5 Mapping

Whenever a mobile robot is required to navigate beyond its sensory horizon, it must either rely on potentially ineffective or misleading local search strategies (such as the 'bug algorithms' [Lum87]) or use some kind of world model to store cues for navigation. Such a world model is generally called a 'map' and can either be provided a priori or built online using a mapping algorithm. The mapping approaches can be separated into world-centric or robot-centric. World-centric systems represent the pose of all objects including the robot of the environment according to a fixed coordinate frame. In indoor scenarios, a corner of a room or a fixed position in the entrance area of an apartment is often used. To specify positions in the operational environment of the robot in outdoor applications, global coordinate systems like the latitude, longitude, and height system, the Earth Centered, Earth Fixed Cartesian coordinate system, the World Geographic Reference System or WGS 84 (GPS) are often used. World-centric mapping is mainly employed for tasks like navigation or path planning while robot-centric approaches are used for piloting tasks such as collision avoidance. Using matrix-based coordinate transformations, it is possible to convert between these different reference frames.

The main problem of generating maps is the inaccuracy of the sensor systems being used for solving the localization problem and measuring objects in the environment of the robots. Therefore, it is very difficult to build global maps based on local ones, to update existing maps or to correspond the objects stored in the map with those measured in the environment of the robot.

Concerning map types, there are numerous ways to model the environment of a vehicle. The data structures found in literature can be broadly divided into the four classes of *metrical*, *grid*, *topological* and *hybrid* maps (see figure 5.1).

Purely metrical maps are probably the most common type. Systems relying on these maps are characterized by using one global, metrically consistent frame of reference. The accuracy of the stored map is approximately equal to the quality of available sensor data. Also, all metrical locations are equally important. Metrical maps comprise geometric features and their spatial locations. The features actually used can range from very basic (such as 3D points calculated from range measurements) over geometrically more

expressive line or box features up to semantically very distinct landmarks, which can be uniquely identified from a large body of sensor data.

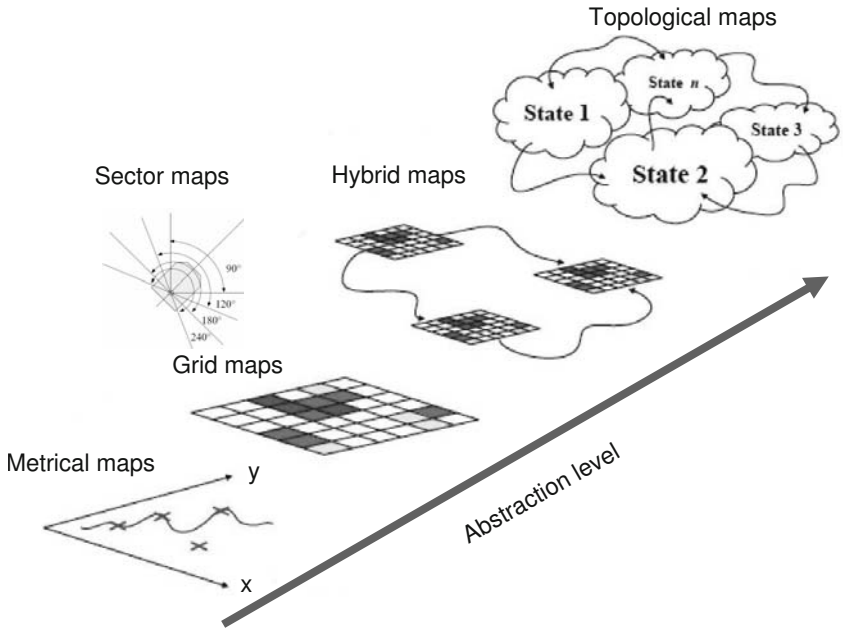


Figure 5.1 Abstraction level of the different mapping approaches

Grid maps, which are very often used for mobile robots, are popular due to their simplicity and intuitive representation. A grid map divides space into adjacent portions of equal metrical sizes. For a two-dimensional map, this results in a square grid, while the three-dimensional grid map resembles a Rubic's cube. Both dimensionalities have been used [Elf89, Mor96], but three-dimensional grid maps are rare due to their excessive storage requirements. Two major variants of grid maps are occupancy grids and elevation maps.

Unfortunately, the highly detailed metrical world representation requires a lot of memory and leads to algorithms with high computational demands. These properties limit the scalability of both grid and metrical maps [Bro87].

Motivated by these drawbacks, researchers aiming at large scale navigation early on began looking at the *topological* world model, which represents the environment in a more compact, qualitative fashion. Topological approaches focus on representing navigation-relevant places and their connections on an abstract level rather than the exact metrical layout of the surroundings. Thus, imprecise localization is less of a problem for topological approaches and algorithms can run faster because they have to cope with

much less data. Topological maps commonly use graphs as their underlying data structure. Graph nodes identify locations of interest and their characteristic features, while knowledge about travel between nodes is encoded in the connecting graph edges.

More recently, many researchers have proposed to attack the problems of autonomous mapping using combinations of the metrical and topological methodologies. Generally, these *hybrid* approaches are designed to combine the benefits of both representational forms, ideally allowing localization and map building with the high precision of metrical maps while retaining the computational tractability and compactness of topological data structures.

In the following, several successful examples of mapping techniques using different map types are presented.

5.1 Metrical maps

In metrical maps, the environment is described with the help of geometrical features. These geometrical features can be 2D or 3D points, lines, polygons or 2D areas like rectangles. As an example for indoor scenarios, lines could describe the walls of rooms. In outdoor scenarios, lines could denote streets or highways.

Metrical mapping offers several advantages. Geometrical features can be maintained over time even if their positions change. Thus it is obvious that this type of map is also suited for dynamic environments. Another advantage is that metrical feature maps offer a more compact description of the surroundings than grid maps. Hence they are superior especially in a scenario with a rather structured environment.

In the following the line-based and plane based metrical maps, which can both be generated by a robot based on distance measurements, are introduced.

5.1.1 Line based metrical maps

Lines extracted from distance measuring sensors form borders between regions that can be divided into either free space, obstacles or unknown regions. Obstacles may be represented by clusters or border lines. In the following, a method is presented to extract both from (laser) distance measurements.

Line segmentation The idea of the line segmentation technique is to cluster a radar scan into groups of distance points that lie close together. Let us take a radar scan $\{r_i,\varphi_i\}$ with $i = 1,\ldots,n$ measured from position $Q = (x_0,y_0,\psi)$. Let $P_i = (r_i,\varphi_i)$. As results, corners C_j, segments S_k and auxiliary clusters H_m will be formed.

