0 \\ \hline 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 10 \\ 1 \end{bmatrix}$$

$$[q]^F = \begin{pmatrix} 0 \\ 0 \\ 10 \\ 1 \end{pmatrix} = [0, -7.07, 7.07, 1]^T$$

Sketch and verify.





Q.2 Given [q]M= {0, 0, 10,1}T, Translate M w.r.t. F by 5 units along axis f1,and -3 units along f2 axis. Find the resulting homogeneous translation matrix and also [q]F.

Solution: Given $[q]^M=[0,0,10,1]^T$, so it pure translation.

$$HCTM_{(Translation)} = \begin{pmatrix} 1 & 0 & 0 & P1 \\ 0 & 1 & 0 & P2 \\ 0 & 0 & 1 & P3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Q.3 If F and M are two orthonormal coordinate frames, translate M along axis f2 by 3 units, rotate M about f3 axis by π rads and hence find $[m1]^F$ after composite transformation. $[m1]^M$= {1,0,0,1}T.

Solution:

Screw Transformation Matrix (STM)

- It is defined as the linear displacement of a frame along an axis followed by the rotation (angular displacement) of the same frame about the same axis (or) vice-versa.
  Let F= {f1,f2,f3} and M={m1,m2,m3} are two orthonormal coordinate frames initially coincident, i.e. T=I_{4x4}.
- Translate M along K^{th} unit vector of F by an amountλ.
  $\Rightarrow$ Given by homogeneous translation matrix Trans(λ,f^k)
- Rotate M about the same K^{th} unit vector of F by ϕ.
  $\Rightarrow$ Given by homogeneous rotation matrix Rot(ϕ,f^k).
- Screw transformation matrix is given by,

Screw(λ,ϕ,f^k)=Rot(ϕ,f^k). Trans(λ,f^k)= Trans(λ,f^k). Rot(ϕ,f^k)

- Special cases

1) If no translation occurs, λ=0, screw transformation matrix consist of pure rotation.
   STM is given by Screw(λ,ϕ,f^k) = Screw(0,ϕ,f^k)= Rot(ϕ,f^k)
2) If no rotation occurs, ϕ=0, screw transformation matrix consist of pure translation.
   STM is given by Screw(λ,ϕ,f^k) = Screw(λ,0,f^k)= Trans(λ,f^k)

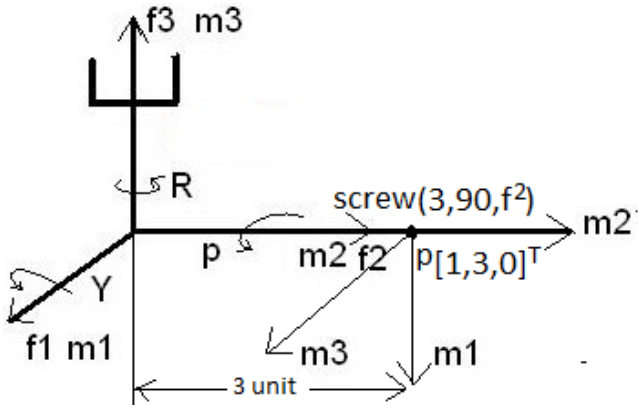
- Screw pitch, ψ= $\phi/(2\pi\lambda)$ threads/unit length

1) For pure rotation, λ=0, the pitch is ∞
2) For pure translation, ϕ=0,pitch is 0.

Note: Rotation and translation are performed on same axis, hence matrix multiplication is commutative.
i.e. AB=BA

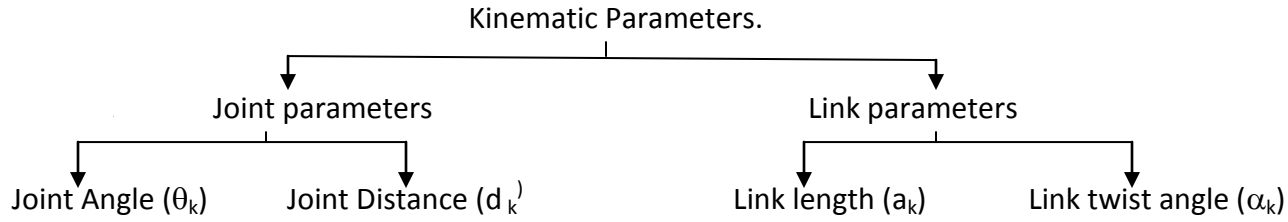
<u>Problem based on screw transformation:</u>

Q.1 If F and M are two frames coincident initially, after performing a screw transformation along f2 axes of F by a distance of λ=3 and rotating by an angle of 90^0 about f2 axes, find $[m3]^F$ after the screw transformation. Also find pitch of the screw. $[m3]^M$= [0, 0, 1, 1]^T.



Parameters of any Robot arm

Kinematic Parameters-parameter associated with the kinematic configuration of each link and joint of the robot arm.



θ_k=angle of K^{th} Joint

d_k =Distance between joints

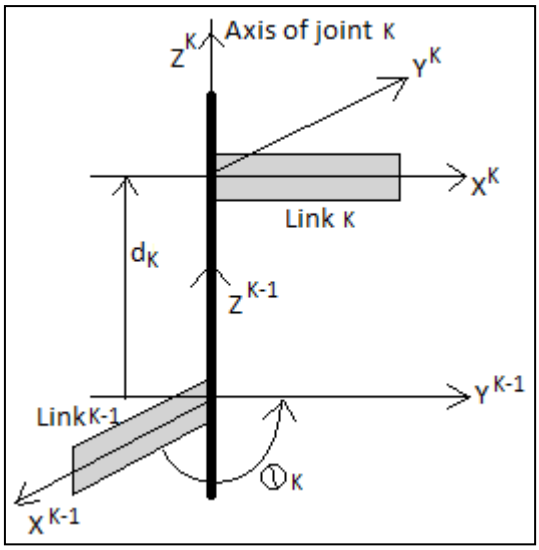
a_k = Length of link K

α_k =Link twist angle

Joint parameters

⇒ Amount of rotation about z^{k-1} to make x^{k-1} parallel to x^k is called as **JOINT ANGLE θk**

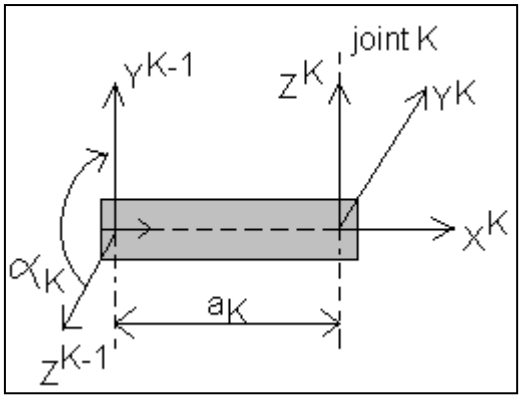
⇒ Amount of Translation along z^{k-1} to make x^{k-1} intersect with xk is called as **JOINT DISTANCE dk**



Link Parameter

⇒ Amount of rotation about x^k to make z^{k-1} parallel to z^k is called as **LINK TWIST ANGLE α^k**

⇒ Amount of translation about x^k to make z^{k-1} intercept with z^k is called as **LINK LENGTH a^K**



Q. Which kinematic parameters are variable for a revolute joint? And which are variable for a prismatic joint?

Kinematic Parameters	Symbol	Types of Joints	
		Revolute	Prismatic
Joint angle	θ^k	Variable	Fixed
Joint Distance	d^k	Fixed	Variable
Link Length	a^K	Fixed	Fixed
Link twist angle	α^k	Fixed	Fixed

According to screw transformation, we get four sub matrixes.

1) **Rotation about z^{k-1} => θ^k**
2) **Transformation along z^{k-1} => d^k**
3) **Rotation about x^k => α^k**
4) **Transformation along x^k => a^K**

Then, successive coordinates transformation T^k_{k-1} **is derive as,**

$$T^{mobile}_{fixed} = T^k_{k-1} = Rot(\theta^k, z^{k-1}).Trans(d^k, z^{k-1}).Rot(\alpha^k, x^k).Trans(a^K, x^k)$$

$$= screw(\theta^k, d^k, z^{k-1}).screw(\alpha^k, a^K, x^k)$$

$$= \left(\begin{array}{cccc} \cos\theta^k & -\sin\theta^k & 0 & 0 \\ \sin\theta^k & \cos\theta^k & 0 & 0 \\ 0 & 0 & 1 & dk \\ \hline 0 & 0 & 0 & 1 \end{array}\right)\left(\begin{array}{cccc} 1 & 0 & 0 & ak \\ 0 & \cos\alpha^k & -\sin\alpha^k & 0 \\ 0 & \sin\alpha^k & \cos\alpha^k & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

$$= \left(\begin{array}{cccc} C\theta^k & -S\theta^k C\alpha^k & S\theta^k S\alpha^k & ak\ C\theta^k \\ S\theta^k & C\theta^k C\alpha^k & -C\theta^k S\alpha^k & ak\ S\theta^k \\ 0 & S\alpha^k & C\alpha^k & dk \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

<u>Denavit-Hertenberg (D-H) Representation</u>

1. It is a systematic representation of assigning the right handed orthonormal coordinate frames to the various links, joints, tool-tip, objects, parts, camera etc.; of the robot.
2. L0={X0,Y0,Z0}
   Lk = { Xk, Yk, Zk}
   Ln = {Xn, Yn, Zn}

Note: Steps (0-8) =>Pass-I Steps (9-13)=>Pass-II

D-H notations are as following,

Step 0: Number the joints from 1 to n,

***Step* 1:** Name the joints from 1 to n,

***Step* 2:** Assign L0 to the base, Align Z^k (Zk is axis of rotation)with the axis of joint k+ 1,

***Step* 3:** Since Z^k and Z^{k-l} do not intersect locate Lk at the intersection of Zk with a common normal between Zk and Zk-l.

***Step* 4:** Mark X^k which normal to Z^k Since Z^k and Z^{k-l} are parallel, point X^k away from Z^{k-1}.

***Step* 5:** Add y^k to form a right-handed coordinate system.

***Step* 6:** Repeat above steps for all joints k.

***Step* 7:** Set Origin of Ln is at the tool tip. Align Zn with the approach vector, yn with the sliding vector,and Xn with normal vector.

Step 8: Locate point bk at the intersection of Xk and Z^{k-l} axes. If they donot intersect, use the intersection of Xk with a common normal between Xk and Z^{k-l} .

Step 9: Compute θ^k

Step 10: Compute dk

Step 11: Compute ak

Step 12: Compute α^k

Step 13: Set k=k+1 .if K<=n,go to step 8 ;else stop

Tool/Hand coordinate system

Approach Vector (Z^n)

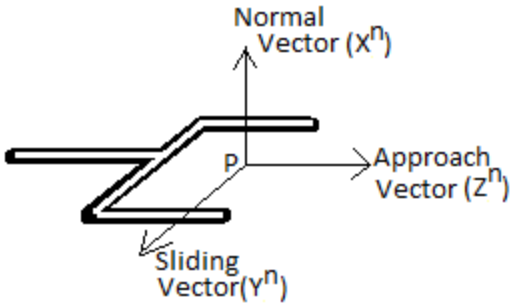
Direction in which tool is pointing or approaching is called as Approach Vector.

Sliding Vector (Y^n)

Direction of opening and closing of tool fingers gives sliding vector.

Normal Vector (X^n)

Vector normal to approach and sliding vector.

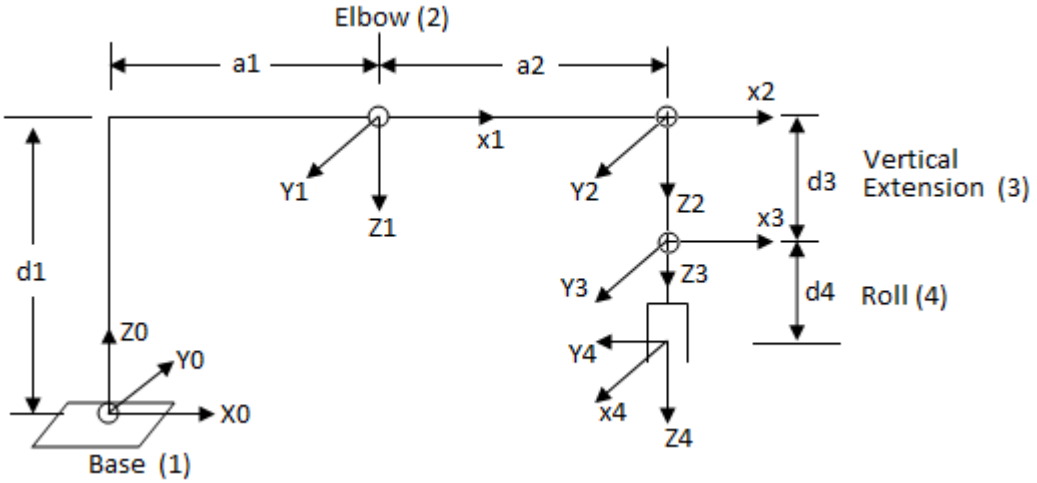


Direct Kinematics of 4-Axis SCARA Robot

Step-1 Specifications

- 4 axis SCARA
- 4=3+1=RRP+Roll
- Used in PCB assembly,
- All axis of rotations are vertical (Z-axis), Minimal configuration industry Robot.

Step-2 Link coordinate Diagram (LCD) [Pass-I of D-H algorithm]



Step-3 Kinematic Parameter Table(KPT) [Pass-II of D-H algorithm]

	Axis	θk	dk	ak	αk	Home
K=1	Base	θ1=q1	d1	a1	π	0
K=2	Elbow	θ2=q2	0	a2	0	0
K=3	Vertical Extension	0	d3 =q3	0	0	d3=100mm
K=4	Roll	θ4=q4	d4	0	0	π/2

Step-4 Arm Equation

$$T_{Base}^{Tool} = T_0^4 = T_0^1\ T_1^2\ T_2^3\ T_3^4$$

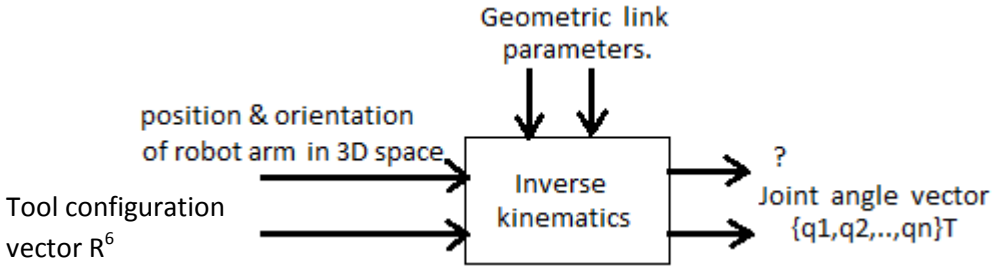
$$= \begin{bmatrix} C1 & S1 & 0 & a1c1 \\ S1 & -c1 & 0 & a1s1 \\ 0 & 0 & -1 & d1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C2 & -S2 & 0 & a2c1 \\ S2 & C2 & 0 & a2s1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C4 & -S1 & 0 & 0 \\ S4 & C4 & 0 & 0 \\ 0 & 0 & 1 & d4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_0^4 = \left[\begin{array}{ccc|c} C_{1-2-4} & S_{1-2-4} & 0 & a1c1+a2c_{1-2} \\ S_{1-2-4} & -c_{1-2-4} & 0 & a1s1+a2s_{1-2} \\ 0 & 0 & -1 & d1-d3-d4 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Note: C_{1-2-4}= cos (θ1-θ2-θ4)

Inverse Kinematics

Definition: Given a desired position P and orientation R for the tool, find values for the joints variable which satisfy the arm equation.



Comparison between Direct Kinematics and Inverse Kinematics

No.	Direct Kinematics	Inverse Kinematics
1	Given the vector of joint variables (joint angle vector $q(t)=\{q1(t),q2(t),....,qn(t)\}^T$ and geometric link parameter 'a' and 'd' where n is DOF) of a robotic manipulator, determine the position and orientation of the tool with respect to a coordinate frame attach to the robot base.	Given a desired position P and orientation R for the tool, find values for the joints variable which satisfy the arm equation.
2	Uniqueness of solution is always guaranteed	Uniqueness of solution is not guaranteed
3	Existence of solution is always guaranteed	Existence of solution is conditional
4	D.K. is used to solve I.K. problems	I.K. is used in practical manipulation.

Tool Configuration Vector (TCV)

- In order to develop a solution to the inverse kinematics problem, the desired tool configuration must be specified as input data.
- Consider the tool orientation information provided by the approach vector or last column of R. (*Note: Why select r3 in formation of TCV from arm equation?*)

$$\text{Arm Equation} = T_{\text{Base}}^{\text{Tool}} = \begin{bmatrix} R11 & R12 & R13 & P1 \\ R21 & R22 & R23 & P2 \\ R31 & R32 & R33 & P3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

r1 r2 r3 — orientation; P1 P2 P3 — position

- The approach vector effectively specifies both the tool Yaw angle and the tool pitch angle, but not the roll angle, because the roll angle represents a rotation about the approach vector.
- The approach vector r3 is a unit vector specifying a direction only.

Definition:

Let P and R denote the position and orientation of the tool frame relative to the base frame where qn represents the tool roll angle. Then the tool configuration vector is a vector W in R6 defined as:

$$W \triangleq \begin{bmatrix} W1 \\ \hline W2 \end{bmatrix} \triangleq \begin{bmatrix} P \\ \hline \left[\text{Exp}\left(\frac{qn}{\pi}\right)\right] \cdot r3 \end{bmatrix}$$

Since the tool configuration vector has only six components, it is a minimal representation of tool configuration in general case. The first three components W1=P, represents the tool-tip position,

While the last three components, $W2 = \left[\text{Exp}\left(\frac{qn}{\pi}\right)\right] \cdot r3$, represents the tool orientation.

