1. Motivation **3**
Content 3
2. Configuration Space **3**
2.1. Degrees of freedom 3
2.1. From workspace to configuration space 4
2.2. Paths 5
2.3. Obstacles 5
3. Bug algorithms **6**
3.1. Base Navigation 6
3.2. Bug algorithms 6
3.3. Application of Bug Strategies for Configuration Spaces Ch 3.2.2 7
4. Classical path planning approaches **7**
4.1. Roadmaps Ch 5 7
4.1.1. Visibility graph method CH. 5.1 7
4.1.2. Voronoi Diagram CH. 5.2 8
4.2. Cell Decomposition Ch 6 8
4.2.1. Trapezoidal decomposition Ch 6.1 8
4.2.2. The Halting Problem 9
4.2.3. Implementation of Vertical Cell Decomposition in a higher dimension 6.3.3 9
4.2.4. k-d Trees 10
4.2.5. Approximate Cell decomposition 10
4.3. Potential field 10
4.3.1. Function of the potential field 11
4.3.2. Bushfire Algorithm Ch 4.3.2 12
4.3.3. Wave Front planner CH 4.5 12
5. Sampling-Based Algorithms Ch7 **13**
5.1. Characteristics of Sampling-Based Planners 13
5.2. Sampling Strategy: Create a uniform distribution 13
5.3. Classic multiple query PRM (Probabilistic Roadmap) 13
5.4. Single Query PRM Ch7.2 15
5.4.1. rapidly-exploring random trees (RRT, by LaValle & Kuffner) Ch 7.2.2 15
5.5. Analysis of PRM: Expansive Spaces Ch 7.4.2 16
5.6. PRM Conclusion 18
5.7. OBPRM: Obstacle-Based PRM 18
6. Kalman Filtering Ch 8 **19**
6.1. Data Fusion example 19
6.2. Observability matrix 20
6.3. Problems with the Kalman filter 21
6.4. Kalman examples 21
6.4.1. Example: Water level in tank 22
6.5. The Extended Kalman Filter (EKF) Ch 8.3 22
6.6. Linearized Motion Model for a Robot 24
6.7. Extended Kalman Filter for SLAM Ch 8.4 24

6.8. Summary 25

7. Unscented Kalman Filter 26

7.1. Filtering Problem: General Problem Statement 26
7.2. Unscented Transformation 26
7.3. Scaled Unscented Transformation (SUT) 27
7.4. Unscented Kalman Filter 27

8. Bayesian and Particle Filter Ch 9 28

8.1. Localization 28
8.2. Bayesian Filtering 29
8.3. Particle Filtering 30

9. Overview Filters 32

1. Motivation

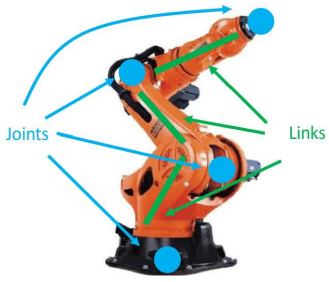
CONTENT

- Complete knowledge
  - Bug Algorithms
  - Configuration Space
  - Potential Functions
- Offline preprocessing
  - Road maps
  - Cell Decompositions
- Probabilistic Methods
  - Sampling-Based Algorithms
  - Kalman Filtering
- Filtering of Uncertainty
  - Bayesian Methods
  - Dynamic Environments

Planning algorithms are even impacting fields as far away from robotics as computational biology. Two major problems are protein folding and drug design. Such molecules are generally flexible. The *docking problem* involves determining whether a flexible molecule can insert itself into a protein cavity. The assembly problem and can be solved by motion planning algorithms.

2. Configuration Space

- Task space: a space where the robot's task can be naturally expressed (eg: entire hall $\mathbb{R}^3 \times S^1$)
- Workspace: captures the „reachable" space of the end effector (eg: reachable points in $\mathbb{R}^3$)
- Both involve some user choice and often are different from Configuration space



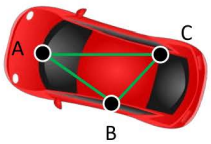
- Rigid body: one-piece robot
- Link: rigid piece, often a part of multi-piece robot
- Links of a multi-piece robot are joined with joints (connectors)

2.1. DEGREES OF FREEDOM

- Configuration: specification of where all points of a robot are
- Degrees of freedom (DOF): smallest number of real-valued (i.e. continuous) coordinates to fully describe configurations of robots

Single body

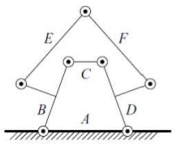
- DOF = total DOF of points - # of constraints
- Example Car: 2 x 3 - 3 = 3 (each of the three points on the car can move in x and y orientation, 3 constraints: A —> B, B —> C, C —> A)



General robots

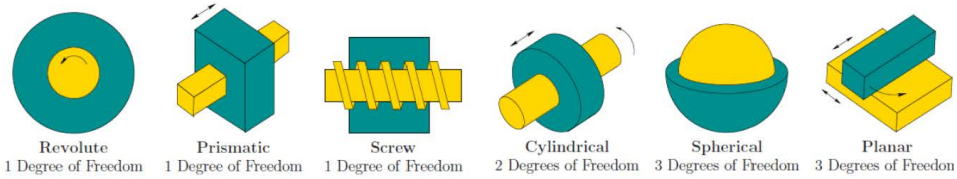
- 2D Chains: Base link is 3D, if fixed, then often 1D, Adding joints generally adds one more dimension
- 3D Chains: Base link is 6D, if fixed, depending on the joint, Then add the DOF of each additional joint
- Closed Chains: also known as a parallel mechanism, is one where the links form one or more closed loops
  - If the mechanism has k links, then one is designated as a stationary "ground" link, and k – 1 links are movable
  - Each movable link has N degrees of freedom: N = 6 for spatial mechanism, N = 3 for planar
  - Each of the n joints places N - f_i constraints on the feasible motion

- Grübeler's Formula: $DOF = N(k-1) - \sum_{i=1}^{n} \left(N - f_i\right) = N(k-n-1) + \sum_{i=1}^{n} f_i$



  - N: 6 for 3D, 3 for 2D
  - k: # of links (including the ground link)
  - n: # of joints
  - f_i: DOF of the joint

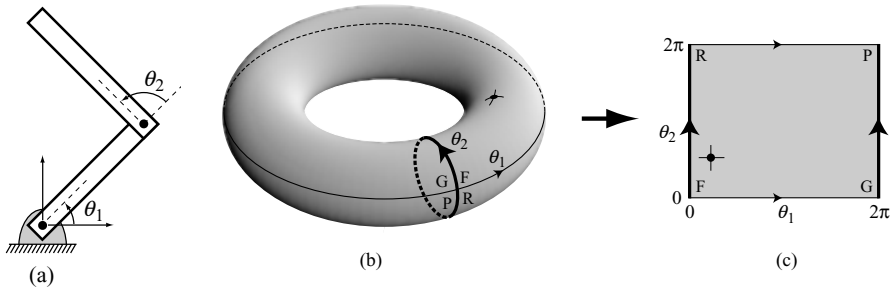
Type of joints
- 2D: Revolute and Prismatic
- 3D: see picture



2.1. FROM WORKSPACE TO CONFIGURATION SPACE

- Configuration of a moving object:
  - specification of the position of every point on the object —> vector of position & orientation $q = (q_1, q_2, \ldots, q_n)$
- Configuration space / C-space:
  - set of all possible configurations of the system —> configuration is simply a point in this abstract configuration space C
  - The number of degrees of freedom of a robot system is the dimension of the configuration space, or the minimum number of parameters needed to specify the configuration.
  - powerful abstraction for solving motion planning problems (finding feasible motions for robots from x_1 to x_G)
  - in C-space: searching for a path in the joint space of 2D position $(x, y) \in \mathbb{R}^2$ and rotation $\theta \in S^1$
  - topology of C: usually not Cartesian

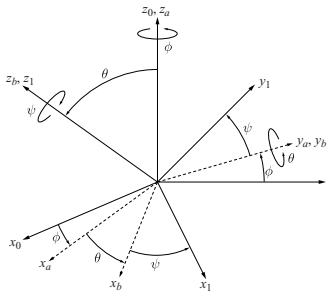
Examples
- screw in nut is 1D because height is coupled to rotation
- (a) Workspace: A two-joint manipulator.
  (b) The configuration of the robot is represented as a point on the toral configuration space.
  (c) Configurations space: The torus can be cut and flattened onto the plane. This planar representation has "wraparound" features where the edge *F R* is connected to *G P* , etc.



- many more on slides

Matrix representations in SO(3)
- nine elements of rotation matrix $R \in SO(3)$ are subject to six constraints —> 3 DOF
- Expectation: SO(3) can be parametrized using three variables —> Euler angles are often used
- Correspond to three successive rotations around z-axis, y-axis and lastly x-axis:

$$
\begin{aligned}
R &= R_{z,\phi} R_{y,\theta} R_{z,\psi} \\
&= \begin{bmatrix} c_\phi & -s_\phi & 0 \\ s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_\phi c_\theta c_\psi - s_\phi s_\psi & -c_\phi c_\theta s_\psi - s_\phi c_\psi & c_\phi s_\theta \\ s_\phi c_\theta c_\psi + c_\phi s_\psi & -s_\phi c_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta \\ -s_\theta c_\psi & s_\theta s_\psi & c_\theta \end{bmatrix}.
\end{aligned}
$$



- no global parametrization because there are some singularities —> define functions like atan, …

Axis-angle parametrization: Rodrigues formula
- efficient algorithm for rotating a vector in space, given an axis and angle of rotation
- can be used to transform all three basis vectors to compute a rotation matrix in SO(3), the group of all rotation matrices, from an axis–angle representation
- In other words, the Rodrigues' formula provides an algorithm to compute the exponential map from so(3), the Lie algebra of SO(3), to SO(3) without actually computing the full matrix exponential

$$\mathbf{R} = \mathbf{I} + (\sin\theta)\mathbf{K} + (1 - \cos\theta)\mathbf{K}^2$$

is the rotation matrix through an angle θ counterclockwise about the axis k.
I the 3 × 3 identity matrix

Unit quaternion (p.494)
- The axis-angle parameterization described above parameterizes a rotation matrix by three parameters. Quaternions, which are closely related to the axis-angle parameterization, can be used to define a rotation by four numbers.
- Quaternions are a popular choice for the representation of rotations in three dimensions because compact, no singularity and naturally reflect the topology to the space of orientations

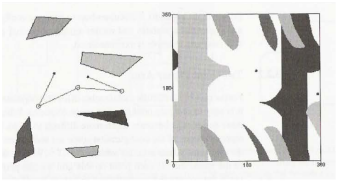
$$u = (u_1, u_2, u_3, u_4) \text{ with } u_1^2 + u_2^2 + u_3^2 + u_4^2 = 1$$

$$(u_1, u_2, u_3, u_4) = \left(\cos\theta/2, n_x \sin\theta/2, n_y \sin\theta/2, n_z \sin\theta/2\right) \text{ with } n_x^2 + n_y^2 + n_z^2 = 1$$

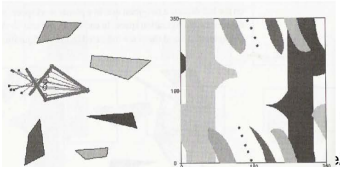
- # DOF = 6
- Topology = $\mathbb{R}^3 \times SO(3)$

2.2. PATHS
- path: continuous curve connecting two configurations q and q':
  $\tau : s \in [0,1] \rightarrow \tau(s) \in C$ such that $\tau(0) = q$ and $\tau(1) = q'$
- trajectory: path parametrized by time: $\tau : t \in [0,T] \rightarrow \tau(t) \in C$
- Constraints: Finite length, Bounded curvature, Smoothness, Minimum length, time, energy, …

Free space topology
- A free path lies entirely in the free space F —> no contact between robot and obstacle
- The moving object and the obstacles are modeled as closed subsets, meaning that they contain their boundaries.
- C-obstacles are closed subsets of the configuration space C
- Free space F is an open subset of C. Hence, each free configuration is the center of a ball of non-zero radius entirely contained in F.



Points cannot be connected



Multiple possibilities because of the periodicity of the angle

/courses/motion

Semi-free space
- semi-free path —> allows robot to contact the boundary of the obstacle
- semi-free space is a closed subset of C. Its boundary is a superset of the boundary of F.

2.3. OBSTACLES
- A configuration q is collision-free, or free, if a moving object placed at q does not intersect any obstacles in the workspace.
- The free space F is the set of free configurations.
- A configuration space obstacle (C-obstacle) is the set of configurations where the moving object collides with workspace obstacles.
- Is the configuration in the free space? Compute the position of the robot and check for collision with any obstacle at that position:
  - if colliding —> configuration within C-space obstacle

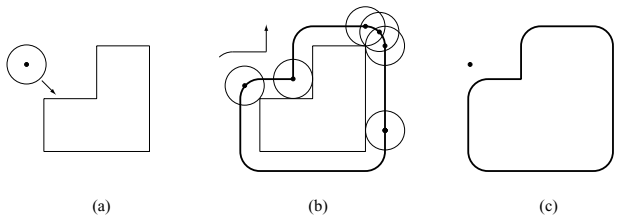


Figure 3.4 (a) The circular mobile robot approaches the workspace obstacle. (b) By sliding the mobile robot around the obstacle and keeping track of the curve traced out by the reference point, we construct the configuration space obstacle. (c) Motion planning for the robot in the workspace representation in (a) has been transformed into motion planning for a point robot in the configuration space.

