Robot Motion Planning

1 Introduction

The main problem is to find a path from one position or configuration to another, without colliding with the environment. Planning algorithms can be rated regarding different aspects.

Performance metrics for path planning algorithms:

- path length
- execution speed
- planning time
- safety (distance to obstacles)
- robustness to disturbances
- probability of success

2 Configuration Space

A configuration describes the state of a robot as a combination of different configurations of its degrees of freedom. For example, a mobile robot moves in directions x, y and can turn its yaw heading θ - a configuration of this robot could be $c = (1, 3, 30°)^{\mathrm{T}}$ The configuration space (*C-space*) is the space of all possible configurations, that a robot can have in a certain workspace.

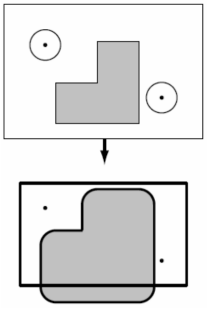
The dimension of the C-space is equal to the degrees of freedom of the robot (minimum number of parameters needed to specify the configuration). Because there can also be angles in the topology of the C-space, it is usually not a cartesian space (e.g. $2\pi = 4\pi$, cyclic repetition along angle axis).

A non-holonomic robot is a robot, that cannot move freely on the configuration space manifold because it is constrained in some way. For example, a car cannot drive sideways.

Point Representation

In order to determine the C-space / free space of configurations, a point representation can be useful. For example, if we have a circular robot with radius r and some obstacles in the workspace, the robot can be reduced to a point if the obstacle boundaries are "blown up" by the radius of the robot. In other words, to get a smaller representation of the robot without changing the properties of the workspace, the obstacles have to get bigger.

⇒ Minkowski sum algorithm



3 Bug Algorithms

Bug algorithms are simple strategies that use local knowledge (e.g. from sensor input) instead of global knowledge.

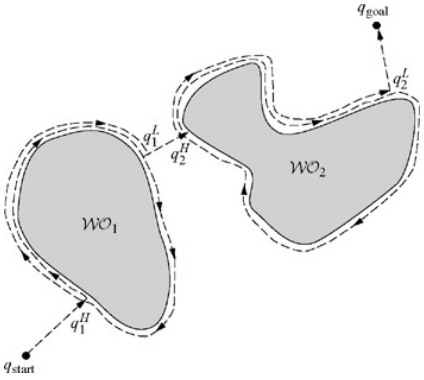
Assumptions:

- robot is a point in a plane
- contact sensor to detect obstacle boundaries

Basic idea: move on a straight line towards goal and follow the boundaries around obstacles

Bug1

1. move towards goal on the *m-line* connecting q_{start} and q_{goal}
2. if an obstacle is detected at a hit-point q_i^H, move left (or right) along its boundary until you return to q_i^H again
3. while circumnavigating the obstacle, calculate and save the distance to the goal at any coordinate
4. determine the closest point to the goal on the boundary and use it as a leave point q_i^L
5. from q_i^L, go straight to the goal again on the m-line q_i^L and q_{goal}
6. if the line that connects q_i^L and the goal intersects the current obstacle, then there is no path to the goal