The tool roll angle is given as, $qn = \pi \text{ Ln } [w4^2+w5^2+w6^2]^{1/2}$

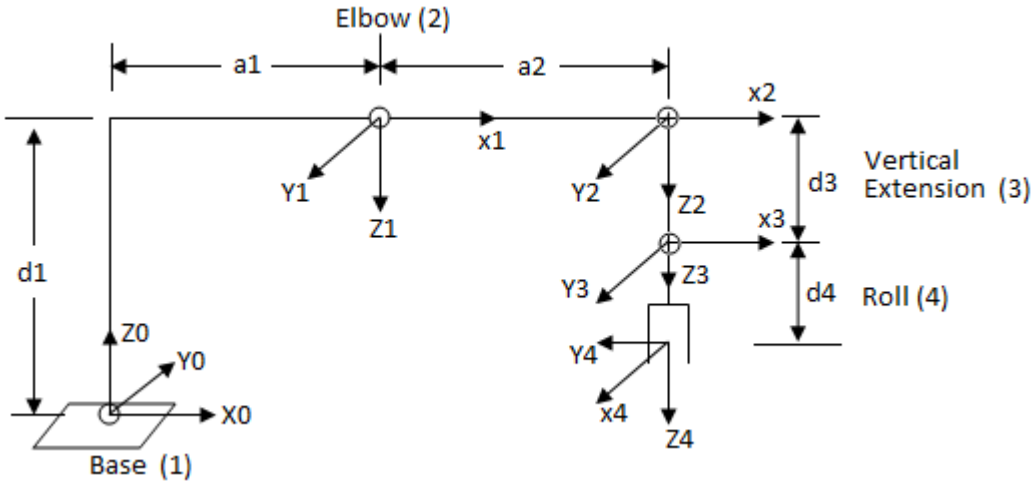
Note: We used exponential because,

1. It is invertible by Ln by natural log.
2. It always returns positive answer for any roll angle.

$$TCV = \begin{bmatrix} W1 \\ W2 \\ W3 \\ W4 \\ W5 \\ W6 \end{bmatrix} \begin{matrix} \} P1 \\ \} P2 \text{ , position} \\ \} P3 \\ \\ \} r3.\exp(qn/\pi) \text{ , orientation} \\ \\ \end{matrix}$$

Inverse kinematics of SCARA Robot

Step-1 Link Coordinate Diagram (LCD)



Step-2 Arm Equation

$$T_0^4 = \left(\begin{array}{ccc|c} C_{1-2-4} & S_{1-2-4} & 0 & a1c1+a2c_{1-2} \\ S_{1-2-4} & -c_{1-2-4} & 0 & a1s1+a2s_{1-2} \\ 0 & 0 & -1 & d1-d3-d4 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

To find,

$q = [q1, q2, q3, q4]^T$

Where, q1=Base angle ,q2=Elbow angle , q3=Vertical extension ,q4= Roll angle

TCV of SCARA robot= W=

$$\left(\begin{array}{c} a1C1+a2C_{1-2} \\ a1S1+a2S_{1-2} \\ \underline{d1-d3-d4} \\ 0 \\ 0 \\ -\exp(q4/\pi) \end{array}\right) \begin{matrix} W1 \\ W2 \\ W3 \\ W4 \\ W5 \\ W6 \end{matrix}$$

a) To extract roll angle (q4).

$\exp(qn/\pi) = [w4^2+w5^2+w6^2]^{1/2}$
By taking natural log,
$qn/\pi = \text{Ln}\ [w4^2+w5^2+w6^2]^{1/2}$
$qn = \pi.\ \text{Ln}[w4^2+w5^2+w6^2]^{1/2}$

$q4=\pi.\ \text{Ln}||w6||$

b) To extract vertical extension(q3)

W3=d1-d3-d4 , from TCV

q3=d3=d1-d4-w3

c) To extract elbow angle(q2)

Using, $W1= a1C1+a2C_{1-2}$
$W2= a1S1+a2S_{1-2}$
By squaring and adding equations, we get
$W1^2+W2^2= a1^2C1^2+a2^2C^21-2+2a1a2C1.C_{1-2}$
$+ a1^2S1^2+a2^2S^21-2+2a1a2S1.S_{1-2}$
$\{C2= C1.C_{1-2}+ S1.S_{1-2}\}$, by trigonometric formula
$W1^2+W2^2= a1^2+ a2^2+2\ a1a2C2$
$C2= \dfrac{W1^2+W2^2- a1^2- a2^2}{2\ a1a2}$

$q2= \pm \underbrace{\text{arc cos}}_{\text{Cos}^{-1}} \left[\dfrac{W1^2+W2^2- a1^2- a2^2}{2\ a1a2}\right]$ { We get two solution for elbow angle as + and -.This is because inverse kinematic isn't unique. }

d) To extract Base angle(q1)
Using, $W1= a1C1+a2C_{1-2}$
$W2= a1S1+a2S_{1-2}$
Now, $W1= a1C1+a2[C1C2+S1S2]$
$W2= a1S1+a2[S1C2-S2C1]$
$W1= (a2S2)S1+(a1+a2C2)C1$
$W2= (a1+a2C2)S1-(a2S2)C1$
Above simultaneous equations can be solved by Cramer's rule to get unknown S1 and C1.

$S1= \dfrac{(a2S2)W1+(a1+a2C2)W2}{(a2S2)^2+(a1+a2C2)^2}$

$C1= \dfrac{(a1+a2C2)W1-(a2S2)W2}{(a2S2)^2+(a1+a2C2)^2}$

$$\tan q1 = \left[\frac{(a2S2)W1+(a1+a2C2)W2}{(a1+a2C2)W1-(a2S2)W2} \right]$$

$$q1 = \underbrace{\arctan}_{\tan^{-1}} \left[\frac{(a2S2)W1+(a1+a2C2)W2}{(a1+a2C2)W1-(a2S2)W2} \right]$$

Questions based on D.K and I.K.

Q1.Compute the joint variable vector q={q1,q2,q3,q4}T for the following tool configuration vector of the SCARA robot where,
W(q)={203.4, 662.7, 557.0, 0, 0, -1.649}T
d= {877, 0, 0, 200}T
a= {425, 375, 0, 0}T
Solution:

a) To extract Roll angle(q4)

q4= π.Ln||W6||
= π.Ln(1.649)
q4=π/2 rads

b) To extract vertical extension (q3)

q3=d1-d4-W3
=877-200-557
q3 =120mm

c) To extract elbow angle (q2)

$$q2 = + \arccos \left[\frac{W1^2+W2^2- a1^2- a2^2}{2\ a1a2} \right]$$

q2 = +/- 60 rads

since IK is not unique,we get two solutions for elbow angle q2=+/- 60.

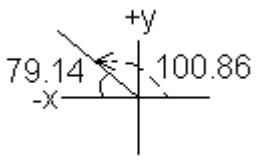
So we select q2=+ 60 rads, to find next angle.

d) To extract Base angle(q1)

$$q1 = \arctan \left[\frac{(a2S2)W1+(a1+a2C2)W2}{(a1+a2C2)W1-(a2S2)W2} \right]$$

$$= \arctan \left[\frac{471959.7}{-90513.138} \right]$$

=79.14
q1=100.86 rads



Q.2 Find the position of tool tip P of SCARA robot when the kinematic parameter given are,

q={π/4, - π /3,120mm, π /2}T

d={877,0,0,200}T

a={425,375,0,0}T

solution:

Q.3 Determine tool configuration vector of SCARA robot,
where, q= { π/6, π/3,150mm, π/2}T
d={877,0,0,200}T
a={425,375,0,0}T
Solution:

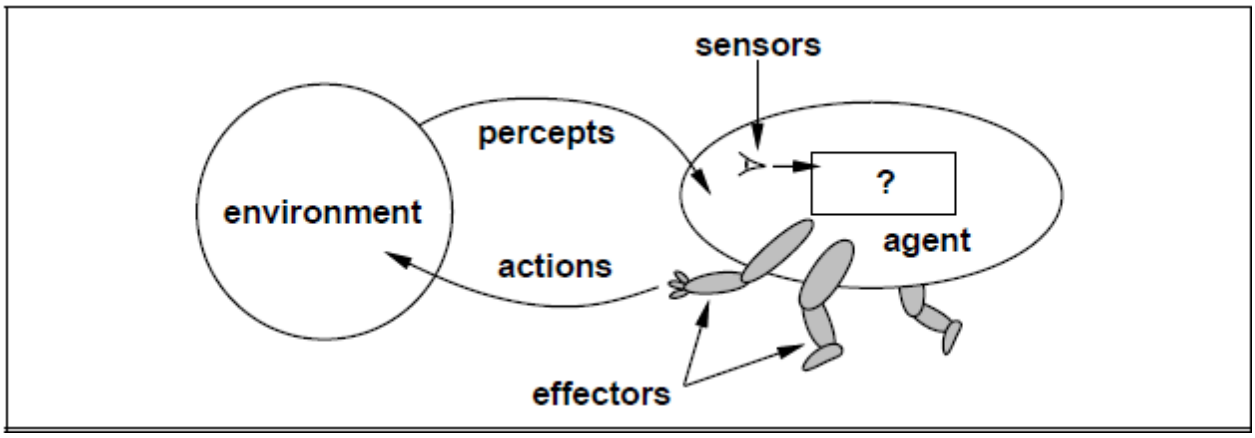
Ch.3. Intelligent Agent

Artificial Intelligent and Agent

The branch of computer science concerned with making computers behave like humans.

"Artificial Intelligence is the study of human intelligence such that it can be replicated artificially."

- **An agent is** anything that can be viewed as **perceiving** its environment through **sensors and acting** upon that environment through **effectors.**
- A human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors.
- A robotic agent substitutes cameras and infrared range finders for the sensors and various motors for the effectors.
- A software agent has encoded bit strings as its percepts and actions.



Concept of Rational Agent

- A **rational agent** is one that does the right thing. As a first approximation, we will say that the right action is the one that will cause the agent to be most successful.
- That leaves us with the problem of deciding *how* and *when* to evaluate the agent's success.
- We use the term **performance measure** for the *how*—the criteria that determine how successful an agent is.
- In summary, what is rational at any given time depends on four things:
  - ✓ The performance measure that defines degree of success.
  - ✓ Everything that the agent has perceived so far. We will call this complete perceptual history the **percept sequence.**
  - ✓ What the agent knows about the environment.
  - ✓ The actions that the agent can perform.
- This leads to a definition of an **ideal rational agent:** *For each possible percept sequence, an ideal rational agent should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*

PEAS: To design a rational agent, we must specify the task environment

Consider, e.g., the task of designing an automated taxi:

- **Performance measure?**
  The performance given by taxi should make it most successful agent that is flawless performance.
  E.g. Safety, destination, profits, legality, comfort . . .
- **Environment?**
  It is a first step in designing an agent. We should specify the environment which suitable for agent action. If swimming is the task for an agent then environment must be water not air.
  E.g. Streets/freeways, traffic, pedestrians, weather . . .
- **Actuators?**
  These are one of the important details of agent through which agent performs actions in related and specified environment.
  E.g. Steering, accelerator, brake, horn, speaker/display, . . .
- **Sensors?**
  It is the way to receive different attributes from environment.
  E.g. Cameras, accelerometers, gauges, engine sensors, keyboard, GPS . . .

Structure of Intelligent agents

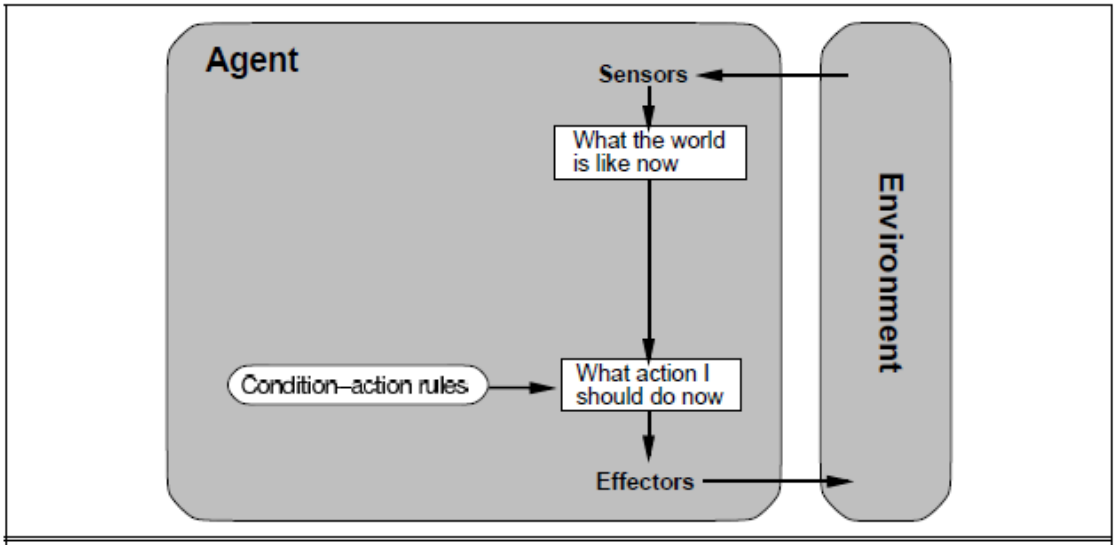
- The job of AI is to design the **agent program:** a function that implements the agent mapping from percepts to actions.
- We assume this program will run on some sort of computing device, which we will call the **architecture.** Obviously, the program we choose has to be one that the architecture will accept and run.
- The architecture might be a plain computer, or it might include special-purpose hardware for certain tasks, such as processing camera images or filtering audio input. It might also include software that provides a degree of insulation between the raw computer and the agent program, so that we can program at a higher level.
- In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the effectors as they are generated.
- The relationship among agents, architectures, and programs can be summed up as follows:

  agent = architecture + program
- **Software agents** (or software robots or **softbot)** exist in rich, unlimited domains. Imagine a softbot designed to fly a flight simulator for a 747.
- The simulator is a very detailed, complex environment, and the software agent must choose from a wide variety of actions in real time.
- Now we have to decide how to build a real program to implement the mapping from percepts to action.
- We will find that different aspects of driving suggest different types of agent program.

Intelligent agents categories into five classes based on their degree of perceived intelligence and capability

1. simple reflex agents
2. model-based reflex agents
3. goal-based agents
4. utility-based agents
5. learning agents

Simple reflex agents

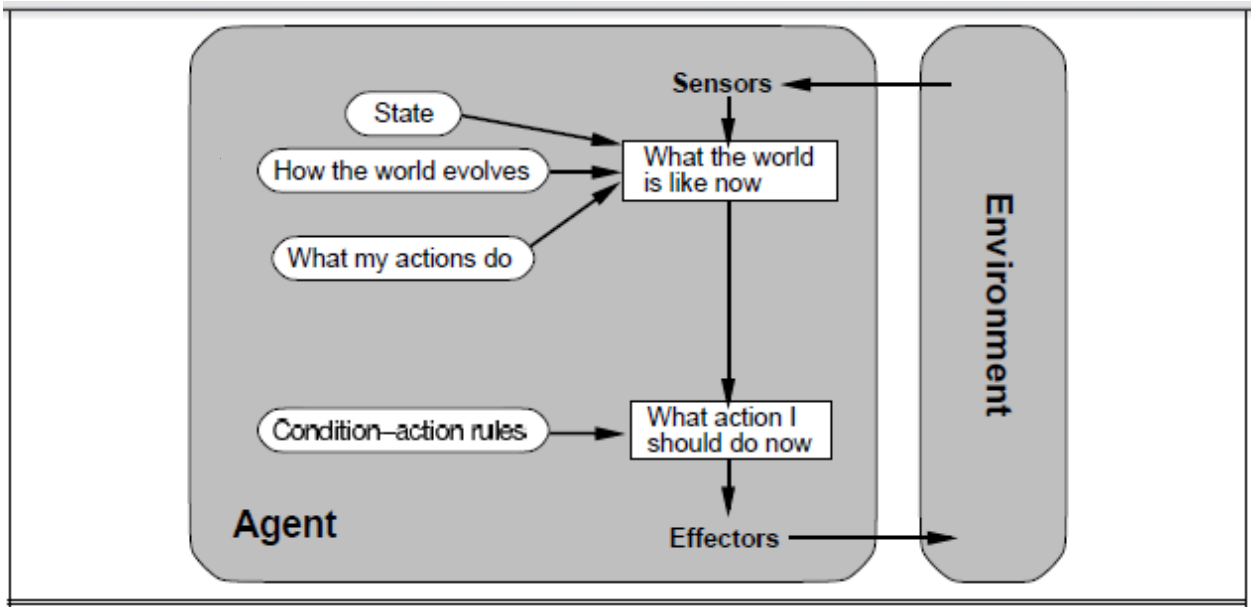



```
function SIMPLE-REFLEX-AGENT(percept) returns action
  static: rules, a set of condition-action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action
```

- Simple reflex agents act only on the basis of the current percept, ignoring the rest of the percept history.
- The agent function is based on the *condition-action rule*: if condition then action.
- This agent function only succeeds when the environment is fully observable.
- Some reflex agents can also contain information on their current state which allows them to disregard conditions whose actuators are already triggered.
- Infinite loops are often unavoidable for simple reflex agents operating in partially observable environments.
- Note: If the agent can randomize its actions, it may be possible to escape from infinite loops.

Model-based reflex agents



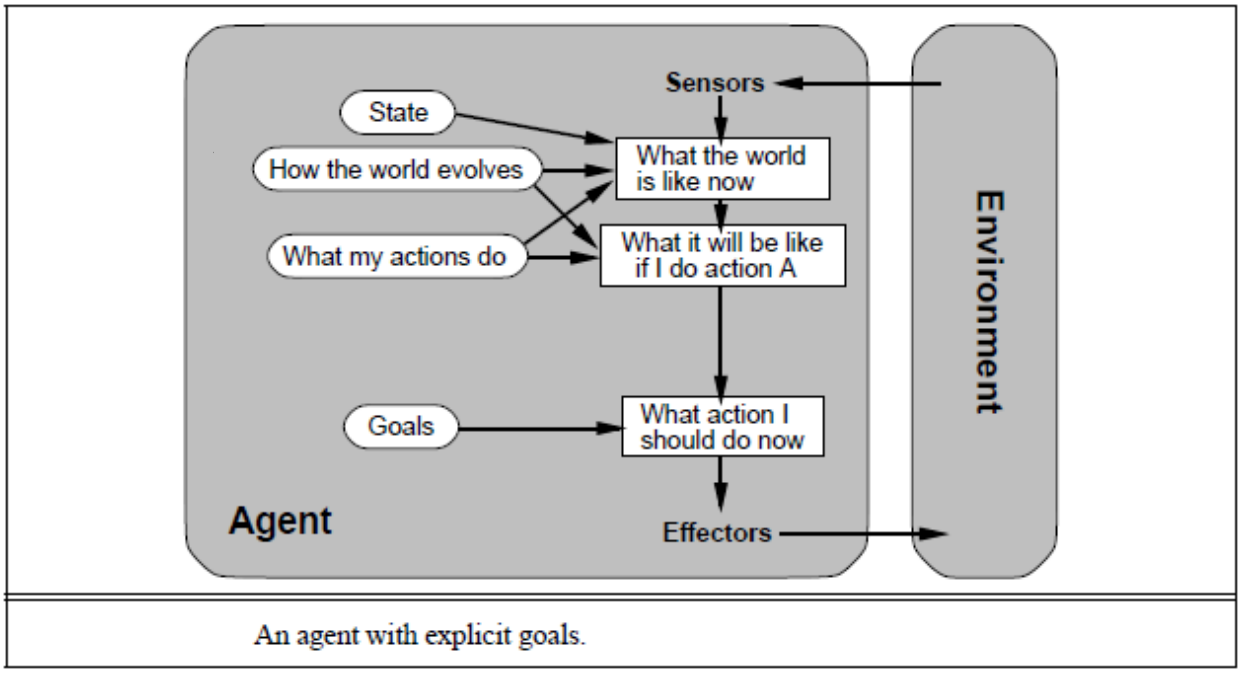

```
function REFLEX-AGENT-WITH-STATE(percept) returns action
  static: state, a description of the current world state
          rules, a set of condition-action rules

  state ← UPDATE-STATE(state, percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  state ← UPDATE-STATE(state, action)
  return action
```

A reflex agent with internal state. It works by finding a rule whose condition matches the current situation (as defined by the percept and the stored internal state) and then doing the action associated with that rule.