- otherwise —> free space

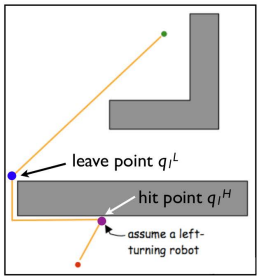
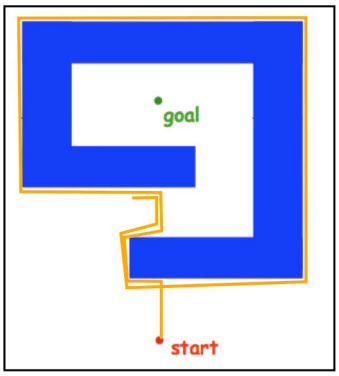
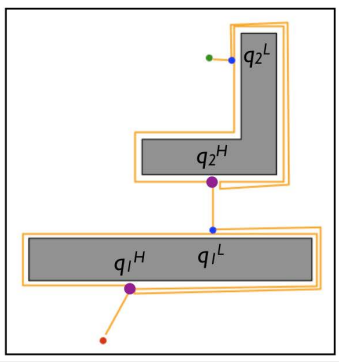
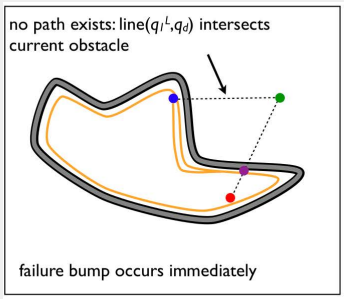
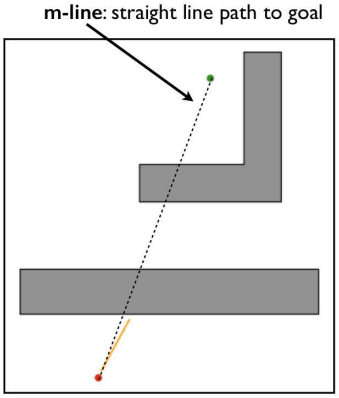
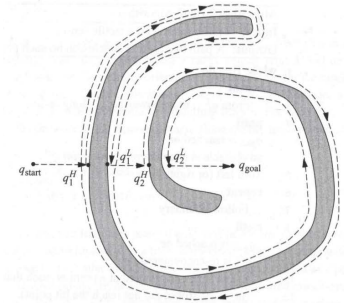
3. Bug algorithms

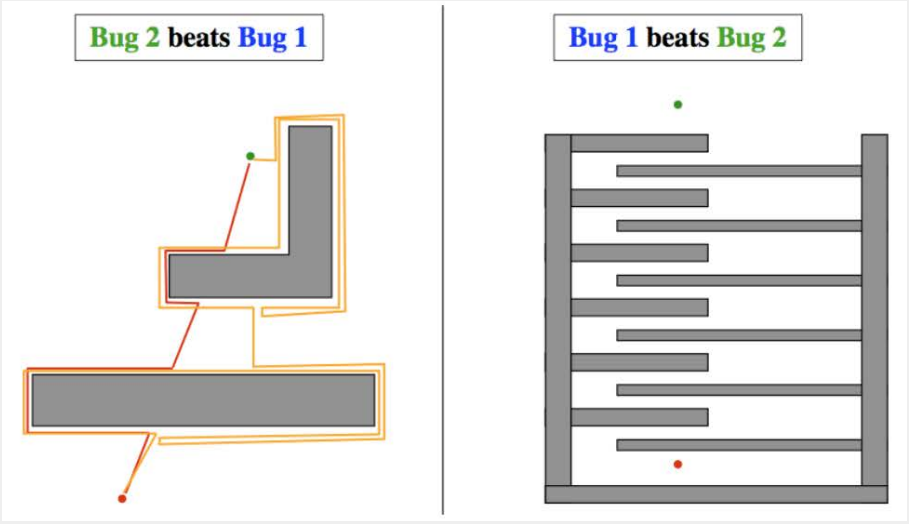
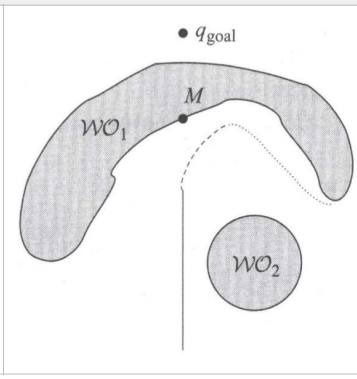
3.1. BASE NAVIGATION

- What is the simplest policy to perform navigation?
  - Random Walk: Move in a random direction until you hit something, then you go in a new direction, stop when you get to the goal, assuming it can be recognized
- What is a simple deliberative policy

3.2. BUG ALGORITHMS

- Assumption: bounded world W
- Known: global goal, measurable distance d(x,y)
- Unknown: obstalte WO_i
- Local sensing: tactile (e.g. bump sensor), distance travelled (odometry: use of data from motion sensors to estimate chafe in position over time)
- Navigation: path start from q_s to goal q_d, as a sequence of hit/leave point paris on obstacles (q_i^H, q_i^L)

Name	Description	Working Case	Problem case
Bug 0	• Head towards goal • When hit point set, follow wall until you can move towards goal again (leave point) • continue from 1		
Bug 1	• Head towards goal • When hit obstacle, circumnavigate, setting leave point as closest to goal • Return to leave point • if bump current obstacle, return fails, else, continue from 1 • Best case: $d\left(q_{start}, q\right)$ • Worst case (n obstacles, p_i: perimeter): $L_{Bug1} \leq d\left(q_{start}, q\right) + 1.5 \cdot \sum_{i=1}^{n} p_i$		 not possible
Bug 2	• Head towards goal on m-line • When hit point set, traverse obstacle until m-line is encountered • Set leave point and exit obstacle • Continue from 1 • Duration (intersects i-th obstacle n_i times): $L_{Bug2} \leq d\left(q_{\text{start}}, q\right) + 0.5 \cdot \sum_{i=1}^{n} n_i p_i$		 n = 2 here

Name	Description	Working Case	Problem case
Comparison			
Tangential Bug (Page 23)	• improvement to Bug 2: shorter path using a range sensor with a 360° infinite orientation resolution • Moves on straight line towards the goal until obstacle is sensed • Moves along the obstacle in distance of sensing radius		

3.3. APPLICATION OF BUG STRATEGIES FOR CONFIGURATION SPACES CH 3.2.2

- grid-based representation of the configuration space
- grid on the torus: for each point on the grid, perform collision check
- left: two-joint arm in a planar workspace, right: configuration space

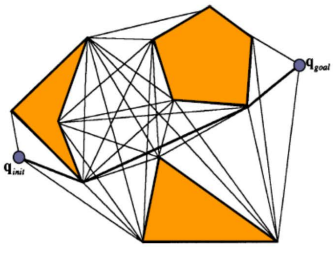
4. Classical path planning approaches

- Construct data structure once and then use data structure to plan subsequent paths more quickly
- **Topological maps —> Roadmaps**: Represent the connectivity of the free space by a network of 1-D curves (connected nodes) and edges represent an adjacency relationship between nodes
- **Geometric models —> Cell Decomposition**: Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells
- **Occupancy grids —> Potential field**: Define a potential function over the free space that has a global minimum at the goal and follow the steepest descent of the potential function

4.1. ROADMAPS CH 5

- A union of one-dimensional curves is a roadmap if for all q_{start} and q_{goal} in free space that can be connected by a path, the following properties hold: Accessibility, Departability, Connectivity

4.1.1. VISIBILITY GRAPH METHOD CH. 5.1



- Defining characteristics: nodes share an edge if they are within line of sight of each other and all points in the robot's free space are within line of sight of at least one node on the visibility map —> properties of accessibility and departability, but connectivity must be proven
- defined in a 2-D polygonal configuration space
- Nodes: v_i of the visibility graph include the start q_{init} and goal location q_{goal} and

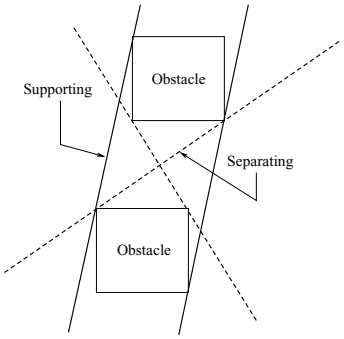
all the vertices of the configuration space obstacles (not part of the offline graph, only added online)
- Graph edges: are straight lines that connect two line-of-sight nodes

Algorithm

```
Input: q_init, q_goal, polygonal obstacles
Output: visibility graph G

for every pair of nodes u,v
      if segment (u,v) is an obstacle edge then insert edge (u,v) into G;
      else
            for every obstacle edge e
                  if segment (u,v) intersects e
                        go to (1);
            insert edge (uv) into G.
```

- Computation time in $O(n^2)$ space:
  - Simple algrorithm: $O(n^3)$
  - More efficient: Rotational Sweep $O(n^2 \log(n))$, Optimal $O(n^2)$

Simplified Visibility Graph
- all edges that do not lie on a supporting or separating line are removed
- It's beneficial because it has fewer edges making the search for the shortest path more efficient
- A supporting line is tangent to two obstacles such that both obstacles lie on the same side of the line.
- A separating line is tangent to two obstacles such that the obstacles lie on opposite sides of the line.



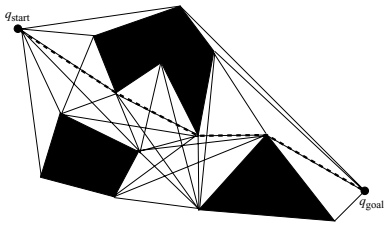


Figure 5.4 The thin solid lines delineate the edges of the visibility graph for the three obstacles represented as filled polygons. The thick dotted line represents the shortest path between the start and goal.

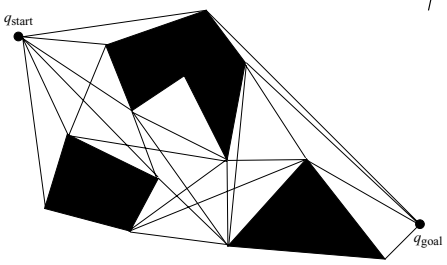
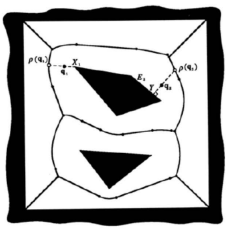


Figure 5.6 Reduced visibility graph.

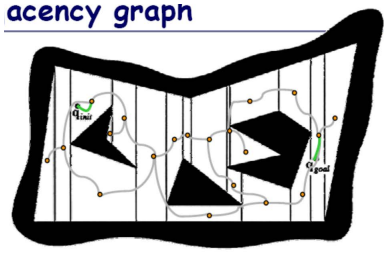
4.1.2. VORONOI DIAGRAM CH. 5.2



- Generate paths that maximizes clearance: set of points where the distance to the two closest obstacle is the same
- applicable mostly to 2-D configuration spaces
- Space $O(n)$ —> Running time $O(n * \log(n))$
- Bushfire method can be used to construct the Voronoi Diagram
  - Input: grid of zeros for free space, ones for obstacles
  - Output: discrete map with each pixel has value equal to the distance to the closest point on the closest obstacle

4.2. CELL DECOMPOSITION CH 6
- Exact cell decomposition: The free space F is represent by a collection of non-overlapping simple cells whose union is exactly F
- Examples of cells: trapezoids, triangles

4.2.1. TRAPEZOIDAL DECOMPOSITION CH 6.1



- vertical line sweeps through space: if hit corner —> separation line
- connect cells by a straight line segment that does not intersect any obstacle —> connecting the midpoints of the vertical extensions to the centroids of each trapezoid
- Running time: $O(n \log(n))$ by planer sweep in space $O(n)$

- mostly for 2-D configurations
- Nodes: cells
- Edges: There is an edge between every pair of nodes whose corresponding cells are adjacent
- **Boustrophedon Decomposition:** extension of algorithm to avoid unnecessary nodes (many small cells are formed that can seemingly be aggregated with neighboring cells)—> keep only lines on critical points (only lines where you have to make decisions)

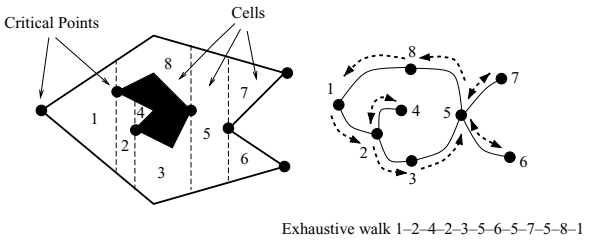


Figure 6.10 The boustrophedon decomposition of a space and its adjacency graph. The nodes represent the cells and the edges indicate the adjacent cells. An exhaustive walk on the graph is generated.

- **Canny's Roadmap algorithm** for round obstacles: Cell boundaries are placed at critical points, which are points where the tangential line touches the object

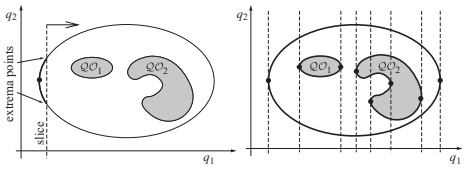


Figure 5.23 Bounded two-dimensional environment with two obstacles $\mathcal{QO}_1$ and $\mathcal{QO}_2$. The left figure contains a single slice, represented by a dashed line, and a partially constructed silhouette. The right figure contains the complete silhouette and slices passing through all critical points.

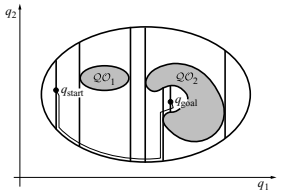


Figure 5.24 Complete silhouette curves traced out with solid lines. A path from start to goal is denoted as a thin curve.

- combinatorial motion planning —> combinatorial partitioning of the environment —> makes algorithms complete

4.2.2. THE HALTING PROBLEM

- Is it possible to determine whether a program will stop?
  - This problem is undecidable by using another algorithm (a computer/turing machine cannot really tell)
- Sketch of proof (via contradiction)
  - Suppose an algorithm H_{TM} can decide whether an algorithm will halt
  - Construct a new algorithm A_{TM} and run on input A_{TM}:
    - If A_{TM} halts (, then $H_{TM}(A_{TM})$ == true, A_{TM} should loop forever
    - If A_{TM} does not halt, then $H_{TM}(A_{TM})$ ==false, A_{TM} should halt
    - Both are contradictions ➔ H_{TM} cannot exist!

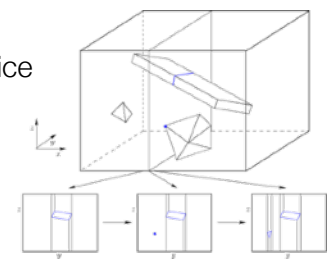
A_{TM} : on input algorithm X
if $H_{TM}(X) ==$ true then loop forever;
else return true;

- Implications: Not all algorithms are complete —> Solution: do not use a full Turing Machine to do the testing
- Combinatorial algorithms are complete (because every single point in C_free is covered, motivates the development to of combinatorial methods for higher dimensions)
- Combinatorial algorithms are algorithms that deal with *combinatorial structures*, which are sets, ordered n-tuples, and any structures that can be built from them, like graphs.

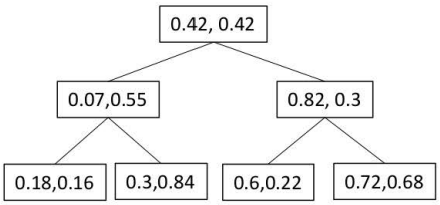
- Example: Piano Mover's Problem
  - Use autonomous robots to move a paint around without hitting other objects
  - Not immediately clear, if such problems are solvable
  - Various algorithms tried it and finally it was solvable with Collin's cylindrical algebraic decomposition

4.2.3. IMPLEMENTATION OF VERTICAL CELL DECOMPOSITION IN A HIGHER DIMENSION 6.3.3

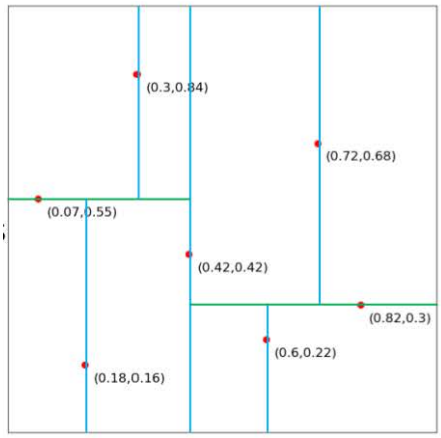
- VCD can be extended to any dimension by recursively applying the sweeping idea
- requirements: C_{obs} is piecewise linear —> C_{obs} is represented as a semi-algebraic model for which all primitives are linear (e.g. polyhedral robot that can translate in R^3 and the obstacles in W are polyhedra)
- Steps:
  1. Pick a dimension x_i
  2. **Plane Sweep:** Sweep the configuration space using a (n-1) dimensional plane orthogonal to x_i axis (unique hyperplane). It produces a polygonal slice of C_{obs} (see three slices in figure).
  3. **Line Sweep**: Sweep a line along the y-axis through the subdivided



volumes and leave boundaries at critical point
4. Repeat 2. to each of the (n-1) dimensional slices