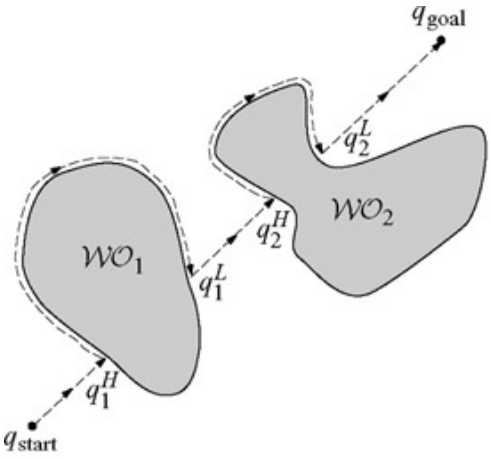


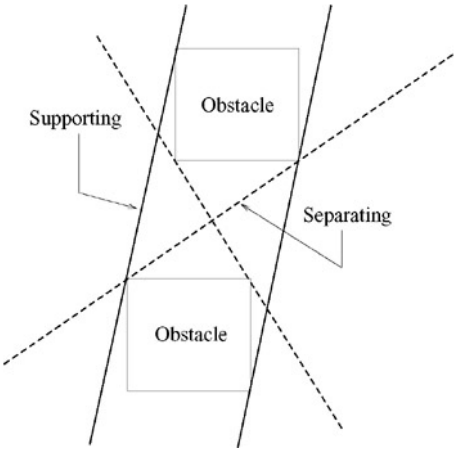
Bug2

The m-line of Bug2 does not change and it connects q_{start} and q_{goal}.

1. move towards goal on the *m-line* connecting q_{start} and q_{goal}
2. if an obstacle is detected at a hit-point q_i^H, move left (or right) along its boundary
3. if a point on the m-line is found and it is closer to the goal than q_i^H, use it as a leave point q_i^L
4. from q_i^L, go straight to the goal again on the m-line
5. if the robot re-encounters the original departure point q_i^H from the m-line, then there is no path to the goal

TU München - Fakultät für Informatik - Robotics, Cognition, Intelligence





Tangent Bug

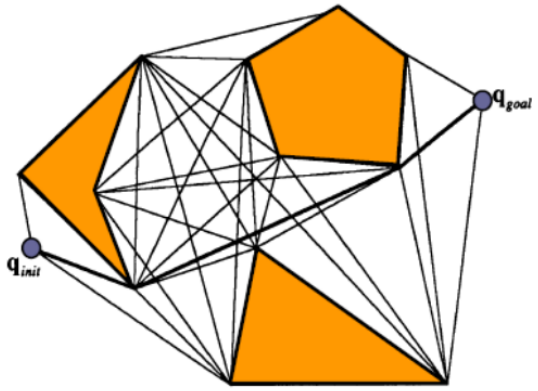
The robot is equipped with a (finite-range) radial distance sensor. If an obstacle is encountered within the range of the sensor, the robot changes its direction tangentially to the border of the obstacle.

4 Roadmapping with Exact Knowledge

The basic idea of a roadmap is to "capture the connectivity of the free space in a network of one-dimensional curves" (*Latombe*). Using this network, paths can be calculated using graph-search algorithms, for example. The advantage of a roadmap is that it only has to be constructed once if the topology of the workspace (i.e. obstacle positions) does not change. On the other hand, roadmaps can be not very efficient in dynamic environments.

Visibility Graph

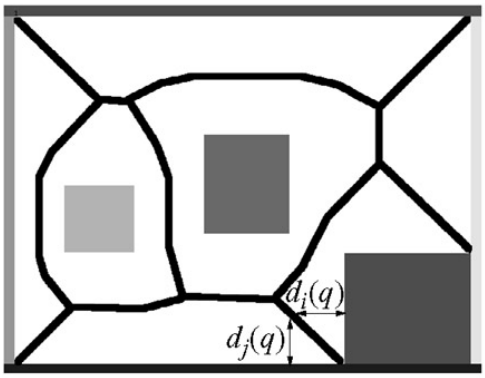
The visibility graph applies to 2-dimensional configuration spaces with polygonal obstacles. The graph's nodes are the initial configuration q_{init} (start) and the goal configuration q_{goal}, as well as the vertices/edges of the obstacles. Two nodes are connected with a straight line if the connecting line does not intersect with an obstacle.



Unfortunately, the visibility graph can contain useless edges. This can be resolved by constructing the reduced visibility graph, where only edges are kept that are separating or supporting lines between two obstacles.

Voronoi Diagram

The goal of the Voronoi diagram is to find paths that maximize the distance to the obstacles. Therefore, the roadmap lines can be curved.