- A model-based agent can handle a partially observable environment.
- Its current state is stored inside the agent maintaining some kind of structure which describes the part of the world which cannot be seen.
- This knowledge about "how the world works" is called a model of the world, hence the name "model-based agent".
- A model-based reflex agent should maintain some sort of internal model that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.
- It then chooses an action in the same way as the reflex agent.

Goal-based agents



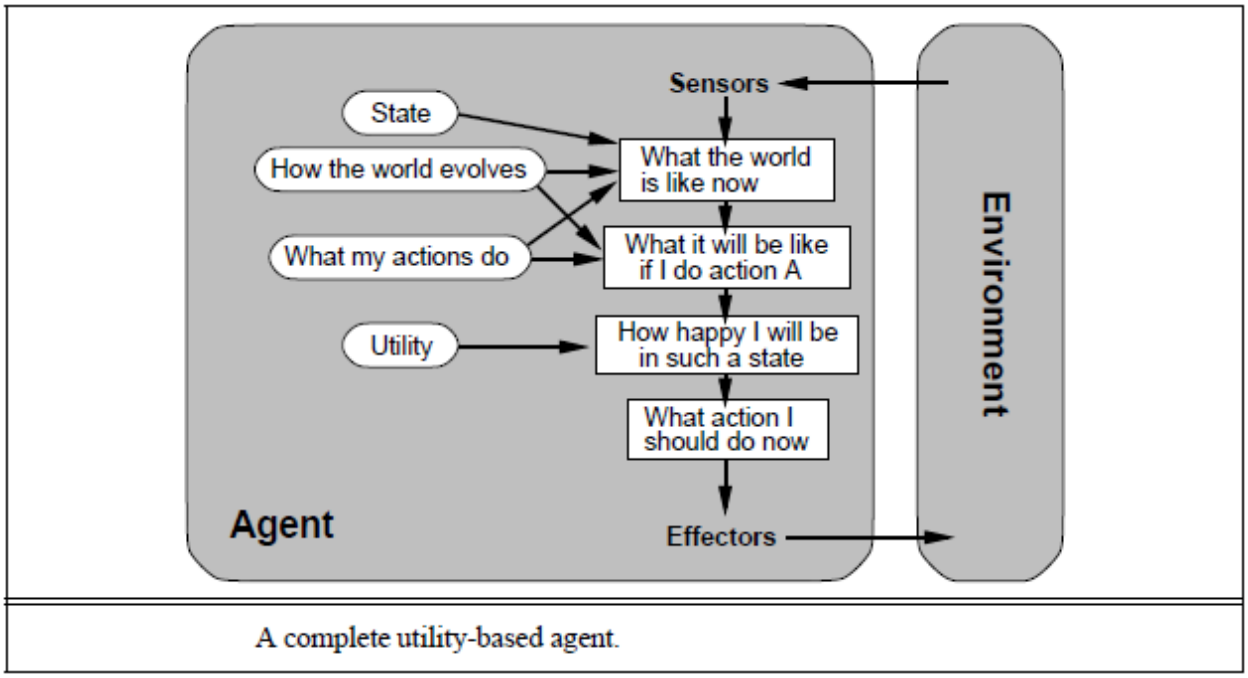
An agent with explicit goals.

- Goal-based agents further expand on the capabilities of the model-based agents, by using "goal" information.
- Goal information describes situations that are desirable.
- This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state.
- Search and planning are the subfields of artificial intelligence devoted to finding action sequences that achieve the agent's goals.
- In some instances the goal-based agent appears to be less efficient; it is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified.

Utility-based agents

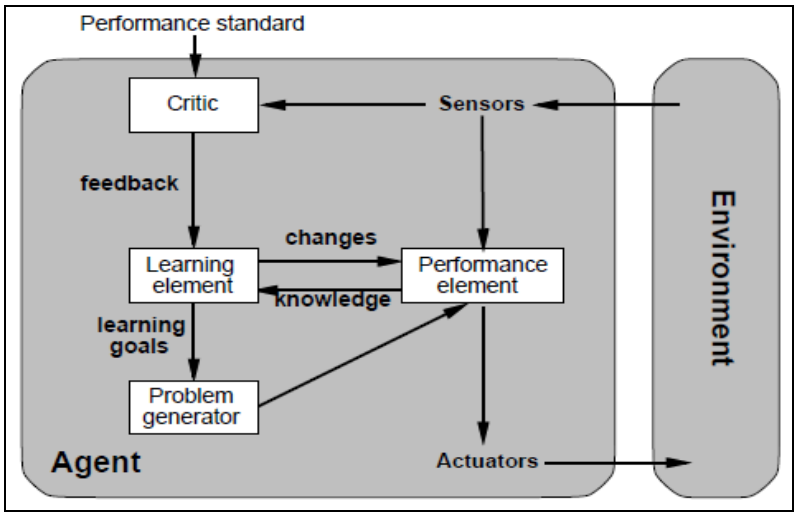
- Goal-based agents only distinguish between goal states and non-goal states.
- It is possible to define a measure of how desirable a particular state is.
- This measure can be obtained through the use of a *utility function* which maps a state to a measure of the utility of the state.
- A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent.
- The term utility can be used to describe how "happy" the agent is.

- A rational utility-based agent chooses the action that maximizes the expected utility of the action outcomes- that is, the agent expects to derive, on average, given the probabilities and utilities of each outcome.
- A utility-based agent has to model and keep track of its environment, tasks that have involved a great deal of research on perception, representation, reasoning, and learning.



A complete utility-based agent.

Learning agents



- Learning has an advantage that it allows the agents to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow.
- 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 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 performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.
- 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.

Properties of environments

Environments come in several flavors. The principal distinctions to be made are as follows:

- **Accessible** vs. **inaccessible**.

If an agent's sensory apparatus gives it access to the complete state of the environment, then we say that the environment is accessible to that agent. An environment is effectively accessible if the sensors detect all aspects that are relevant to the choice of action. An accessible environment is convenient because the agent need not maintain any internal state to keep track of the world.

- **Deterministic** vs. **nondeterministic**(**stochastic**)

If the next state of the environment is completely determined by the current state and the actions selected by the agents, then we say the environment is deterministic. In principle, an agent need not worry about uncertainty in an accessible, deterministic environment. If the environment is inaccessible, however, then it may *appear* to be nondeterministic. This is particularly true if the environment is complex, making it hard to keep track of all the inaccessible aspects. Thus, it is often better to think of an environment as deterministic or nondeterministic *from the point of view of the agent*.

- **Episodic** vs. **nonepisodic (Sequential)**.

In an episodic environment, the agent's experience is divided into "episodes." Each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself, because subsequent episodes do not depend on what actions occur in previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.

- **Static** vs. **dynamic**.

If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. If the environment does not change with the passage of time but the agent's performance score does, then we say the environment is semidynamic.

- **Discrete** vs. **continuous**.

If there are a limited number of distinct, clearly defined percepts and actions we say that the environment is discrete. Chess is discrete—there are a fixed number of possible moves on each turn. Taxi driving is continuous—the speed and location of the taxi and the other vehicles sweep through a range of continuous values.

Ch.4 Common Sensing Techniques for Reactive Robots

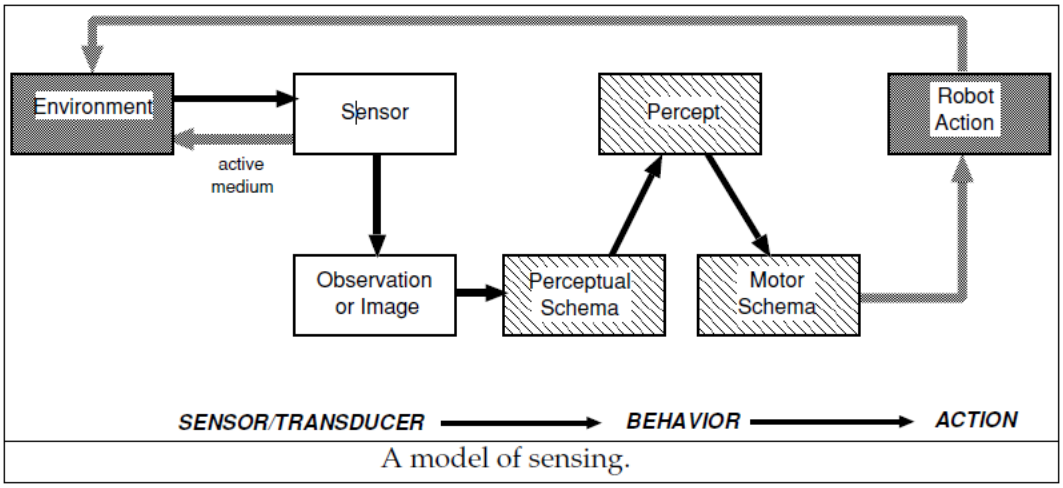
Reactive robot

Reactive robots are behavior-based system, which means they use relatively little internal variable state to model the environment. They receive input through sensors from environment and extract behavior from that input and decide actions.

Sensor

A **sensor** is a device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument. For example, a mercury-in-glass thermometer converts the measured temperature into expansion and contraction of a liquid which can be read on a calibrated glass tube. A thermocouple converts temperature to an output voltage which can be read by a voltmeter. For accuracy, most sensors are calibrated against known standards.

- The *sensor* is a device that measures SENSOR some attribute of the world.
- The term *transducer* is often used interchangeably with *sensor*.
- A transducer is the mechanism, or element, of the sensor that transforms the energy associated with what is being measured into another form of energy.
- A sensor receives energy and transmits a signal to a display or a computer. Sensors use transducers to change the input signal (sound, light, pressure, temperature, etc.) into an analog or digital form capable of being used by a robot.
- In a reactive robot (see figure below), the sensor observation is intercepted by a perceptual schema which extracts the relevant percept of the environment for the behavior.
- This percept is then used by the motor schema, which leads to an action.



A model of sensing.

Active Sensor and Passive Sensor

- A sensor is often classified as being either *passive sensor* or *active sensor*.
- Passive sensors rely on the environment to provide the medium for observation, e.g., a camera requires a certain amount of ambient light to produce a usable picture.
- Active sensors put out energy in the environment to either change the energy or enhance it. Sonar sends out sound, receives the echo, and measures the time of flight.

- An X-ray machine emits X-rays and measures the amount blocked by various types of tissue.
- Although a camera is a passive device, a camera with a flash is active.
- The term *active sensor* is not the same as *active sensing*. Active sensing connotes the system for using an effector to dynamically position a sensor for a "better look."
- A camera with a flash is an active sensor; a camera on a pan/tilt head with algorithms to direct the camera to turn to get a better view is using active sensing.

Logical sensors

- A powerful abstraction of sensors is *logical sensors*.
- A logical sensor is a unit of sensing or module that supplies a particular percept.
- It consists of the signal processing from the physical sensor and the software processing needed to extract the percept; it is the functional building block for perception.
- A logical sensor can be easily implemented as a perceptual schema.
- An overlooked aspect of a logical sensor is that it contains all available alternative methods of obtaining that percept.
- Logical sensor can be implemented as a perceptual schema, where the methods are the alternative means of generating the percept and the coordinated control strategy contains the knowledge as to when a particular method is appropriate.
- ***Logical sensors*** provide data without depending on hardware devices. For example, a logical sensor could provide data about the user's current location by using a service that looks up an IP address in a table. Logical sensors are implemented as sensor drivers.

Attributes of a sensor

1. **Field of view and range.**
    - Every exteroceptive sensor has a region of space that it is intended to cover.
    - The widths of that region are specified by the sensor's *field of view*, often abbreviated as FOV.
    - The field of view is usually expressed in degrees; the number of degrees covered vertically may be different from the number of degrees covered horizontally.
    - Field of view is frequently used in photography, where different lenses capture different size and shape areas. The distance that the field extends is called the range.
    - The *field of view (FOV)* can be thought of in terms of egocentric spherical coordinates, where one angle is the *horizontal FOV* and the other is the *vertical FOV*.
    - The other aspect is the *range*, or how far the sensor can make reliable measurements.
    - In spherical coordinates, this would be the values of r that defined the depth of the operating range.
    - Field of view and ranges are obviously critical in matching a sensor to an application.
    - If the robot needs to be able to detect an obstacle when it's 8 feet away in order to safely avoid it, then a sensor with a range of 5 feet will not be acceptable.
2. **Accuracy, repeatability, and resolution.**
    - Accuracy refers to how correct the reading from the sensor is.
    - But if a reading for the same conditions is accurate only 20% of the time, then the sensor has little repeatability.
    - If the sensor is consistently inaccurate in the same way (always 2 or 3 cm low), then the software can apply a bias (add 2 centimeters) to compensate.

- If the inaccuracy is random, then it will be difficult to model and the applications where such a sensor can be used will be limited.
- If the reading is measured in increments of 1meter, that reading has less *resolution* than a sensor reading which is measured in increments of 1 cm.

3. **Responsiveness in the target domain.**
   - Most sensors have particular environments in which they function poorly.
   - Another way of viewing this is that the environment must allow the signal of interest to be extracted from noise and interference (e.g., have favorable signal-to-noise ratios).
   - Sonar is often unusable for navigating in an office foyer with large amounts of glass because the glass reflects the sound energy in ways almost impossible to predict.
4. **Power consumption.**
   - Power consumption is always a concern for robots.
   - Since most robots operate off of batteries, the less power they consume, the longer they run. For example, the battery life on a Nomad 200, which carries five batteries, was improved from four hours to six by shutting off all sensors.
5. **Hardware reliability.**
   - Sensors often have physical limitations on how well they work. For example, Polaroid sonars will produce incorrect range reading when the voltage drops below 12V.
   - Other sensors have temperature and moisture constraints which must be considered.
6. **Size.**
   - The size and weight of a sensor does affect the overall design.
   - Size of sensor should be proportionate w.r.t. robot hardware**.** E.g. Webcam on Laptop.
7. **Computational complexity.**
   - Computational complexity is the estimate of how many operations an algorithm or program performs.
   - It is often written as a function O, called the "order," where O(x) means the number of operations is proportional to *x.*
8. **Interpretation reliability.**
   - The designer should consider how reliable the sensor will be for the ecological conditions and for interpretation.
   - The robot will often have no way of determining when a sensor is providing incorrect information. As a result the robot may "hallucinate" (think it is seeing things that are not there) and do the wrong thing.

<u>Proprioceptive sensor</u>

- The robot measures a signal originating from within using proprioceptive sensors. These sensors are responsible for monitoring self-maintenance and controlling internal status.
- Common uses of proprioceptive measurements are for battery monitoring, current sensing, and heat monitoring.

Examples of Proprioceptive Sensors:

- Global Positioning System (GPS) - The drawbacks of GPS sensors include a slow refresh rate and sensitivity to blackouts.

- Inertial Navigation System (INS) - The drawbacks of INS include the tendency to drift over time and with temperature.
- Shaft Encoders - A shaft encoder, also known as a rotary encoder, is an electro-magnetic device that works as a transducer to convert the angular position of a shaft or axle to an analog or digital code.
- Compass - A compass sensor is used to detect direction and accurately correct motion.
- Inclinometer - An inclinometer sensor measures the tilt or angle of an axis.

The Global Positioning System (GPS)

- **The Global Positioning System (GPS)** is a satellite-based navigation system made up of a network of 24 satellites placed into orbit by the U.S. Department of Defense.
- GPS works in any weather conditions, anywhere in the world, 24 hours a day.
- There are no subscription fees or setup charges to use GPS.
- GPS systems work by receiving signals from satellites orbiting the Earth.
- GPS has three parts,
    1) Satellite (which send signal to base station in form of Latitude, Longitude and altitude with time)
    2) Base station (Base station at earth, triangulate that signal and determine user position.)
    3) User receiver (it is device which receives position of user).

Proximity Sensor

- A **proximity sensor** is a sensor able to detect the presence of nearby objects without any physical contact.
- A proximity sensor often emits an electromagnetic or electrostatic field, or a beam of electromagnetic radiation (infrared, for instance), and looks for changes in the field or return signal.
- Proximity sensors can have a high reliability and long functional life because of the absence of mechanical parts and lack of physical contact between sensor and the sensed object.
- Since the sensor is mounted on the robot, it is a straightforward computation to translate a range relative to the sensor to a range relative to the robot at large.
- Most proximity sensors are active. Sonar, also called ultrasonic, is the most popular proximity sensor, with infrared, bump, and feeler sensors not far behind.

Sonar or ultrasonic

- Sonar refers to any system for using sound to measure range.
- Sonars for different applications operate at different frequencies.
- They are active sensors which emit a sound and measure the time it takes for the sound to bounce back.
- The *time of flight* (time from emission to bounce back) along with the speed of sound in that environment (remember, even air changes density with altitude) is sufficient to compute the range of the object.

- A robotic sonar transducer is shown in diagram. The transducer is about the size and thickness of a dollar coin, and consists of a thin metallic membrane.
- A very strong electrical pulse generates a waveform, causing the membrane on the transducer to produce a sound.
- The sound is audible as a faint clicking noise, like a crab opening and closing its pinchers. Meanwhile a timer is set, and the membrane becomes stationary.
- The reflected sound, or *echo*, vibrates the membrane which is amplified and then threshold on return signal strength; if too little sound was received, and then the sensor assumes the sound is noise and so ignores it. If the signal is strong enough to be valid, the timer is tripped, yielding the time of flight.

Infrared sensor

- An infrared sensor is an electronic device that emits and/or detects infrared radiation in order to sense some aspect of its surroundings.
- Infrared sensors can measure the heat of an object, as well as detect motion.
- Infrared sensor units are capable of emitting infrared light, which cannot be seen by the human eye, and are able to turn on devices when the infrared light is disturbed. Selecting an infrared sensor with an appropriate range will ensure the sensor covers the desired area so nothing can get around it.
- Infrared (IR) radiation is part of the electromagnetic spectrum, which includes radio waves, microwaves, visible light, and ultraviolet light, as well as gamma rays and X-rays.
- The IR range falls between the visible portion of the spectrum and radio waves. IR wavelengths are usually expressed in microns, with the lR spectrum extending from 0.7 to 1000 microns. Only the 0.7-14 micron band is used for IR temperature measurement.

