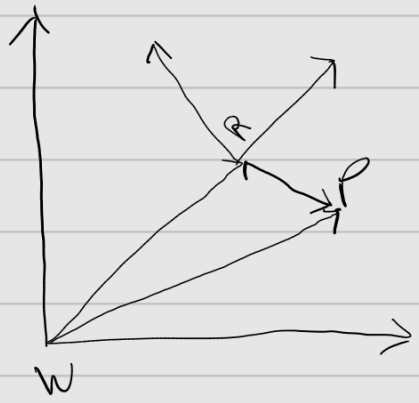


${}^{W}P = {}^{W}\xi_{R} \cdot {}^{R}P$

Closure: composition of 2 ξ is ξ

Associativity: $(\xi_1 \oplus \xi_2) \oplus \xi_3 = \xi_1 \oplus (\xi_2 \oplus \xi_3)$

identity: $\xi \oplus 0 = 0 \oplus \xi = \xi$

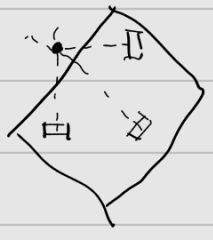
Can be described as first rotation then translation.

Homogeneous

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Pose in 2D is represented by (x, y, θ)

Instantaneous Center of Rotation (ICR)



if they don't intersect at same point, Robot can't move.

Rotations in 3D

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

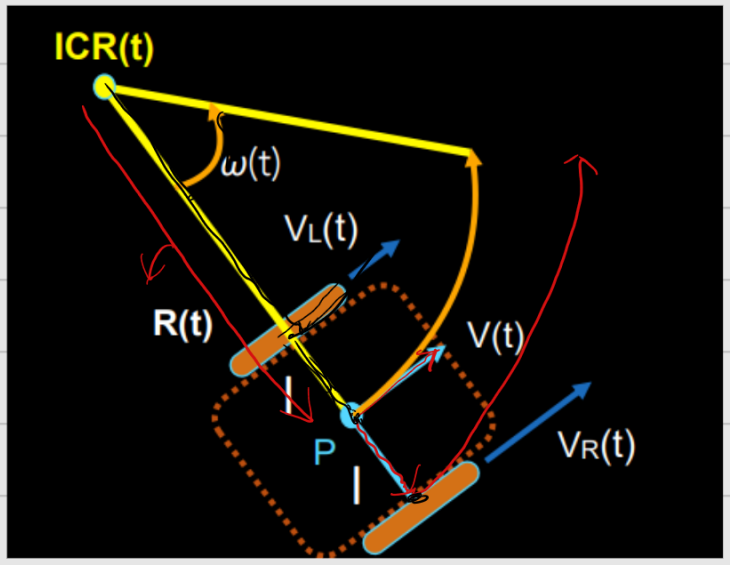
$$R_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

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

Differential Drive Robots

$$R(t) = l\,\frac{V_R(t)+V_L(t)}{V_R(t)-V_L(t)}$$

$$w(t) = \frac{V_R(t)-V_L(t)}{2l}$$



Idea to compute transform:

1. move Robot to ICR $(x,y) = (0,R)$
2. Rotate by $\Theta(t)$
3. move again to $(0,-R)$ with respect to new frame