To construct the Voronoi diagram, one can use the Brushfire algorithm, which calculates a distance map on a grid:

- grid initialization: free space = 0, obstacle space = 1
- for each point in the grid, assign the distance to the closest obstacle point
- every point, where the distance to two (or more) different obstacles is the same, lies on the Voronoi diagram

("wavefront from obstacles, Voronoi diagram where two wavefronts meet")

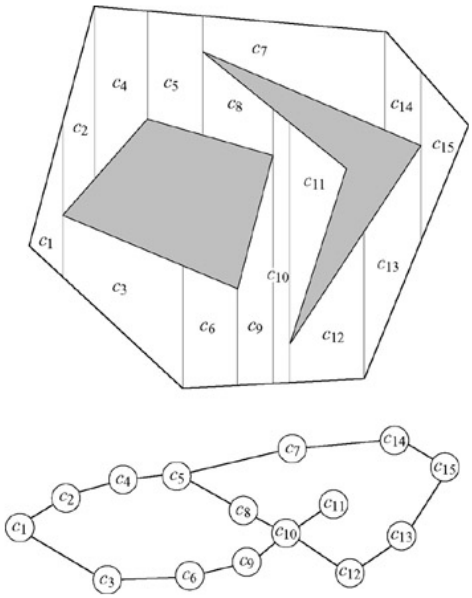
Cell Decomposition

Main idea: decompose the free space into simple cells and represent the connectivity of the free space $\mathcal{F}$ by the adjacency of these cells. It is called an exact cell decomposition if the union of all cells is exactly $\mathcal{F}$, meaning there is no overlap between cells.

The shape of the cells can be triangles, trapezoids, ...

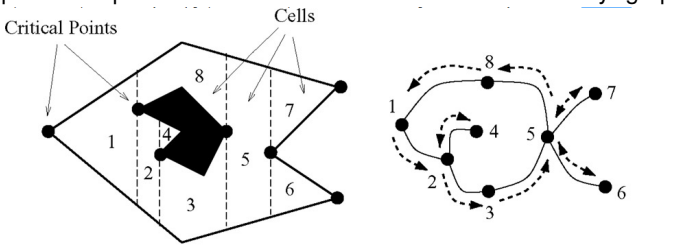
Trapezoidal Decomposition

Free space is decomposed into trapezoids and triangles. Then, the adjacency graph of the cells is calculated.



Boustrophedon Decomposition

Points on the obstacles, from which a separating line can be drawn in the upper and lower direction are called critical points. Then, an exhaustive walk through the critical points is performed in order to obtain a connectivity graph.

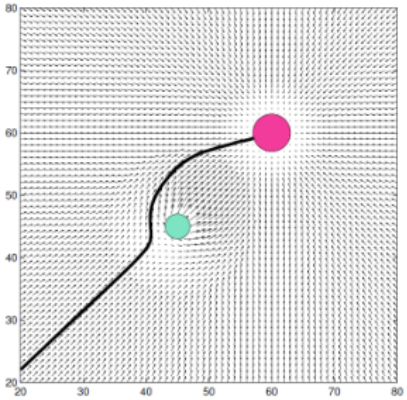


Exhaustive walk 1–2–4–2–3–5–6–5–7–5–8–1

Potential Field Method

The C-space is turned into a potential field, where the obstacles are surrounded by a repulsive field and the goal location by an attractive field. To navigate, the robot applies a force proportional to the negative gradient of the field - this is called gradient descent.

The advantage of potential field methods is that they are easy to compute. On the other hand, they can suffer from local minima (where robot gets stuck), and they don't consider dynamic constraints in their initial form (forces can be too high).



5 Roadmapping with Random Sampling

Instead of looking at the whole space, sampling based methods are usually more efficient because they only require point-wise evaluations. They are probabilistically complete, meaning the probability that they will produce a solution approaches 1 as more time is spent. However, they are not as robust as methods with full knowledge of the space and cannot determine if there is no solution to the path planning problem.