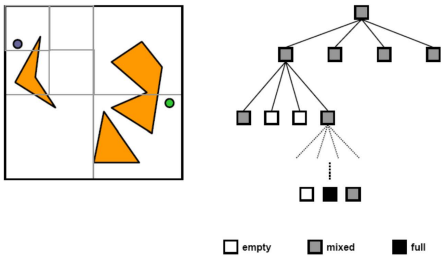


- The 2-cells in a slice are actually slices of 3-cells in the 3D decomposition. The only places in which these 3-cells can critically change is when the sweeping plane stops at some x value.
- The roadmap is constructed by placing a sample point in the center of each 3-cell and 2-cell. The vertices are the sample points, and edges are added to the roadmap by connecting the sample points for each case in which a 3-cell is adjacent to a 2-cell.
- This same principle can be extended to any dimension, but the applications to motion planning are limited because the method requires linear models (or at least it is very challenging to adapt to nonlinear models; in some special cases, this can be done).
- Canny's Algorithm in higher dimensions
  - basic idea: project directed from R^n to R^2 many times
  - drops computation time to singly exponential *(a^n)*
  - still not very practical but theoretically much better



4.2.4. K-D TREES

- A data structure for storing points in k dimensions
- Assume a tree like structure
- Useful for finding points with certain properties
- Can be used for solving 1-NN
- Construction of a k-d tree for n points
  - Pick dimension i, pick a point with coordinates x = (x_1, … , x_i, … , x_k)
  - Split the points based on x_i (greater or less than)
  - Repeat the above two steps recursively: Increase i (modulo k) each time (I.e., pick a new dimension each time)
  - Depth: *log(n)* if balanced
  - Construction takes *O(k n log(n))* time
  - Each dimension needs sorting *~O(n log(n))*
  - Can speed up to *O(n log(n))*
- Very efficient, but: relies on complete knowledge about environment, if higher level node is removed —> reconstruct the tree again, changes in spaces —> reorder everything

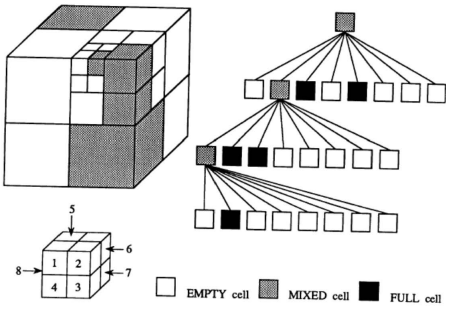


4.2.5. APPROXIMATE CELL DECOMPOSITION

- F is represented by a collection of non overlapping cells whose union is contained in F
- does not need to be balanced, adapts to the structure

<u>Quadtree decomposition</u>

- first node: whole space
- divide space into subspaces —> subsides
- only mark if node is empty, mixed or full
- graph only extends to lower entries if it has a mixed entry
- if obstacle is removed from graph —> just remove entries in the tree up to the node where obstacle was removed
- boundaries are fix (don't adapt to the obstacles)



<u>Octree decomposition</u>

- extend to higher dimensions: subdivide volume into 8 subvolumes
- not as optimal, but don't need to be rebalance if information is removed from system

4.3. POTENTIAL FIELD

- mostly used for mobile robots

- Define a potential function over the free space that has a global minimum at the goal and follow the steepest descent of the potential function

4.3.1. FUNCTION OF THE POTENTIAL FIELD

- potential field: scalar function over the free space
- Navigation: applies a force proportional to the negated gradient of the potential field —> combination of repulsive and attractive forces
- Navigation Function is a ideal potential field that
  - has a global minimum at the goal
  - has no local minima
  - grows to infinity near obstacles
  - is smooth
- simplest function would be cone
- high gradient = high velocities, low gradient = low velocities

Designing the free space function

- u(x) = k*|x| —> not a good function because velocity will not slow down at the goal and will overshoot
- u(x) = x^2 —> better function because velocity is proportional to the gradient of the function, no constant gradient, but system will slow down, while approaching the goal ⚡ robot velocity may (very far away from the goal) reach and exceed the maximum velocity of the function due to high slope
- Switch from quadratic to line function once the maximum velocity is reached:

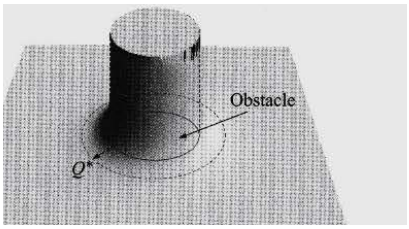
$$U_{\text{att}}(q) = \begin{cases} \frac{1}{2}\zeta d^2\left(q, q_{\text{goal}}\right), & d\left(q, q_{\text{goal}}\right) \le d^*_{\text{goal}} \\ d^*_{\text{goal}}\zeta d\left(q, q_{\text{goal}}\right) - \frac{1}{2}\zeta\left(d^*_{\text{goal}}\right)^2, & d\left(q, q_{\text{goal}}\right) > d^*_{\text{goal}} \end{cases}$$

$$\nabla U_{\text{att}}(q) = \begin{cases} \zeta\left(q - q_{\text{goal}}\right), & d\left(q, q_{\text{goal}}\right) \le d^*_{\text{goal}} \\ d^*_{\text{goal}}\zeta, & d\left(q, q_{\text{goal}}\right) > d^*_{\text{goal}} \end{cases}$$

- q_{goal}: position of the goal
- d_{goal}: distance from goal
- d_{goal}^*: threshold distance from the goal where the planner switches between conic and quadratic potentials
- if gradient is 0: minimum, saddle point or maximum —> calculate next derivative:
  - < 0 —> minimum,
  - > 0 —> maximum
  - if saddle point: push of the robot would keep it rolling so that it doesn't stop at a step (saddle point)

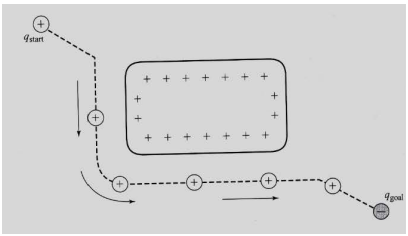
Designing the obstacle function

- Obstacle = repulsive force: keeps robot away from obstacle, the closer the robot is to an obstacle, the stronger the repulsive force should be

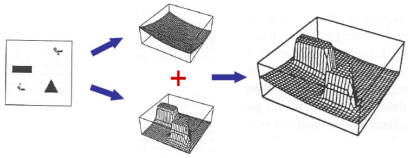


$$U_{\text{rep}}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{D(q)} - \frac{1}{Q^*}\right)^2, D(q) \le Q^* \\ 0, D(q) > Q^* \end{cases}$$

$$\nabla U_{\text{rep}}(q) = \begin{cases} \eta\left(\frac{1}{Q^*} - \frac{1}{D(q)}\right)\frac{1}{D^2(q)}\nabla D(q), D(q) \le Q^* \\ 0, D(q) > Q^* \end{cases}$$

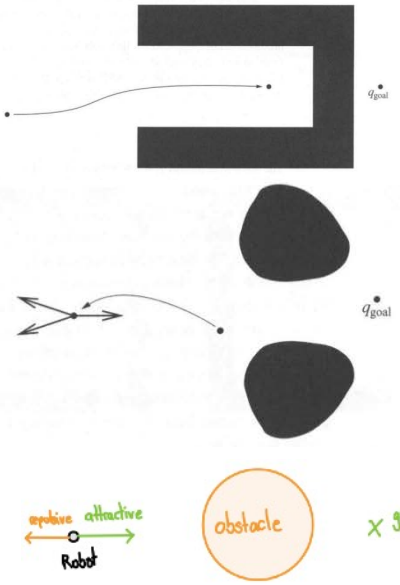


- Q*: distance from the obstacle that define how large the influence of the obstacle on the robot ist
- eta:
- D(q): distance to the obstacle
- goes to infinity close to the obstacle



- Resulting force = combination of attractive and repulsive field
- can start with empty map: guided by the force of the goal and the obstacles

Local Minima Problem Ch 4.4



- local minima can appear if a specific form if a obstacle exists
- **U-shape:** the closer the robot comes to the wall, the higher will be the repulsive form —> attractive = repulsive force —> stuck in the local minimum
  - even if slightly moved to left and right: robot will stay at the place
  - —> increase η or Q* (but robot will not reach goal in the U-shape)
  - solution: run the algorithm and if goal is not reached, increase Q* and run again
- **Two obstacles**: at same point repulsive forces of the two obstacles = attractive force —> robot won't continue despite if the obvious path
  - not a sattle point because if we move slightly the robot, will not move further
  - solution: decrease Q*
- **Saddle point**: no local minima, because slightly push to the right or left will create a s small motion so that the robot can go around the obstacle —> no need to change the parameters but just slightly movement

Summary: Potential field method (Khatib 1986)

- may not find a global solution but can find specific solution if parameters are changed

$$F_{att} = -k_{att}(x - x_{goal})$$

$$F_{rep} = \begin{cases} k_{rep}\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2}\frac{\partial \rho}{\partial x} & \text{if } \rho \le \rho_0, \\ 0 & \text{if } \rho > \rho_0 \end{cases}$$

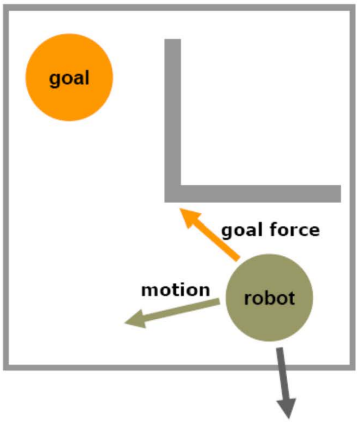
k_{att}, k_{rep} : positive scaling factors

x : position of the robot

ρ : distance to the obstacle

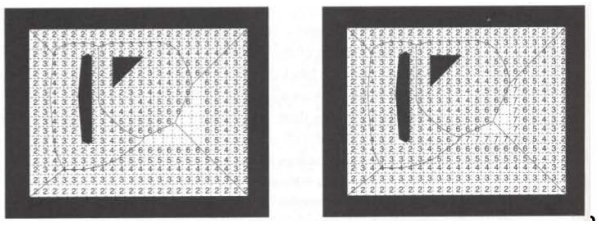
ρ_0 : distance of influence

[Khatib, 1986]



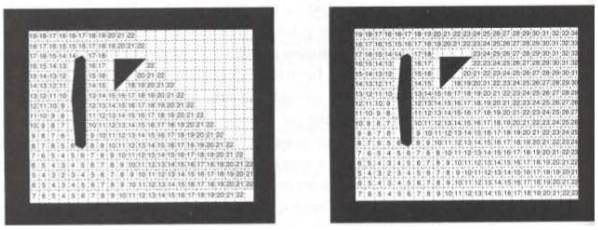
4.3.2. BUSHFIRE ALGORITHM CH 4.3.2

- method to compute distance on a grid (map representation)
- Implementation:
  - all cells near neighbored to obstacles labeled with 2
  - all empty cells neighbored to 2 are labeled with 3
  - iterate as long as there are empty cells
- Four (respects Manhattan distance function) or eight point connectivity
- Gradient: vector which point to the lowest neighbor —> compute repulsive function



4.3.3. WAVE FRONT PLANNER CH 4.5

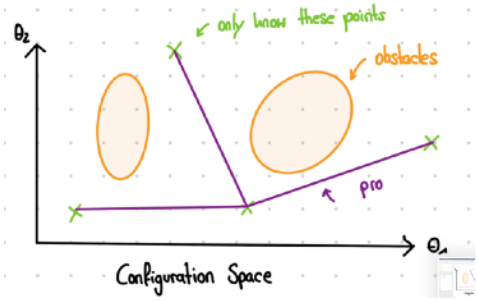
- simple solution to local minima problem
- Implementation:
  - goal labeled with two
  - all neighbors to goal labeled with 3
  - …
  - determine path via gradient descent start from the start: go to pixel with lower value



5. Sampling-Based Algorithms Ch7

Difficulty with classical approaches:

- Running time increases exponentially with the dimension of the configuration space: For a d-dimension grid with 10 grid points on each dimension, there 10^d grid cells
- Several variants of the path planning problem have been proven to be PSPACE-hard
- not feasible to have complete knowledge about all obstacles in the environment
- Classical approaches not possible in > 3D



—> solution: sampling based methods — instead of representing C_free explicitly and globally, we instead "probe" the space locally, as necessary

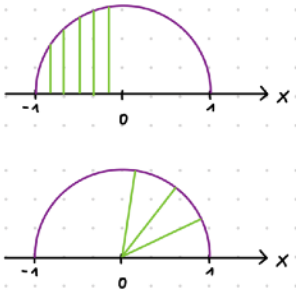
- Sampling-based methods employ a variety of strategies for generating samples (collision-free configurations of the robot) and for connecting the samples with paths to obtain solutions to path-panning problems
- Not sample entire space, but rely on very few samples in the free space

5.1. CHARACTERISTICS OF SAMPLING-BASED PLANNERS

- No attempt to explicitly construct the boundaries of the configuration space obstacles or represent cells of the free space, but rely on whether a configuration is in collision with obstacle or not
- Heuristic algorithm: Unreliable (Example: potential field)
- Give up completeness:
  - Completeness requires that the planner always answers a path-panning query correctly. A complete algorithm finds a path if one exists and reports no otherwise (Example: Canny's roadmap method)
  - Complete planners cannot be implemented for robots with more than 3 degrees of freedom due to their high combinatorial complexity
  - Weaker form of completeness: *probabilistic completeness* —> If there is a solution path, the algorithm will find it with high probability

5.2. SAMPLING STRATEGY: CREATE A UNIFORM DISTRIBUTION

- Straight interval
  - Pick a random interval on the unit interval [0, 1] —> create unit square/cubes
  - Scale on unit interval: $x_1 = (a-b)x + a$
- Circle:
  - 1. Method: Pick x uniform at random from [-1,1] , Set $y = \sqrt{1-x^2}$ —> Intervals of same widths are sampled with equal probabilities
  - 2. Method: Pick θ uniformly at random from [0, 2 pi], Set $x = \cos\theta$ and $y = \sin\theta$ —> Circular arcs of same angles are sampled with equal probabilities.
  - Both are uniform in some sense. For sampling orientations in 2-D, the second method is usually more appropriate.