The procedure is shown in algorithm 5.1.

Algorithm 5.1 Line segmentation

Initialization: $i := 1$; $j := 1$; $k := 1$; $m := 1$; $H_1 := (P_1)$
while $i < n - 1$ **do**
  if $|P_{i+1} - P_i| < d$ **then**
    $H_m := H_m \cup P_{i+1}; i := i + 1;$
  else if $|P_{i+2} - P_i| < d$ **then**
    $H_m := H_m \cup P_{i+2}; C_j := (P_{i+1}); j := j + 1; i := i + 1;$
  else if $|P_{i+2} - P_{i+1}| < d$ **then**
    $m := m + 1; H_m := (P_{i+1},P_{i+2}); i := i + 2;$
  else
    $C_j := (P_{i+1}); j := j + 1; m := m + 1; H_m := (P_{i+2});$
  end if
end while
while $m > 0$ **do**
  if number in $(H_m) \leq c$ **then**
    $C_j := H_m; j := j + 1; m := m - 1;$
    (* c is the maximal number in a cluster*)
  else
    $S_k := H_m; j := j + 1; m := m - 1;$
  end if
end while

Iterative end point fit [DH73]

Take a segment S_k with points $P_q,\ldots,P_r$ according to figure 5.2 and form the distances $h_{r-1},\ldots,h_{q-1}$. They are calculated following figure 5.3 for two points P_q and P_r. Let ψ be the angle of the line connecting both points with respect to the common coordinate system.

$$\tan\psi = \frac{y_q - y_r}{x_q - x_r} \tag{5.1}$$

$$h_i = ((y_i - y_q) - (x_i - x_q)\tan\psi)\cos\psi \tag{5.2}$$

$$\Rightarrow \boxed{h_i = (y - i - y_q)\cos\psi - (x_i - x_q)\sin\psi} \tag{5.3}$$

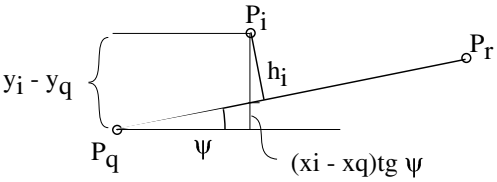


Figure 5.2 Distance to line

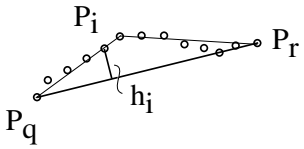


Figure 5.3 Iterative endpoint fit

Take the line between P_q and P_r. If $h_j = \max(h_i) > \varepsilon$ for $r < i < q$; (*ε is the allowed fuzziness in deviations from a line*) then form two lines (P_r,P_j) and (P_j,P_q) and repeat the calculation, else the line (P_r,P_q) is established. The effort is equal to the number of lines to be inserted between P_q and P_r.

Regression line Find a line through N points $P_q, \ldots, P_r$ so that the sum of the squared distances to that line gets minimal. There are $h_i = (y_i - y_q)\cos\psi - (x_i - x_q)\sin\psi$ the distances to that line. Set for a moment $P_q = (0,b)$, then the distances become $h_i = (y_i - b)\cos\psi - x_i \sin\psi$.

The sum of the squares is

$$\sum_i h_i^2 = \sum_i (y_i - b)^2 \cos^2\gamma - 2(y_i - b)x_i \sin\gamma\cos\gamma + x_i^2 \sin^2\gamma \tag{5.4}$$

This is to be minimized with respect to b:

$$\frac{\partial(\sum(\ldots))}{\partial b} \stackrel{!}{=} 0$$

$$\sum_i -2(y_i - b)\cos^2\gamma + 2x_i \sin\gamma\cos\gamma \stackrel{!}{=} 0 \tag{5.5}$$

$$\sum_i -(y_i - b)\cos\gamma + x_i \sin\gamma = 0 \tag{5.6}$$

$$\sum_i b - y_i + x_i \tan\gamma = 0 \tag{5.7}$$

$$N \cdot b = \sum_i y_i - \tan\gamma \sum_i x_i \tag{5.8}$$

$$y_s = \tan\gamma x_s + b \tag{5.9}$$

This is a line through the center of gravity of the points. The next step now is to minimize the squares of distances: The squares of the distances are

$$\sum_i h_i^2 = \sum_i (y_i - b)^2 \cos^2\gamma - 2(y_i - b)x_i \sin\gamma\cos\gamma + x_i^2 \sin^2\gamma \tag{5.10}$$

The angle γ is to be minimized:

$$\frac{\partial(\sum(\ldots))}{\partial\gamma} \stackrel{!}{=} 0$$

$$\sum_i^N -2(y_i-b)\cos\gamma\sin\gamma - 2(y_i-y_s)(x_i-x_s)(-\sin^2\gamma+\cos^2\gamma) + \sum_i^N 2(x_i-x_s)^2\sin\gamma\cos\gamma \stackrel{!}{=} 0 \quad (5.11)$$

$$\sum_i^N (y_i-y_s)^2\tan\gamma + (x_i-x_s)(y_i-y_s)(1-\tan^2\gamma) + (x_i-x_s)^2\tan\gamma = 0 \quad (5.12)$$

$$\sum_i^N (y_i-y_s)(x_i-x_s)\tan^2\gamma + \sum_i^N(-(y_i-y_s)^2+(x_i-x_s)^2)\tan\gamma - \sum_i^N (y_i-y_s)(x_i-x_s) = 0 \quad (5.13)$$

The last equation is a quadratic equation in $z = \tan\gamma$:

$$a\cdot z^2 + b\cdot z - a = 0 \quad (5.14)$$

$$a = \sum_i^N (y_i-y_s)(x_i-x_s) \qquad b = \sum_i^N (y_i-y_s)(x_i-x_s) \quad (5.15)$$

$$\boxed{z_{1,2} = -\frac{b}{2a} \pm \sqrt{1+b^2/4a^2}} \quad (5.16)$$

Shift of line end points Given a line through the center of gravity for a couple of points, find proper endpoints Q and R in the vicinity of P_q and P_r. Figure 5.4 shows the situation.
Let $\psi \approx \gamma$. Then

$$h_s = (y_s-y_q)\cos\psi - (x_i-x_s)\sin\psi \quad (5.17)$$

$$\sum h_i - h_s = \sum (y_i-y_s)\cos\psi - (x_i-x_s)\sin\psi \quad (5.18)$$

$$\sum h_i - h_s = \cos\psi(\sum y_i - N\cdot y_s) - \sin\psi(\sum x_i - N\cdot x_s) = 0 \quad (5.19)$$