Multi-Query

A multiple query approach tries to capture the connectivity of the free space as good as possible, such that multiple, different queries for paths can be answered very fast. In other words: create a roadmap that is suitable for as many use cases as possible.

PRM - Probabilistic Roadmaps

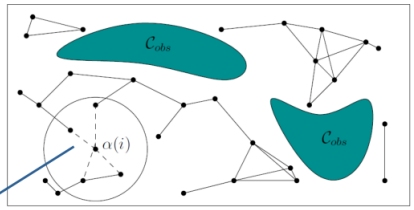
Basic steps for constructing PRMs:

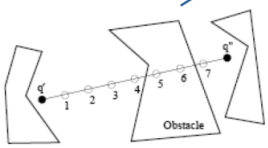
1. sample vertices and keep vertices that do not lie on an obstacle
2. find neighbour vertices

   k-nearest neighbour or

   neighbours within a specified radius
3. connect neighbouring vertices with edges (lines) (and check for collisions on connecting line using e.g. discretized line search)
4. add vertices and edges until roadmap is dense enough

1. Sample vertex
2. Find neighbors
3. Add edges



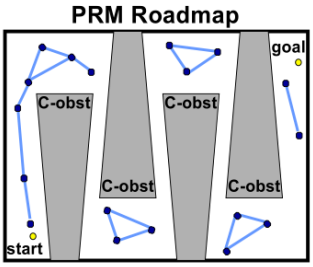


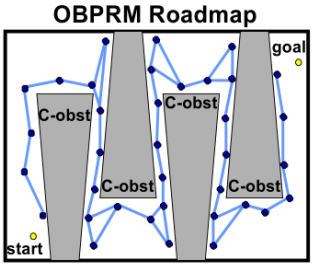
Step 3: Check edges for collisions, e.g., using discretized line search

Drawbacks: PRMs don't perform well when there are narrow passages.

OBPRM - Obstacle-Based PRM

Obstacle-based PRMs are constructed by sampling only close to obstacles. During sampling, the first goal is to find a point that lies inside an obstacle. Then, another point is sampled at an arbitrary distance to the first point. Using step-wise approximation, a point sufficiently close to the obstacle border is searched.



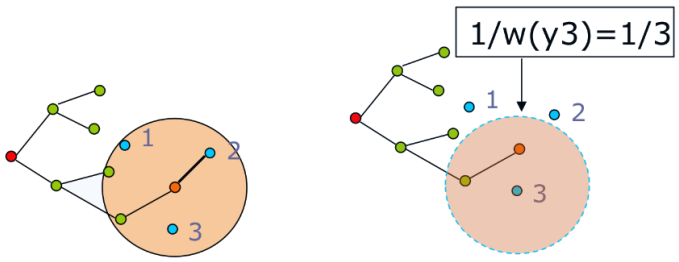


Single-Query

Single query planners try to solve a single query as fast as possible, without trying to cover the whole free space.

Weighted Randomized Tree Expansion

1. expand trees from start and goal
2. pick a node with probability $1/w(x)$, with $w(x)$ being the amount of neighbors within radius (measurement for exploration around x)
3. sample k points $(y_1, ..., y_k)$ around x
4. add y_i to the tree if
   (a) $1/w(y_i) > 1/w(x)$
   (b) y_i is collision free
   (c) y_i can see x
5. if a pair of nodes from start tree and goal tree are close and can see each other, then connect them and terminate



RRT - Rapidly Exploring Random Trees

1. pick q_{start} as the first node
2. pick a random target location (every n-th iteration, choose q_{goal})
3. find closest vertex in roadmap
4. extend this vertex towards target location
5. repeat steps until q_{goal} is reached

Note to point 2: q_{goal} is not chosen as target every time, because this would possibly lead the tree into an obstacle instead of exploring the free space.

