- ***Mathematical:***

α	0°	45°	90°	135°	180°	360°
$sin\alpha$	0	$\frac{\sqrt{2}}{2}$	1	$\frac{\sqrt{2}}{2}$	0	0
$cos\alpha$	1	$\frac{\sqrt{2}}{2}$	0	$-\frac{\sqrt{2}}{2}$	-1	1

$sin^2\theta + cos^2\theta = 1$; $\sin(\theta_1 + \theta_2) = sin\theta_1 cos\theta_2 + cos\theta_1 sin\theta_2$; $\cos(\theta_1 + \theta_2) = cos\theta_1 cos\theta_2 - sin\theta_1 sin\theta_2$

Vectors:

Dot product: $a \cdot b = a_1b_1 + a_2b_2 + \cdots + a_nb_n$; Cross product: $a \times b = (y_1z_2 - y_2z_1)i - (x_1z_2 - x_2z_1)j + (x_1y_2 - x_2y_1)k$

- ***Bug Algorithm:***

Assume only local knowledge of environment and a global goal with essentially tactile sensing

Behaviors: Boundary follow / motion to goal

Bug 1:

- Towards to goal, remember closest point if encounter with obstacle, return the closest point and continue
- Boundary: $D + 1.5\sum P$. Worst case to travel half of boundary of obstacle to find closest point

Bug 2:

- Along m-line intersect with obstacle, follow obstacle until encounter the m-line again closer to the goal
- Boundary: $D + 0.5\sum n_i P_i$. n_i is the number of intersection with ith obstacle. Worst case to have $\frac{n}{2}$ inner loop

Bug 1 vs Bug 2:

- Bug 1 is an exhausting algorithm, while Bug 2 is a greedy algorithm. Bug 1 is safe and reliable, Bug 2 better in some case
- As the intersection number between m-line and obstacle increasing, the Bug 1 is likely to outperform than Bug 2

Tangent Bug:

- Improvement to the Bug 2 algorithm determines a shorter path to the goal using a range sensor with finite distance sensing
- Motion-to-goal: Move in straight line to goal until sense obstacle, move to an intermediate point O_i according to some distance until reach the goal or a minimum M_i in which case switch to boundary following
- Boundary-following: define d_{reach} (shortest distance between obstacle and goal) and $d_{following}$ (shortest distance between sensed boundary and the goal) and continue move around the obstacle till $d_{reach} < d_{following}$. Then switch to motion to goal
- Issue: the robot does not know the obstacles boundary a priori

- ***Representation:***

Boundary point: represent the object by specifying details about points on the boundary

Primitive: Use primitives(mathematical functions) for defining objects. e.g. Circle - $x^2 + y^2 \le 1$

Polygons and polyhedral use linear primitives (e.g., half spaces)

A generalization of polygons and polyhedral that uses **nonlinear primitives**

Boundary representation are composed of two parts: topology and geometry(surface, curves and points). The main topological items are: faces(bounded portion of a surface), edge(a bounded piece of a curve) and vertices(lies at a point)

A boundary representation of a nonconvex polygon may be directly encoded by listing the vertices a specific order.

- Rigid Body Transformation:

Only change **position** and **rotation**, NOT change shape

Rigid body translation: $h(x, y) = (x + x_t, y + y_t)$

2D rotation: $R(\theta) = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$ Rotate followed by translate: $T = R(\theta) + t = \begin{bmatrix} cos\theta & -sin\theta & x_t \\ sin\theta & cos\theta & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

2D rotation about an arbitrary point: $T = T(x, y) * R(\theta) * T(-x, -y) = \begin{bmatrix} cos\theta & -sin\theta & x_r(1 - cos\theta) + y_r sin\theta \\ sin\theta & cos\theta & y_r(1 - cos\theta) - x_r sin\theta \\ 0 & 0 & 1 \end{bmatrix}$

Translate by $T(-x, -y)$, Rotate by θ, then translate by $T(x, y)$

Rigid body translation in 3D: $h(x, y, z) = (x + x_t, y + y_t, z + z_t)$

3D rotation: There are 3 axis of rotations: x, y, z. Each rotation is **counter clockwise**