This means a shift of the line end points by h_s:

$$P_q \longrightarrow Q = (x_q+\Delta x, y_q+\Delta y) \qquad \Delta x = h_s\sin\psi \quad (5.20)$$

$$P_r \longrightarrow R = (x_r+\Delta x, y_r+\Delta y) \qquad \Delta y = h_s\cos\psi \quad (5.21)$$

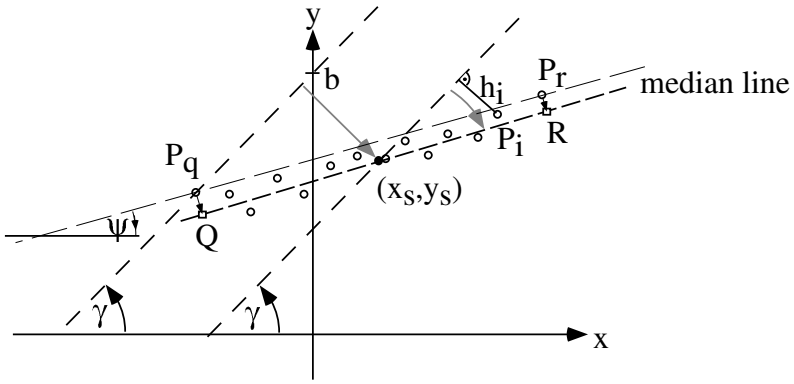


Figure 5.4 Fitting the regression line

Fusion of lines Let the vehicle take radar pictures from different positions P_0 and P_1. Let (P_i,P_k) be a line extracted from radar scan R_1 at P_0 and (P_i',P_k') extracted from radar scan R_2 at P_1. Let the angles be ψ and $\psi' = \psi + \varepsilon$. Figure 5.5 shows two cases of fusions:

- Let $h_i' < \delta$ for P_i' with respect to the line (P_i,P_k) and P_i' between P_i and P_k and moreover $|P_i',P_k| > |P_i',P_k|$ then both lines are condensed into one line (P_i,P_k').
- Let $|P_k,P_i'| < d$ then both lines are fused into one line (P_i,P_k'), too.

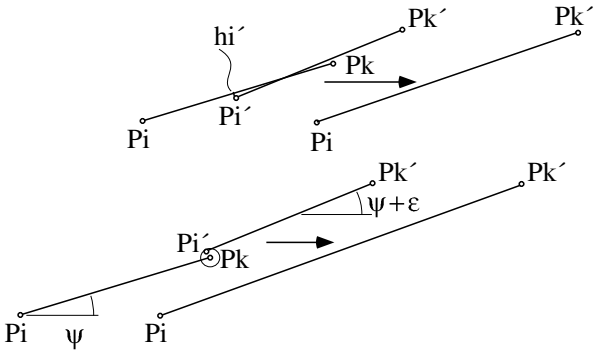


Figure 5.5 Fusion of lines

Obstacles vs. free space Introducing a direction in a line allows to differentiate between free space and possible obstacles for a vehicle, as shown in figure 5.6: to the right of a line from P_1 to P_2 there is a free space open, otherwise it could not have been constructed. To the left is unknown territory, possibly the line is the border of an obstacle. A room may be described by a polygon with lines oriented clockwise; they might also describe isolated obstacles if the lines are oriented counter clockwise.

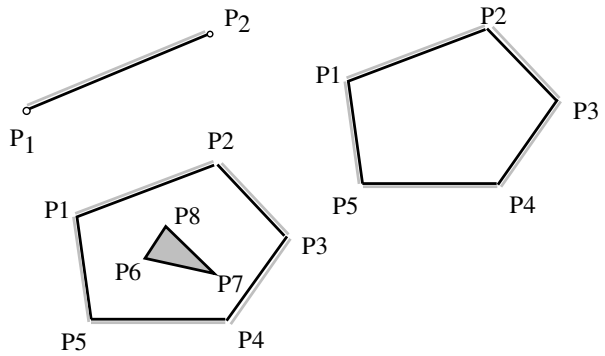


Figure 5.6 Defining free space

Given a radar scan with some lines extracted, a certain amount of free space is established. Driving around, this free space can be enlarged as shown in figure 5.7. In general some parts of the environment remain unknown.

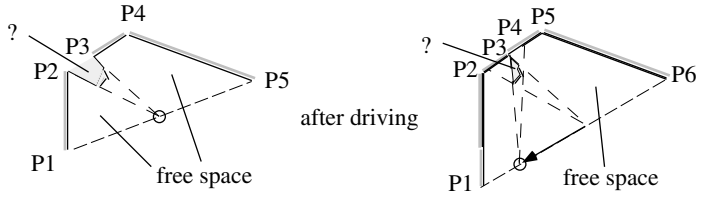


Figure 5.7 Enlarging free space by driving around

5.1.2 Plane based metrical maps

Planes are often used as features to define reliable landmarks such as floor, ceiling, walls, doors and big objects of furniture like desks and cupboards. The main problem of this environment representation is the extraction of planes. A typical plane extraction algorithm [WGS03] is based on a 3D

point cloud. The corresponding data acquisition is based on 3D sensor systems as rotating 2D laser scanners (see figure 5.8) or time-of-flight cameras. Figure 5.9 shows a typical 3D point cloud of a real cluttered indoor scene.

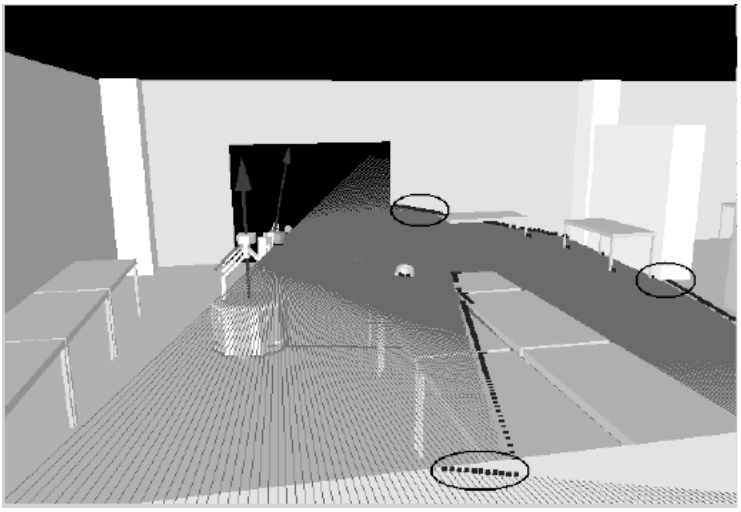


Figure 5.8 3D data acquisition in a virtual indoor scene. The laser beams are visualized as rays and the measurement points as black dots.