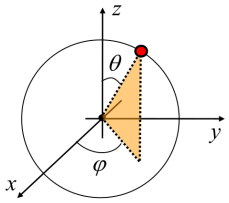


- Sphere:
  - Spherical patches of same areas are sampled with equal probabilities.
  - Suppose U_1 and U_2 are chosen uniformly at random from [0,1].
- Unit quaternion:

$$\left(\cos\xi/2, n_x \sin\xi/2, n_y \sin\xi/2, n_z \sin\xi/2\right) \text{ with } n_x^2 + n_y^2 + n_z^2 = 1$$

$$\begin{cases} n_z = U_1 \\ n_x = R\cos(2\pi U_2) \\ n_y = R\sin(2\pi U_2) \end{cases}$$

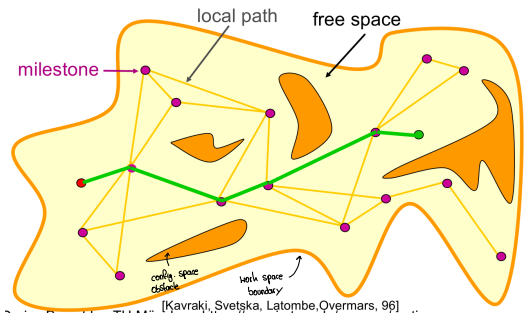
where $R = \sqrt{1-U_1^2}$



5.3. CLASSIC MULTIPLE QUERY PRM (PROBABILISTIC ROADMAP)

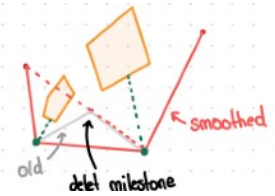
- Probabilistic Roadmaps for Path Planning in High- Dimensional Configuration Spaces, L. Kavraki et al., 1996.
- Assumption: Obstacles are static —> Roadmaps don't need to be modified
- Multiple query: No knowledge a priori about start and end point, can be anywhere —> Many queries to be processed in the same environment
- Example: Navigation in a static virtual environment, Robot manipulator arms in a work cell

- Planning divided into two phases:
    1. learning phase, during which a roadmap in free space is built and is stored as a graph
    2. query phase, during which user-defined query configurations are connected with the precomputed roadmap

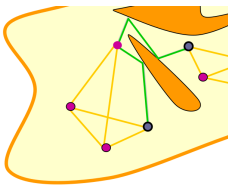


Implementation of Learning Phase

- Connection Step
    - **Set milestones**: Create n uniform distribute milestones: If milestone is in free space —> keep it, if outside of workspace or in the obstacle —> try again
    - **Find local paths**: If there is no collision along the local path, connect two milestones with straight line doing dense sampling (make sure there is no collision with many tests). This leads to a significant reduction of collision tests to build a roadmap, finding a solution from start to goal via green line (in the image above).
    - **Smoothing the resulting paths**: milestone with the maximum distance from line is the new milestone and delete the others if path is collision free.
    - If collision: repeat and use milestone with highest distance to the line
    - —> fewer velocity changes —> more efficient BUT homotopic paths needs to be preserved



- Expansion Step (resampling)
    - easy scenes give large set of milestones gives well connected roadmap ⚡ more constrained situations: roadmap consists of several small connected components
    - expand milestones in those regions: If you hit obstacle, continue in random direction (random walk) with max. distance L and then create a new milestone —> still keeps samples low but helps us to get samples in the problematic areas
    - Could solve problem to get through narrow passages

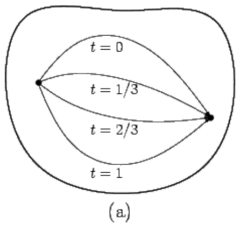


Uniform sampling

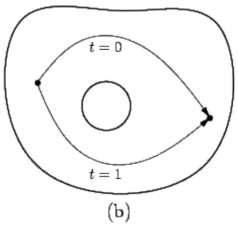
```
Input: geometry of the moving object & obstacles
Output: roadmap G = (V, E)

V <— Ø and E <— Ø,
repeat
      q < a configuration sampled uniformly at random from C.
      if CLEAR(q)then
      Add q to V.
            N, <a set of nodes in V that are close to q.
            for each q'e N,, in order of increasing d(q,q')
                  if LINK (q',q) then
                  Add an edge between q and q' to E.
```

Homotopic paths

- two paths τ and τ' with the same endpoints are homotopic if one can be continuously deformed into the other:
  $h : [0,1] \times [0,1] \rightarrow F$ with $h(s,0) = \tau(s)$ and $h(s,1) = \tau$
- A homotopic class of paths contains all paths that are homotopic to one another
- During the deformation, path should not intersect with any obstacle
- Homotopy ensures that after the path correction, the space topology gets visited from the same side as usually planned. While deforming, no other object may be intersected. It is important for a surveillance robot, which needs to observe e.g., specific walls to check for paintings.



Homotopic example



non-homotopic paths

Query processing in an online step

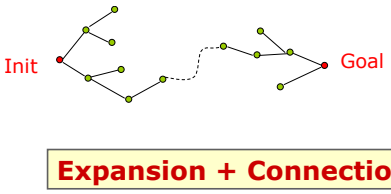
- During the query phase, paths are to be found between arbitrary input start and goal configurations, using the roadmap constructed in the learning phase.
- Connect q_{init} and q_{goal} to the roadmap
- Start at q_{init} and q_{goal}, Perform a random walk, and try to connect with one of the milestones nearby
- Try multiple times
- If a path is returned, the answer is always correct.
- If no path is found, the answer may or may not be correct. We hope it is correct with high probability.

Why does it work? —> Intuition
- A small number of milestones almost cover the entire configuration space

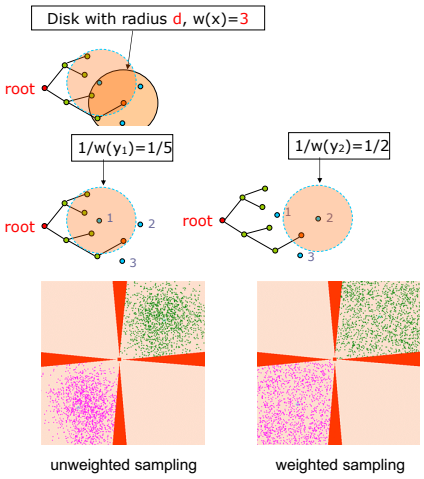
Drawbacks of Multi-Query Methods
- PRM is known as „multi-query“ sampling-based method because after initial roadmap is built, multiple quires can be executed on the same roadmap
- lot of useless information stores if we want to run a single query —> developed single-query methods

5.4. SINGLE QUERY PRM CH7.2

- Multiple Query: start and end point anywhere in the free space —> create many paths in the space to find the shortest —> create a roadmap that captures the connectivity of the free space and then answer multiple user-defined queries very fast
- Single Query: is there a possibility to connect start and end point? use the environment only once, eventually not necessary to explore whole space —> don't focus on the exploration of the entire free space, just if its possible to find a path
- Implementation:
  - 1. Grow two trees from Init position and Goal configurations.
  - 2. **Expansion**: Randomly sample nodes around existing nodes.
  - 3. **Connection**: Connect a node in the tree rooted at Init to a node in the tree rooted at the Goal (if distance is below a threshold L and x in Init tree can see y in Goal tree can see)



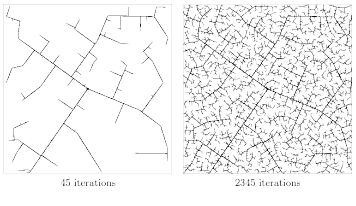
- ⚡ if nodes are distributed randomly they could not expand but go back —> specify the Expansion
- Specified Expansion:
  - 1. Pick a node x with probability 1/w(x).
  - 2. Randomly sample k points around x.
  - 3. For each sample y, calculate w(y), which gives probability 1/w(y)
  - —> high probability for nodes with few nodes in the convergence circle
  - Sampling distribution
    - Weight $w(x)$ = no. of neighbors
    - Roughly $\Pr(x) \sim 1 / w(x)$
  - —> place nodes close to the obstacle: possible to go trough narrow passages (right figure)
- Termination Condition: The program iterates between **Expansion** and **Connection**, until
  - two trees are connected —> success
  - max number of expansion & connection steps is reached —> no success



5.4.1. RAPIDLY-EXPLORING RANDOM TREES (RRT, BY LAVALLE & KUFFNER) CH 7.2.2

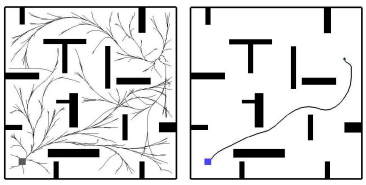
- single-query planning algorithm that efficiently covers the space between q_{init} and q_{goal}, probabilistically complete, not a roadmap, but a tree

Construction
- grows a tree rooted at the starting configuration by using random samples from the search space. As each sample is drawn, a connection is attempted between it and the nearest state in the tree. If the connection is feasible, this results in the addition of the new state to the tree.
- With uniform sampling of the search space, the probability of expanding an existing state is proportional to the size of its Voronoi region. As the largest



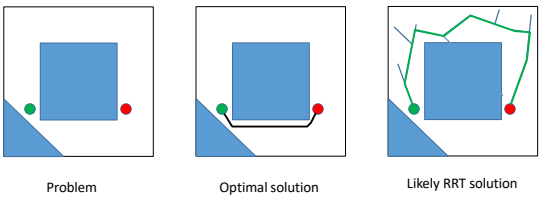
Voronoi regions belong to the states on the frontier of the search, this means that the tree preferentially expands towards large unsearched areas
- Extend the tree from existing nodes in random directions with decreasing distances. Stop edge close to the obstacle
- If obstacle: Stop at the border of the obstacle and move the node to the boundary —> nodes very close to obstacles, good for narrow passages
- Growth limit Δq: limiting the length of the edge —> growth limited to this parameter, if no growth limit: explore whole space
- generate nodes close to the goal —> preference of growing to the goal

Kinodynamic RRT

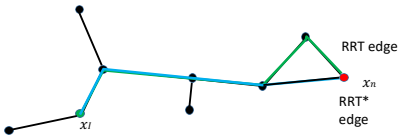


- Besides solving single-query problems faster, Kinodynamic RRT is suitable for solving problems for systems with **differential constraints** (e.g. non-holonomic systems such as cars) ⚡ Standard PRM and RRT cannot be applied
- respect the differential constraints —> Need to compute path more carefully: needs to solve a boundary value problem (differential equations)
- —> Paths are not lines anymore, but curves (in germany: clothoids (curvature is getting narrower))

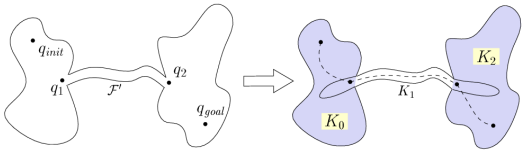
When does PRM and RRT work well? —> Re-Wiring



- PRM: good for reasonably high-dimensional problems because problems are „easy“ / ε, α, β - expansive
- but its also possible to construct instance to make PRM/RRT produce long paths —> need to keep re-wiring the graph structure: RRT* (Karaman & Frazzoli)
  - for each new sample x_n, check its log(n) neighborhood
  - if there are better paths from x_1 to x_n, pick that path
  - RRT* is an asymptotically optimal sampling based algorithm —> As the number of samples goes to infinity, an optimal path from x_I to x_G



- Problem: Robot traveling between two spaces —> Many samples wasted with uniform samples because many samples are not part of the free space —> Extensions: Three possibilities
  - 1. Divide space into subspaces (expansive components)
  - 2. Use geometric transformations to increase the expansiveness of a free space, *e.g.*, smaller variance of uniform samples in narrow passages
  - 3. Integrate the new planner with other planner for multiple-query path planning problems.



5.5. ANALYSIS OF PRM: EXPANSIVE SPACES CH 7.4.2

- Path Planning in Expansive Configuration Spaces, D. Hsu, J.C. Latombe, & R. Motwani, 1999.
- How to determine the number of configurations that should be generated to ensure that, with probability exceeding a given constant, each roadmap is connected
- Issues of PRM: Coverage and Connectivity —> find a method to find properties and derive the correct number of milestones
- Make sure that
  - **coverage is adequate:** It means that milestones are distributed such that almost any point of the configuration space can be connected by a straight line segment to one milestone. —> uniform distribution
  - **milestones are connected**: There should be a one-to-one correspondence between the connected components of the roadmap and those of F —> not two separated graphs
    - Connectivity is difficult to capture when there are narrow passages ⚡ narrow passages are difficult to define
  - Characterize coverage & connectivity? —> α, β, ε - Expansiveness in order to capture the complexity of a configuration space due to narrow passages

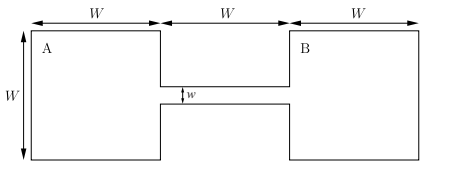
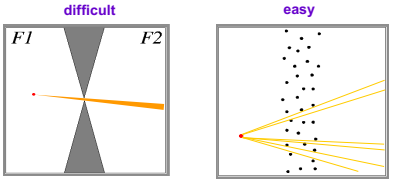
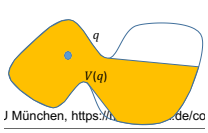
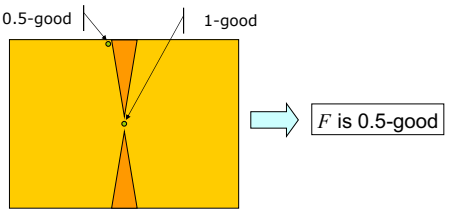


Figure 7.19 An example of an $(\epsilon, \alpha, \beta)$-expansive $\mathcal{Q}_{free}$ with $\epsilon, \alpha, \beta \approx w/W$. The points with the smallest ϵ are located in the narrow passage between square A and square B. Each such point sees only a subset of $\mathcal{Q}_{free}$ of volume approximately $3wW$. Hence $\epsilon \approx w/W$. A point near the top right corner of square A sees the entire square; but only a subset of A, of approximate volume wW, contains points that each see a set of volume $2wW$; hence $\alpha \approx w/W$ and $\beta \approx w/W$. (From Hsu [192].)

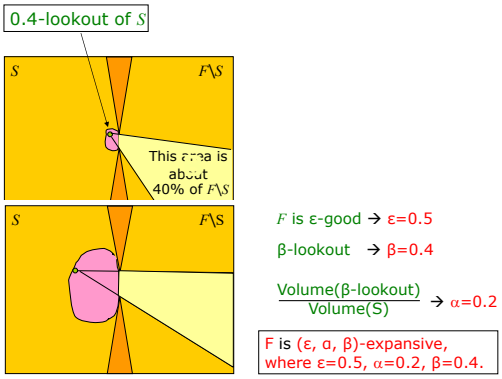


- Definition: Visibility set S of q
  - $\text{reach}(S) = \{x \in F \mid \exists y \in S \text{ such that } \overline{xy} \subset F\}$
  - All configurations that can be connected to q by a straight line path in F
  - All configurations seen by q
- Definition: ϵ-good
  - $\epsilon \leq \frac{S}{F}$
  - $F = C_{\text{free}}$ is ε-good if given any point $q \in F$, $\mu(V(q)) \geq \varepsilon\mu(F)$ (μ measures a volume)
  - Every free configuration sees at least ϵ fraction of the free space, ϵ in (0,1]



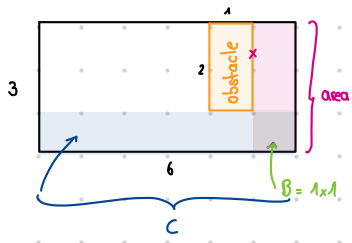


- Definition: lookout of a subset S
  - Subset of points in S that can see at least β fraction of $F \backslash S$, β is in (0,1].
  - $\beta = \frac{B}{F/S}$
  - $\alpha = \frac{A}{S}$
  - A and B as big as possible: they will reduce the number of try outs to get a fully connected graph



- Definition (ε, α, β)-expansive: The free space F is (ε, α, β)-expansive if
  - 1. free space F is ε-good
  - 2. for each subset S of F, its β-lookout is at least α fraction of S
  - The first condition of definition ensures that a certain fraction of F is visible from any configuration in F. The second condition ensures that each subset S ⊆ F has a large lookout set. It is reasonable to think of S as the union of the reachability sets of a set V of points. Large values of α and β indicate that it is easy to choose random points from S such that adding them to V results in significant expansion of S . This is desirable since it allows for a quick exploration of the entire space.
  - bigger ε, α and β —> lower cost of constructing a roadmap with good connectivity and coverage

3.2 b) Explain briefly adding for each parameter a small drawing how expansivness of a space can be parametrized with the (ε,α,β)?