$$w(t)\,(R(t)+l) = V_R(t)$$
$$w(t)\,(R(t)-l) = V_L(t)$$

$$\frac{w(t)\,(R(t)+l)}{w(t)\,(R(t)-l)} = \frac{V_R(t)}{V_L(t)}$$

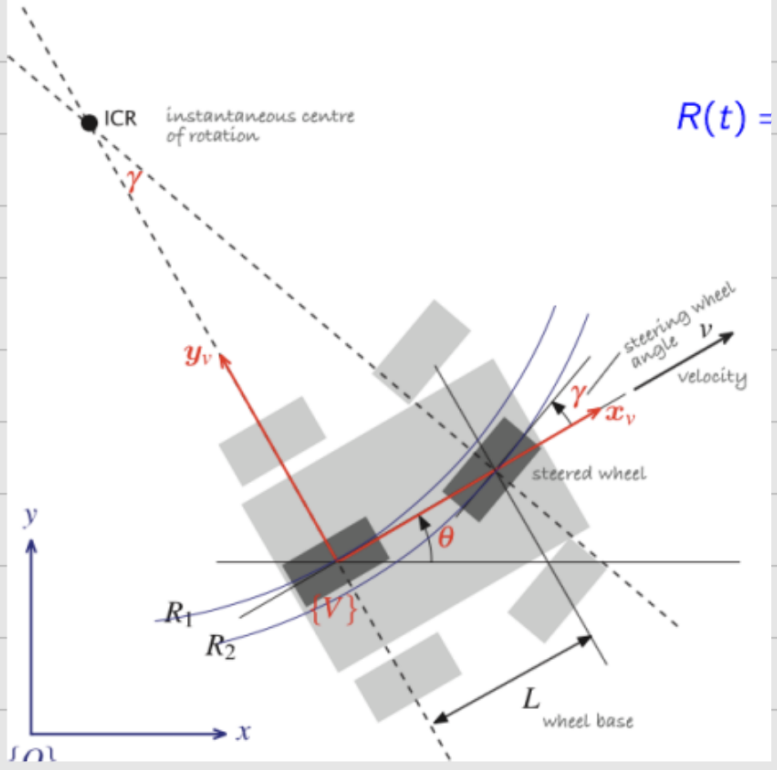
$$\Leftrightarrow V_L(t)\,(R(t)+l) = V_R(t)\,(R(t)-l)$$
$$\Leftrightarrow V_L(t)\,R(t) + V_L(t)\,l = V_R(t)\,R(t) - V_R(t)\,l$$
$$\Leftrightarrow R(t)\,(V_L(t)-V_R(t)) = -l\,(V_R(t)+V_L(t))$$
$$\Leftrightarrow R(t) = l\,\frac{V_R(t)+V_L(t)}{V_R(t)-V_L(t)}$$

$$w(t) = \frac{V_R(t)}{R(t)+l} = \frac{V_R(t)}{l\,\frac{V_R(t)+V_L(t)}{V_R(t)-V_L(t)} + l} =$$

$$= \frac{V_R(t)}{l\left(\frac{V_R(t)+V_L(t)+V_R(t)-V_L(t)}{V_R(t)-V_L(t)}\right)} = \frac{V_R(t)\,(V_R(t)-V_L(t))}{l\,2V_R(t)}$$