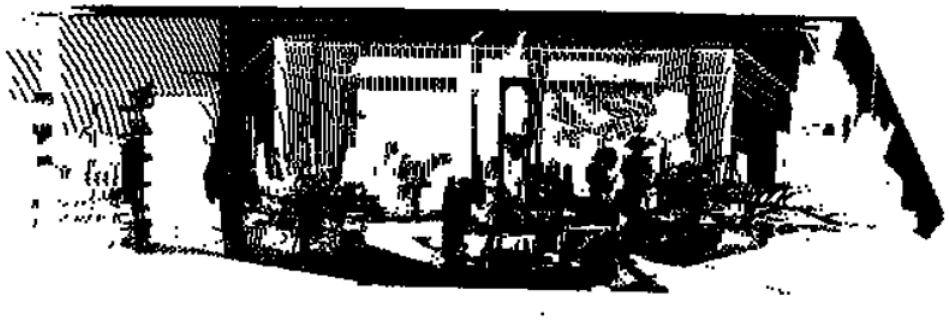


Figure 5.9 3D point cloud of a typical indoor scene

The goal is to approximate the input data by a set of planar patches so that each set of points is optimally represented in a least square sense by its plane. Features represented by a large amount of samples (e. g. corridor walls) are reduced to one large planar patch. Figure 5.10 shows a result of this process applied to the 3D point cloud shown in figure 5.9.

Figure 5.10 Planes extracted from the 3D point cloud of figure 5.9. The ground and ceiling plane are marked by a bounding box.

The following gives a summary of the different steps of the extraction method. The complete strategy is described in more detail in [Ast07]. Figure 5.11 shows a flow chart of the whole procedure and algorithm 5.2 sketches the different processing steps.

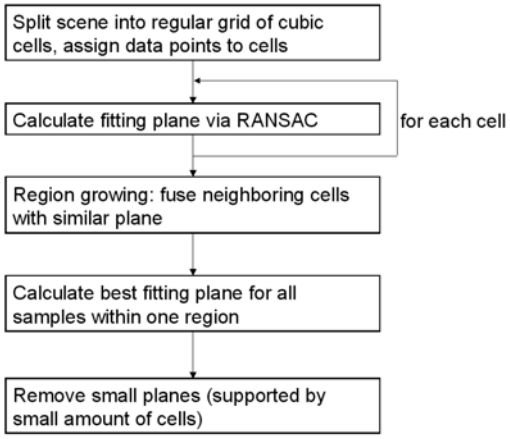


Figure 5.11 Flow chart of RANSAC based plane extraction

In step 1 and 2 the whole point cloud is split into a regular grid of cubic cells (see figure 5.12) in order to perform plane fitting locally. In this way smaller features as chairbacks and screens can be extracted. Big planes from floor, walls and ceiling are detected via region growing in step 4. The RANSAC algorithm in step 3 repeatedly calculates planes approximating the local set of points within one cell. As the cells are handled independently, this step can be executed in parallel on modern CPUs. In each iteration three (not colinear)

points are selected randomly within the local set. Then a plane through these points is calculated and the number of inliers are counted (points in the cell which lie within a certain distance threshold to this plane). This number of supporting points is the maximization criterion: the calculated plane is used as best plane when its number of supporting points is bigger than the one of the best plane calculated during the previous iterations.

Algorithm 5.2 RANSAC algorithm for plane extraction

Given: a set of 3D distance measurement samples (data points)
Return: a set of planes approximating disjunctive subsets of the input points
Step 1: split the whole 3D scene into a regular grid of cubic cells
Step 2: assign the input points to the corresponding cells
Step 3: calculate fitting plane for each cell:
for every cell **do**
  find approximating plane using RANSAC algorithm (output: best fitting plane after certain number of RANSAC iterations)
  remove outliers with respect to the best RANSAC plane
  calculate least-square fitting plane for inliers
end for
Step 4: region growing – fuse matching planes of neighboring cells:
for every cell **do**
  compare plane parameters with those of all neighboring cells
  if the angle between plane normals is below the angular threshold and the distance of the center of gravity of points of neighboring cell to the plane is below the distance threshold **then**
    mark both cells as belonging to the same region
  end if
end for
for all regions **do**
  calculate best fitting plane for all points of cells belonging to the same region
end for
Step 5: remove small planes supported only by a small amount of cells

This procedure is motivated by the fact that the repeated random selection of points and plane approximation converges to a "good" fitting plane if such a plane exists at all (model valid) and the number of iterations is high enough.

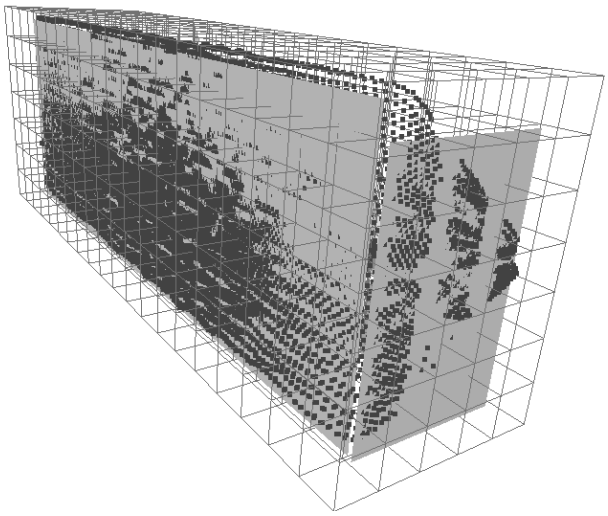


Figure 5.12 Splitting of a 3D point cloud into local sets via cubic grid cells

The plane fitting for all inliers within one cell as well as for all samples of one merged region is calculated via *principal component analysis*. A plane is described by its normal $\overrightarrow{n}$ and distance from origin d (equation 5.22).

$$< \overrightarrow{n}, \overrightarrow{x} >= d \tag{5.22}$$

According to [WGS03], the normal of the least-square fitting plane is the eigenvector of the smallest eigenvalue of covariance matrix A (equation 5.23).