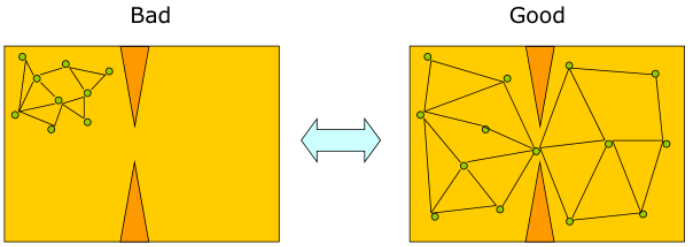
For faster execution, the tree can be grown from both q_{start} and q_{goal}

How To Sample

- unit line, square, cube: pick a random number $r \in [0, 1]$ for each dimension
- intervals shifted by c & scaled by s: $sr + c,\ r \in [0, 1]$
- circle: pick an angle $\theta \in [0, 2\pi]$ for a unit sphere uniformly at random
- rotations: depends on representation, in axis-angle notation the axis and the angle are sampled separately

Coverage, Connectivity, ϵ, α, β - Expansiveness

A PRM has good coverage if the milestones are distributed in such a way that (almost) any point in the free C-space can be connected to one milestone via a straight line. Also, the connectivity should be good, meaning that every milestone should be reachable from any other milestone. Especially with narrow passages, the connectivity can be hard to capture.



The coverage and connectivity are characterized by the expansiveness of the space (given by ϵ, α, β).

Definition from Hsu's paper: A free space F is (α, β, ϵ)-expansive, if it satisfies these conditions:

1. $\mu(V(p)) \geq \epsilon, \quad \forall p \in F$
   $\Rightarrow$ each point p must see at least a ϵ fraction of the free space
   F $\rightarrow$ F is ϵ-good
2. $\beta - \text{lookout}(S) = \{q \in S \mid \mu(V(q) \backslash S) \geq \beta \cdot \mu(F \backslash S)\}$
   $\Rightarrow$ for any $S \subseteq F$, the $\beta - \text{lookout}(S)$ is defined as the subset of points p that can see at least a β-fraction of the complementary space of S ($\equiv F \backslash S$). α is then defined as the relative volume of these points p to S: $\frac{\mu(\beta - \text{lookout}(S))}{\mu(S)}$

with $\epsilon, \alpha, \beta \in (0, 1]$, $V(\cdot)$ as the set of visible points of a point and $\mu(\cdot)$ denoting the volume of a set of points.

Burschka-Style:

1. let F be the free space (set of points); $\mu(G)$ the volume/ area of some set G; $reach(x)$ the space ($\subseteq F$) that can be seen from some point x; $reach(G)$ the space that can be seen from any point in a set G
2. find point $x \in F$ with smallest $\mu(reach(x))$
3. $S = reach(x)$
4. $\epsilon = \frac{\mu(S)}{\mu(F)}$
5. choose some set $Z \subseteq S$ that can reach a lot and has an easy to calculate area

6. $\alpha = \frac{\mu(Z)}{\mu(S)}$

7. $\beta = \frac{\mu(reach(Z) \setminus S)}{\mu(F \setminus S)}$ (not really according to the definition, but best way to do it in the exam)

8. the goal is to keep α and β balanced

Using these values, the number n of samples that is needed for a good connectivity can be calculated:

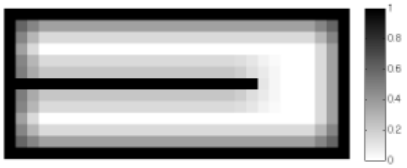
$$n = \frac{8 \ln(\frac{8}{\epsilon\alpha\gamma})}{\epsilon\alpha} + \frac{3}{\beta},$$

where $1 - \gamma$ measures the probability that the uniformly sampled milestones have the correct connectivity ($\gamma \in [0, 1]$).