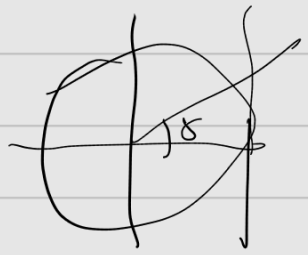
$$= \frac{V_R(t)-V_L(t)}{2l}$$

The bicycle model



$R(t) =$

$R(t) = R_2(t)$

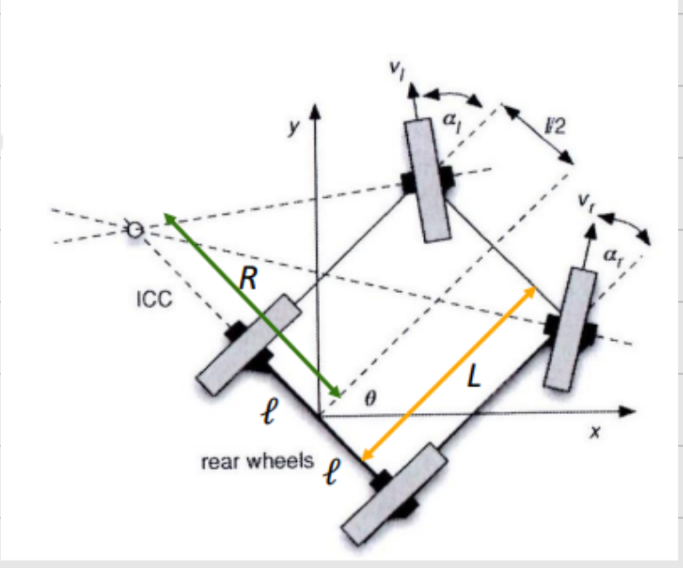


$R_2(t) \cdot \tan(\gamma) = L$

$R_2(t) = \frac{L}{\tan(\gamma)}$

$v(t) = R_2(t)\,\omega(t)$

$\omega(t) = \frac{v(t)}{R_2(t)}$



$R(t) - \ell = \frac{L}{\tan(\alpha_1(t))}$

$R(t) + \ell = \frac{L}{\tan(\alpha_2(t))}$

Localization

Map representations

- Continuous representation
- Decomposition (discretization)

Continuous line based

- Use sets of lines in a known map, which the robot can then observe and fit with known map to figure location

Exact polygon decomposition

- Represent map as sets of polygons. Only a few points are kept in memory

Approximate cell decomposition

- Represent map in discretized grid and mark grid squares where object is.

Adaptive cell decomposition

- Same as before but increased resolution near object borders

Topological representation

- Represent a map as a set of nodes with connections (possible paths)

Continuous vs. Discrete

Continuous	Discrete
• precision bound by sensors • single hypothesis (might get lost) • compact and efficien	• Precision bound by discretization resolution • multiple hypothesis • requires more memory and process power

Approaches to map localization

- Markov
- Kalman

Markov

- Multiple estimates of position
- Can start in unknown place
- Can handle ambiguous situations
- Computationally heavy to update all positions in probability space

Kalman

- Single estimate
- Requires known start position
- Robot odometry and sensors need to be very precise
- Cannot recover if gets lost

Markov (discretized)

Sensor Mode (SEE)

$$P(l|i) = \frac{P(i|l)\,p(l)}{p(i)}$$

$p(l)$ → previous belief of being in l

- Update after reading a sensor

Action Model (ACT)

$$p(l_t|o_t) = \int p(l_t|l'_{t-1}, o_t)\, p(l'_{t-1})\, dl'_{t-1}$$

↙

Sum over all probabilities of reaching l_t given that we were at l'_{t-1} and observed o_t

- Reads value from odometry and updates value.

Drawbacks of Markov

- for a pose 2D (x, y, θ) needs a 3D discretization map
  - Can be improved with randomized sampling to improve complexity.
    approximates belief state by keeping track of a small subset of states
    sampling is weighted around peaks with highest probability.

Kalman

Nothing much said

Scan Matching

- If we know point correspondences subtract mean to each other
- Calculate $W_{2\times 2} = \sum_{i}^{N} x'_i p_i$
- Perform SVD on $W_{2\times 2} \Rightarrow U \begin{bmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{bmatrix} V^T$, $\theta_1 \geq \theta_2 \geq \theta_3$
- If W is full Ranks $R = UV^T$, $t = \mu_x - R\mu_p$
  and $E(R,t) = \sum_{i=1}^{N} (\|x_i\|^2 + \|y_i\|^2) - 2(\sum_j \theta_j)$

Iterative closest points (ICP)

- If we don't know correspondences we use ICP

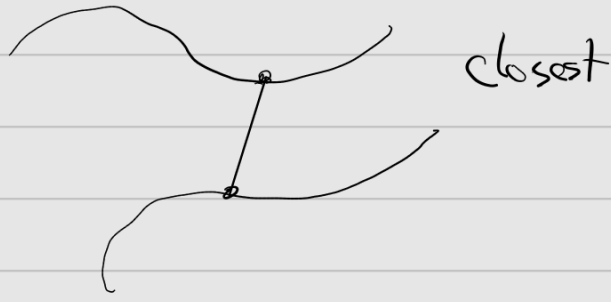
Algorithm:
- Determine correspondence
- Compute SVD and get R and t
- Apply to points in P
- Compute error and see if less than previous error
- If yes, iterate again