$$A = \begin{pmatrix} \sum_{i=0}^{N} w_i x_i^2 & \sum_{i=0}^{N} w_i x_i y_i & \sum_{i=0}^{N} w_i x_i z_i \\ \sum_{i=0}^{N} w_i x_i y_i & \sum_{i=0}^{N} w_i y_i^2 & \sum_{i=0}^{N} w_i y_i z_i \\ \sum_{i=0}^{N} w_i x_i z_i & \sum_{i=0}^{N} w_i y_i z_i & \sum_{i=0}^{N} w_i z_i^2 \end{pmatrix} \tag{5.23}$$

Here, $x_i = x_i^{\text{raw}} - \overline{x}$, $y_i = y_i^{\text{raw}} - \overline{y}$ and $z_i = z_i^{\text{raw}} - \overline{z}$ are input points centered around mean and weights w_i represent measurement uncertainty of the samples (set to 1 as uncertainty is not yet taken into account). d is given by equation 5.24

$$d =< \overrightarrow{cog}, \overrightarrow{n} > \tag{5.24}$$

where $\overrightarrow{cog} = (\overline{x},\overline{y},\overline{z})^T$ is the center of gravity of points belonging to the fitted plane. As A is a square symmetric matrix, its eigenvalues can be computed efficiently via *singular value decomposition*.

Table 5.13 lists all adjustable parameters used by the described algorithm. One question is how to choose reasonable values for the size of cubic cells and the minimum number of points within each cell needed for extracting stable features. Naturally, the density of 3D samples decreases with increasing distance between sensor and object (see figure 5.14). Assuming an angular scan resolution of $\alpha = 0.5°$ horizontally and vertically and a distance between adjacent laser beams of e =10 cm, a regular grid of 3×3 samples fits within a cell of 20×20×20 cm^3. In this case the maximum distance between sensor and objects is $d = \frac{e}{\tan \alpha} \approx 11.5$ m. This example shows that objects which are further away than ≈ 11.5 m from the sensor generate too few samples for reliable plane extraction. Consequently, cell size and minimum number of samples have to be chosen depending on sensor setup, environmental conditions and size of features of interest.

step	parameter	value
1	cell size	200 mm
3	min. amount of points in a cell to start RANSAC alg.	10
3	number of RANSAC iterations	50
3	max. point-to-plane distance for inliers	50 mm
4	angular threshold for normals of neighboring planes	15°
4	distance threshold for cog of one plane to neighb. plane	50 mm
5	min. amount of supporting cells for plane filtering	10

Figure 5.13 Parameters used for plane extraction

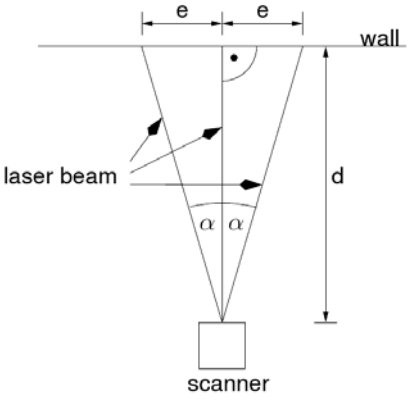


Figure 5.14 Distance between adjacent sampling points depends on distance between sensor and object

An example run of the described algorithm is shown in figure 5.10, with input data from figure 5.9. The scan resolution is 0.5° horizontally and 0.3° vertically and the number of samples is about 92000. The parameters have been chosen as shown in table 5.13. The plane extraction procedure took less than 2 seconds. For visualization, the resulting planes are clipped by the bounding boxes of their supporting samples. As expected, most stable features arise from floor and ceiling, followed by walls and cupboards. The window frames in the central part of the figure are represented by several small planes where region growing sometimes failed due to chosen plane orientation thresholds.

The output of the presented algorithm is a collection of 3D planes. At a higher level of environmental representation, these planes can now be used to extract semantically meaningful features such as the walls of a room and objects of furniture. However, corresponding strategies for this are beyond the scope of this book.

5.1.3 Feature-based metrical maps

Geometric invariants of a scan At first, the regions of the map and the current scans which can be matched must be determined as the algorithms try to do partial matching. At least the extending information in the current scan cannot be part of the global map and therefore cannot be matched. The relative movement between two scans can be estimated using odometry, as the typical error does not accumulate over time if we do not try to estimate the global pose of the robot. This helps to identify the correct region. Another way of finding a good initial estimate is the usage of invariant attributes, that allow to compare two consecutive scans, despite to slight changes in position.

An example of invariants is the afore mentioned **box feature**. Slight variations in furniture or in the position of the vehicle in the room do not change the boxes seen. Another invariant is the **center of gravity** of a scan. Slight variations in the position and a turning at the same place will not change the center of gravity.

Let (x_i,y_i) be a scan taken in an robot centered coordinate system, then the center of gravity is given by

$$x_s = 1/N \sum_{N}^{i=1} x_i \tag{5.25}$$

$$y_s = 1/N \sum_{N}^{i=1} y_i \tag{5.26}$$

The center of gravity is an example of the concept of general **moments** $M_{p,q} = 1/N \sum_{N}^{i=1} (x_i)^p (y_i)^q$. For instance $M_{10} = x_s$ and $M_{01} = y_s$. Second order moments are the **moments of inertia** with respect to an axis under angle ψ through the center of gravity (x_s,y_s): Let d_i be the distance to the axis from point P_i with coordinates (x_i,y_i) then

$$f(\psi) = \sum_{i=1}^{N} (d_i)^2 \tag{5.27}$$

$\Rightarrow$ The line through (x_s,y_s) under an angle ψ is given by

$$y = m \cdot x + b \tag{5.28}$$

Then the distance of the line to the point(x_s,y_s) is calculated as shown:

$$y_s = m \cdot x_s + b \tag{5.29}$$

$$m = \tan \psi \tag{5.30}$$

$$b = y_s - m \cdot x_s \tag{5.31}$$

$$y_q = m \cdot x_i + b \tag{5.32}$$

$$\cos \psi = d_i / (y_i - y_q) \tag{5.33}$$

$$d_i = (y_i - y - q) \cdot \cos \psi \tag{5.34}$$

$$(d_i)^2 = (y_i - \tan \psi \cdot x_i - y_s - \tan \psi \cdot x_s)^2 \cdot \cos^2 \psi \tag{5.35}$$

Figure 5.15 shows the relation.