- Commonly used is yaw-pitch-roll referring to rotation about z, y, x axis

Yaw: $R_z(\alpha) = \begin{pmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$ Pitch: $R_y(\beta) = \begin{pmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{pmatrix}$ Roll: $R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\gamma & -sin\gamma \\ 0 & sin\gamma & cos\gamma \end{pmatrix}$

Order is important: Rotate γ about x axis, rotate β about y axis, rotate α about z axis:

$$R(\alpha,\beta,\gamma) = R_z(\alpha)R_y(\beta)R_x(\gamma) = \begin{pmatrix} cos\alpha cos\beta & cos\alpha sin\beta sin\gamma - sin\alpha cos\gamma & cos\alpha sin\beta cos\gamma + sin\alpha sin\gamma \\ sin\alpha cos\beta & sin\alpha sin\beta sin\gamma + cos\alpha cos\gamma & sin\alpha sin\beta cos\gamma - cos\alpha sin\gamma \\ -sin\beta & cos\beta sin\gamma & cos\beta cos\gamma \end{pmatrix}$$

Euler angles: any rotation can be represented by no more than three rotations about coordinate axes, - no two successive rotations are about the same axis

Property of rotation matrix: rotation matrix R is an orthogonal matrix -> $R^T R = RR^T = I \rightarrow \det(R) = \pm 1$, In right hand coordinate: $\det(R) = 1$

Special Orthogonal matrices: $SO(n) = \{R \in R^{n\times n} | RR^T = I \wedge \det(R) = 1\}$

Gimbal lock: Problem of loss of a degree of freedom in 3D space. How to avoid: not to use gimbals entirely -> Use quaternion methods to derive orientation and velocity

- ***Quaternion:***
  - Something extend from complex number
  - $a + bi + cj + dk$, and $i^2 = -1, j^2 = -1, k^2 = -1$. So that: $ij = k, jk = i, ki = j, ji = -k, kj = -i, ik = -j$
  - Sum: $q_1 + q_2 = (a_1 + a_2,\ v_1 + v_2)$. Production: $q_1 q_2 = (a_1 a_2 - v_1 \cdot v_2,\ a_1 v_1 + a_2 v_2 + v_1 \times v_2)$. If vectors are 0, back to real number, otherwise complex

    $v_1 \cdot v_2 = b_1 b_2 + c_1 c_2 + d_1 d_2;\ v_1 \times v_2 = (c_1 d_2 - c_2 d_1)i + (d_1 b_2 - d_2 b_1)j + (b_1 c_2 - b_2 c_1)k$
  - $q_1 q_2 \neq q_2 q_1 \rightarrow v_1 \times v_2 \neq v_2 \times v_1$. It is opposite
  - Conjugate: $q = a + bi + cj + dk,\ q^* = a - bi - cj - dk$ ➔ $qq^* = q^* q = a^2 + b^2 + c^2 + d^2$
  - Quaternion rotation: $S_{q\left(N,\frac{\theta}{2}\right)}(v) = q(N,\frac{\theta}{2}) \cdot v \cdot q^*(N,\frac{\theta}{2})$

    Unit Quaternion: $q(N,\theta) = (cos\theta,\ sin\theta N)$. Unit quaternion rotation :$q = (cos\frac{\theta}{2},\ sin\frac{\theta}{2}N)$
  - Pros of quaternion: Does not suffer gimbal lock; Concatenating rotation is computationally faster and numerically more stable; Angle and axis is simply to extract

    Cons of quaternion: Not intuitive; Cannot visualize, need to convert to rotation matrix to visualize
- ***Kinematic Chains of Bodies:***
  - Kinematic: study if possible movements and configurations of a system
  - Link: Each rigid body in a chain of rigid bodies; Joint: Connect two rigid bodies and enforces constrains
  - Forward kinematics: Position of the end effector in terms of joint angles → Given the joint angles, find the position of the end-effector
  - Inverse kinematics: Joint angles in terms of the position of the end effector → Given the position of the end-effector, determine the joint angles
  - Application of T_i moves A_i from its body frame to body frame of A_{i-1}

    $$T_i = \begin{pmatrix} cos\theta_i & -sin\theta_i & a_{i-1} \\ sin\theta_i & cos\theta_i & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
  - For I = 3, we can have