CCD Cameras

- Computer vision on reactive robots is most often from a video camera, which uses CCD (charged couple device) technology to detect visible light.
- A video camera, such as a camcorder, is arranged so that light falls on an array of closely spaced metal-oxide semiconductor (MOS) capacitors.
- Interestingly, the MOS capacitors are rectangular, not square, so there is some distortion in creating the image.
- The capacitors form a shift register, and output is either a line at a time (“line transfer”) or the whole array at one time (“frame transfer").
- The output of most consumer video signals is analog, and must be digitized for use by a computer.
- Consumer digital cameras post an analog signal, but the update rate is too slow at this time for real-time reactive robot control.
- The A/D conversion process takes longer than the CCD array can sense light, so the camera device can either have many frame buffers, which create a pipeline of images (but is expensive), or have a low frame rate.

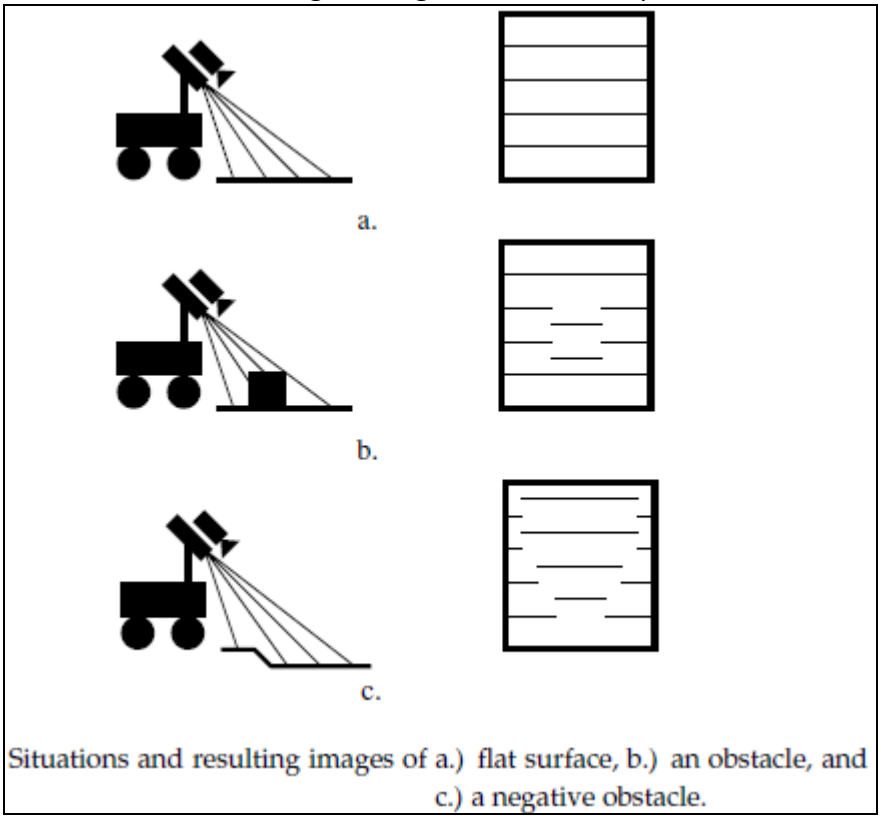
Stereo camera pairs

- Using two cameras to extract range data is often referred to as *range from stereo, stereo disparity, binocular vision,* or just plain “stereo.”
- One way to extract depth is to try to superimpose a camera over each eye as in **Fig.a**.

- Each camera finds the same point in each image, turns itself to center that point in the image, then measures the relative angle.
- The cameras are known as the *stereo pair*.

Light stripers

- *Light striping, light stripers* or *structured light detectors* work by projecting a colored line (or stripe), grid, or pattern of dots on the environment.
- Then a regular vision camera observes how the pattern is distorted in the image. For example, in **Fig.a**, the striper projects four lines.
- The lines should occur at specific, evenly spaced rows in the camera image if the surface is flat.
- If the surface is not flat, as shown in **Fig.b**, the lines will have breaks or discontinuities.
- A vision algorithm can quickly scan each of the designated rows to see if the projected line is continuous or not.
- The location of the breaks in the line gives information about the size of the obstacle.
- The vision algorithm can also look for where the dislocated line segments appear, since the distance in image coordinates is proportional to the depth of the object.
- The relative placement of the lines indicates whether the object is above the ground plane (an obstacle) or below (NEGATIVE OBSTACLE a hole or *negative obstacle*) as shown in **fig.c**.
- The more lines or finer-grained grid, the more depth information.



Situations and resulting images of a.) flat surface, b.) an obstacle, and c.) a negative obstacle.

Laser

- A **laser** is a device that emits light (electromagnetic radiation) through a process of optical amplification based on the stimulated emission of photons.
- The term "laser" originated as an acronym for *Light Amplification by Stimulated Emission of Radiation*.
- The emitted laser light is notable for its high degree of spatial and temporal coherence, unattainable using other technologies.

- Spatial coherence typically is expressed through the output being a narrow beam which is diffraction-limited, often a so-called "pencil beam."
- Laser beams can be focused to very tiny spots, achieving a very high irradiance. Or they can be launched into a beam of very low divergence in order to concentrate their power at a large distance.
- Lasers are employed in applications where light of the required spatial or temporal coherence could not be produced using simpler technologies.

Ch.5 Problem solving

PROBLEM-SOLVING AGENTS

Intelligent agents are supposed to act in such a way that the environment goes through a sequence of states that maximizes the performance measure.

What is Problem solving agent?

- It is a kind of Goal-based Agents
- 4 general steps in problem-solving:

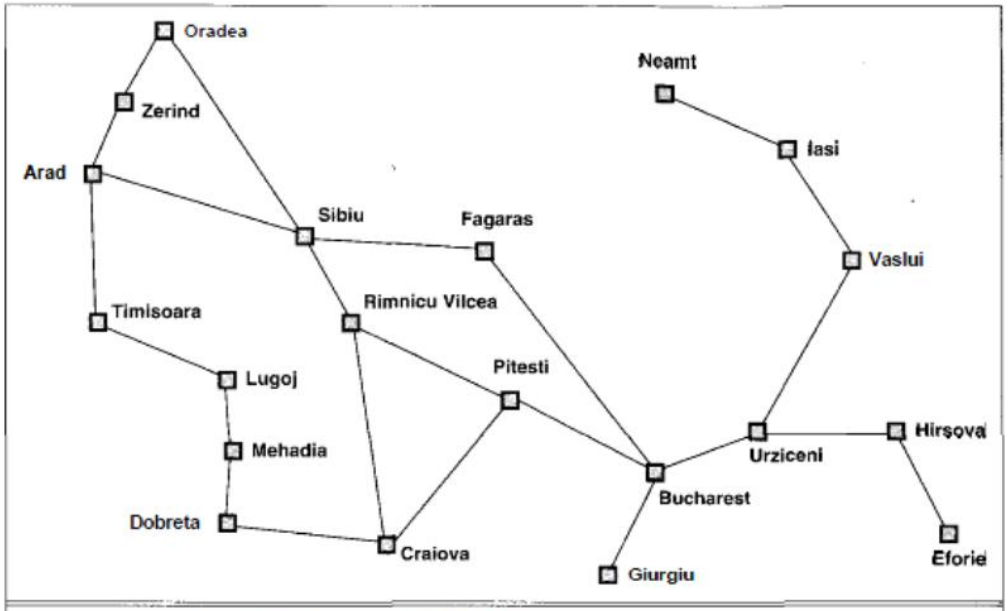
  1. Goal Formulation

  2. Problem Formulation

  3. Search

  4. Execute

- E.g. Driving from Arad to Bucharest ...
- ***Note**: In this chapter we will consider one example that "A map is given with different cities connected and their distance values are also mentioned. Agent starts from one city and reach to other."*



A simplified road map of Romania.

Subclass of goal-based agents

- goal formulation
- problem formulation
- example problems
  - Toy problems
  - Real-world problems
- search
  - search strategies
  - Constraint satisfaction
- solution

Goal Formulation

Goal formulation, based on the current situation, is the first step in problem solving. As well as formulating a goal, the agent may wish to decide on some other factors that affect the desirability of different ways of achieving the goal. For now, let us assume that the agent will consider actions at the level of driving from one major town to another. The states it will consider therefore correspond to being in a particular town.

- Declaring the Goal: Goal information given to agent *i.e. start from Arad and reach to Bucharest.*
- Ignoring the some actions: agent has to ignore some actions that will not lead agent to desire goal. *i.e. there are three roads out of Arad, one toward Sibiu, one to Timisoara, and one to Zerind. None of these achieves the goal, so unless the agent is very familiar with the geography of Romania, it will not know which road to follow. In other words, the agent will not know which of its possible actions is best, because it does not know enough about the state that results from taking each action.*
- Limits the objective that agent is trying to achieve: Agent will decide its action when he has some added knowledge about map. *i.e. map of Romania is given to agent.*
- Goal can be defined as set of world states: The agent can use this information to consider subsequent stages of a hypothetical journey through each of the three towns, to try to find a journey that eventually gets to Bucharest. *i.e. once it has found a path on the map from Arad to Bucharest, it can achieve its goal by carrying out the driving actions.*

Problem Formulation

Problem formulation is the process of deciding what actions and states to consider, given a goal.

- **Process of looking for action sequence (number of action that agent carried out to reach to goal) is called search. A search algorithm takes a problem as input and returns a solution in the form of an action sequence. Once a solution is found, the actions it recommends can be carried out. This is called the execution phase. Thus, we have a simple "formulate, search, execute" design for the agent.**

Well-defined problem and solutions.

A *problem* is defined by four items:

Initial state: *The initial state that the agent starts in.* e.g., the initial state for our agent in *Romania might be described as "In(Arad)".*

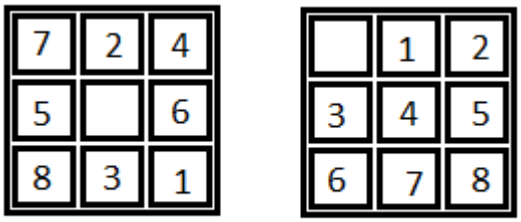
Successor function **S(x)** = A description of the possible actions available to the agent. The most common formulation uses a successor function , given a particular state x, SUCCESSOR-FN(x) returns a set of <action, successor> ordered pair where each action is one of the legal actions in state x and each successor is a state

that can be reached from x by applying the action. e.g. ,from state *In(Arad),*the successor function for Romania problem would return {<Go(Zerind),In(Zerind)>, <Go(sibiu),In(sibiu)>, <Go(Timisoara),In(Timisoara)>}.

Goal test=It determines whether a given state is a goal state.

path cost (additive)=Function that assigns a numeric cost to each path. e.g., sum of distances, number of actions executed, etc. Usually given as c(x, a, y), the step cost from x to y by action a, assumed to be ≥ 0.
"A *solution* is a sequence of actions leading from the initial state to a goal state".

Example:
The 8-puzzle consist of a 3x3 board with 8 numbered tiles and a blank space.A tile adjacent to the blank space can slide into the space.



Initial state Final State

The standard formulation is as follows:
States: A state description specifies the location of each of the eight tiles and blank is one of the nine squares.
Initial Function: any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states.
Successor function: This generates the legal states that result from trying the four actions (blank *moves Left, Right, Up or Down*).
Goal State: This checks whether the state matches the goal configuration shown in figure.
Path cost: Each step cost 1; so the path cost is the number of steps in path.

There are two types of searching strategies are used in path finding,

1) Uninformed Search strategies.
2) Infirmed Search strategies.

Uninformed Search:
Uninformed search means that they have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from non-goal state.
Uninformed strategies use only the information available in the problem definition

1) Breadth-first search
2) Uniform-cost search
3) Depth-first search
4) Depth-limited search
5) Iterative deepening search

Note: (Only for Understanding)

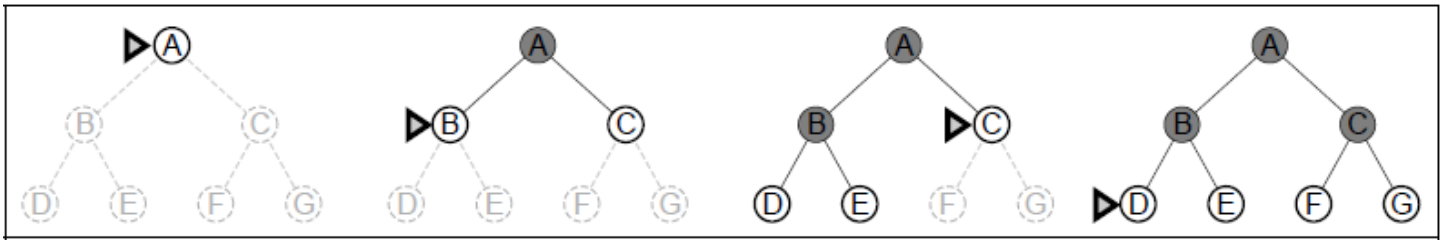
1) It is important to understand the distinction between nodes and states.
   A node is book keeping data structure used to represent the search tree.
   A state corresponds to a configuration of the world.
2) We also need to represent the collection of nodes that have been generated but not yet expanded, this collection is called the **fringe.**
3) In AI ,where the graph is represented implicitly by the initial state and successor function and is frequently infinite , its complexity expressed in terms of three quantities:
   b ; the branching factor or maximum number of successor of any node.
   d; the depth of shallowest goal node; and
   m; the maximum length of any path in the state space.

Breadth First Search (BFS Algorithm)

- **Breadth First Search** (BFS) searches breadth-wise in the problem space.
- Breadth-First search is like traversing a tree where each node is a state which may be a potential candidate for solution.
- Breadth first search expands nodes from the root of the tree and then generates one level of the tree at a time until a solution is found.
- It is very easily implemented by maintaining a queue of nodes.
- Initially the queue contains just the root.
- In each iteration, node at the head of the queue is removed and then expanded.
- The generated child nodes are then added to the tail of the queue.

Algorithm:

1. Place the starting node.
2. If the queue is empty return failure and stop.
3. If the first element on the queue is a goal node, return success and stop otherwise.
4. Remove and expand the first element from the queue and place all children at the end of the queue in any order.
5. Go back to step 1.

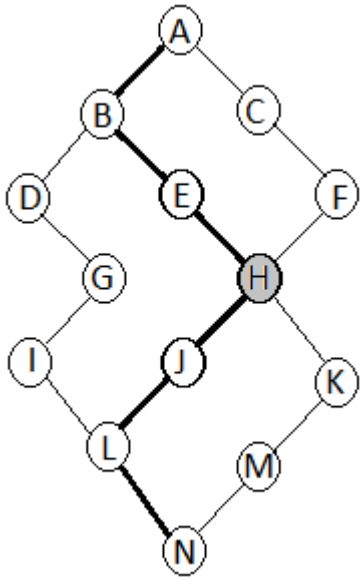


Advantages:
Breadth first search will never get trapped exploring the useless path forever.
If there is a solution, BFS will definitely find it out.
If there is more than one solution then BFS can find the minimal one that requires less number of steps.

Disadvantages:
If the solution is farther away from the root, breath first search will consume lot of time.

Bidirectional search

- Bidirectional Search, as the name implies, searches in two directions at the same time: one forward from the initial state and the other backward from the goal.
- This is usually done by expanding tree with branching factor b and the distance from start to goal is d.
- The search stops when searches from both directions meet in the middle.
- Bidirectional search is a brute-force search algorithm that requires an explicit goal state instead of simply a test for a goal condition.
- Once the search is over, the path from the initial state is then concatenated with the inverse of the path from the goal state to form the complete solution path.
- The time complexity of Bidirectional Search is O(b^d/2) since each search need only proceed to half the solution path.
- Since at least one of the searches must be breadth-first in order to find a common state, the space complexity of bidirectional search is also O(b^d/2). As a result, it is space bound in practice.



Advantages

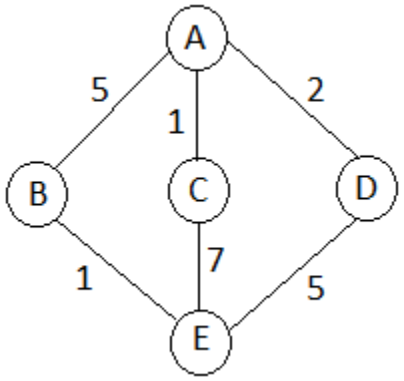
- The merit of bidirectional search is its speed. Sum of the time taken by two searches (forward and backward) is much less than the $O(b^d)$ complexity.
- It requires less memory.

Disadvantages

- Implementation of bidirectional search algorithm is difficult because additional logic must be included to decide which search tree to extend at each step.
- One should have known the goal state in advance.
- The algorithm must be too efficient to find the intersection of the two search trees.
- It is not always possible to search backward through possible states.

Uniform Cost Search

- If all the edges in the search graph do not have the same cost then breadth-first search generalizes to uniform-cost search.
- Instead of expanding nodes in order of their depth from the root, uniform-cost search expands nodes in order of their cost from the root.
- At each step, the next step n to be expanded is one whose cost g(n) is lowest where g(n) is the sum of the edge costs from the root to node n. The nodes are stored in a priority queue.
- Whenever a node is chosen for expansion by uniform cost search, a lowest-cost path to that node has been found.
- The worst case time complexity of uniform-cost search is $O(b^c/m)$, where c is the cost of an optimal solution and m is the minimum edge cost. Unfortunately, it also suggests the same memory limitation as breadth-first search.