From here the moment of inertia through (x_s,y_s) with respect to angle ψ is

$$f(\psi) = \sum_{i=1}^{N} (d_i)^2 \tag{5.36}$$

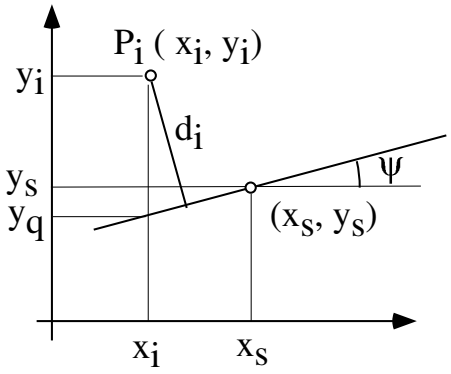


Figure 5.15 Distance with respect to the center of gravity and the angle *psi*

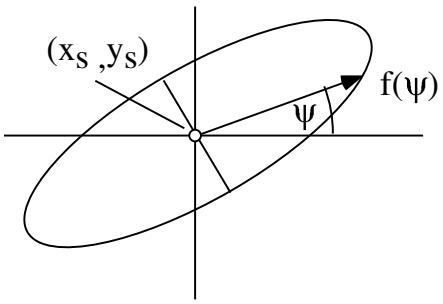


Figure 5.16 $f(\psi)$ forming the inertia ellipsis

Running through all angles ψ $f(\psi)$ forms an ellipsis, regardless of the given point cloud. The quotient of the length of the small and the long axis of the ellipsis as well as their direction are invariants of the point cloud as sketched in figure 5.16.

As the influence of points far away from the center of gravity is rather strong, a smoothing function is introduced: let d_0 be the medium distance to the center of gravity and d_i be calculated as above. Then d_i is smoothed by a function as shown in figure 5.17.
Here, a parameter α defines the steepness of the function described by

$$d_0 = \frac{1}{N} \cdot \sum_{i=1}^{N} \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2} \tag{5.37}$$

$$d_i := \frac{d_i}{1 + \exp\left(\alpha \cdot (d_i - d_0)/d_0\right)} \tag{5.38}$$

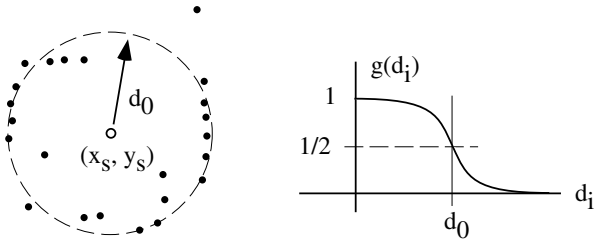


Figure 5.17 Left: center of gravity in a scan and medium distance of scan points right: a smoothing function, a parameter α describes the steepness of the function

While analyzing the data gathered during a scan of a laser distance sensor, one is able to distinguish four different kind of features as depicted in figure 5.18:

1. false edges: a sudden change in distances but no real edge
2. limit wall/ round edge
3. edge between two walls
4. real jump edge

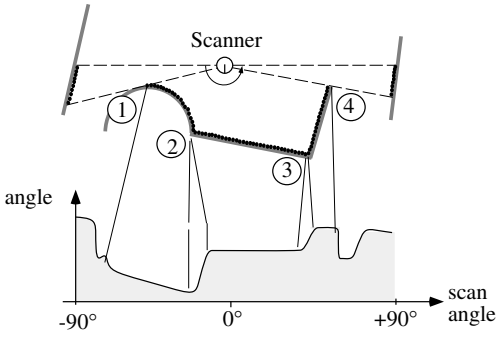


Figure 5.18 Different types of anchor points

Other than that, virtual edges occur. They can be found by applying a histogram based criterion. First of all, the scan image is rotated according to the main preferential orientation. Afterwards histograms are generated based on the scan data as illustrated in figure 5.19. Hence one is now able to determine the virtual intersection points of wall segments. The edge extraction process is presented in figure 5.20.

Using a scan (r_i,ϕ_i), segmentation can be performed according to algorithm 5.3, assuming $\Delta r_i = r_{i+1} - r_i$. Feature numbers 2 through 4 as well as virtual edges serve as anchor points of a scan. The distance of two anchor points P_i and P_j (with their coordinates (r_i,ϕ_i) and respectively (r_j,ϕ_j)) can be determined as shown in equation 5.39.

$$d_{ij} = \sqrt{(r_i)^2 + (r_j)^2 - 2r_i r_j \cos(\phi_j - \phi_i)} \tag{5.39}$$

These distances are invariants of a scene.

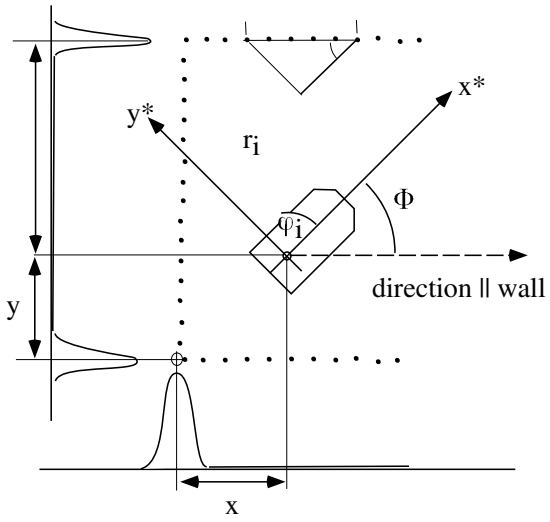


Figure 5.19 Anchor points from wall edges

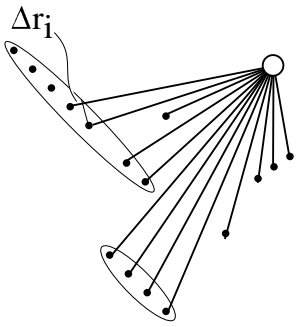


Figure 5.20 Finding edges in a radar scan

Algorithm 5.3 Segmentation

```
for i = 1 to N do
  if |Δr_i| ≤ s then
    form segment of (r_i,φ_i) and (r_{i+1},φ_{i+1})
    i := i + 1
  else if |Δr_i| > s and |(r_{i+2} − r_i)| ≤ s then
    add (r_{i+2},φ_{i+2}) to segment
    mark (r_{i+1},φ_{i+1}) as singular point
    i := i + 2
  else if |Δr_i| > s and |δr_{i+1}| < s then
    start new segment for (r_{i+1},φ_{i+1})
    mark edge at (r_i,φ_i)
    i := i + 1
  else if |Δr_i| > s and |δr_i| > s then
    false edge detected
    mark (r_{i+1},φ_{i+1}) is a singular point
    i := i + 2
  else
    mark (r_i,φ_i) is a singular point
    i := i + 1
  end if
end for
```