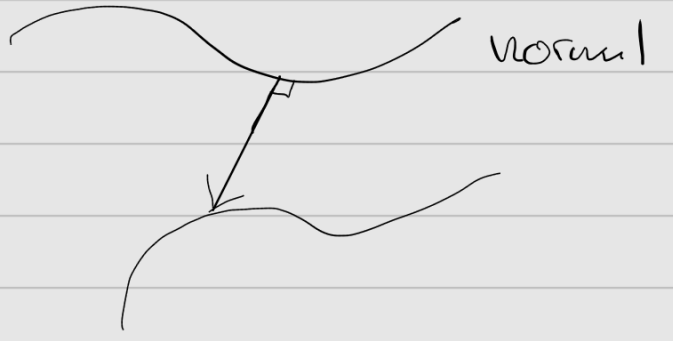
Both sets might have points without correspondences outliers. So how do we select points from each set?
- Use all
- Uniform sub-sampling (same density in both sets, avoids overweighting dense areas)
- Random sampling
- Feature based sampling (color?)
- Normal space sampling

We can weight points so that points that were further from robot weight less in the error metric

How do we associate the points?

- Closest point
- Normal shooting
- Closest compatible
- projection based





Compatible:

- normals
- colors
- local curvature
- thighest order derivatives

Outliers:

- Ignore correspondences with high distance (threshol)
- reject pairs incosisten with neighbors

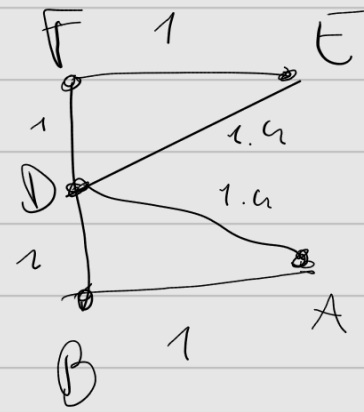
Path planning

Approaches:
- Use free c-space
- Build a graph: Vertices are poses in c-space
  - Use a minimum cost path algorith

A*

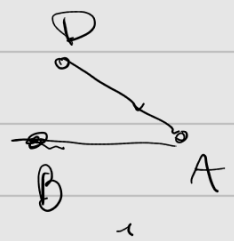
$f^*(n) = g(n) + h^*(n)$

From A to E

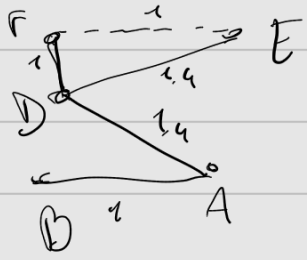


f^* is the direct distance

Start



$f^*(D)$ has the lowes direct distance so we expand D



$f^*(E) = (1.4) + (1.4) + 0$ has the lowes direct distance, so we reached destination

Combinatorial Planning
(Building planning graphs)

Methods:
- Visibility Graphs
- Voronoi diagrams
- Approximate space decomposition
- Exact spatial decomposition

Visibility graphs

Given polygonal objects, we can travel through vertices of polygons until we reach end.

for each point on the graph, swipe lines to to get visibility edges.

Guaranteed to converge to optimal path but might collide with edges

Generalized voronoi diagram

- Computes locus points (points equidistant to object borders)

- Tends to be biased towards open spaces
- easy to generalize to 3D

- Good for imprecise robots
- Uses intersection of focus lines

Approximate space decomposition

- Discretize c-space in mesh grid, where each cell is a node.
- nodes that touch objects are not taken into consideration

- Depending on grid size, might not be able to computed path.

Quad-trees iterative decomposition

- Same idea as adaptive cell decomposition
- cells near borders have higher resolution

Exact cell decomposition

- Vertical lines are draw in free c-space and middle points are considered nodes

---

Above algorithms are not complete because they rely on discretization (resolution)

Sampling based planning

- blindly explores free c-space
- Needs a colision detection tool to probe free c-space
- Needs a local planer to check if sampled configuration can be connected to another nearby

Probabilistic Road Maps (PRM)

- Take random samples in free c-space
- Use k-nn or radius to look for nearby nodes
- try to connect vertices with local planner:
  - holonomic: checks if line of sight is colision free
  - non-holonomic: checks if vertex can be reached with non holonomic constraints
- keep generating and adding nodes until road-map is dense enough