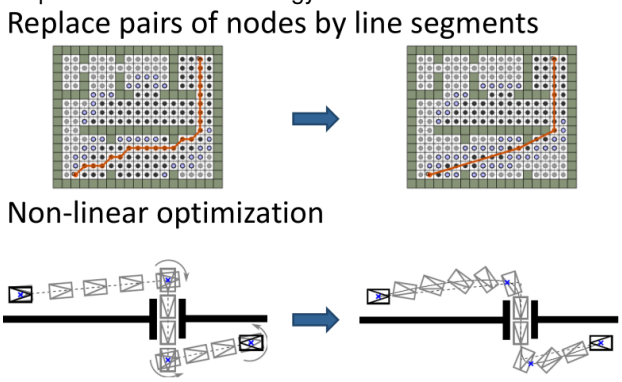
Path and Map Smoothing

The paths obtained by random sampling techniques are not necessarily the most efficient paths. Path smoothing techniques are used to optimize the path for length, energy, distance to obstacles etc.

To grow the distance to obstacles, the map can be smoothed by itself before planning. For example, a 2D-convolution with a Gaussian filter kernel leads to a non-zero probability distribution around the edges of the obstacles (edge smoothing).



After planning, the path can be optimized, too. For example, the number of "zick-zacks" of the path can be reduced by connecting pairs of nodes that are in sight of each other with straight lines. Furthermore, the path can be also optimized regarding the energy efficiency for the robot using some non-linear optimization technique to minimize the energy cost.



6 Probabilistic Robotics

Kalman Filter

predict/ time update

mean	$\hat{x}_{k+1}^- = A\hat{x}_k + Bu_k$	(1)
covariance $\sigma_3^2 = \sigma_1^2 + \sigma_2^2$	$P_{k+1}^- = AP_kA^\top + Q$	(2)

correct/ measurement update

Kalman Gain	$K_k = P_k^- H^\top (HP_k^- H^\top + R)^{-1}$	(3)
mean	$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$	(4)
covariance $\sigma_3^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}$	$P_k = (I - K_kH)P_k^-$	(5)

with

$\hat{x}$ - state estimate mean

$\hat{x}^-$ - predicted state estimate mean

A - state propagation/ transfer function

u - control input

B - mapping from control space to state space

P - state covariance

Q - process noise/ uncertainty covariance

K - Kalman Gain

R - measurement noise covariance

z - measurement

Observability

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

The system is observable if $rank(O) = n$ (O has n linearly independent rows).

Extended Kalman Filter

The (first order) EKF allows to use non-linear system models for Kalman filtering. The EKF adapts techniques from calculus, namely multivariate Taylor Series expansions, to linearize a model about a working point.

predict/ time update

mean	$\hat{x}_{k+1}^- = f(\hat{x}_k, u_k, 0)$	(6)
covariance	$P_{k+1}^- = A_{k+1}P_kA_{k+1}^\top + W_{k+1}QW_{k+1}^\top$	(7)

correct/ measurement update

Kalman Gain	$K_k = P_k^- H_k^\top (H_kP_k^- H_k^\top + V_kRV_k^\top)^{-1}$	(8)
mean	$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0))$	(9)
covariance	$P_k = (I - K_kH_k)P_k^-$	(10)

with

$f(x, u, n)$ - state transfer function

n - process noise

A - Jacobian of f with respect to x

W - Jacobian of f with respect to n

$h(x, v)$ - mapping from state space to measurement space

v - measurement noise

H - Jacobian of h with respect to x

V - Jacobian of h with respect to v

Unscented Kalman Filter

Instead of using first order approximations of the Taylor series (EKF), the unscented filter extends to the second (or higher) order approximations. EKF may cause significant error for highly nonlinear systems because of the propagation of uncertainty through the nonlinear system. However, higher order unscented EKFs tend to only provide performance benefits when the measurement noise is small.

The idea is to produce several sampling points (Sigma Points) from the current state and noise distributions. Then, propagating these points through the nonlinear map to get more accurate estimation of the mean and covariance of the mapping results. In this way, it avoids the need to calculate the Jacobian, hence incurs only the similar computation load as the EKF.