Depth First Search:

Depth-first search (**DFS**) is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

Algorithm:

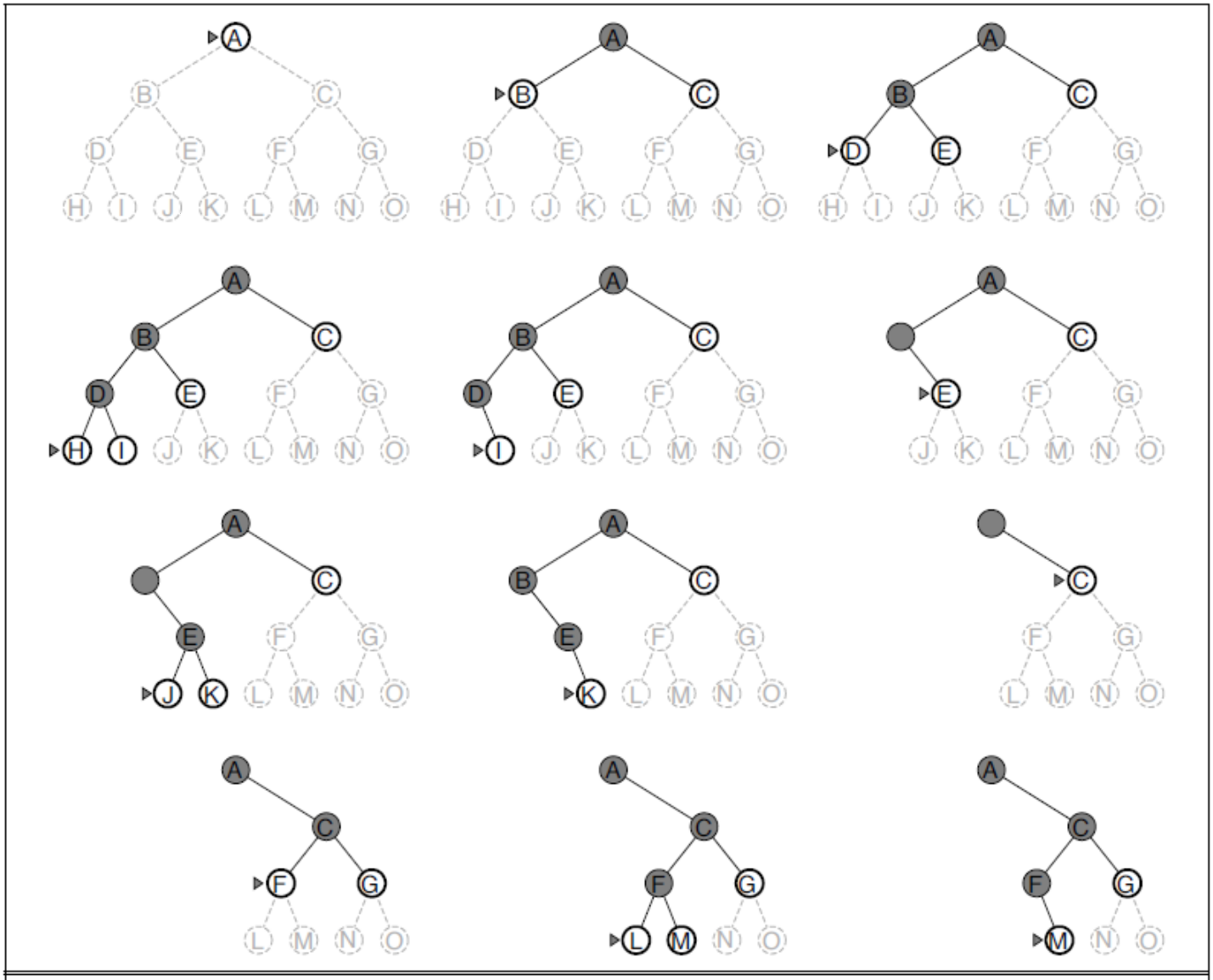
1. Push the root node onto a stack.
2. Pop a node from the stack and examine it.
    - If the element sought is found in this node, quit the search and return a result.
    - Otherwise push all its successors (child nodes) that have not yet been discovered onto the stack.
3. If the stack is empty, every node in the tree has been examined – quit the search and return "not found".
4. If the stack is not empty, repeat from Step 2.

Advantages:

- If depth-first search finds solution without exploring much in a path then the time and space it takes will be very less.
- The advantage of depth-first Search is that memory requirement is only linear with respect to the search graph. This is in contrast with breadth-first search which requires more space.

Disadvantages:

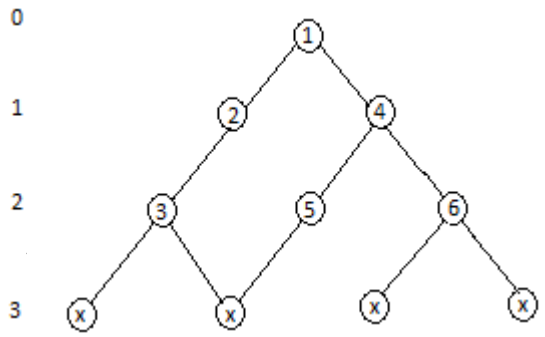
- Depth-First Search is not guaranteed to find the solution.
- It is not complete algorithm, if it go into infinite loops.



Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors, and M is the only goal node.

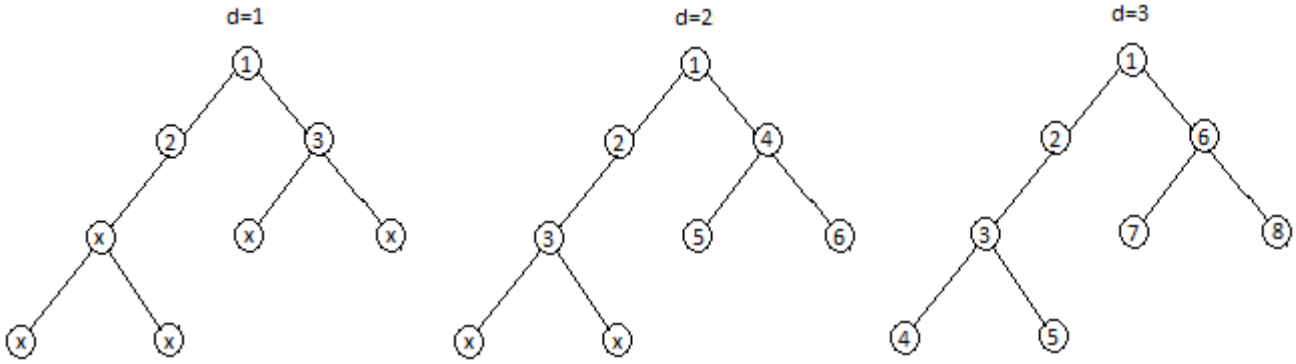
Depth Limited Search:

- Depth limited search (DLS) is a modification of depth-first search that minimizes the depth that the search algorithm may go.
- In addition to starting with a root and goal node, a depth is provided that the algorithm will not descend below.
- Any nodes below that depth are omitted from the search.
- This modification keeps the algorithm from indefinitely cycling by halting the search after the pre-imposed depth.
- The time and space complexity is similar to DFS from which the algorithm is derived.
- Space complexity: O(bd) and Time complexity : $O(b^d)$



Iterative Deepening Search:

- Iterative Deepening Search (IDS) is a derivative of DLS and combines the feature of depth-first search with that of breadth-first search.
- IDS operates by performing DLS searches with increased depths until the goal is found.
- The depth begins at one, and increases until the goal is found, or no further nodes can be enumerated.
- By minimizing the depth of the search, we force the algorithm to also search the breadth of a graph.
- If the goal is not found, the depth that the algorithm is permitted to search is increased and the algorithm is started again.



Comparing Search Strategies:

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Informed Search techniques:

- A strategy that uses problem-specific knowledge beyond the definition of the problem itself.
- Also known as "heuristic search," informed search strategies use information about the domain to (try to) (usually) head in the general direction of the goal node(s)
- Informed search methods: Hill climbing, best-first, greedy search, beam search, A, A*.

Best First Search:

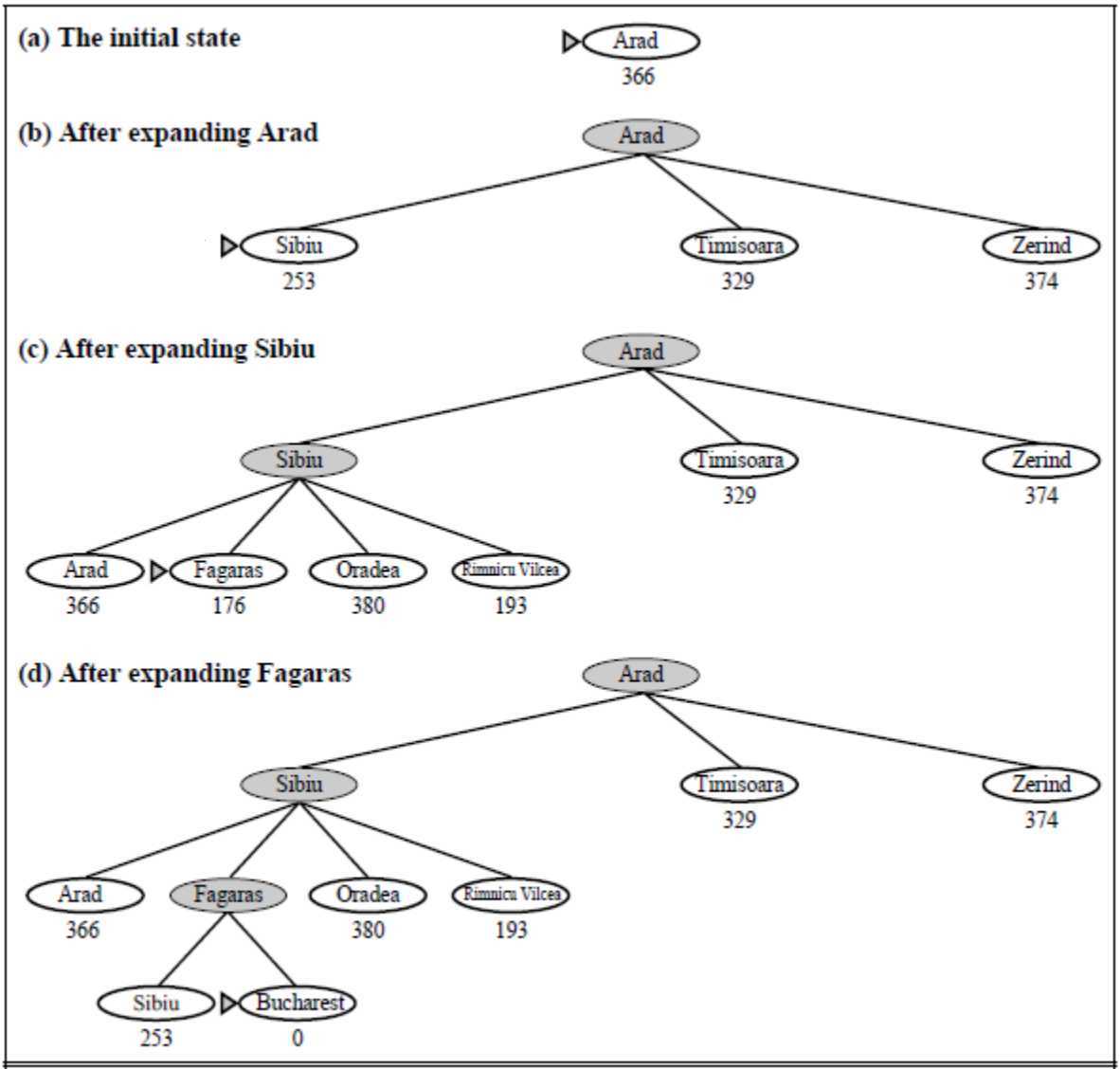
- It is an algorithm in which a node is selected for expansion based on an evaluation function f(n).
- Traditionally the node with the lowest evaluation function is selected.
- Not an accurate name…expanding the best node first would be a straight march to the goal.
- Choose the node that *appears* to be the best.
- There is a whole family of Best-First Search algorithms with different evaluation functions.
  - Each has a heuristic function h(n).
- *h(n) = estimated cost of the cheapest path from node n to a goal node*
- Example: in route planning the estimate of the cost of the cheapest path might be the straight line distance between two cities.
- Quick Review,
  - g(n) = cost from the initial state to the current state n
  - h(n) = estimated cost of the cheapest path from node n to a goal node
  - f(n) = evaluation function to select a node for expansion (usually the lowest cost node)
- Best-First Search can be represented in two ways,
  - Greedy Best-First Search
  - A* search

Greedy Best-First Search:

- Greedy Best-First search tries to expand the node that is closest to the goal assuming it will lead to a solution quickly
  - f(n) = h(n)
  - aka "Greedy Search"
- Implementation
  - Expand the "most desirable" node into the fringe queue.
  - Sort the queue in decreasing order of desirability.
- Example: consider the straight-line distance heuristic h_{SLD}
  - Expand the node that appears to be closest to the goal.
- h_{SLD}(In(Arid)) = 366
- Notice that the values of h_{SLD} cannot be computed from the problem itself
- It takes some experience to know that h_{SLD} is correlated with actual road distances
  - Therefore a useful heuristic.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}—straight-line distances to Bucharest. (Refer Pg.no.49 for Map of Romania)

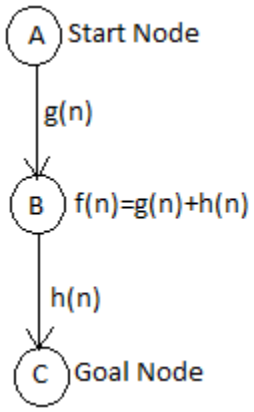


Stages in a greedy best-first tree search for Bucharest using the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

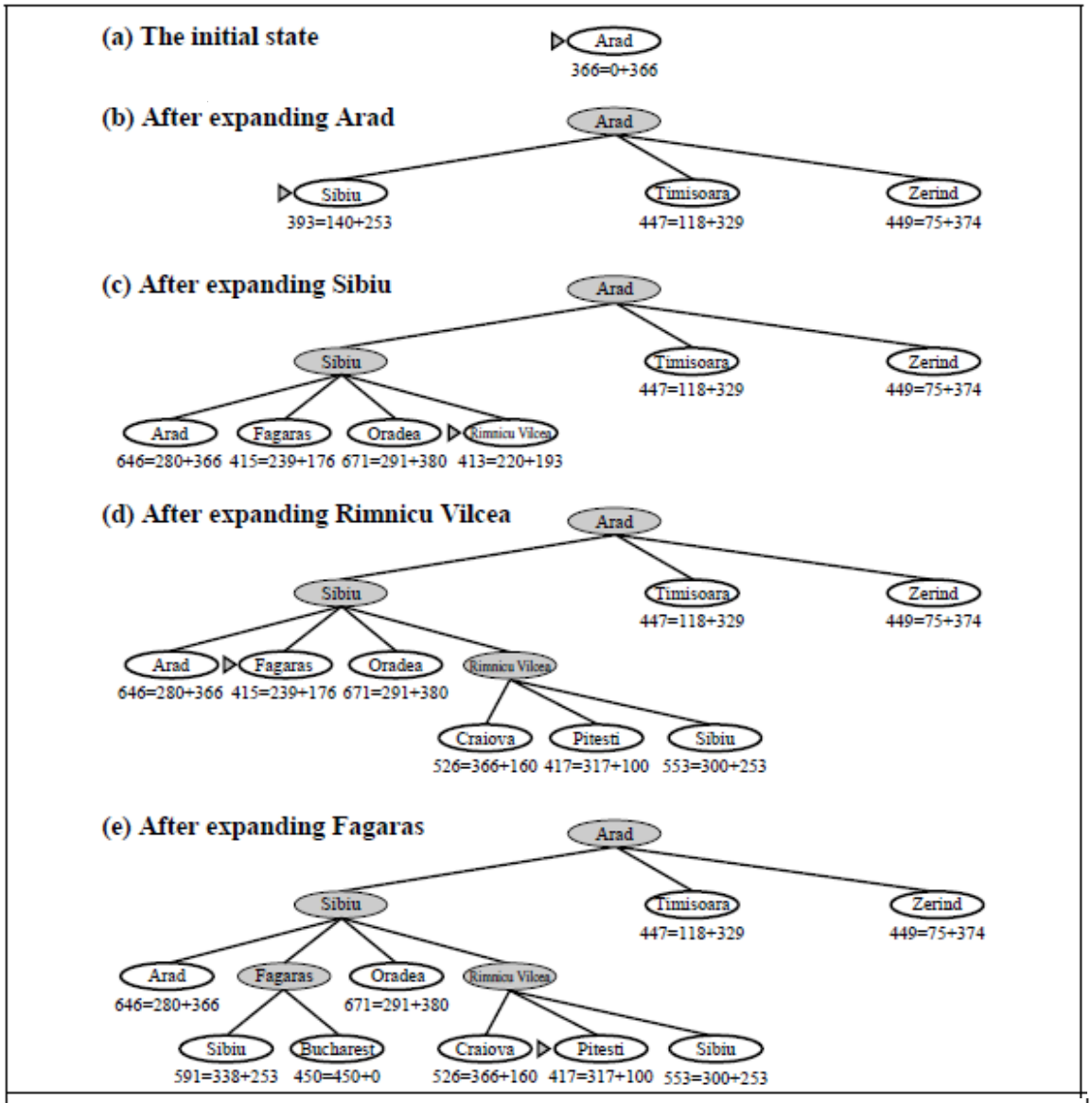
So the path is: Arad-Sibiu-Fagaras-Bucharest

A* search

- A* (A star) is the most widely known form of Best-First search
  - It evaluates nodes by combining g(n) and h(n).
  - f(n) = g(n) + h(n).
  - Where
    - g(n) = cost so far to reach n.
    - h(n) = estimated cost to goal from n.
    - f(n) = estimated total cost of path through n.
- When h(n) = actual cost to goal
  - Only nodes in the correct path are expanded
  - Optimal solution is found
- When h(n) < actual cost to goal
  - Additional nodes are expanded



  - Optimal solution is found
- When h(n) > actual cost to goal
  - Optimal solution can be overlooked.



Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Pg.no.58

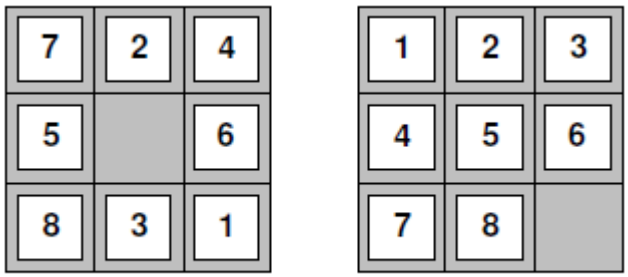
Heuristic Function:

It is a technique which evaluation the state and finds the significance of that state w.r.t. goal state because of this it is possible to compare various states and choose the best state to visit next.

Heuristics are used to improve efficiency of search process. The search can be improved by evaluating the states. The states can be evaluated by applying some evaluation function which tells the significance of state to achieve goal state. The function which evaluates the state is the heuristic function and the value calculated by this function is heuristic value of state.