- Non optimal, non complete
- heuristics might be used for sampling strategy

Rapidly Exploring Random Trees (RRT)

- Root of tree starts at p_{start}
- Generate configuration (using collision detection)
- use distance criterion to find nearest neighbor
- not optimal and not complete
- we can force RTT to move to p_{find} an incremental amount every nth iteration (RTT trick)

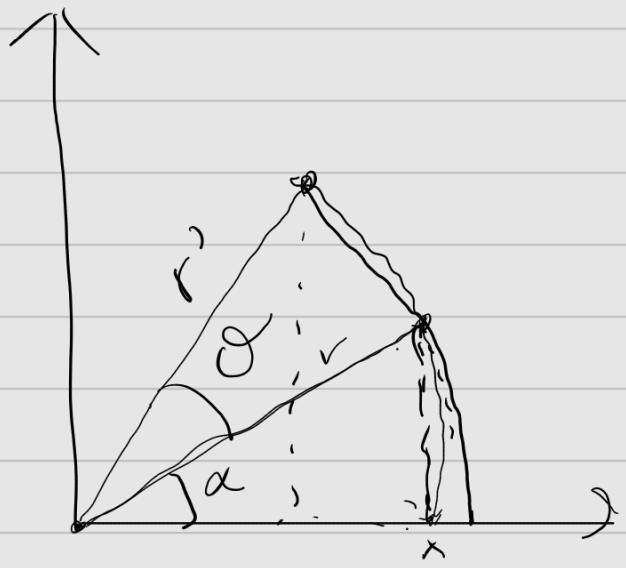
RTT-Connect

- Grow 2 RTTs, one from start and one from end

RTT*

- Rewires the tree as it grows using some distance criterion

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



$$\begin{cases} x = r\cos(\alpha) \\ y = r\sin(\alpha) \end{cases}$$

$$\begin{cases} x' = r'\cos(\theta+\alpha) \\ y' = r'\sin(\theta+\alpha) \end{cases}$$

$$\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$$
$$\sin(\alpha+\beta) = \cos(\alpha)\sin(\beta) + \sin(\alpha)\cos(\beta)$$

we know $r' = r$ hence

$$x' = \overbrace{r\cos(\alpha)}^{x}\cos(\theta) - \overbrace{r\sin(\alpha)}^{y}\sin(\theta)$$
$$y = \underbrace{r\cos(\alpha)}_{x}\sin(\theta) + \underbrace{r\sin(\alpha)}_{y}\cos(\theta)$$

$$\begin{aligned} x' &= x\cos(\theta) - y\sin(\theta) \\ y' &= x\sin(\theta) + y\cos(\theta) \end{aligned} \Rightarrow \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Final

① Transforms

- ${}^{A}\xi_{C}$:= Pose of $\underline{C}$ relative to $\underline{A}$,
- $p^{w} = ({}^{A}\xi_{B} \oplus {}^{B}\xi_{w}) \cdot {}^{w}p$, ${}^{w}p$ is p from the perspective of w. We can do compositions between reference systems to get a different perspective on the or $\underline{p}$
- Properties:

Closure: $\xi_1 \oplus \xi_2 = \xi_3$

Associativity: $(\xi_1 \oplus \xi_2) \oplus \xi_3 = \xi_1 \oplus (\xi_1 \oplus \xi_2)$

Identity: $\xi \oplus 0 = 0 \oplus \xi = \xi$, 0 is the null relative pose

Not-commutative!!