Martin: The indices in the slides seemed mixed up. I hope I corrected them right. I also changed the notation so it is similar to KF and EKF.

init

mean vector
$$\hat{x}^a_{k-1} = \begin{bmatrix} x^\top_{k-1} & 0 & 0 \end{bmatrix}^\top \tag{11}$$

contains mean values of state (x), process noise (v) and measurement noise (n)

covariance matrix
$$P^a_{k-1} = \begin{bmatrix} P_{k-1} & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & R \end{bmatrix} \tag{12}$$

contains covariances of state, process, and measurement

sigma points
$$\chi^x_{k-1} = \begin{bmatrix} \hat{x}^x_{k-1} & \hat{x}^x_{k-1} \pm \sqrt{(n_x + \lambda) P^x_{k-1}} \end{bmatrix} \tag{13}$$

do the same for v and n

first-order weights
$$W^{(m)}_0 = \frac{\lambda}{n_x + \lambda} \tag{14}$$

second-order weights
$$W^{(c)}_0 = \frac{\lambda}{n_x + \lambda} + (1 - \alpha^2 + \beta) \tag{15}$$

$$W^{(m)}_i = W^{(c)}_i = \frac{\lambda}{2(n_x + \lambda)} \tag{16}$$

weighting of sigma points ($x \in \mathbb{R}^{n_x}; i = 1, ..., 2n_x;$)

predict/ time update

sigma points
$$\chi^{x-}_k = f(\chi^x_{k-1}, \chi^v_{k-1}) \tag{17}$$

mean state
$$\hat{x}^-_k = \sum_{i=0}^{2n_x} W^{(m)}_i \chi^{x-}_{i,k} \tag{18}$$

covariance
$$P^-_k = \sum_{i=0}^{2n_x} W^{(c)}_i \left[\chi^{x-}_{i,k} - \hat{x}^-_k\right] \left[\chi^{x-}_{i,k} - \hat{x}^-_k\right]^\top \tag{19}$$

correct/ measurement update

pred. meas.
$$Y^-_k = h(\chi^x_{k-1}, \chi^n_{k-1}) \tag{20}$$

mean meas.
$$\hat{y}^-_k = \sum_{i=0}^{2n_x} W^{(m)}_i Y^-_{i,k} \tag{21}$$

meas. cov.
$$P_{y_k y_k} = \sum_{i=0}^{2n_x} W^{(c)}_i \left[Y^-_{i,k} - \hat{y}^-_k\right] \left[Y^-_{i,k} - \hat{y}^-_k\right]^\top \tag{22}$$

cross cov.
$$P_{x_k y_k} = \sum_{i=0}^{2n_x} W^{(c)}_i \left[\chi^{x-}_{i,k} - \hat{x}^-_k\right] \left[Y^-_{i,k} - \hat{y}^-_k\right]^\top \tag{23}$$

Kalman Gain
$$K_k = P_{x_k y_k} P^{-1}_{y_k y_k} \tag{24}$$

mean state
$$\hat{x}_k = \hat{x}^-_k + K_k(y_k - \hat{y}^-_k) \tag{25}$$

covariance
$$P_k = P^-_k - K_k P_{y_k y_k} K^\top_k \tag{26}$$

Bayes Filter

noch nicht fertig...
not yet written...

Particle Filter

The state representations of the Bayes and Kalman filter are restricted to probability density or Gaussian functions. They cannot model multiple hypotheses in their initial form, for example the uncertainty when a robot could be in one of multiple locations. (Multi-hypothesis KF could do that, but it's only a combination of multiple KFs).

The particle filter uses random sampling techniques together with probabilistic state estimation. The basic principle is:

1. if no initial distribution is given, start with a uniform sample distribution
2. apply motion model
3. apply sensor model
4. resample using Monte Carlo method (sample more into areas where the density of samples (certainty) is higher
5. repeat from 2