The heuristic function is represented as h(n)

Eg. 8-puzzle problem

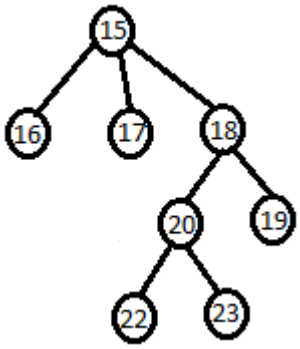


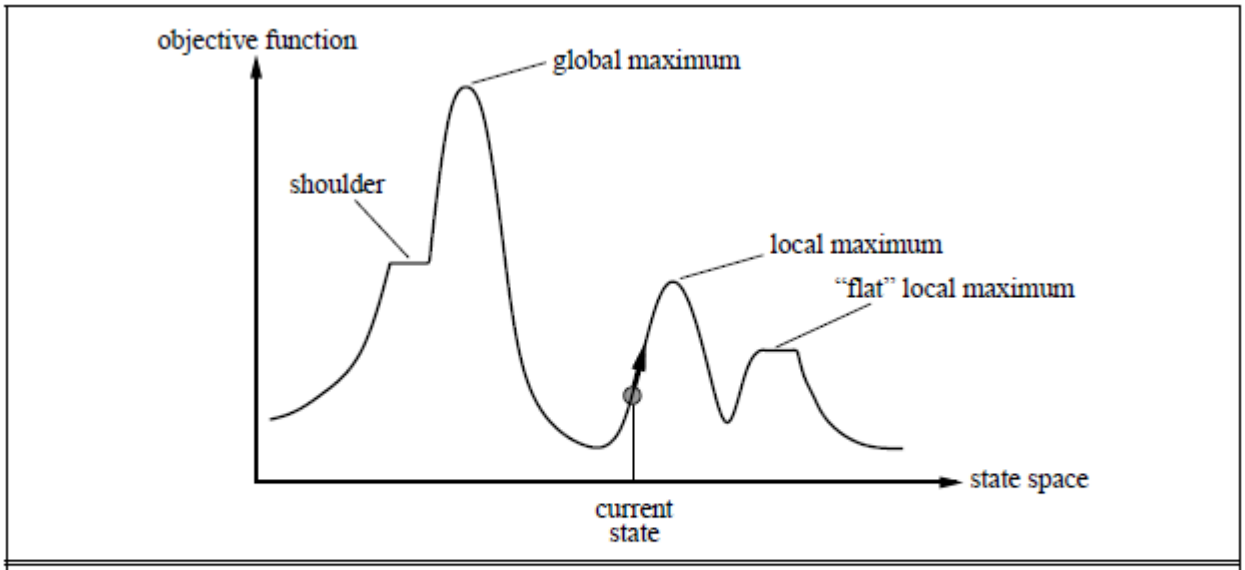
Admissible heuristics
$h1(n)$ = number of misplaced tiles
$h2(n)$ = total Manhattan distance (i.e., no. of squares from desired location of each tile)
In the example
$h1(S) = 6$
$h2(S) = 2 + 0 + 3 + 1 + 0 + 1 + 3 + 4 = 14$

If $h2$ dominates $h1$, then $h2$ is better for search than $h1$.

Hill climbing:

- **Hill climbing** is a mathematical optimization technique which belongs to the family of local search.
- It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution.
- If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.
- Hill climbing is good for finding a local optimum (a solution that cannot be improved by considering a neighboring configuration) but it is not guaranteed to find the best possible solution (the global optimum) out of all possible solutions (the search space).
- Global maximum is the best possible solution and the objective of this search to reach at global maximum (highest peak on hill).





A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

Problems in Hill climbing.

Local maxima:

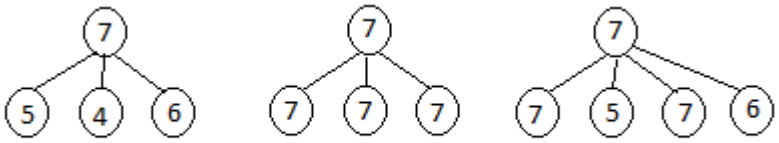
- Local maxima = no uphill step
  - Algorithms on previous slide fail (not complete)
  - Allow "random restart" which is complete, but might take a very long time.

Ridges:

- A "ridge" which is an area in the search that is higher than the surrounding areas, but cannot be searched in a simple move.

Plateau:

- All steps equal (flat or shoulder)
- A plateau is encountered when the search space is flat, or sufficiently flat that the value returned by the target function is indistinguishable from the value returned for nearby regions due to the precision used by the machine to represent its value.
- In such cases, the hill climber may not be able to determine in which direction it should step, and may wander in a direction that never leads to improvement.



(a) Local maxima (b) Platue (c) Ridge

a). Successor of current state are heuristically worst than the current state.
b).Successor of current state are heuristically equivalent to the current state.
c).Successor of current state are heuristically worst or heuristically equivalent than the current state.

6. Knowledge Representation

• ***Knowledge*** is a general term.
An answer to the question, *"how to represent knowledge"*, requires an analysis to distinguish between knowledge "how" and knowledge "that".

- Knowing *"how to do something".*
  e.g. "how to drive a car" is Procedural knowledge.
- Knowing *"that something is true or false".*
  e.g. "that is the speed limit for a car on a motorway" is Declarative knowledge.

Knowledge and Representation are distinct entities that play a central but distinguishable role in intelligent system.

- Knowledge is a description of the world.
  - It determines a system's competence by what it knows.
- Representation is the way knowledge is encoded.
  - It defines the performance of a system in doing something.

• Different types of knowledge require different kinds of representation.
The Knowledge Representation ***models/mechanisms*** are often based on:

- *Logic*
- *Rules*
- *Frames*
- *Semantic Net*

• Different types of knowledge require different kinds of ***reasoning***.

Knowledge-based agent:

- A knowledge-based agent includes a knowledge base and an inference system.
- A knowledge base is a set of representations of facts of the world.
- Each individual representation is called a **sentence**.
- The sentences are expressed in a knowledge **representation language**.
- The agent operates as follows:

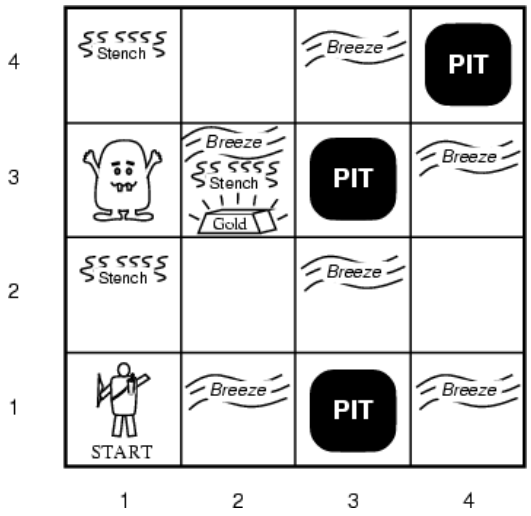
  1. It TELLs the knowledge base what it perceives.
  2. It ASKs the knowledge base what action it should perform.
  3. It performs the chosen action.

The WUMPUS WORLD Environment

- The Wumpus computer game
- The agent explores a cave consisting of rooms connected by passageways.
- Lurking somewhere in the cave is the Wumpus, a beast that eats any agent that enters its room.
- Some rooms contain bottomless pits that trap any agent that wanders into the room.
- Occasionally, there is a heap of gold in a room.
- The goal is to collect the gold and exit the world without being eaten.

Typical Wumpus World:

- The agent always starts in the field [1,1].
- The task of the agent is to find the gold, return to the field [1,1] and climb out of the cave.



WUMPUS WORLD PEAS description

Performance:
Points are awarded and/or deducted:
– find gold: +1000
– death by Wumpus: -1000
– death by endless pit: -1000
– each action: -1
– picking up the arrow: -10

Environment:
– 4x4 grid.
– Agent starts at 1,1 (lower left).
– Agent starts facing to the right.
– The gold and the Wumpus are placed at random locations in the grid. (can't be in start room).
– Each room other than starting room can be a bottomless pit with probability 0.2

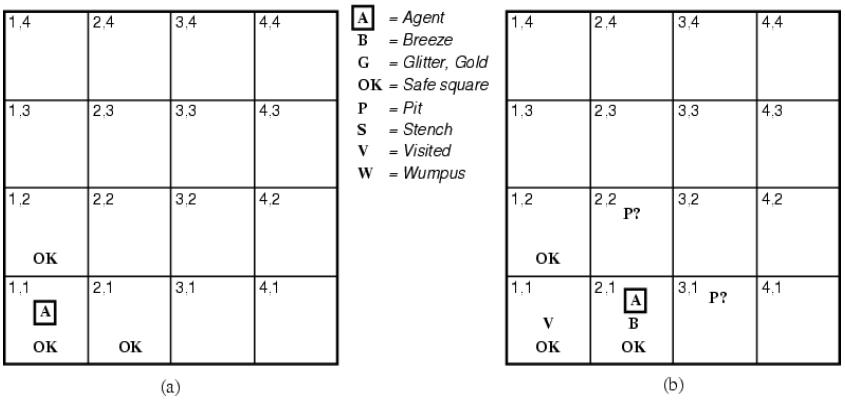
Actuators:
● Turn left 90°
● Turn right 90°
● Move forward into room straight ahead.
  – blocked by walls.
  – Eaten by a live Wumpus immediately.
  – Fall into pit immediately.
● Grab an object in current room.
● Shoot arrow in straight line (only allowed once).

Sensor:
● Agent can smell, detect a breeze, see gold glitter, detect a bump (into a wall) and hear the Wumpus scream when it is killed.
● Sensors:

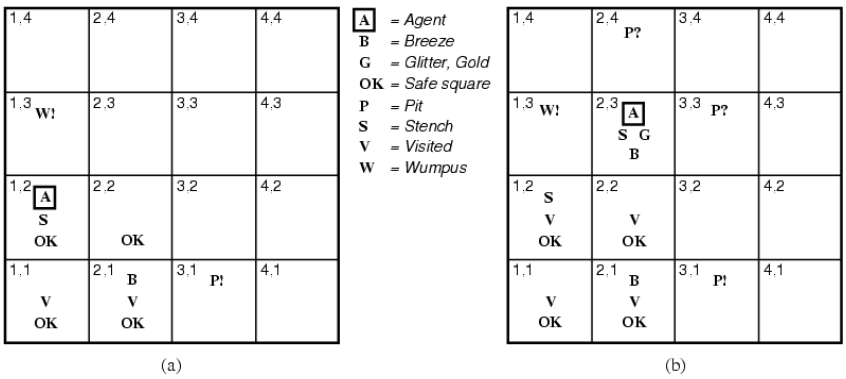
– *Stench*: a neighboring room holds the Wumpus.
– *Breeze*: a neighboring room holds a pit.
– Glitter: The room the agent is in has some gold.
– *Bump*: The agent just banged into a wall.
– *Scream*: The Wumpus just died (shot with arrow).

Based on sensors agent get following percept: **Percept (Breeze, Stench, Glitter, Bump, Scream).**

The Wumpus agent's first step



Later



Percept Sequence:
Percept (1,1)=(None, None, None, None, None,)
Percept (2,1)=(**Breeze**, None, None, None, None,)
Percept (1,1)=(None, None, None, None, None,)
Percept (1,2)=(None, **Stench**, None, None, None,)
Percept (2,2)=(None, None, None, None, None,)
Percept (3,2)=(**Breeze**, None, None, None, None,)
Percept (2,3)=(None, None, **Glitter**, None, None,)

Propositional Logic (PL)

- A simple language that is useful for showing key ideas and definitions
- User defines a set of propositional **symbols**, like *P* and *Q*. User define the semantics of each of these symbols. For example,
  - P means "It is hot"
  - Q means "It is humid"
  - R means "It is raining"
- A **sentence** (also called a formula or well-formed formula or wff) is defined as:
  1. A symbol
  2. If S is a sentence, then ~S is a sentence, where "~" is the "not" logical operator
  3. If S and T are sentences, then (S v T), (S ^ T), (S => T), and (S <=> T) are sentences, where the four logical connectives correspond to "or," "and," "implies," and "if and only if," respectively
  4. A finite number of applications of (1)-(3)
- Examples of PL sentences:
  - (P ^ Q) => R (here meaning "If it is hot and humid, then it is raining")
  - Q => P (here meaning "If it is humid, then it is hot")
  - Q (here meaning "It is humid.")
- Given the truth values of all of the constituent symbols in a sentence, that sentence can be "evaluated" to determine its truth value (True or False). This is called an **interpretation** of the sentence.
- A **model** is an interpretation (i.e., an assignment of truth values to symbols) of a set of sentences such that each sentence is True. A model is just a formal mathematical structure that "stands in" for the world.
- A **valid** sentence (also called a **tautology**) is a sentence that is True under *all* interpretations. Hence, no matter what the world is actually like or what the semantics is, the sentence is True. For example "It's raining or it's not raining."
- An **inconsistent** sentence (also called **unsatisfiable** or a **contradiction**) is a sentence that is False under *all* interpretations. Hence the world is never like what it describes. For example, "It's raining and it's not raining."

Example:

Q.1) consider following set of facts.

I. **Rani is hungry.**
II. **If rani is hungry she barks.**
III. **If rani is barking then raja is angry.**

Convert into proposition logic statements.

Sol: step1: we can use following propositional symbols

P: Rani is hungry

Q: Rani is Barking

R: Raja is Angry.

Step2: The propositional logic statements are,

I. P
II. P=>Q
III. Q=>R

Predicate Logic:

First-Order Logic or first order Predicate Logic (FOL or FOPL) Syntax

- User defines these primitives:
  - **Constant symbols** (i.e., the "individuals" in the world) E.g., Mary, 3
  - **Function symbols** (mapping individuals to individuals)
    E.g., father-of(Mary) = John, color-of(Sky) = Blue
  - **Predicate symbols** (mapping from individuals to truth values) E.g., greater(5,3), green(Grass), color(Grass, Green)

- FOL supplies these primitives:
  - **Variable symbols**. E.g., x, y
  - **Connectives**. Same as in PL: not (~), and (^), or (v), implies (=>), if and only if (<=>)
  - **Quantifiers**: Universal (∀) and Existential (∃)
    - Universal quantification corresponds to conjunction ("and") in that (Ax):P(x) means that P holds for all values of x in the domain associated with that variable.

      E.g., (∀x) dolphin(x) => mammal(x)

    - Existential quantification corresponds to disjunction ("or") in that (∃x)P(x) means that P holds for some value of x in the domain associated with that variable.

      E.g., (∃x) mammal(x) ^ lays-eggs(x)

    - Universal quantifiers usually used with "implies" to form "if-then rules."

      E.g., (∀x) cs540-student(x) => smart(x) means "All cs540 students are smart.

      "You rarely use universal quantification to make blanket statements about every individual in the world: (∀x)cs540-student(x) ^ smart(x) meaning that everyone in the world is a cs540 student and is smart.

    - Existential quantifiers usually used with "and" to specify a list of properties or facts about an individual.

      E.g., (∃x) cs540-student(x) ^ smart(x) means "there is a cs540 student who is smart."

      A common mistake is to represent this English sentence as the FOL sentence:

      (∃x) cs540-student(x) => smart(x) But consider what happens when there is a person who is NOT a cs540-student.

    - Switching the order of universal quantifiers does not change the meaning:
      (∀x)(∀y)P(x,y) is logically equivalent to (∀y)(∀x)P(x,y). Similarly, you can switch the order of existential quantifiers.
    - Switching the order of universals and existential *does* change meaning:

- Everyone likes someone: (∀x):(∃y):likes(x,y)
- Someone is liked by everyone: (∃y)(∀x)likes(x,y)

Translating English to FOL

- Every gardener likes the sun.
  (∀x) gardener(x) => likes(x,Sun)
- You can fool some of the people all of the time.
  (∃x) (person(x) ^ (At)(time(t) => can-fool(x,t)))
- You can fool all of the people some of the time.
  (∀x) (person(x) => (Et) (time(t) ^ can-fool(x,t)))
- All purple mushrooms are poisonous.
  (∀x) (mushroom(x) ^ purple(x)) => poisonous(x)
- No purple mushroom is poisonous.
  ~(∃x) purple(x) ^ mushroom(x) ^ poisonous(x)
  or, equivalently,
  (∀x) (mushroom(x) ^ purple(x)) => ~poisonous(x)
- There are exactly two purple mushrooms.
  (∃x)(∃y) mushroom(x) ^ purple(x) ^ mushroom(y) ^ purple(y) ^ ~(x=y) ^ (∀z) (mushroom(z) ^ purple(z)) => ((x=z) v (y=z))
- Deb is not tall.
  ~tall(Deb)
- X is above Y if X is on directly on top of Y or else there is a pile of one or more other objects directly on top of one another starting with X and ending with Y.
  (∀x)(∀y) above(x,y) <=> (on(x,y) v (∃z) (on(x,z) ^ above(z,y)))

Resolution

It is a mechanism to infer some fact or to reach to some conclusion using logic.in resolution to prove some fact true we resolve that the negation of that fact is not true.

Following steps are in resolution;

1) Convert English statements to either propositional logic or predicate logic statements.
2) Convert logical statement to CNF (Conjunctive Normal Form).
3) Negate the conclusion.
4) Resolve the negation of conclusion is not true using resolution tree.

Conjunctive Normal Form (CNF)

Following are the steps to convert logic into CNF.
In CNF the fact will be connected only with 'V' (conjunctive).If we eliminate implication (=>), Universal (∀), and Existential (∃), then the statement is converted to CNF.

1) **Eliminate implication '=>'**

   - a=>b = ~aVb
   - (a∧b)=>c =~(a∧b)Vc

2) Eliminate '∧'

- a∧b=a
  b
- a∧b∧c=a
  b
  ~c
- a∧(bVc) = a
  bVc
- aV(b∧c) = aVb
  aVc

3) Eliminate '∃'

We can eliminate ∃ by substituting for the variable a reference to a function that produces a desired value.

Eg.There is a teacher

∃x: Teacher(x) ,then by eliminating '∃' we get,

"Teacher (s1)" where s1 is a function that somehow produces a value that satisfies the predicate teacher.

4) **Eliminate '∀'**

To eliminate '∀' convert the fact into prefix normal form in which all the universal quantifiers are at the beginning of formula.

Eg. All students are intelligent. ∀x: Student(x)->Intelligent(x)

After eliminating ∀x we get,

Student(x)->Intelligent(x).

Problems based on Resolution. (Refer class notes)

Forward Reasoning/Chaining

Forward chaining is one of the two main methods of reasoning when using inference rules (in artificial intelligence) and can be described logically as repeated application of *modus ponens*.

Forward chaining starts with the available data and uses inference rules to extract more data (from an end user for example) until a goal is reached. An inference engine using forward chaining searches the inference rules until it finds one where the antecedent (**If** clause) is known to be true. When found it can conclude, or infer, the consequent (**Then** clause), resulting in the addition of new information to its data.

Inference engines will iterate through this process until a goal is reached.