${}^{A}\xi_{B} \oplus {}^{B}\xi_{C} = {}^{A}\xi_{C}$ Composition

$\ominus {}^{A}\xi_{B} = {}^{B}\xi_{A}$ inverse

Note: if ${}^{i}\xi_{j}$ are in homogenous then ${}^{i}\xi_{j} \oplus {}^{j}\xi_{K} = {}^{i}\xi_{j} \cdot {}^{j}\xi_{K} = {}^{i}\xi_{K}$

Non-homogeneous 3D

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

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

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

Homogeneous 2D

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

$$tr(\vec{v}) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}, \vec{v} = (t_x, t_y)$$

② Poses, workspace, C-space

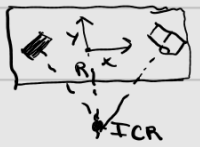
- Rigid: a body is rigid if constituent points maintain the same relative position
- A pose of rigid body in 2D is defined by position and orientation (x, y, θ) w/ respect to world reference frame. ($(x, y, z, \alpha, \beta, \gamma)$ in 3D)
- topology of a workspace is a 3D volume

③ Forward Kinematics

- Holonomic: Can move in any direction independant of orientation (RoboMaster vs. Car)
- ICR: Instantaneous Center of Rotation ⇒ intersection point between all tangent steerad wheels.
  - ↳ If more than 1 point, system can't move.

- $\tan(\gamma) = \frac{O}{A}$, $\sin(\gamma) = \frac{O}{H}$, $\cos(\gamma) = \frac{A}{H}$, $Arc = \theta \cdot r$

$\vec{V} = \vec{\omega} \cdot r$



R is the distance from ICR to Robots referance frame

(4) Dead Reckoning (odometry)

- Process of (incrementally, discrete timesteps) determining own position in the absence of external infrastructure, using a know position and knowledge/estimate of velocities
- Prone to accumulated error over time (odometry drift)
  - ↳ innacurate measurements, wheel slippage, integration error approximations
- Systematic/Deterministic errors can be corrected through calibration

(5) closed-loop control

- Open-loop control: decisions without observing world or consequences of previous actions
- Closed-loop control: feedback based decisions
- PID controller: takes error as input, makes decision based K_p, K_d, K_i
  - ↳ K_p: Proportional gain := how quickly to act given error => $K_p \cdot error$
  - ↳ K_d: derivative gain := considers system inertia => $d = \frac{e - e_{last}}{dt}$, $K_d \cdot d$
  - ↳ K_i: integral gain := K_p might not be enough to put error to zero => $i \mathrel{+}= \frac{e}{dt}$, $K_i \cdot i$
- K_p makes us go faster towards goal, K_d makes as slow down near goal (prevents oscilations) and K_i makes us align with goal if external forces are applied
- We can stack PIDs, feeding the result of one to another
- Bang-bang controller: a controller that only outputs 1 of 2 options depending on error.

(6) Localization

- Sensor aliasing: When the sensor information is not sufficient to determine position
- Cannot just use odometry because of accumulated errors.

- Global localization: 'kidnaping'; position tracking: known initial position.

- Map representations need to take into account:
  - precision vs computational complexity: more precision ≡ more computation
  - precision vs features precision (data from robot)
  - precision vs application: how precise do we need it to be

- Map representations:
  - Continuous line based: Architecture of map + set of lines
  - Exact cell decomp. w/ polygons: for polygons in map, lines that pass through vertices
  - Approximate cell decomp: grid map with polygons and mark cells where polygons touch
  - Adaptive cell decomp.: like before but with increased resolution near borders
  - Topological decomp.: given map, mark nodes connected nodes in different sections (fully connected graph)

- Belief of position can be represented as a distribution in some probability space
- Continuous: precision bound by sensor, typically single hypothesis, compact and efficient
- Discrete: precision bound by resolution, multiple hypothesis, computationally heavy

- Markov: multiple estimates, can start at unknown, can recover from ambiguous situations, computational heavy
- Markov: uses explicit discrete representation, on each update all states are updated
- Kalman: single estimate, requires initial known, very precise, cannot recover if robot gets lost

- Sensor model (SEE): update after sensor reading := $p(l|i) = \frac{p(i|l)\,p(l)}{p(i)}$, i := input, l := state
- Action model (ACT): updating after moving (odometry) := $p(l_t|o_t) = \int p(l_t|l'_{t-1}, o_t) \cdot p(l'_{t-1})\, dl_{t-1}$

- For discretized map (markov), the pose in 2D originates $n \times m \times 3$ matrix of possible states

- Randomized sampling (Monte carlo): sample states around current distribution and compute probabilites after ACT and SEE. repeat