(A=S in lecture) area A the point can see

must be seen from any point in B

ε: minimum visibility → pick point that can see the smallest area of the free space

$\varepsilon = \frac{A}{F} = \frac{3}{3 \cdot 6 - 2 \cdot 1} = \frac{3}{18-2} = \frac{3}{16}$

α: region in confined space which allows me to look in the free space

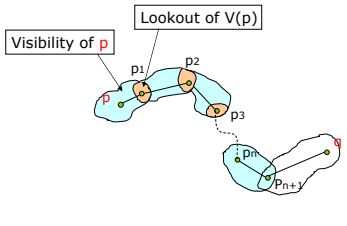
lookout region $\alpha = \frac{B}{S} = \frac{1}{3}$

β: ratio of region that can connect to visible space where I'm connecting

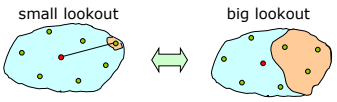
$\beta = \frac{C}{F-S} = \frac{6}{16-3} = \frac{6}{13}$

- Theorem: A roadmap of $n = \frac{8 \ln(8/\varepsilon\alpha\gamma)}{\varepsilon\alpha} + \frac{3}{\beta}$ uniformly-sampled milestones has the correct connectivity with probability at least 1 - gamma (gamma: failure rate —> exponential dependence of success) (each subgraph is a connected graph)

- Definition: Linking sequence



- p_{n+1} is chosen from the lookout of the subset seen by $p, p_1, \ldots, p_n$
- place a point p1 in the lookout region of the parent point —> defines how much of the volume is visible
- C-Space with a larger lookout set has a higher probability of constructing a linking sequence
- Lemma: In an expansive space with large ε,α, and β, we can obtain a linking sequence that covers a large fraction of the free space, with high probability.



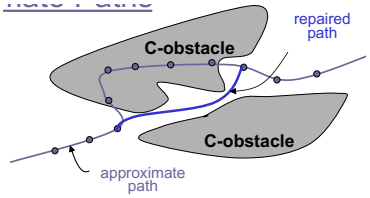
- **Theorem 1**: Probability of achieving good connectivity increases exponentially with the number of milestones (in an expansive space).
  - If (ε, α, β) decreases → then need to increase the number of milestones (to maintain good connectivity)
- **Theorem 2**: Probability of achieving good coverage, increases exponentially with the number of milestones (in an expansive space).
- Probabilistic completeness: In an expansive space, the probability that a PRM planner fails to find a path when one exists goes to 0 exponentially in the number of milestones (~ running time).

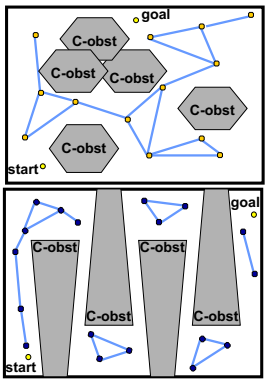
Summary
- Main result: If a C-space is expansive, then a roadmap can be constructed efficiently with good connectivity and coverage.
- Limitation in practice: It does not tell you when to stop growing the roadmap. A planner stops when either a path is found or max steps are reached.

5.6. PRM CONCLUSION

- A relatively small number of milestones and local paths are sufficient to capture the connectivity of the free space → Exponential convergence in expansive free space (probabilistic completeness)
- Checking sampled configurations and connections between samples for collision can be done efficiently → Hierarchical collision checking
- Problem: Even with the best sampling methods, roadmaps may not contain valid solution paths
  - may lack critical cfgs in narrow passages
  - may contain approximate paths that are 'nearly' valid

- Repairing/Improving Approximate Paths
  - Create initial roadmap Extract *approximate path P*
  - Repair P (push to C-free)
  - Focus search around P
  - Use OBPRM-like techniques



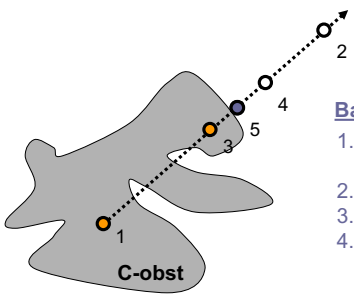


Pros:
- PRMs are *probabilistically complete*
- PRMs apply easily to high-dimensional C-space
- PRMs support fast queries w/ enough preprocessing
- Many success stories where PRMs solve previously unsolved problems

Cons:
- PRMs don't work as well for some problems: unlikely to sample nodes in *narrow passages,* hard to sample/connect nodes on constraint surfaces
- Our work concentrates on improving PRM performance for such problems.

5.7. OBPRM: OBSTACLE-BASED PRM

- To Navigate Narrow Passages we must sample in them: most PRM nodes are where planning is easy (not needed)
- Idea: Can we sample nodes near C-obstacle surfaces?
  - we cannot explicitly construct the C-obstacles but we do have models of the (workspace) obstacles
  - 1. Find a point in S's C-obstacle (robot placement colliding with S)
  - 2. Select a random direction in C-space
  - 3. Find a free point in that direction
  - 4. Find boundary point between them using binary search (collision checks)



  - Note: we can use more sophisticated heuristics to try to cover C-obstacle
- Not for dynamic scenes

6. Kalman Filtering Ch 8

In this chapter, we begin to consider cases for which the robot's knowledge of the world derives from measurements provided by imperfect, noisy sensors.
Improve the knowledge about the world by moving the robot in the environment.
Based on G. Welch & G. Bishop, "An Introduction to the Kalman Filter"

Why Kalman Filtering? Kalman Filter is a tool that predicts values using a bunch of mathematical equations under the assumptions that input data is in the form of Gaussian distribution and then followed by the application of linear equations to that Gaussian distribution. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown.
Kalman filter (KF) techniques have been successfully used to predict and estimate errors in position of linear systems like ships.

What should we do if the sensors do not agree on the measurement? The mean of the Gaussian with smaller sigma should be trusted more. Weight the mean according to sigma and combine the measurements.

Idea of the Kalman Filter:
- The main idea behind the Kalman filter is that you do not just have an estimate for a parameter x but also have some estimate for the uncertainty in your value for x
- This is represented by the variance/covariance of the estimate P_x
- There are many advantages to this, as it allows you a means for estimating the confidence in your robot's ability to execute a task (e.g. navigating through a tight doorway)
- In the case of the KF, it also provides a nice mechanism for optimally combining data over time
- This optimality condition assumes we have **linear models**, and the error characteristics of our sensors can be modeled as **zero-mean, Gaussian noise**

6.1. DATA FUSION EXAMPLE

- Use code from Team 1 and Team 2 to obtain two different measurements $Z = [z_1, z_2]^T$ for the range r to a beacon
- Variance in each of these sensor measurements is R_1 and R_2, respectively
- Q: How should we fuse these measurements in order to obtain the "best" possible resulting estimate for r ?
- We'll define "best" from a lest-squares perspective...
- We have 2 measurements that are equal to r plus some additive zero-mean Gaussian noise v_1 and v_2

$$z_1 = r + N\left(0, R_1\right) = r + v_1$$
$$z_1 = r + N\left(0, R_1\right) = r + v_1$$

- We want to fuse these measurements to obtain a new estimate for the range r hat
- Using a weighted least-squares approach, the resulting sum of squares error will be $e = \sum_{i=1}^{n} w_i \left(\hat{r} - z_i\right)^2$
- Minimizing the error: $\frac{\partial e}{\partial \hat{r}} = \frac{\partial}{\partial \hat{r}} \sum_{i=1}^{n} w_i \left(\hat{r} - z_i\right)^2 = 2 \sum_{i=1}^{n} w_i \left(\hat{r} - z_i\right) = 0 \Rightarrow \hat{r} = \frac{\sum_{i=1}^{n} w_i z_i}{\sum_{i=1}^{n} w_i}$

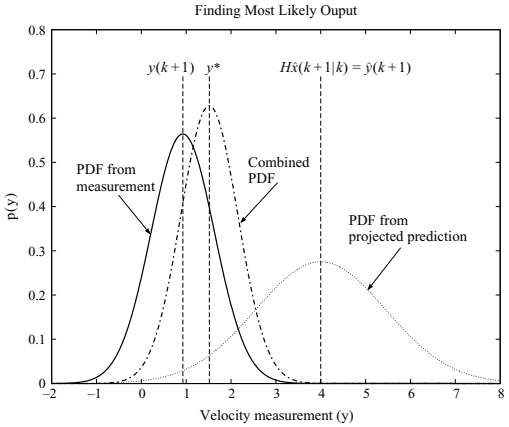


Figure 8.6 Measurements and predictions are then merged. The PDF plotted with the dot-dashed line results from the combination of the measurement PDF and the PDF of the prediction projected into output space, where the combination is computed using theorem 8.2.1. The most likely output y^* is the value at which this combined distribution reaches its peak.

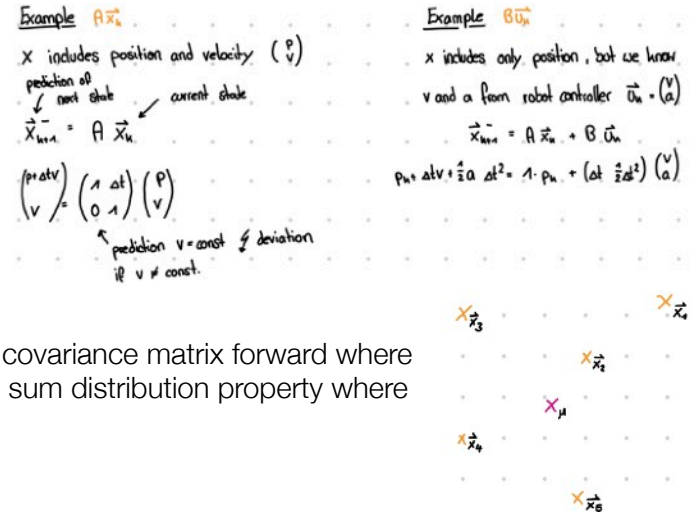
- We obtain: $\hat{r} = \frac{\frac{z_1}{R_1} + \frac{z_2}{R_2}}{\frac{1}{R_1} + \frac{1}{R_2}} = \frac{R_2}{R_1 + R_2} z_1 + \frac{R_1}{R_1 + R_2} z_2$ with $w_i = \frac{1}{\sigma_i^2} = \frac{1}{R_i}$
- Rewrite as **Kalman equation**: $\hat{r} = z_1 + \frac{R_1}{R_1 + R_2}(z_2 - z_1)$ with the Kalman gain K = [0,1]
  - R_1 —> infinity: rely on second estimate
- first set r_hat to z_1 and the shift to z_2 according to the R values or if we think of this as adding a new measurement (z, P) to our current estimate (r, R) of the state we would get (—> prediction) :

$$\hat{r}_{k+1} = \hat{r}^-_{k+1} + \frac{P^-_{k+1}}{P^-_{k+1} + R}\left(z_{k+1} - \hat{r}^-_{k+1}\right) = \hat{r}^-_{k+1} + K_{k+1}\left(z_{k+1} - \hat{r}^-_{k+1}\right)$$

- For merging Gaussian distributions, the update rule is: $\frac{1}{\sigma_3^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} = \frac{\sigma_1^2 + \sigma_2^2}{\sigma_1^2 \sigma_2^2} \Rightarrow \sigma_3^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2}$
- which if we write in our measurement update equation form we get:

$$P_{k+1} = \frac{P^-_{k+1} R_{k+1}}{P^-_{k+1} + R_{k+1}} \equiv P^-_{k+1} - K_{k+1} P^-_{k+1}$$

Time Update



- **Step 1** of the time update phase is only our prediction based upon the linear state update equation that we have $\vec{x}^-_{k+1} = A\vec{x}_k + B\vec{u}_k$
  - A: state transition matrix, x_k: current state (either position, position + velocity or position + velocity + acceleration) —> estimate v and a
  - B: process parameter u_k from robot controller —> know v and a —> better prediction if we have access to the control values
- **Step 2** of the time update phase comes from projecting our covariance matrix forward where we only add the process noise variance Q due to the normal sum distribution property where $\sigma_3^2 = \sigma_1^2 + \sigma_2^2$
  - Covariance matrix: $P_k = \frac{1}{N}\sum_{i=1}^{N}\left(\vec{x}_i - \vec{\mu}_k\right)\left(\vec{x}_i - \vec{\mu}_k\right)^T$
  - Prediction of next covariance (model error Q): $P_{k+1} = \frac{1}{N}\sum_{i=1}^{N}\left(\vec{x}_{i,k+1} - \vec{\mu}_{k+1}\right)\left(\vec{x}_{i,k+1} - \vec{\mu}_{k+i}\right)^T + Q$
  - No control value (u_k =0), replace x_{k+1}:

$$P_{k+1} = \frac{1}{N}\sum_{i=1}^{N}\left[A\left(\vec{x}_{i,k} - \vec{\mu}_k\right)\right]\left[A\left(\vec{x}_{i,k} - \vec{\mu}_k\right)\right]^T + Q = AP_kA^T + Q$$

  - Q will flatten the current prediction (flatter gaussian)

  - 1. H: measurement matrix (e.g. x is a position (x, y) and z is rotated version (x', y') —> constant linear error corrected by measurement error, here: rotation matrix)
  - 2. add measurement z_k, otherwise the distribution will flatten

6.2. OBSERVABILITY MATRIX

- A system with state vector x of dimension n is observable if the observability matrix has row rank n (i.e. n linearly independent rows)
- Example:

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \qquad x = \begin{pmatrix} p \\ v \end{pmatrix}$$

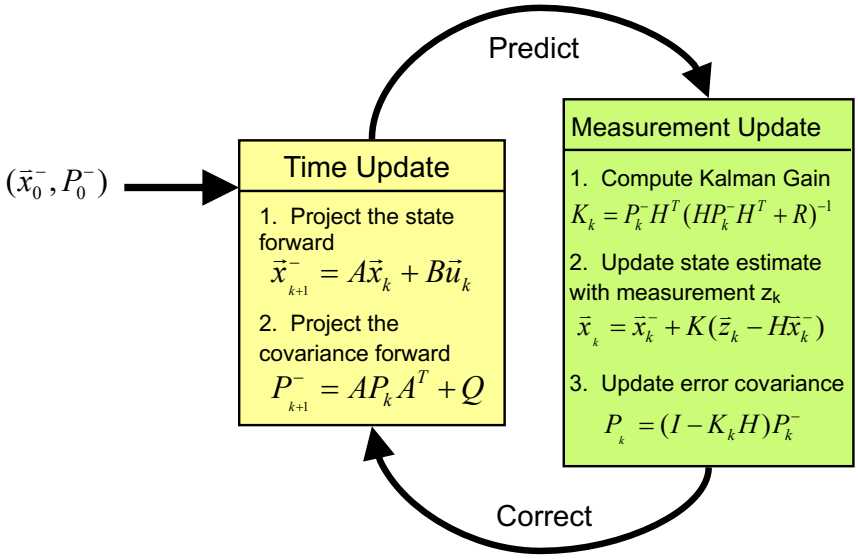
Measurement: position p

$$H = \begin{pmatrix} 1 & 0 \end{pmatrix} \qquad O = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Measurement: speed v

$$H = \begin{pmatrix} 0 & 1 \end{pmatrix} \qquad O = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

$$\mathbf{O} = \begin{bmatrix} \mathbf{H} \\ \mathbf{HA} \\ \vdots \\ \mathbf{HA}^i \\ \vdots \\ \mathbf{HA}^{n-1} \end{bmatrix}$$



- Smaller Q —> Trust chosen model
- Smaller R —> Trust measurements

6.3. PROBLEMS WITH THE KALMAN FILTER

- For many applications, the time update and measurement equations are NOT linear
- As a consequence, the KF is not applicable
- However, the KF is such a nice algorithm that maybe if we linearize around the non-linearities, we can still get good performance in practice
- This line of thought lead to the development of the Extended Kalman Filter (EKF)
- By relaxing the linear assumptions, the use of the KF is extended dramatically
- Life Rule: There is no such thing as a free lunch
- We can no longer use the word “optimal” with the EKF

6.4. KALMAN EXAMPLES

- The Kalmanfilter is a very powerful tool when it comes to controlling noisy systems. The basic idea of a Kalman filter is: Noisy data in —> hopefully less noisy data out
- The applications of a Kalman filter are numerous:
  - Tracking objects (e.g., missiles, faces, heads, hands)
  - Fitting Bezier patches to (noisy, moving, ...) point data
  - Economics
  - Navigation
  - Many computer vision applications: Stabilizing depth measurements, Feature tracking, Cluster tracking
  - – Fusing data from radar, laser scanner and stereo-cameras for depth and velocity measurements

Predict-Update Equations

- $\hat{x}$: Estimated state.
- F: State transition matrix (i.e., transition between states).
- u : Control variables.
- B : Control matrix (i.e., mapping control to state variables)
- P : State variance matrix (i.e., error of estimation).
- Q : Process variance matrix (i.e., error due to process).
- y : Measurement variables.
- H : Measurement matrix (i.e., mapping measurements onto state).
- K : Kalman gain.
- R : Measurement variance matrix (i.e., error from measurements).
- Subscripts are as follows: t|t current time period, t - 1|t - 1 previous time period, and t|t - 1 are intermediate steps.