Extraction of anchor point type Let us assume the segments $\alpha(\phi_i)$ are given. Parts that are parallel to the ϕ-axis correspond with parts with a constant angle with respect to the orientation of the vehicle. Traversing from one parallel part to another, one denotes an edge (type 3). Straight parts not parallel to the ϕ-axis denote curved parts in the scene. Going from one part to another denotes a (type 2) edge. A discontinuity in the r_i-values denotes a real jump-edge (type 4). Hence, the scan can be transformed into a graph with vertices annotated with the type of edge and the edges with the distances as shown in figure 5.21 taken from [Web02]. The graph at hand is a compact description of a scene, invariant to slight changes in position and the orientation of the vehicle. In order to find the location of the vehicle, a graph corresponding to the scene just captured can be matched with graphs generated earlier. This approach remains within manageable dimensions as long as there are only a few graphs to be compared. This is the case if the vehicle position is approximately known.

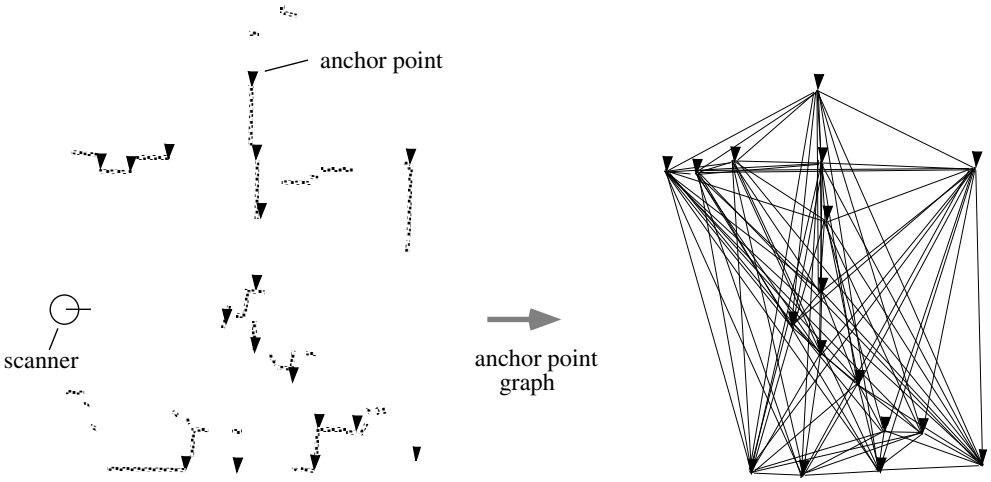


Figure 5.21 Generation of an Anchor Point graph (AP-graph) based on laser scanner measurements

Graph comparison

As one can see in figure 5.21, the anchor points found earlier can be connected in order to form a complete graph. This graph does not simply include the anchor points, but rather their relation is expressed by the help of edges that contain distance information as well. Assume two complete graphs G and $G*$ of a scene as given. Let us further assume that G features n anchor points and $n(n-1)$ distances and $G*$ contains m anchor points and $m(m-1)$ distances. The comparison of the two graphs can thus be reduced to a subgraph matching problem.

The momentary scan may involve 'ghost edges': Those can be either temporary object edges of obstacles (rather rare, only occurring with large obstacles) or the more common case of real edges that are obscured by obstacles.

In general only partial matching can be performed. In order to still be able to assign corresponding scenes, a particular search beginning with large distances in a data bank has to be implemented. Hence, if the number of found matches exceeds a preset threshold, the scene can be assumed as recognized.

Feature matching, continuous values Let there be two scenes with invariant features $M_i,\ldots,M_k$ and $M_i^*,\ldots,M_k^*$. Further, let the range of possible values of each feature M_i be continuous and thus $M_i \in [m_{ia},m_{ie}]$. Therefore, scenes are described by feature vectors $\mathbf{M^T} = (M_1,\ldots,M_k)$ and

$\mathbf{M}^{*\mathbf{T}} = (M_1^*, \ldots, M_k^*)$ interpreted as points in a room of k dimensions. A comparison is then based on the distance of the end points. In order to do that, the values have to be normalized $M_i \in (m_{ia}, m_{ie}) \rightarrow M_i \in [0,1]$:

$$M_i \longrightarrow \frac{M_i - m_{ia}}{m_{ie} - m_{ia}} \tag{5.40}$$

This assures all features are treated equally and the vector end-points define points in a k-dimensional unit cube. The Euclidean distance between M and M^* can be denoted as

$$d = \sqrt{\sum_{i=1}^{k} (M_i - M_i*)^2} \tag{5.41}$$

$$d \in [0, \sqrt{k}] \tag{5.42}$$

Feature matching, discrete range of values Let us assume the existence of a discrete range of values for the features $M_i \in (m_{i1}, m_{i2}, \ldots, m_{in})$. If a relation $m_{i1} < m_{i2} < \cdots < m_{in}$ exists, then $m_{ir} \longrightarrow r$ and normalized $M_i \longrightarrow (M_i - 1)/(n - 1)$. If however this is not the case, only complete equivalence is considered: let $M_i = m_{ir}$ and $M_i^* = m_{is}$ then $d_i = (M_i - M_i^*) = (1 - \delta_{rs})$. If this equivalence is a condition sine qua non then

$$d = \delta_{rs} \cdot \sqrt{\sum_{j \neq i} (M_j - M_j^*)^2} + \sqrt{k} \cdot (1 - \delta_{rs}) \tag{5.43}$$

and otherwise $d = d_{\max}$.

5.2 Grid maps

Grid maps model the environment as a regular grid of cells with constant areas. The cells are filled with information extracted from noisy sensors. In order to produce consistent maps, this process requires either a known robot pose or a good pose estimate. The sensors used are predominantly ranging sensors like sonar or laser scanners. The lack of precise knowledge is expressed through various kinds of regions: high occupancy regions indicate objects, while lower values most likely represent free space. The advantage of this specific kind of map is the robustness and easy implementation. Their disadvantage is that they rely on exact pose estimation.