    $x = a_1 cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) + a_3 \cos(\theta_1 + \theta_2 + \theta_3)$

    $y = a_1 sin\theta_1 + a_2 \sin(\theta_1 + \theta_2) + a_3 \sin(\theta_1 + \theta_2 + \theta_3)$
  - Inverse kinematic with 2 solutions:

    $cos\theta_2 = \frac{1}{2a_1 a_2}((x^2 + y^2) - (a_1{}^2 + a_2{}^2))$

    $sin\theta_2 = \pm\sqrt{1 - cos^2\theta_2}$

    $cos\theta_1 = \frac{1}{x^2+y^2}(x(a_1 + a_2 cos\theta_2) \pm y a_2 \sqrt{1 - cos^2\theta_2})$

    $sin\theta_1 = \frac{1}{x^2+y^2}(y(a_1 + a_2 cos\theta_2) \mp x a_2 \sqrt{1 - cos^2\theta_2})$
- ***Configuration Space:***
  - Definition: The **configuration** of a moving object is a specification of the position of every point on the object

    The configuration space C is the set of all possible configurations → Configuration space is the space that robot can reach
  - The **dimension** of a configuration space: the minimum number of a parameters needed to specify the configuration of object

    Degrees of freedom (Dof) of a moving object
  - Dof for a space: $d = \frac{n(n+1)}{2}$

    2-dimensional space: degree is 3, with 2 position parameters and 1 rotation parameters

    3-dimensional space: degree is 6, with 3 position parameters and 3 rotation parameters

    4-dimensional space: degree is 10, with 4 position parameters and 6 rotation parameters

    Dof = $\sum(freedom\ of\ bodies) - \#\ of\ independent\ constraints = m(N - 1 - J) + \sum_{i=1}^{J} f_i$

    m is space degree, N is number of bodies(include ground), J is number of joints, f is total number of joints degrees
  - Degree of freedom of different joints:

    Revolute joint: 1; Prismatic joint: 1; Universal: 4; Spherical: 3
  - One joint can only connect 2 rigid bodies
- ***Topology:***

- Two spaces are topologically equivalent if one can be smoothly deformed to the other without cutting and gluing
- Topologically distinct 1-dimensional space: “circle”, “line”, “closed interval”
- Topologically distinct 2-dimensional space: “plane”, “sphere”, “surface of torus”, “surface of a cylinder”
- Examples:

Mobile robot translating in palne	$\mathbb{R}^2$	Rigid body translating in 3D	$\mathbb{R}^3$
Mobile body trans and rotate 2d	SE(2)	Rigid body trans and rotation	SE(3)
n-joint revolute arm	$\mathbb{T}^n$	Planar mobile robot with n-joint arm	$SE(2) \times \mathbb{T}^n$
Prismatic joint	$\mathbb{R}$	Boundary of circle in 2D	S^1
Torus	$\mathbb{T} = s^1 \times s^1$	Boundary of sphere in 3D	S^2
		2D rotation	$SO(2)$
		3D rotation	$SO(3)$
		$S^1 \times S^1 \ldots \times S^1 = \mathbb{T}^n \neq S^n$	$SE(3) = \mathbb{R}^3 \times SO(3)$

PRM：

1. Add q_{init}, q_{goal} to roadmap vertex set V
2. Repeat: for $q \leftarrow Sample()$, add to roadmap vertex set V if $\boldsymbol{isSampleCollisionFree(q)}$ = true
3. For each pair neighboring $Sample(q_a, q_b) \in V \times V$

   Generate path from $\boldsymbol{LocalPath(q_a, q_b)}$

   If $\boldsymbol{isPathCollisionFree(path)}$ = true: add (q_a, q_b) to road map edge set E
4. Search $Graph(V, E)$ for path q_{init} to q_{goal}

Pros: Computationally efficient, Solves high-dimensional problems, Easy to implement, Applications in many different areas

Cons: Not guarantee completeness (Not always find solution or report no solution exist)

Probabilistic completeness:

1. When has solution, probability to find a solution when time goes infinite
2. When no solution may not able to determine solution does not exist

Path smoothing:

Repeatedly replace long paths by short paths

Roadmap with no circle: Edge is added to roadmap only if connects two different roadmap components

Lazy PRM:

1. Generate samples and construct edges
2. Find minimum path to check if is edge collision free
3. Remove collision edge and redo procedure

Narrow-Passage Problem

1. Probability of generating samples vis uniform sampling in a narrow passage is low due to the small volume of the narrow passage.