Predict:

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{F}_t\hat{\mathbf{x}}_{t-1|t-1} + \mathbf{B}_t\mathbf{u}_t$$
$$\mathbf{P}_{t|t-1} = \mathbf{F}_t\mathbf{P}_{t-1|t-1}\mathbf{F}_t^T + \mathbf{Q}_t$$

Update:

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t\left(\mathbf{y}_t - \mathbf{H}_t\hat{\mathbf{x}}_{t|t-1}\right)$$
$$\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t^T\left(\mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}_t\right)^{-1}$$
$$\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_{t|t-1}$$

Possible values of K

- since $K_k = P_k^- H^T \left(HP_k^- H^T + R\right)^{-1}$, there are two extreme values K can take

Case 1: K —> 0	Case 2: K —> H^{-1}
• happens if R —> ∞ (sensor is lost/crappy sensor) or P_k = 0 (the state estimation is 100% accurate) • The prior estimation is assumed to be exact, therefore the measurement is neglected • KF will blend the resulting mean closer to predicted state mean • Update of the state estimate: $x_k = x_k^-$ • Update of the error covariance: $P_k = P_k^- = 0$	• happens if R = 0 (the measurement is 100% correct) or P_k —> ∞ (prediction is not valuable at all) • The measurement is fully trusted and the • prior estimation is dropped. • KF will blend the mean closer to mean measurements. • Update of the state estimate: $x_k = H^{-1} z_k$ • Update of the error covariance: $P_k = 0$

Model Definition Process

The Kalman filter removes noise by assuming a pre-defined model of a system. Therefore, the Kalman filter model must be meaningful. It should be defined as follows:

1. **Understand the situation:** Look at the problem. Break it down to the mathematical basics. If you don't do this, you may end up doing unneeded work.
2. **Model the state process:** Start with a basic model. It may not work effectively at first, but this can be refined later.
3. **Model the measurement process:** Analyze how you are going to measure the process. The measurement space may not be in the same space as the state (e.g., using an electrical diode to measure weight, an electrical reading does not easily translate to a weight).
4. **Model the noise:** This needs to be done for both the **state** diode to measure weight, an electrical reading does not easily translate to a weight).
4. **Model the noise:** This needs to be done for both the **state** and **measurement** process. The base Kalman filter assumes Gaussian (white) noise, so make the variance and covariance (error) meaningful (i.e., make sure that the error you model is suitable for the situation).
5. **Test the filter:** Often overlooked, use synthetic data if necessary (e.g., if the process is not safe to test on a live environment). See if the filter is behaving as it should.
6. **Refine filter:** Try to change the noise parameters (filter), as this is the easiest to change. If necessary go back further, you may need to rethink the situation.

6.4.1. EXAMPLE: WATER LEVEL IN TANK

see Subject MI63: Kalman Filter Tank Filling

6.5. THE EXTENDED KALMAN FILTER (EKF) CH 8.3

- The Extended Kalman (EKF) is a sub-optimal extension of the original KF algorithm
- The EKF allows for estimation of non-linear processes or measurement relationships
- This is accomplished by linearizing the current mean and covariance estimates (similar to a first order Taylor series approximation)
- Replace predictions function by non-linear functions

EKF Time Update Phase

- However when we propagate the covariance ahead in time, the underlying function needs to be linear in order to properly combine the Gaussian uncertainty in our state x
- Solution: Use a Taylor series
- Let's say we know the uncertainty of a variable x, and we want to compute the uncertainty of y=f(x)
- We can then use the Jacobian to linearly approximate y
- So the covariance is projected ahead as —->
- where A is the Jacobian of f with respect to x (state) and W is the Jacobian of f with respect to w (sensor measurements: velocity)

$P_{k+1}^- = AP_kA^T + Q$

Kalman Filter

$P_{k+1}^- = AP_kA^T + WQW^T$

Extended Kalman Filter

EKF Robot Implementation Example

- Assume that we have a mobile robot using odometry and range measurements to landmark to estimate its position and orientation: $\vec{x} = [x, y, \theta]^T$
- Assume that the odometry provides a velocity estimate V and an angular velocity estimate ω that are both corrupted by gaussian noise
- We can write the state update equation as, which is obviously non-linear in the state

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k+1} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k} + \begin{bmatrix} V\cos\theta\Delta t \\ V\sin\theta\Delta t \\ \omega\Delta t \end{bmatrix}$$

- Calculate the Jacobian A and W

$$f(\vec{x}) = \begin{bmatrix} f_1(\vec{x}) \\ f_2(\vec{x}) \\ f_3(\vec{x}) \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k+1} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k} + \begin{bmatrix} V\cos\theta\Delta t \\ V\sin\theta\Delta t \\ \omega\Delta t \end{bmatrix} \qquad W = \frac{\partial f}{\partial \vec{w}} = \begin{bmatrix} \Delta t\cos\theta & 0 \\ \Delta t\sin\theta & 0 \\ 0 & \Delta t \end{bmatrix} \qquad A = \frac{\partial f}{\partial \vec{x}} = \begin{bmatrix} 1 & 0 & -V\sin\theta\Delta t \\ 0 & 1 & V\cos\theta\Delta t \\ 0 & 0 & 1 \end{bmatrix}$$

EKF Measurement Update Phase

- Again, in the measurement update we can have a non-linear relationship between our measurements z_k and state x_k: $\vec{z}_k = h\left(\vec{x}_k, \vec{v}_k\right)$ and once again we will assume that the noise is zero $\vec{z}_k = h\left(\vec{x}_k, 0\right)$
- To propagate uncertainty, we shall again have to calculate the appropriate Jacobians
  - H is the Jacobian matrix of partial derivatives of h with respect to x
  - V is the Jacobian matrix of partial derivatives of h with respect to measurement noise v
- These are then substituted into the original KF as appropriate

	KF	EKF
• Computing the Kalman Gain:	$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$	$K_k = P_k^- H^T (HP_k^- H^T + VRV^T)^{-1}$
• State Update:	$\bar{x}_k = \bar{x}_k^- + K_k(\bar{z}_k - H\bar{x}_k^-)$	$\bar{x}_k = \bar{x}_k^- + K_k(\bar{z}_k - h(\bar{x}_k^-, 0))$
• Covariance Update:	$P_k = (I - K_k H)P_k^-$	$P_k = (I - K_k H_k)P_k^-$

EKF Robot Implementation Example

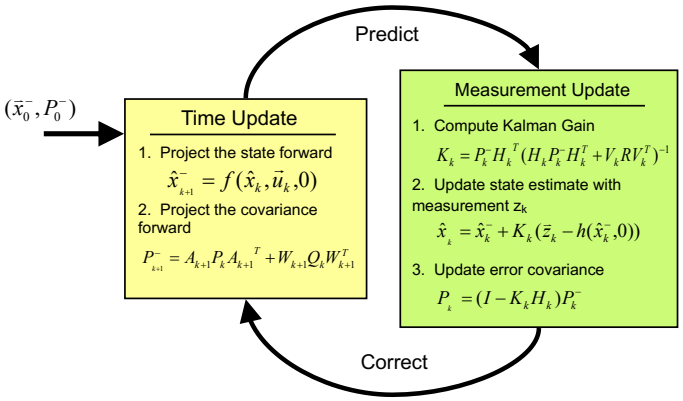
- Let's go back to our beacon example for the Aibo. From this, our range measurement can be written as $z_k = \sqrt{(x_k - x_b)^2 + (y_k - y_b)^2}$, where x_b and y_b are constants
- The Jacobians H and V can then be calculated as

$$H = \frac{\partial z}{\partial \vec{x}} = \begin{bmatrix} \frac{x_k^-}{\sqrt{(x_k - x_b)^2 + (y_k - y_b)^2}} & \frac{y_k^-}{\sqrt{(x_k - x_b)^2 + (y_k - y_b)^2}} & 0 \end{bmatrix} \text{ and } V = 1 \text{ (just linear addition of noise)}$$

- Note that if this were the only measurements available then θ would be unobservable

The discrete extended Kalman Filter

control input u is neglected here



Time Update:

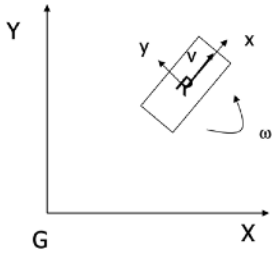
$$\bar{x}_{t|t-1} = f(\bar{x}_{t-1}, 0)$$
$$P_{t|t-1} = F_t P_{t-1} F_t^T + G_t Q_t G_t^T$$

Measurement Update:

$$K_t = P_{t|t-1} H_t^T \left[U_t R_t U_t^T + H_t P_{t|t-1} H_t^T\right]^{-1}$$
$$\bar{x}_t = \bar{x}_{t|t-1} + K_t\left(y_t - h(\bar{x}_{t|t-1}, 0)\right)$$
$$P_t = P_{t|t-1} - K_t H_t P_{t|t-1}$$

$$F_t = \left.\frac{\partial f(x_t)}{\partial x_t}\right|_{(x_t = \bar{x}_{t|t-1})} \qquad H_t = \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{(x_t = \bar{x}_{t|t-1})}$$
$$G_t = \left.\frac{\partial f(v_t)}{\partial v_t}\right|_{(v_t = \bar{v})} \qquad U_t = \left.\frac{\partial h(n_t)}{\partial n_t}\right|_{(n_t = \bar{n})}$$

K_t is Kalman Gain, Q is the variance process noise, R is the variance of the measurement noise

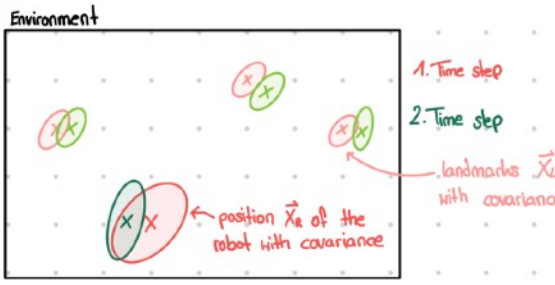


6.6. LINEARIZED MOTION MODEL FOR A ROBOT

- EKF is well suited to the problem of localizing a mobile robot equipped with sensors that can detect range and bearing to previously mapped landmarks in the environment
- Velocities of the robot from a global perspective:
  - $\dot{x}_t = V_t \cos\phi_t$
  - $\dot{y}_t = V_t \sin\phi_t$
  - $\dot{\phi}_t = \omega_t$
- Discrete time state estimate (including noise):
  - $\hat{x}_{t+1} = \hat{x}_t + \left(V_t + w_{V_t}\right)\delta t \cos\hat{\phi}_t$
  - $\hat{y}_{t+1} = \hat{y}_t + \left(V_t + w_{V_t}\right)\delta t \sin\hat{\phi}_t$
  - $\hat{\phi}_{t+1} = \hat{\phi}_t + \left(\omega_t + w_{\omega_t}\right)\delta t$
- The indirect Kalman filter derives the pose equations from the estimated error of the state:
  - $x_{t+1} - \hat{x}_{t+1} = \tilde{x}_{t+1}$
  - $y_{t+1} - \hat{y}_{t+1} = \tilde{y}_{t+1}$
  - $\phi_{t+1} - \hat{\phi}_{t+1} = \widetilde{\phi}_{t+1}$
- From the error-state propagation equation, we can obtain the State propagation and noise input functions A and W: $\tilde{X}_{t+1} = F_t\tilde{X}_t + G_tW_t$
- $\begin{bmatrix}\tilde{x}_{t+1}\\ \tilde{y}_{t+1}\\ \widetilde{\phi}_{t+1}\end{bmatrix} = \begin{bmatrix}1 & 0 & -V_m\delta t\sin\hat{\phi}\\ 0 & 1 & V_m\delta t\cos\hat{\phi}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\tilde{x}_t\\ \tilde{y}_t\\ \widetilde{\phi}_t\end{bmatrix} + \begin{bmatrix}-\delta t\cos\phi_R & 0\\ -\delta t\sin\phi_R & 0\\ 0 & -\delta t\end{bmatrix}\begin{bmatrix}w_{V_t}\\ w_{\omega_t}\end{bmatrix}$
- compute the standard covariance propagation equation: $P_{t+1/t} = F_tP_{t/t}F_t^T + G_tQ_tG_t^T$

6.7. EXTENDED KALMAN FILTER FOR SLAM CH 8.4

One common approach to solving the SLAM (simultaneous localization and mapping) problem is to use a Kalman filter to simultaneously estimate the position of a moving vehicle along with the positions of landmarks seen by the vehicle.