- Landmark model: given observed landmark, sample points from distribution of points given landmark

(7) Scan Matching

$X = \{x_1, \dots, x_N\}$

$P = \{p_1, \dots, p_n\}$

$E(R,t) = \frac{1}{N}\sum_i^N \|x_i - (Rp+t)\|^2$

$X' = \{x_i - \mu_x\}$

$P' = \{p_i - \mu_p\}$

$W = \sum_i^N x_i' p_i'^T \Rightarrow 2\times 2$ matrix

$W = U \cdot diag(\theta_i) \cdot V^T \Rightarrow SVD$

$R = UV^T, \quad t = \mu_x - R\mu_p$

$E(R,t) = \sum_i^N (\|x_i'\|^2 + \|g_i\|^2) - 2\sum_i \theta_i$

(1) selecting points:
- Use all
- Uniform sub-sampling (Voxel base)
- Random sampling
- Feature based sampling
- Normal space sampling (uniformly dist. points)

(2) Weighting correspondances
- i.e. weight point far from robot by less

(3) Data association:
- Closest point: stable, slow convergence, computationally expensive
- Normal shooting (based on neighboring points): just for smooth surfaces
- Closest compatible (based on neighboring): based on normals, colors, local curvature, etc.

(4) Rejecting outliers
- reject pairs w/ distances > threshold
- reject pairs inconsistent w/ neighbors

Iterative closest Point (ICT)
- * sample subsets
- • Determine correspondences
- * discard outliers
- • Compute R, t
- * weight correspondences
- • Compute error, if < t, iterate

(8) Path planning

Idea:

1. build free c-space
2. Compute a graph
3. Use transvers algorith

Combinatorial planning

- Assuming holonomic and polygon objects

- **Visibility graphs**
  - From start swipe lines to get vertices
  - From each vertice swipe lines to get others
  - Optimal, can lead to collision due to errors, complicated in more than 2 dimensions, finding a safe ath might be more important, complete

- **Approximate cell decomposition**
  - Same idea as exact cell decomp.
  - Easy to build, might falsely report no available path due to resolution, high res. increases computations, not complet, not optimal

(3) transverse algorithm

- A* is based on heuristics.
- At each iteration we expand the node with min cost, where cost is given by path from start to possible expanding node plus heuristic for that node to finish

- **Generalized Voronoi diagrams**
  - Locust points are equidistant to closest obstacles in c-space
  - Connect start and finish to closest locust lines
  - Difficult in higher dimensions, not optimal and attraction to open spaces, unstable in small changes in obstacles, complete

- **Quad-trees iterative decomposition**
  - Same idea as adaptive cell decomp
  - Recursived devides mixed cells not Coplek, not optimal

Exact cell decomp.

- Exactly decompose free space into finite cells
- For each vertex, shoot ray up/down (divides into trapezoids)
- put 1 node inside trapezoid and in every line

- Combinatorial are quickly intractable, but complete

Sampling based planning

- Weaker guarantees but more efficient
- No definite explicit C-space needed

- Collision-detection tool
  - probes C-space to see if point lies in free

- local planner
  - See if sample configuration can be connected by another with feasible path

Probabilistic Road Maps (PRM)

- Sample, use collision detection, select nearby vertices and use local planner to connect
- keep adding configurations until dense enough
- For non-holonomic, local planner takes into account robot constraints.
- Not optimal, does not build c-space, scalable, issues in narrow passages

Rapid Exploring Random Trees (RTT)

- Root tree at start
- Generate random config and validate using collision detection
- Detect closet vertex to random config w/ distance criterion,
- From nearest move a bit and assess collisions
- If vertex is reach, add it to tree and the line of parameters connecting
- Not optimal, not complete, doesn't build C-space, not scalable

Trick: every nth iteration try move a bd amount towards finish node

RTT-Connect (Bidirectional)

- Have 2 trees growing from start node and end node
- Avoids problems with narrow passages in convex sets

RTT*

Same as RTT, but at every iteration tries to improve tree configuration rewiring nodes