For example, suppose that the goal is to conclude the color of a pet named Fritz, given that he croaks and eats flies, and that the rule base contains the following four rules:

1. **If** X croaks and eats flies - **Then** X is a frog
2. **If** X chirps and sings - **Then** X is a canary
3. **If** X is a frog - **Then** X is green
4. **If** X is a canary - **Then** X is yellow

(Only for Understanding)This rule base would be searched and the first rule would be selected, because its antecedent (**If** Fritz croaks and eats flies) matches our data. Now the consequent (**Then** X is a frog) is added to the data. The rule base is again searched and this time the third rule is selected, because its antecedent (**If** Fritz is a frog) matches our data that was just confirmed. Now the new consequent (**Then** Fritz is green) is added to our data. Nothing more can be inferred from this information, but we have now accomplished our goal of determining the color of Fritz.

Backward Reasoning/Chaining

Backward chaining (or **backward reasoning**) is an inference method that can be described (in lay terms) as working backward from the goal(s). Backward chaining is implemented in logic programming by SLD resolution. And can be described logically as repeated application of *modus tolens*.

Backward chaining starts with a list of goals (or a hypothesis) and works backwards from the consequent to the antecedent to see if there is data available that will support any of these consequents.[2] An inference engine using backward chaining would search the inference rules until it finds one which has a consequent (**Then** clause) that matches a desired goal. If the antecedent (**If** clause) of that rule is not known to be true, then it is added to the list of goals (in order for one's goal to be confirmed one must also provide data that confirms this new rule).

For example, suppose that the goal is to conclude the color of my pet Fritz, given that he croaks and eats flies, and that the rule base contains the following four rules:

An Example of Backward Chaining.

1. **If** X croaks and eats flies – **Then** X is a frog
2. **If** X chirps and sings – **Then** X is a canary
3. **If** X is a frog – **Then** X is green
4. **If** X is a canary – **Then** X is yellow

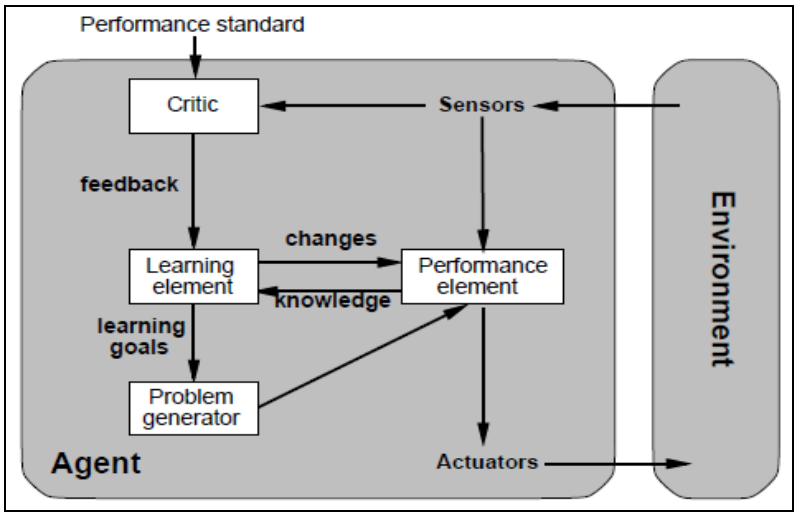
(Only for Understanding)This rule base would be searched and the third and fourth rules would be selected, because their consequents (**Then** Fritz is green, **Then** Fritz is yellow) match the goal (to determine Fritz's color). It is not yet known that Fritz is a frog, so both the antecedents (**If** Fritz is a frog, **If** Fritz is a canary) are added to the goal list. The rule base is again searched and this time the first two rules are selected, because their consequents (**Then** X is a frog, **Then** X is a canary) match the new goals that were just added to the list. The antecedent (**If** Fritz croaks and eats flies) is known to be true and therefore it can be concluded that Fritz is a frog, and not a canary. The goal of determining Fritz's color is now achieved (Fritz is green if he is a frog, and yellow if he is a canary, but he is a frog since he croaks and eats flies; therefore, Fritz is green).

7. Learning

Learning denotes changes in a system that enable the system to do the same task more efficiently next time.
Learning is an important feature of Intelligence".
Learning is a branch of artificial intelligence, is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases.

General model of Learning Agent:

Learning agents



- Learning has an advantage that it allows the agents to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow.
- 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 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 performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.
- 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.

Forms of Learning:

Supervised learning:

- Supervision: The data (observations, measurements, etc.) are labeled with pre-defined classes. It is like that a "teacher" gives the classes (supervision).

- Test data are classified into these classes too.

Unsupervised learning:

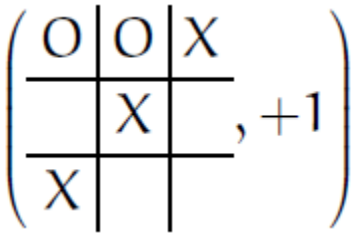
- Unsupervised learning (clustering)
  - Class labels of the data are unknown
  - Given a set of data, the task is to establish the existence of classes or clusters in the data.

Reinforcement Learning

Learning from feedback (+ve or -ve reward) given at end of a sequence of steps. Unlike supervised learning, the reinforcement learning takes place in an environment where the agent cannot directly compare the results of its action to a desired result. Instead, it is given some reward or punishment that relates to its actions. It may win or lose a game, or be told it has made a good move or a poor one. The job of reinforcement learning is to find a successful function using these rewards.

Inductive Learning:

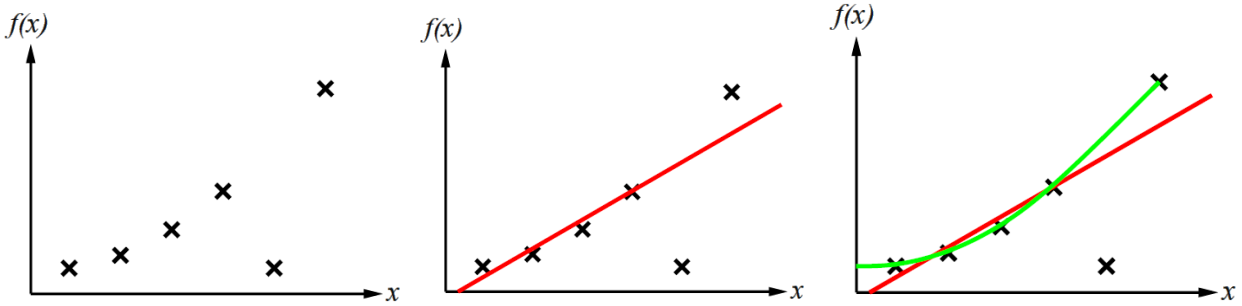
- Simplest form: learn a function from examples
  - f is the target function
  - An example is a pair (x; f(x)), e.g.
  - Problem: find a hypothesis h such that h _ f, given training set of examples
- Highly simplified model of real learning:
  - Ignores prior knowledge
  - Assumes a deterministic, observable environment
  - Assumes examples are given
  - Assumes that the agent wants to learn f



Inductive Learning Method

Construct/adjust h to agree with f on training set
h is consistent if it agrees with f on all examples

Curve fitting example:

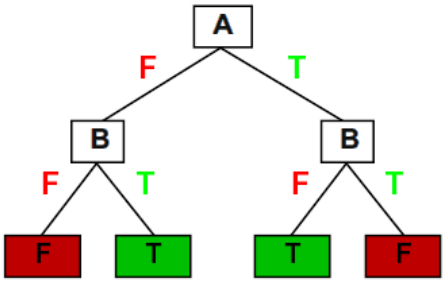


Decision Tree

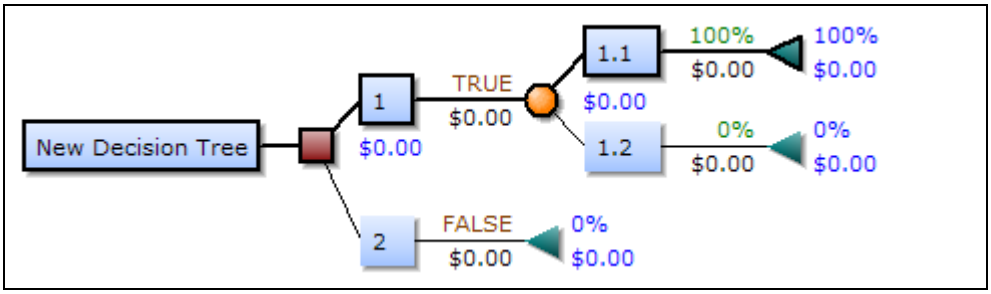
- A **decision tree** is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.
- It is one way to display an algorithm.
- Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal.
- Another use of decision trees is as a descriptive means for calculating conditional probabilities.
- Decision trees can express any function of the input attributes.

E.g. for Boolean features, truth table row is path to leaf

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- In decision analysis, a "decision tree" is used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.
- A decision tree consists of 3 types of nodes:-
    1. Decision nodes - commonly represented by squares
    2. Chance nodes - represented by circles
    3. End nodes - represented by triangles



8. Uncertain Knowledge and Reasoning

Uncertainty:

Many times in complex world theory of agent and events in environment are contradicted to each other, and this result in reduction of performance measure. E.g. "let agent's job is to leave the passenger on tome, before the flight departs. But agent knows the problems it can face during journey. Means Traffic, flat tire, or accident. In these cases agent cannot give its full performance. "This is called as uncertainty.

Probability:

⇒Objective probability
- Averages over repeated experiments of random events
  – E.g. estimate P (Rain) from historical observation
- Makes assertions about future experiments
- New evidence changes the reference class

⇒Subjective / Bayesian probability
- Degrees of belief about unobserved event: state of knowledge

– E.g. agent's belief that it will rain, given the season
- Estimate probabilities from past experience
- New evidence updates beliefs

⇒Fuzzy logic handles degrees of truth, not uncertainty
  – Wet (Grass) is true to degree 0:2
- Fuzzy sets: degree of membership—rough vs. crisp (usual) sets

Probability Basics

Priori probability

The prior probability of an event is the probability of the event computed before the collection of new data. One begins with a prior probability of an event and revises it in the light of new data. For example, if 0.01 of a population has schizophrenia then the probability that a person drawn at random would have schizophrenia is 0.01. This is the prior probability. If you then learn that that there score on a personality test suggests the person is schizophrenic, you would adjust your probability accordingly. The adjusted probability is the posterior probability.

Bayes' Theorem:

Bayes' theorem considers both the prior probability of an event and the diagnostic value of a test to determine the posterior probability of the event. The theorem is shown below:

$$P(D \mid T) = \frac{P(T \mid D)P(D)}{P(T \mid D)P(D) + P(T \mid D')P(D')}$$

where P(D|T) is the posterior probability of Diagnosis D given Test result T, P(T|D) is the conditional probability of T given D, P(D) is the prior probability of D, P(T|D') is the conditional probability of T given not D, and P(D') is the probability of not D'.

Conditional Probability Formulae

Definition of conditional probability
P(a|b) =P(a ^ b)/P(b) if P(b) 6= 0

– Probability of observing event a given evidence (knowledge / observation) of event b.

Bayesian Networks (IMP)

⇒ A simple, graphical notation for conditional independence assertions
- Compact specification of full joint distributions

⇒ Syntax
- a set of nodes, one per variable Xl
- a directed, acyclic graph (link _ "directly influences")
- a conditional probability distribution (CPD) for each node given its parents.
  P(Xi|Parents(Xi))

⇒ Simplest case: CPD is a conditional probability table (CPT)
- Giving distribution over Xi for each combination of parent values.

Example of Bayesian network

Suppose that there are two events which could cause grass to be wet: either the sprinkler is on or it's raining. Also, suppose that the rain has a direct effect on the use of the sprinkler (namely that when it rains, the sprinkler is usually not turned on). Then the situation can be modeled with a Bayesian network (shown). All three variables have two possible values, T (for true) and F (for false).

The joint probability function is:

P(*G*,*S*,*R*) = P(*G* | *S*,*R*)P(*S* | *R*)P(*R*)

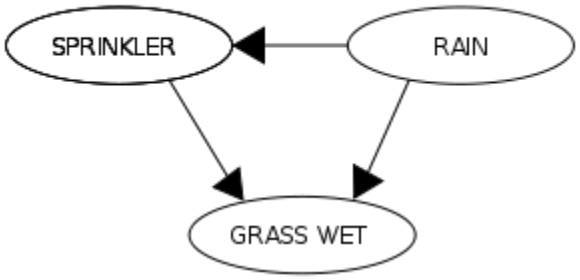
where the names of the variables have been abbreviated to *G = Grass wet*, *S = Sprinkler*, and *R = Rain*.

The model can answer questions like "What is the probability that it is raining, given the grass is wet?" by using the conditional probability formula and summing over all nuisance variables:

$$\mathrm{P}(R = T \mid G = T) = \frac{\mathrm{P}(G = T, R = T)}{\mathrm{P}(G = T)} = \frac{\sum_{S \in \{T,F\}} \mathrm{P}(G = T, S, R = T)}{\sum_{S,R \in \{T,F\}} \mathrm{P}(G = T, S, R)}$$

$$= \frac{(0.99 \times 0.01 \times 0.2 = 0.00198_{TTT}) + (0.8 \times 0.99 \times 0.2 = 0.1584_{TFT})}{0.00198_{TTT} + 0.288_{TTF} + 0.1584_{TFT} + 0_{TFF}} \approx 35.77\%.$$

	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99



	RAIN	
T	F	
0.2	0.8	

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

9. Planning

Planning is a key ability for intelligent systems, increasing their autonomy and flexibility through the construction of sequences of actions to achieve their goals. It has been an area of research in artificial intelligence for over three decades. Planning techniques have been applied in a variety of tasks including robotics, process planning, web-based information gathering, and autonomous agents and spacecraft mission control.

A Simple Planning Agent:

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

Problem Solving Agents + Knowledge-based Agents = Planning Agents

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

Algorithm of a simple planning agent:

1. Generate a goal to achieve
2. Construct a plan to achieve goal from current state
3. Execute plan until finished
4. Begin again with new goal

The agent first generates a goal to achieve, and then constructs a plan to achieve it from the current state. Once it has a plan, it keeps executing it until the plan is finished, then begins again with a new goal.

Logic Based Planning (only for understanding)

Situation Calculus

Situation calculus is a version of first-order-logic (FOL) that is augmented so that it can reason about actions in time.

- Add *situation variables* to specify time. A **situation** is a snapshot of the world at an interval of time when nothing changes
- Add a special predicate *holds(f,s)* that means "f is true in situation s"
- Add a function *result(a,s)* that maps the current situation *s* into a new situation as a result of performing action *a*.

Example:

The action *"agent-walks-to-location-y"* could be represented by:

(Ax)(Ay)(As)(at(Agent,x,s) -> at(Agent,y,result(walk(y),s))

9.2.1.1 Frame Problem

Actions in Situation Calculus describe what they *change*, not what they *don't change*. So, how do we know what's still true in the new situation? To fix this problem we add a set of frame axioms that explicitly state what doesn't change.

Example:

When the agent walks to a location, the locations of most other objects in the world do not change. So for each object (i.e. a bunch of bananas), add an axiom like:
(Ax)(Ay)(As)(at(Bananas,x,s) -> at(Bananas,x,result(walk(y),s))

Solving planning problems using situation calculus

Using situation calculus a planning algorithm is represented by logical sentences that describe the three main parts of a problem.

1. Initial State: a logical sentence of a situation So. For the shopping problem it could be:

   At(Home,So) /\ -Have(Milk,So) /\ -Have(Bananas,So) /\ -Have(Drill,So)

2. Goal State: a logical query asking for a suitable situation. For the shopping problem, a query is:

   $ s At(Home,s) /\ Have(Milk,s) /\ Have(Bananas,s) /\ Have(Drill,s)

3. Operators: a set of descriptions of actions.

   Ex. A successor-state action with the Buy(Milk) action
   " a,s Have(Milk,Result(a,s)) <=> [(a = Buy(milk) /\ At(supermarket,s) \/ (Have(Milk,s) /\ a NOT= Drop(Milk))]

Metric Path Planning

- Objective: determine a path to a specified goal
- Metric methods:
  - Tend to favor techniques that produce an optimal path
  - Usually decompose path into subgoals called waypoints
- Two components to metric methods for path planning:
  - Representation (i.e., data structure)
  - Algorithm

Configuration Space

- Configuration Space (abbreviated: “Cspace”):
  - Data structure that allows robot to specify position and orientation of objects and robot in the environment
  - “Good Cspace”: Reduces # of dimensions that a planner has to deal with
  - Typically, for indoor mobile robots:
    - Assume 2 DOF for representation
    - Assume robot is round, so that orientation doesn’t matter
    - Assumes robot is holonomic (i.e., it can turn in place)
      - (Although there is much research dealing with path planning in nonholonomic robots)
  - Typically represents “occupied” and “free” space
    - “Occupied” _object is in that space
    - “Free” _space where robot is free to move without hitting any modeled object.

Metric Maps use Cspace

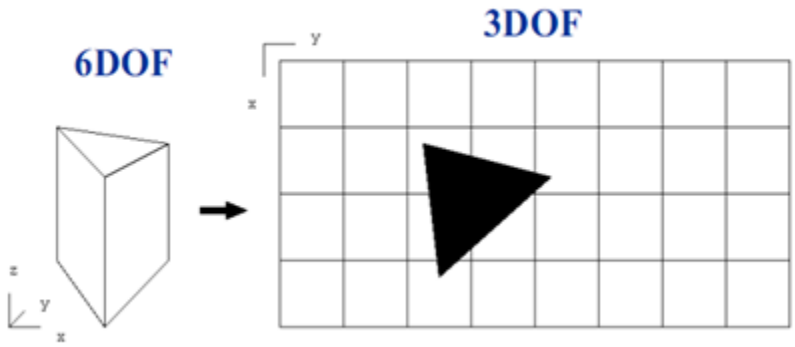
World Space: physical space robots and obstacles exist in

– In order to use, generally need to know (x,y,z) plus Euler angles: 6DOF

• Ex. Travel by car, what to do next depends on where you are and what direction you're currently heading

• Configuration Space (Cspace)

– Transform space into a representation suitable for robots, simplifying assumptions.