- Typical problem: Unknown map for the robot —> Robot needs to build the map while moving in the environment
- State vector: location of the vehicle together with the locations of each of the landmarks, State vector can be grown as new landmarks are discovered
- Position of the robot $\overrightarrow{X}_R = [x, y, \theta]^T$
- State vector $\overrightarrow{X}_k = [\overrightarrow{X}_R\ \overrightarrow{X}_{L_1}\ \dots\ \overrightarrow{X}_{L_n}]^T$, where n is the number of Landmarks
- Covariance matrix: $P = \begin{bmatrix}P_{RR} & P_{RL_1} & \cdots & P_{RL_N}\\ P_{L_1R} & P_{L_1L_1} & \cdots & P_{L_1L_N}\\ \vdots & \vdots & \ddots & \vdots\\ P_{L_NR} & P_{L_NL_1} & \cdots & P_{L_NL_N}\end{bmatrix}$ where P_{RR} is the uncertainty from the robot to itself, P_{RL_1} the uncertainty from the robot to the first landmark, …
- Kinematic equations for landmark propagation
  - Prediction of the robot motion:

$$l\hat{x}_{t+1} = \hat{x}_t + \left(V_t + w_{V_t}\right)\delta t\cos\hat{\phi}_t$$

$$\hat{y}_{t+1} = \hat{y}_t + \left(V_t + w_{V_t}\right)\delta t\sin\hat{\phi}_t$$

$$\hat{\phi}_{t+1} = \hat{\phi}_t + \left(\omega_t + w_{\omega_t}\right)\delta t$$

  - Prediction of the landmarks will be constant (no change over time):

$$l\hat{x}_{L_it+1} = \hat{x}_{L_it}$$

$$\hat{y}_{L_it+1} = \hat{y}_{L_it}$$

$$\hat{\phi}_{L_it+1} = \hat{\phi}_{L_it}$$

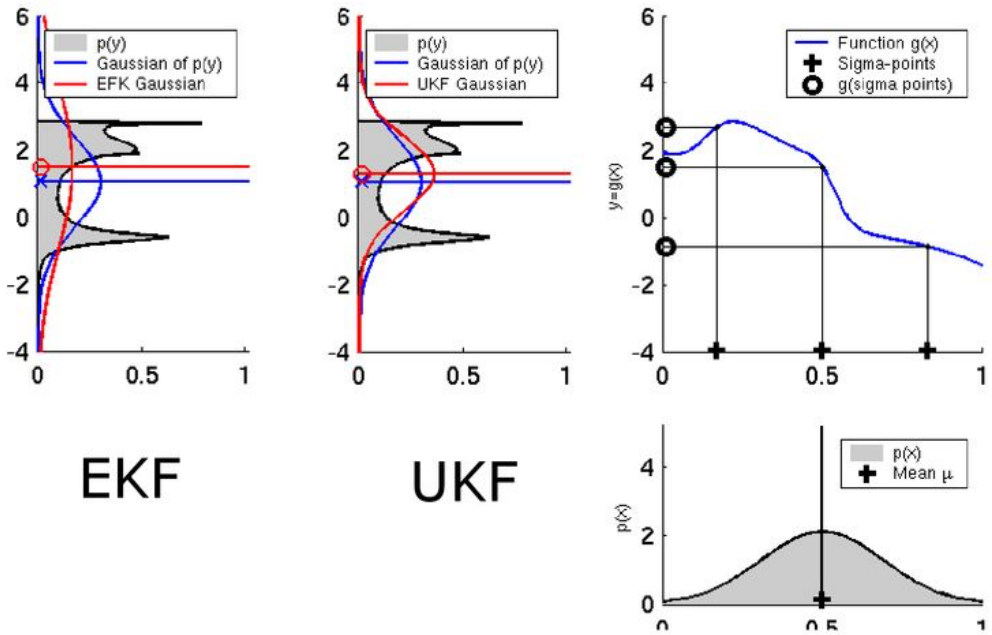
- Sensor equations for update: $\tilde{X} = \begin{bmatrix}\tilde{X}_R^T & \tilde{X}_{L_1}^T & \cdots & \tilde{X}_{L_i}^T & \cdots & \tilde{X}_{L_n}^T\end{bmatrix}$

$$H = [H_R\ \ 0\ \cdots\ 0\ \ H_{L_i}\ \ 0\ \cdots\ 0]$$

- Modify entries in H to 0 for landmarks I don't see (here only landmark i can be seen)
- Very powerful because covariance update records *shared information* between landmarks and robot positions
- Solve Data Association step —> which entries are currently visible and which not
- Intuition: chose data close to the predicted position: $v_{ij} = y_i - h_i(x(k+1 \mid k), j)$
- Uncertainties need to be considered in the choice of data: $\chi^2_{ij} = v_{ij}^{\top} S_{ij}^{-1} v_{ij}$, where $S_{ij} = H_i(k+1,j)P(k+1 \mid k)H_i(k+1,j)^T + R_i(k+1)$
- Mahanalobis norm
  - for a group of values with mean: $\mu = \left(\mu_1, \mu_2, \mu_3, \ldots, \mu_p\right)$ and a multivariate vector: $x = \left(x_1, x_2, x_3, \ldots, x_p\right)$
  - $D_M(x) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}$
  - If covariance matrix equals identity matrix then Euclidean distance. If covariance matrix is diagonal then normalized Euclidean distance

6.8. SUMMARY

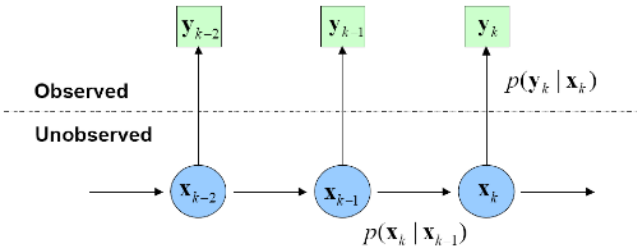
- For linear processes with Gaussian noise, the KF provides a provably optimal means for fusing measurement information
- For our purposes, the plain KF is to restrictive as both our motion and measurement equations are highly non-linear
- The linear constraints can be lifted, but the optimality of the filter is lost (although performance is still quite good in practice)
- This is the basis for the Extended Kalman Filter (EKF) – one of the most used estimation algorithms in mobile robotics



7. Unscented Kalman Filter

In cases the non-linearity too big, then even EKF will not converge. This chapter's filters can handle non-linearities.

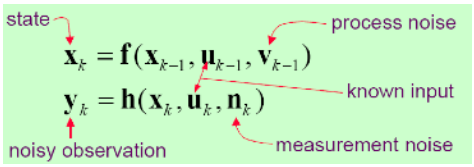
7.1. FILTERING PROBLEM: GENERAL PROBLEM STATEMENT



- Filtering is the problem of sequentially estimating the states (parameters or hidden variables) of a system as a set of observations become available on-line, where x_i is the state and y_i the observation
- $\vec{z} = \tilde{H}\vec{x} \quad \vec{z} = \tilde{h}(\vec{x})$
- Solution of sequential estimation problem given by
  - Posterior Density: $p\left(X_k \mid Y_k\right)$ with $X_k = \{x_1, x_2 \ldots, x_{k-1}, x_k\}$ and $Y_k = \{y_1, y_2 \ldots, y_{k-1}, y_k\}$
  - Marginal of the Posterior: $p\left(x_k \mid Y_k\right)$
  - One does not need to keep track of the history of the state sequence

- Dynamic State Space Model
  - General discrete time non-linear, non-Gaussian dynamic system
  - Assumption: States are markovian, i.e.,

$$p\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{x}_{k-2}, \cdots, \mathbf{x}_0\right) = p\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}\right)$$



- Solution by Extended Kalman Filter —> Crude Approximation:
  - EKF uses first order terms of the Taylor series expansion of the nonlinear functions.
  - Large errors are introduced when the models are highly non-linear
  - Local linearity assumption breaks down when the higher order terms become significant.

7.2. UNSCENTED TRANSFORMATION

The unscented transformation (UT) is a method for calculating the statistics of a random variable which undergoes a nonlinear transformation and builds on the principle that it is easier to approximate a probability distribution than an arbitrary nonlinear function.

- Problem: Propagating an $\boldsymbol{n_x}$ dimensional random variable $\boldsymbol{x}$ through a nonlinear function, $g : R^{n_x} \rightarrow R^{n_y}$, $y = g(x)$
- Formulation: Gaussian peak is shifted slightly. Sample the new distribution by choosing sample points and projecting the sampling points (Sigma points) through the function g. Assume x has mean $\bar{x}$ and covariance P_x. A set of $2n_x + 1$ weighted samples or sigma points $S_i = \{W_i, \chi_i\}$ are chosen as follows:

$$\chi_0 = \bar{x} \text{ (center)} \quad W_0 = \kappa / \left(n_x + \kappa\right) \quad i = 0$$

$$\chi_i = \bar{x} + \left(\sqrt{\left(n_x + \kappa\right) P_x}\right)_i \quad W_i = 1 / \left\{2\left(n_x + \kappa\right)\right\} \quad i = 1, \cdots, n_x$$

$$\chi_i = \bar{x} - \left(\sqrt{\left(n_x + \kappa\right) P_x}\right)_i \quad W_i = 1 / \left\{2\left(n_x + \kappa\right)\right\} \quad i = n_x + 1, \cdots, 2n_x$$

- where k is a scaling parameter and $\left(\sqrt{\left(n_x + \kappa\right) P_x}\right)_i$ is the i-th row or column of the matrix square root of $\left(n_x + \kappa\right) P_x$. W_i is the weight associated with the i-th point such that $\sum_{i=0}^{2n_x} W_i = 1$.
- Each sigma point is propagated through the nonlinear function $Y_i = g\left(\chi_i\right) \quad i = 0, \cdots, 2n_x$ and the estimated mean and covariance are computed as follows

$$\bar{y} = \sum_{i=0}^{2n_x} W_i Y_i \qquad P_y = \sum_{i=0}^{2n_x} W_i \left(Y_i - \bar{y}\right)\left(Y_i - \bar{y}\right)^T$$

- These estimates of the mean and covariance are accurate to the second order of the Taylor series expansion of g(x) for any nonlinear function
- Advantages:
  - The EKF only calculates the posterior mean and covariance accurately to the first order with all higher order moments truncated; however, UT calculates the mean and covariance to the second order.
  - No need for continuous function as state or measurement function but any function can be used because projection through the non-linearity, no derivative taken.
- More Information
- A cloud of 5000 samples drawn from a Gaussian prior is propagated through an arbitrary highly nonlinear function and the true posterior sample mean and covariance are calculated, which can be regarded as a ground truth of the two approaches, EKF and UT.

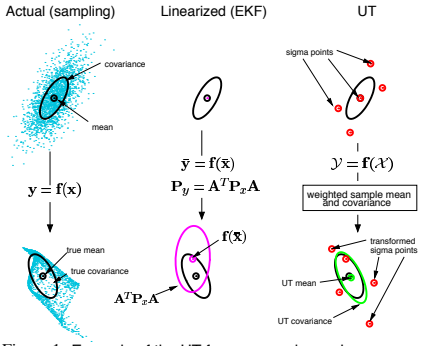


Figure 1: Example of the UT for mean and covariance propagation. a) actual, b) first-order linearization (EKF), c) UT.

The scaling factor κ

- As the dimension of the state-space increases, the radius of the sphere that bounds all the sigma points increases. Even though the mean and covariance of the prior distribution are still captured correctly, it does so at the cost of sampling non-local effects.
- These points are asymmetrically distributed about the mean. Higher order effects such as skew become more significant as the dimension increases.
- The sigma points can be scaled towards and away from the mean of the prior distribution by a proper choice of κ: $\left|\chi_i - \bar{x}\right| \propto \sqrt{\left(n_x + \kappa\right)}$

7.3. SCALED UNSCENTED TRANSFORMATION (SUT)

This improvement overcomes dimensional scaling effects by calculating the transformation of a scaled set of sigma points of the form $\chi_i' = \chi_0 + \alpha\left(\chi_i - \chi_0\right)$, where is a positive scaling parameter which can be made arbitrary small to minimize higher order effects.

The scaled unscented transformation can be written as:

$$S(W,\chi) \mapsto S'(W',\chi'):$$
$$\chi_i' = \chi_0 + \alpha(\chi_i - \chi_0)$$
$$Y_i' = f\left[\chi_i'\right]$$
$$W_i' = \begin{cases} W_0/\alpha^2 + \left(1 - 1/\alpha^2\right) & i = 0 \\ W_i/\alpha^2 & i \neq 0 \end{cases}$$
$$\bar{y} = \sum_{i=0}^{2n_x} W_i' Y_i'$$
$$P_y = \sum_{i=0}^{2n_x} W_i' \left\{Y_i' - \bar{y}\right\}\left\{Y_i' - \bar{y}\right\}^T + \left(1 + \beta - \alpha^2\right)\left\{Y_0' - \bar{y}\right\}\left\{Y_0' - \bar{y}\right\}^T$$

β is a parameter which minimizes the effects from high order terms

The sigma point selection and scaling can be combined into a single step by setting $\lambda = \alpha^2\left(n_x + \kappa\right) - n_x$ and selecting the sigma points are set by:

$$\chi_0 = \bar{x}$$
$$\chi_i = \bar{x} + \left(\sqrt{(n_x + \lambda)P_x}\right)_i \qquad i = 1,\ldots,n_x$$
$$\chi_i = \bar{x} - \left(\sqrt{(n_x + \lambda)P_x}\right)_i \qquad i = n_x + 1,\ldots,2n_x$$
$$W_0^{(m)} = \lambda / \left(n_x + \lambda\right)$$
$$W_0^{(c)} = \lambda / \left(n_x + \lambda\right) + \left(1 - \alpha^2 + \beta\right)$$
$$W_i^{(m)} = W_i^{(c)} = 1 / \left\{2\left(n_x + \lambda\right)\right\} \qquad i = 1,\ldots,2n_x$$

7.4. UNSCENTED KALMAN FILTER

The unscented kalman filter (UKF) is a straightforward application of the scaled unscented transformation, where the state variable $x_t^a = \left[x_t^T v_t^T n_t^T\right]^T$ is an augmented vector and the covariance P_t^a is also an augmented matrix.

1. Initialization

$$\bar{x}_0 = E\left[x_0\right] \quad P_0 = E\left[\left(x_0 - \bar{x}_0\right)\left(x_0 - \bar{x}_0\right)^T\right]$$
$$\bar{x}_0^a = E\left[x^a\right] = \left[\bar{x}_0^T \quad 0 \quad 0\right]^T$$
$$P_0^a = E\left[\left(x_0^a - \bar{x}_0^a\right)\left(x_0^a - \bar{x}_0^a\right)^T\right] = \begin{bmatrix} P_0 & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & R \end{bmatrix}$$

2. Updating

(a) Computing Sigma Points

$$\chi_{t-1}^a = \left[\bar{x}_{t-1}^a \quad \bar{x}_{t-1}^a \pm \sqrt{\left(n_a + \lambda\right)P_{t-1}^a}\right]$$

b) Time update

$$\chi_{t|t-1}^x = f\left(\chi_{t-1}^x, \chi_{t-1}^v\right)$$
$$\bar{x}_{t|t-1} = \sum_{i=0}^{2n_a} W_i^{(m)} \chi_{i,t|t-1}^x$$
$$P_{t|t-1} = \sum_{i=0}^{2n_a} W_i^{(c)}\left[\chi_{i,t|t-1}^x - \bar{x}_{t|t-1}\right]\left[\chi_{i,t|t-1}^x - \bar{x}_{t|t-1}\right]^T$$
$$Y_{t|t-1} = h\left(\chi_{t|t-1}^x, \chi_{t-1}^n\right)$$
$$\bar{y}_{t|t-1} = \sum_{i=0}^{2n_a} W_i^{(m)} Y_{i,t|t-1}$$

c) Measurement Update

$$P_{\bar{y}_t,\bar{y}_t} = \sum_{i=0}^{2n_a} W_i^{(c)}\left[Y_{i,t|t-1} - \bar{y}_{t|t-1}\right]\left[Y_{i,t|t-1} - \bar{y}_{t|t-1}\right]^T$$
$$P_{x_t,y_t} = \sum_{i=0}^{2n_a} W_i^{(c)}\left[\chi_{i,t|t-1} - \bar{x}_{t|t-1}\right]\left[Y_{i,t|t-1} - \bar{y}_{t|t-1}\right]^T$$
$$K_t = P_{x_t,y_t} P_{\bar{y}_t,\bar{y}_t}^{-1}$$
$$\bar{x}_t = \bar{x}_{t|t-1} + K_t\left(y_t - \bar{y}_{t|t-1}\right)$$
$$P_t = P_{t|t-1} - K_t P_{\bar{y}_t,\bar{y}_t} K_t^T$$

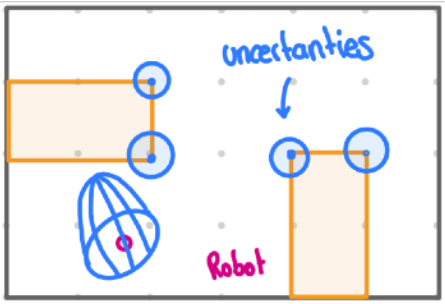
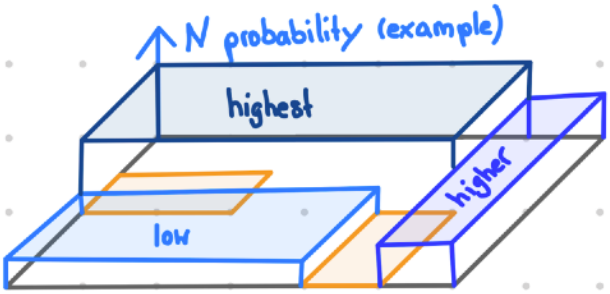
where, $x^a = \left[x^T \quad v^T \quad n^T\right]^T$, $\chi^a = \left[\left(\chi^x\right)^T\left(\chi^v\right)^T\left(\chi^n\right)^T\right]^T$

λ composite scaling parameter, $n_a = n_x + n_v + n_n$,
Q is process noise cov. R is measurement noise cov.
K is kalman gain, W is weights

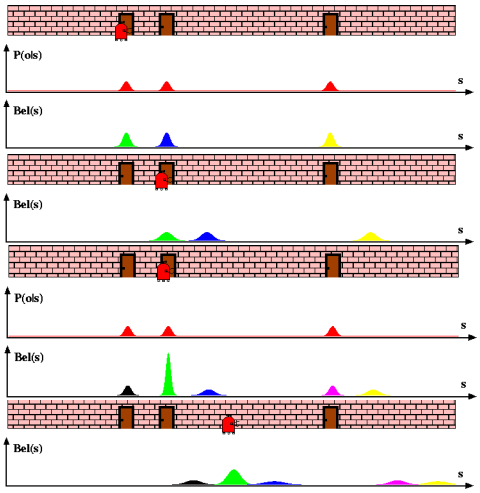
8. Bayesian and Particle Filter Ch 9

Kalman filter is a bayesian filter with the constraint that the distribution is normal. Baysian filter allows to have arbitrary distributions.