Gaussian Sampling

1. $q_a \leftarrow$ generate configuration uniformly at random
2. $r \leftarrow$ generate distance form Gaussian distribution
3. $q_b \leftarrow$ generate configuration uniformly at random at distance r from q_a
4. If q_a is collision free and q_b is not, return q_a
5. If q_b is collision free and q_a is not, return q_b
6. Otherwise, return $null$

Bridge-based Sampling

If q_b and q_b are both NOT collision free, then create Path between q_b and q_b

And find q at the midpoint of the Path then check if q is collision free

Visibility-based Sampling

Generate samples that create new components or join existing components

1. Generate a q uniformly at random and if isCollisionFree() is true
2. If q belongs to a new roadmap component then return
3. If q connects two roadmap components then return

4. Otherwise return $null$

Importance Sampling

Construct roadmap using given sampling strategy

Identify roadmap nodes that lie in regions that are hard to connect

Sample more in these regions

- Tree-based Motion Planning

  Useful for problems that involves kinematic and dynamic constrains

  General idea: Graw a tree in the free configuration space from q_{init} to q_{goal}

  Rapidly-exploring Random Tree (RRT)

  - Generate q_{rand} at random sample and find q_{near} which is the nearest configuration in Tree from q_{rand}
  - Generate Path from q_{rand} to q_{near} and if $dist \le step$, add q_{rand} to Tree and add Path as edge to Tree
  - Else find a configuration q_{new} in the Path has $dist \le step$, add q_{new} to Tree and add (q_{new}, q_{near}) as edge
  - If find solution $dist(q_{new}, q_{goal}) \approx 0$, return. Otherwise repeat

  Aspect of improvement:

  Only take small step between q_{rand} and q_{near} → Take several steps until q_{rand} is reached or a collision is found

  Expansive-Space Tree (EST)

  - Push the tree frontier in the free configuration space
  - EST associate a weight $w(q)$ with each tree configuration
  - $w(q)$ is a running estimate on importance of selecting q as the tree configuration from which to add a new tree branch
  - $w(q) = \frac{1}{1+\deg(q)} = \frac{1}{(1+\text{number of neighbors near } q)}$
  - Select q in T with probability $w(q)/\sum_{q' \in T} w(q')$
  - Sample a collision-free configuration near q_{near}
  - Generate path from q to q_{near}
  - If path is collision-free, then add to the Tree

  Bi-directional Trees

  - Rooted at q_{start} and q_{goal}
  - Fewer configuration in each tree, imposes less of a computational burden
  - Each tree explores a different part of the configuration space

- PRM provides global sampling of the configuration space. But if sampling is sparse, roadmap is disconnected, and dense sampling is impractical in high-dimensional spaces
- Tree planner provides fast local exploration of area around root. But tree growth slow down significantly in high-dimensional spaces even bi-directional trees offers some improvements
- Sampling-based Roadmap of Trees (SRT)
  - Hierarchical planner
  - Top level performs global sampling (PRM-based)
  - Bottom level performs local sampling (Tree-based)
  - Combines advantages of global and local sampling
    - Idea: Sample some configurations in space, and expand configurations into Trees, then connect nearest trees
  - Combine Sampling with Some Estimation of Coverage
    - EST: Use density of nodes to guide expansion (density bias)
    - SBL: Uses some coverage estimates and density of nodes
  - KPIECE (Sampling and some estimation of coverage):
    - Keeps tract of coverage by using discretization and by distinguishing the boundary from the covered space
    - Keeping of coverage can be done in a hierarchical fashion
    - Projections may be used
  - Path-Directed Subdivision Trees (PDST):
    - Incrementally grows a tree
    - Samples are paths instead of configurations
    - Density used to limit redundant growth
    - Measures coverage spatially and uses deterministic priority scheme to guide expansion. No distance measure is needed
    - Excellent planner for robots with dynamics