Major Cspace Representations

• Idea: reduce physical space to a Cspace representation which is more amenable for storage in computers and for rapid execution of algorithms

• Major types

–Meadow Maps

–Generalized Voronoi Graphs (GVG)

–Regular grids, quadtree

Meadow Maps (Hybrid Vertex-graph Free-space)

• Transform space into convex polygons

– Polygons represent safe regions for robot to traverse

• Important property of convex polygons:

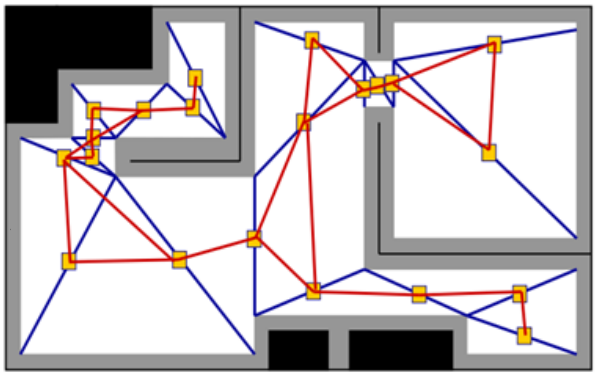
– If robot starts on perimeter and goes in a straight line to any other point on the perimeter, it will not go outside the polygon

• Path planning:

– Involves selecting the best series of polygons to transit through

Example Meadow Map

1. Grow objects
2. Construct convex polygons
3. Mark midpoints; these become graph nodes for path planner
4. Path planner plans path based upon new graph

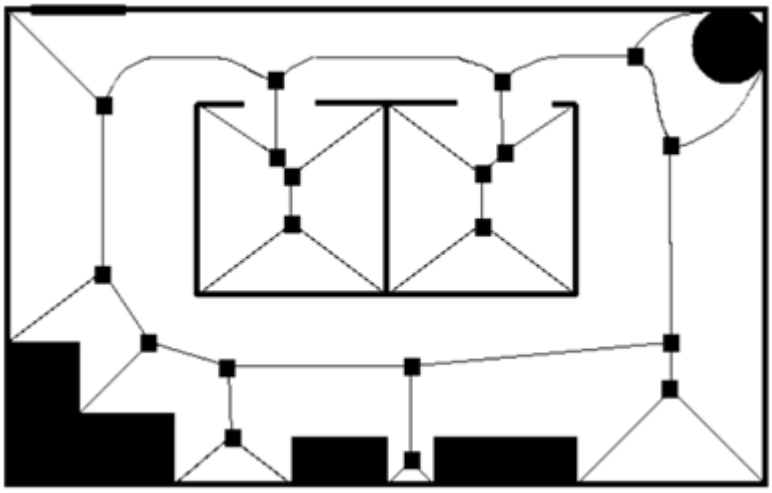


Generalized Voronoi Diagrams (GVGs)

• GVGs:
– Popular mechanism for representing Cspace and generating a graph
– Can be constructed as robot enters new environment
• Basic GVG approach:
–Generate a Voronoi edge, which is equidistant from all points
– Point where Voronoi edge meets is called a Voronoi vertex
– Note: vertices often have physical correspondence to aspects of environment that can be sensed
– If robot follows Voronoi edge, it won't collide with any modeled obstacles don't need to grow obstacle boundaries

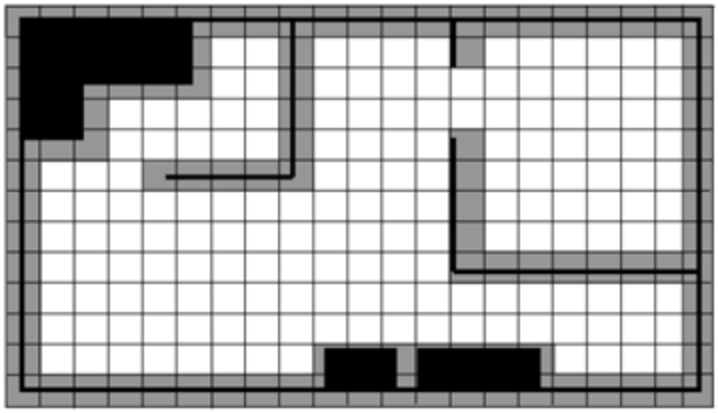
Example Generalized Voronoi Graph (GVG)

• Imagine a fire starting at the boundaries, creating a line where they intersect. Intersections of lines are nodes.
• Result is a relational graph.



Regular Grids / Occupancy Grids

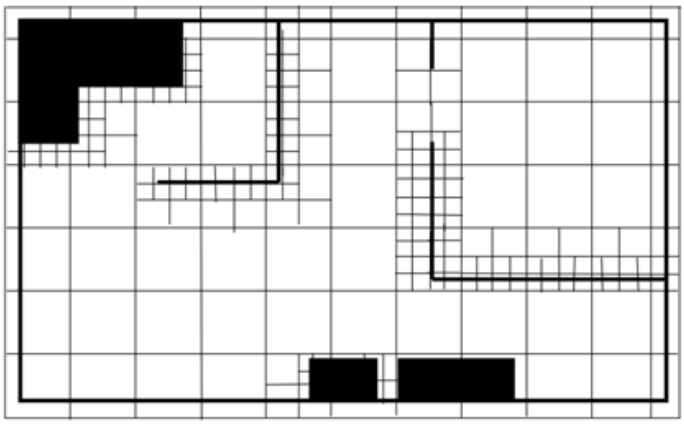
• Superimposes a 2D Cartesian grid on the world space (bigger than pixels, but same idea)
• If there is any object in the area contained by a grid element, that element is marked as occupied
• Center of each element in grid becomes a node, leading to highly connected graph
• Grid nodes are connected to neighbors (either 4-connected or 8-connected).



Quadtrees

- Representation starts with large area (e.g., 8x8 inches)
- If object falls into part of grid, but not all of grid, space is subdivided into for smaller grids
- If object doesn't fit into sub-element, continue recursive subdivision
- 3D version of Quadtree – called an Octree.

Example Quadtree Representation
(Not all cells are subdivided as in an actual quadtree representation (too much work for a drawing by hand!, but this gives basic idea)

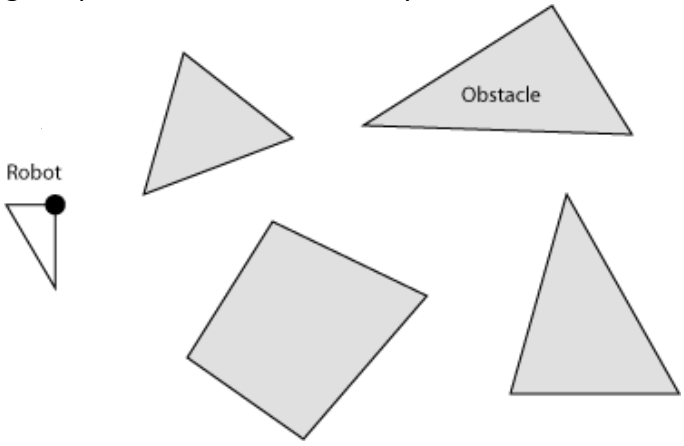


Graph Based Planners

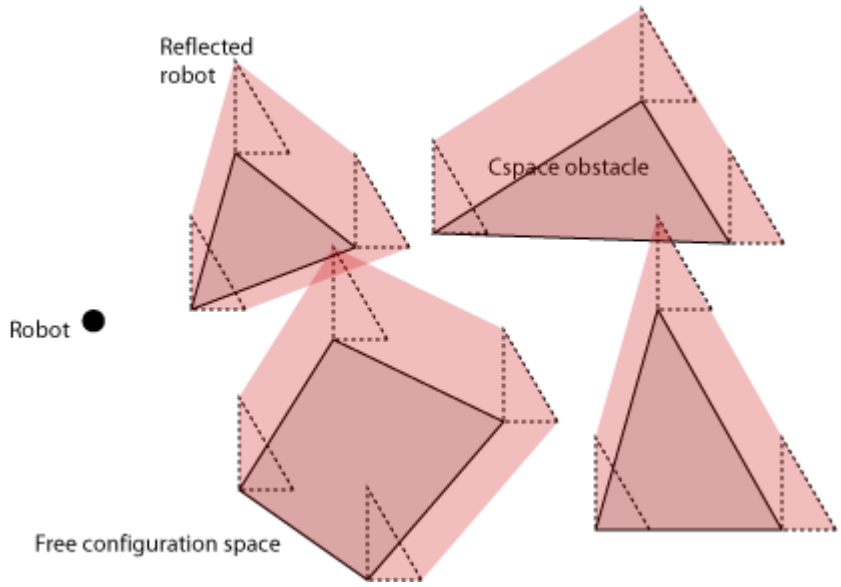
- Finding path between initial node and goal node can be done using graph search algorithms
- Graph search algorithms: found in networks, routing problems, etc.
- However, many graph search algorithms require visiting each node in graph to determine shortest path
  – Computationally tractable for sparsely connected graph (e.g., Voronoi diagram)
  – Computationally expensive for highly connected graph (e.g., regular grid)
- Therefore, interest is in "branch and bound" search
  – Prunes off paths that aren't optimal
- Classic approach: A* ("A Star") search algorithm
  – Frequently used for holonomic robots

Planar translation of rigid bodies(only for understanding)

Here is a triangular planar robot that can only translate, and some polygonal obstacles:



We will describe the configuration of the robot by the location of a reference point attached to the robot. The configuration space is therefore $\mathbf{R}^2$. As we drag the robot around by its reference point, there will be intersections between the robot and the obstacles. We mark those (*x*,*y*) configurations as collisions in the configuration space, as shown in pink below.



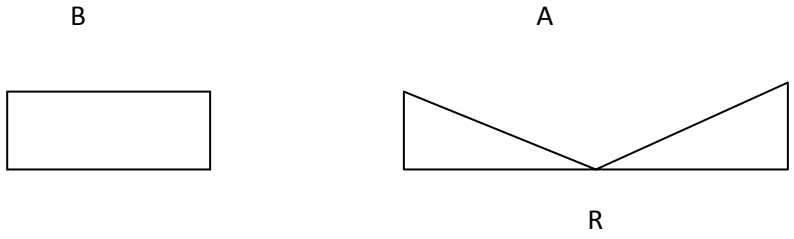
There's a pattern to the set of points where there are collisions, and we can use this pattern to derive a graphical method for constructing the configuration space obstacles. Imagine an obstacle that is just a point. For what configurations of the robot does that particular point collide with the robot? Slide the robot edges around the point, marking the location of the reference point as you slide. You'll find that you get a shape in configuration space that looks like a flipped version of the robot -- a reflection of the robot in *x* and *y* across horizontal and vertical lines through the reference point.

Notice that the boundaries of the polygonal obstacles are unions of point obstacles. The configuration space obstacles are therefore unions of flipped robots. So, to construct the configuration space obstacles, flip the robot, drag it around the boundary of all the physical obstacles, and that will give you the shapes of the configuration space obstacles.

It's not surprising that the c-space obstacles are larger than the original physical obstacles -- the robot has 'shrunk' to a point.

Example:

Q) Draw the Configuration space induced by the translation of the concave mobile part A other the fixed obstacle B



10. Introduction and overview of Robotics paradigms

Robotic Paradigms

- *A paradigm is a philosophy or a set of assumptions and/or techniques which characterize an approach to a class of problems.* Robotic paradigms are defined by the relationship among 3 basic primitives: **sense, plan,** and **act.** Other was to define the paradigm is to look how sensory data is distributed and processed.
- **Hierarchical Paradigm :**
  - oldest paradigm
  - heavy planning, a top-down paradigm
  - in each step, robot make plans
  - robot construct a global world model, which has a *closed-world* assumption and suffers from *frame problem*.
- **Reactive Paradigm:**
  - Popular mainly from 1988-92, since then hybrid model was used more.
  - Two main trends among researchers :
    - They investigate biological and cognitive sciences for living creature behaviors,
    - Decrease in cost, fast execution time.
  - Behaviors are composed of **sense->act** couplings and behaviors combine for the actions.
  - Throwing away **planning** phase was not good for general purpose robots.
- **Hybrid Deliberative/Reactive Paradigm:**
  - Task is divided into subtasks, behaviors are determined. **plan-> sense-act**

DESIGNING A REACTIVE IMPLEMENTATION

Overview

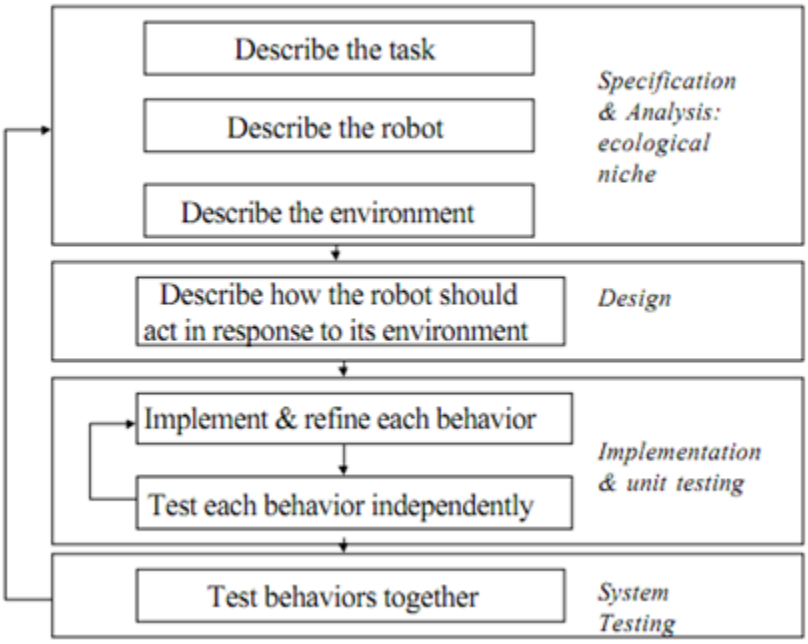
- Reactive robotics is frustrated from 2 important deficits:
  - Design of behaviors is more an art than science.
  - Integration of well-designed behaviors is a very challenging subject. Emergent behaviors etc.<>
- Many applications are best thought as *a **sequence of behaviors.** For example picking up and disposing a soda can.*
- In this chapter, OOP is introduced based on the schema theory.
- Establishing the ecological niche of a reactive robot is important.
- 2 techniques for managing behavior sequence: (Both are similar to IRM - Innate Releasing Mechanism)
  - finite state automata
  - *scripts*
- ***Program a small set of generic behaviors rather than designing a large set of specialized behaviors..***

Behaviors as Objects in OOP

- Schema theory is used as a bridge between concepts in biological intelligence and robotics.
- **Schema** is a class. **Perceptual schema**, **motor schema** and **behavior** are all **derived** from schema class.
- Schema class contains data structures and other schemas. Optionally, a *coordinated control program* exists, to control any methods of derived classes.

- **Behaviors** are composed of at least one perceptual schema, one motor schema. These classes act as **methods** for the Behavior class.
- **Perceptual schema** contains at least one method, which transforms sensor data into **percept** data structure.
- **Motor schema** contains at least one method which transforms **percept** data structure into a vector (action).
- Perceptual schema is linked to the sensors, while Motor schema is linked to actuators.
- **Behavior** may be composed of several Perceptual schemas, Motor schemas and even behaviors.
- *Primitive Behavior* is composed of only one PS and one MS.
- *Abstract Behavior* is assembled from other behaviors or have multiple PS and MS.
- ***Example : A primitive move-to-goal behavior***
    - The objective is to place right colored trashes into right colored bins.
    - *move-to-goal(goal-color)* is the basic behavior. [Better than *move-to-red* and *move-to-blue*]
    - perceptual schema is *extract-goal (goal-color)*. - A Method
    - motor schema is pfields.attraction(goal-angle, goal-strength). - A Method
    - percept data structure has *goal-angle* and *goal-strength* fields.
    - **Affordance** of color is used to extract where the goal is in the image.
    - The affordance of the Coke is the color while the information extracted from perception is the angle and size. This information (percept) is given to motor schema to move.
- ***Example: An abstract follow-corridor behavior****:* Different class diagrams and structures to create same behavior.
- ***Where do releasers go in OOP?***
    - Perception serves two purposes: to release a behavior and to guide it.
    - The releaser itself may be a perceptual schema and behavior is instantiated when releaser condition is satisfied OR the behavior is always active and when releaser condition is not satisfied coordinated control program short-circuits processing.
    - **The releaser must be designed to support correct sequence.**

<u>Steps in Designing a Reactive Behavior</u>



1. **Describe the task**. The purpose of this step is to specify what the robot has to do to be successful.
2. **Describe the robot.** The purpose of this step is to determine the basic physical abilities of the robot and any limitations. The designer is usually handed some fixed constraints on the robot platform which will impact the design.
3. **Describe the environment.**
   This step is critical for two reasons.
   - First, it is a key factor in determining the situatedness of the robot.
   - Second, it identifies perceptual opportunities for the behaviors, both in how a perceptual event will instantiate a new behavior, and in how the perceptual schema for a behavior will function.
4. **Describe how the robot should act in response to its environment** (behavioral decomposition ecological niche).
   - Ideally, testing occurs in simulation prior to testing on the robot in its environment
   - **Behavior table** is constructed which includes, *releaser, Behavior name, Motor Schema, Percept,* and *Perceptual Schema*
5. **Implement & refine each behavior**. By this point, the designer has an overall idea of the organization of the reactive system and what the activities are. This step concentrates on the design of each individual behavior. As the designer constructs the underlying algorithms for the motor and perceptual schemas, it is important to be sure to consider both the normal range of environmental conditions the robot is expected to operate in (e.g., the steady-state case) and when the behavior will fail.
6. **Test each behavior independently**.
   - It is important to remember that **the simulators often only model the mechanics of the robot, not the perceptual abilities.**
7. **Test behaviors together.** The final step of designing and implementing a reactive system is to perform integration testing, where the behaviors are combined. This also includes testing the behaviors in the actual environment.

In most simulation environments, motor schema is simulated correctly; however the simulation of perception and sensing mechanisms is not realistic.

Case Study: Unmanned Ground Robotics Competition

- In this section, the design steps of reactive behavior are realistically implemented.

Assemblages of Behaviors

- Concurrent run and in a sequence, how to formally define? The coordinated control program member of the *abstract behavior* expresses the releaser for the component behaviors.
- Most important point to remember is **to make the world *release***, rather than rely on internal model.
- ***Finite State Automata :***
  - states
  - inputs or alphabets : releasers
  - transition defines the new state due to current state and releaser
  - Obstacle avoidance should be always on and is implicit in FSA.
- ***A Pick Up the Trash FSA***
  - Behavior table does not show the sequence and control of behaviors, so FSA can be used.
  - A common mistake is done when all releasers are not shown in FSA for the sake of previous state should have that releaser.
  - It is difficult to show the concurrency of behaviors using FSA.

- ***Implementation Examples***
    - Schema-theoretic implementations use potential field methodologies and vector summation for effector's control.
    - Advantage of FSA is that they are abstract and can be implemented in a number of ways.
    - Inhibition and suppression cannot be well represented using FSA.
- Sometimes, robots fail in their task, these events are called *exceptions.* Another behavior or other parameters can be substituted in such situations.