8.1. LOCALIZATION

Incremental localization	Global localization
• feature points (corners, …) have uncertainties due to the uncertainty of the robot position and the sensors • uncertainty can be improved • uncertainty of position is n-dimensional gaussian • incremental localization: know at the beginning where we are, want to know how the robot moves • —> optimal tool to run SLAM: KF or EKF	• Global localization: initially no idea where the robot is • Initial distribution not gaussian, not smooth, • Could map probability distribution on a set of KF BUT better ways
	

- **Why not just use Kalman Filter?** Would be enough for incremental localization. Robot is not able to identify the door it has sensed —> cannot associate an observation of a door uniquely to the doors given in the map —> Data-Association problem —> probabilistic localization

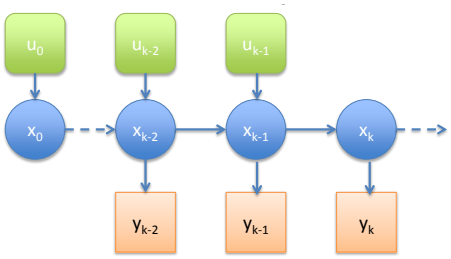


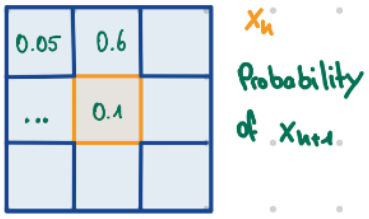
Probability properties

- $P(x, y) = P(x \mid y)P(y)$
- $P(x) = {}_{y}P(x, y) = {}_{y}P(x \mid y)P(y)$
- If x is independent of z given y: $P(x \mid y, z) = P(x \mid y)$
- Markov: $P\left(x_k \mid x_{k-1}, \ldots, x_0\right) = P\left(x_k \mid x_{k-1}\right)$

8.2. BAYESIAN FILTERING

—> probabilistically estimates a dynamic system's state (person's or object's location) from noisy updates



Given			Find
Sequence of measurements/ sensor readings (e.g. corners)	$y_1, \ldots, y_k$	$\{y_{1:k}, x_{0:k-1}\}$	$P(x_k \mid y_{1:k}, u_{0:k-1})$ Most likely state x at step k given a sequence of commands and measurements —> tells us how to use the sensory input to update the most recent estimate (prior) to obtain a new estimate (posterior)
Sequence of commands/control values (until last known step k-1)	$u_o, \ldots u_{k-1}$		
Observation model/Sensor model/map	$P(y_k \mid x_k)$	most likely sensor reading given state x 	
Motion model/dynamic model	$P(x_{k+1} \mid x_K)$	what are typical motions of the robot Example: 	
Prior probability	$P(x_0)$	See localization, uniform if unknown	

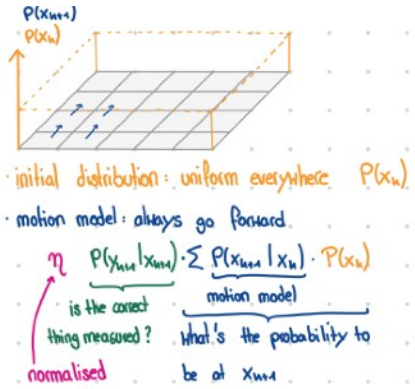
- Bayes Theorem: $P(x \mid y) = P(y \mid x)P(x)/P(y)$
- Think of x as the state of the robot and y as the data we know

$P(x_k \mid u_{0:k-1}, y_{1:k}) \longrightarrow$ *Posterior Probability Distribution*

$P(x_k \mid u_{0:k-1}, y_{1:k}) = P(y_k \mid x_k, u_{0:k-1}, y_{1:k-1})\ P(x_k \mid u_{0:k-1}, y_{1:k-1})\ /\ P(y_k \mid u_{0:k-1}, y_{1:k-1})$

$= \eta_k P(y_k \mid x_k) \int_{x_{k-1}} P(x_k \mid u_{k-1}, x_{k-1})\ P(x_{k-1} \mid u_{0:k-2}, y_{1:k-1})$

observation | state prediction | recursive instance



- Suppose instead state space is partitioned and probability in partition is constant

$$P(x \mid y) = P(y \mid x)P(x)/P(y) \approx P\left(x_i \mid y\right) = \eta P\left(y \mid x_i\right) P\left(x_i\right)$$

- Sum over all locations: $\eta = \sum_i P\left(y \mid x_i\right) P\left(x_i\right)$ (sum of resulting probability is 1)
- Thus, updating from observations is a simple multiplication of prior probability by likelihood of observation

$$P\left(x_{i,k} \mid u_{0:k-1}, y_{k-1}\right) = \sum_j P\left(x_{i,k} \mid u_{k-1}, x_{j,k-1}\right) P\left(x_{j,k-1} \mid u_{0:k-2}, y_{k-1}\right)$$

- Thus, updating using dynamical model is simply a discrete convolution (blurring) of the prior by the driving noise of the planned motion

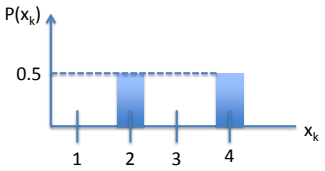
Motion

- Suppose we have $P\left(x_k\right)$ and $P\left(x_{k+1} \mid x_k, u_k\right) \text{ -> } P\left(x_{k+1}\right) = P\left(x_{k+1} \mid x_k, u_k\right) P\left(x_k\right) dx_k$ (probability distribution for x_{k+1} given the command u_k and all the previous states x_k

State space X = { 1, 2, 3, 4}

$P(x_{k+1} \mid x_k, u_k)$	$X_{k+1}=1$	$X_{k+1}=2$	$X_{k+1}=3$	$X_{k+1}=4$
$X_k=1$	0.25	0.5	0.25	0
$X_k=2$	0	0.25	0.5	0.25
$X_k=3$	0	0	0.25	0.75
$X_k=4$	0	0	0	1

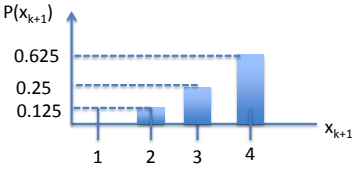
Prior $P(x_k)$



Probability that robot is in state 4 in the next step

$$P(x_{k+1}=4) = \sum_{x' \in X} P(x_{k+1}=4 \mid x'_k, u_k) P(x'_k)$$
$$= P(x_{k+1}=4 \mid x'_k=1, u_k) P(x'_k=1) +$$
$$P(x_{k+1}=4 \mid x'_k=2, u_k) P(x'_k=2) +$$
$$P(x_{k+1}=4 \mid x'_k=3, u_k) P(x'_k=3) +$$
$$P(x_{k+1}=4 \mid x'_k=4, u_k) P(x'_k=4)$$
$$= 0 \times 0 + 0.25 \times 0.5 + 0.75 \times 0 + 1 \times 0.5$$
$$= 0.625$$

$$P(x_{k+1}) = \int P(x_{k+1} \mid x_k, u_k) P(x_k) dx_k$$



Discrete Bayes Filter Algorithm

Algorithm **Discrete_Bayes_filter**($u_{0:k-1}$, $y_{1:k}$, $P(x_0)$)

1. $P(x) = P(x_0)$
2. **for** i=1:k
3. **for all** states $x \in X$
4. $P'(x) = \sum_{x' \in X} P(x \mid u_{i-1}, x') P(x')$ Prediction given prior dist. and command
5. **end for**
6. η=0
7. **for all** states $x \in X$
8. $P(x) = P(y_i \mid x) P'(x)$ Update using measurement
9. η = η + P(x)
10. **end for**
11. **for all** states $x \in X$ Normalize to 1
12. P(x) = P(x) / η
13. **end for**
14. **end for**

	Kalman Filter	Multi-hypothesis tracking	Grid-based (fixed/ variable)
Sensors	Gaussian	Gaussian	Non-Gaussian
Posterior	Gaussian	Multi-modal	Piecewise constant
Efficiency (memory)	++	++	+/-
Efficiency (time)	++	++	0/+
Implementation	+	0	+/0
Accuracy	++	++	+/++
Robustness	-	+	++
Global Localization	No	Yes	Yes

8.3. PARTICLE FILTERING

- Represent posterior/belief by a set of random samples instead of functions
- Estimation of non-Gaussian, nonlinear processes
- Examples: Monte Carlo filter, Survival of the fittest, Condensation, Bootstrap filter, Particle filter

The Likelihood Function

- Generating the sensor likelihood is essentially a sensor simulation
  - can be expensive – pre-compute
  - approximate
- A good fast approximation is often a weighted sum of
  - a nominal model that is fast to compute
  - other deviations that are modeled as random elements

Resampling

- Given: Set S of weighted samples.
- Wanted : Random sample, where the probability of drawing x_i is given by w_i (sparse for low probability, dense for high probability)
- Typically done n times with replacement to generate new sample set S'

Particle Filter Algorithm

$$Bel(x_t) = \eta\, p(z_t \mid x_t) \int p(x_t \mid x_{t-1}, u_{t-1})\, Bel(x_{t-1})\, dx_{t-1}$$

draw x^i_{t-1} from $Bel(x_{t-1})$

draw x^i_t from $p(x_t \mid x^i_{t-1}, u_{t-1})$

Resampling Algorithm

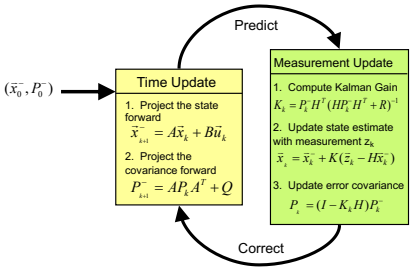
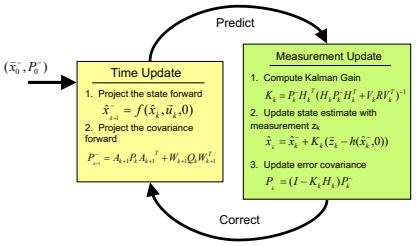
1. Algorithm **systematic_resampling**(S,n):
2. $S' = \emptyset, c_1 = w^1$
3. **For** $i = 2 \dots n$ — *Generate cdf*
4. $\quad c_i = c_{i-1} + w^i$
5. $u_1 \sim U[0, 1/n], i = 1$ — *Initialize threshold*
6. **For** $j = 1 \dots n$ — *Draw samples …*
7. $\quad$ **While** ($u_j > c_i$) — *Skip until next threshold reached*
8. $\quad\quad i = i + 1$
9. $\quad S' = S' \cup \{< x^i, 1/n >\}$ — *Insert*
10. $\quad u_j = u_j + 1/n$ — *Increment threshold*
11. **Return** S'

Also called **stochastic universal sampling**

Monte Carlo Localization

- Monte-Carlo-Localization(a, z, N, map)
  - S = N samples from P(X(t)) from previous call
  - for i = 1 to N
    - S[i] = sample from P(X(t+1) | X(t) = S[i], A = a)
    - W[i] = 1
    - for j = 1 to M do
      - z* = expected-sensor-reading(j,S[i],map)
      - W[i] = W[i] * P(Z = z(j) | Z* = Z*)
  - S = weighted-sample-with-replacement(N, S, W)
  - return S
- Note that S is a discrete representation of the probability of robot location

9. Overview Filters

	Kalman Filter	Extended Kalman Filter	Unscented Kalman Filter
Difference	KF requires a linear state transition and linear relation between state and measurement	EKF can deal with non-linear state transition and non-linear relation state-measurement through linear approximation by first-order Taylor expansion	UKF applies the scaled unscented transformation, which calculates the mean and covariance to the second order, thus can deal with highly non-linear problems
Prediction and update equation			b) Time update $\chi_{t\|t-1}^x = f(\chi_{t-1}^x, \chi_{t-1}^v)$ $\bar{x}_{t\|t-1} = \sum_{i=0}^{2n_a} W_i^{(m)} \chi_{i,t\|t-1}^x$ $P_{t\|t-1} = \sum_{i=0}^{2n_a} W_i^{(c)} [\chi_{i,t\|t-1}^x - \bar{x}_{t\|t-1}][\chi_{i,t\|t-1}^x - \bar{x}_{t\|t-1}]^T$ $Y_{t\|t-1} = h(\chi_{t\|t-1}^x, \chi_{t-1}^n)$ $\bar{y}_{t\|t-1} = \sum_{i=0}^{2n_a} W_i^{(m)} Y_{i,t\|t-1}$ c) Measurement Update $P_{\tilde{y}_t\tilde{y}_t} = \sum_{i=0}^{2n_a} W_i^{(c)} [Y_{i,t\|t-1} - \bar{y}_{t\|t-1}][Y_{i,t\|t-1} - \bar{y}_{t\|t-1}]^T$ $P_{x_t y_t} = \sum_{i=0}^{2n_a} W_i^{(c)} [\chi_{i,t\|t-1} - \bar{x}_{t\|t-1}][Y_{i,t\|t-1} - \bar{y}_{t\|t-1}]^T$ $K_t = P_{x_t y_t} P_{\tilde{y}_t\tilde{y}_t}^{-1}$ $\bar{x}_t = \bar{x}_{t\|t-1} + K_t(y_t - \bar{y}_{t\|t-1})$ $P_t = P_{t\|t-1} - K_t P_{\tilde{y}_t\tilde{y}_t} K_t^T$ where, $x^a = [x^T \quad v^T \quad n^T]^T$, $\chi^a = [(\chi^x)^T (\chi^v)^T (\chi^n)^T]^T$ λ composite scaling parameter, $n_a = n_x + n_v + n_n$, Q is process noise cov. R is measurement noise cov. K is kalman gain, W is weights
Name and explanation of methods	• K is the Kalman Gain	• H is the Jacobian matrix of partial derivatives of h with respect to x • V is the Jacobian matrix of partial derivatives of h with respect to measurement noise v • A is the Jacobian of f with respect to x • W is the Jacobian of f with respect to w	
Limits of applicability	KF, EKF and UKF require the error characteristic to modeled as zero-mean, Gaussian noise.		
	Linear models (linear state transition and linear relation between state and measurement)	Cannot deal with highly non-linear models	