5.2.1 Occupancy grid maps

For many tasks it is useful to raster the environment. In general, a square raster with tiles of d cm width is used. Each tile gets a pair of indices (i,k), with d adjusted to the task at hand.

It is simple to build a grid map from laser radar distance readings. The dimension of a tile should be the resolution of the grid map. With $\Delta r \approx \pm 2.5$ cm and a radar cone of 0.5° up to a distance of $r = s \cdot 360/(2\pi \cdot 0.5) = 570$ cm a radar point occupies one tile and the cone denotes free space as shown in figure 5.22.

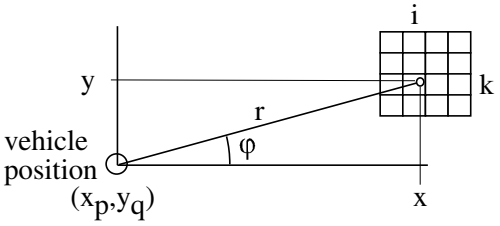


Figure 5.22 Building a grid map from laser radar measurements

Let $x = r\cos\varphi$ and $y = r\sin\varphi$ be the coordinates of the radar point and (x_p,y_q) the coordinates of the vehicle in a 2D scenario. The corresponding tile with index i,k and center (x_i,y_k) is given by

$$x_i - \frac{d}{2} < x + x_p \leq x_i + \frac{d}{2} \qquad x_i = i \cdot d \tag{5.44}$$

$$y_k - \frac{d}{2} < y + y_q \leq y_k + \frac{d}{2} \qquad y_k = k \cdot d \tag{5.45}$$

Each of the tiles described above is marked as free space (white) or obstacle (hatched) or partly free space (gray). Figure 5.23 shows a typical grid map with the mentioned parameters. If tiles are marked as occupied for vehicle navigation, there should remain a safety clearance of n of free space between the vehicle and the occupied region. E.g. suppose that $n = 7$ cm; with $d = n/\sqrt{2} \Rightarrow d = 5$ cm. In this case a grid map with 400 tiles/m^2 will be generated (for a field of 100 m×100 m, $4 \cdot 10^6$ tiles will be built up).

For a compact description of grid maps a **quadtree** representation can be applied. The environment is split recursively into blocks of $2^i \times 2^i$ tiles as shown in figure 5.24.

A node in the tree represents a $2^i \times 2^i$ square part of the environment. It is either a free space or represents an obstacle or is a mixture of both. If a node is marked as obstacle or free it will not be split any further. A mixed

node will be recursively split in 4 equal areas, represented as 4 new nodes of the tree. If the whole environment is made up of $2^k \times 2^k$ tiles, then in general the quadtree will have much less than 2^{2k} nodes. The procedure is shown in algorithm 5.4.

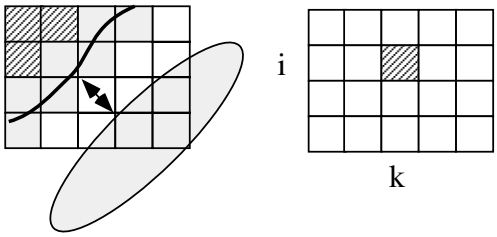


Figure 5.23 A typical grid map. White tiles represent free space, hatched tiles obstacles, and gray ones mixed spaces.

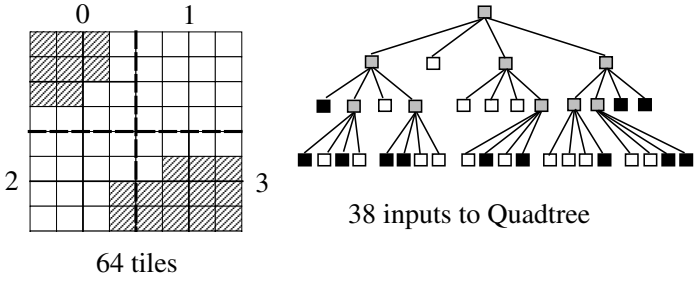


Figure 5.24 Quadtree representation of a grid map with 64 tiles.

Algorithm 5.4 Quadtree representation of a grid map

if $n = 0$ **then**
  it is a leaf node
else
  the map becomes the root node of a quad tree
end if
if the map is not uniformly colored **then**
  split the $2^n \times 2^n$ tiles into four maps of $2^{n-1} \times 2^{n-1}$ tiles each
  handle them the same way
else if a map is uniformly colored **then**
  it is a leaf node of the tree, denoting a block of $2^k \times 2^k$ tiles
end if

The concept may also be extended to three dimensions: splitting a 3D-space of $2^n \times 2^n \times 2^n$ voxels recursively into blocks of $2^k \times 2^k \times 2^k$. In figure 5.25 a 3D object is represented in a grid map.

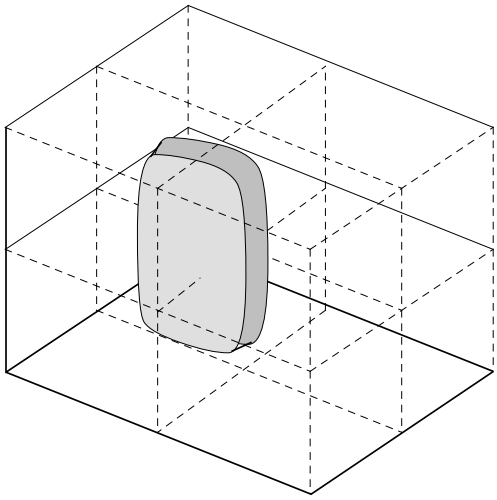


Figure 5.25 An object in a 3D grid

Probabilities Especially when using ultrasonic sensors with their rather unreliable measurements, it makes sense to include obstacle probability in the specification of the raster points. Performing more than one independent measurement, the probability of the presence of an obstacle at the specified location can be determined in a very simple manner. For each cell $C(i,j)$ the likelihood of occupancy can be computed as:

$$C(i,j) = \frac{\text{hits}(i,j)}{\text{hits}(i,j) + \text{miss}(i,j)} \tag{5.46}$$

where $\text{hits}(i,j)$ represents the number of scans that indicated the presence of an obstacle at this cell while $\text{miss}(i,j)$ stands for the times the cell appeared to be empty. Thus the computation effort affiliated with e. g. Bayesian Filters can be reduced to the simple counter approach above.

5.3 Sector maps

A sector map widens the rigid structure of a grid map and allows to partition space in a more flexible way (see [AKSB07]). A sector map is divided up into

one or more *sectors*, hence its name. The most common separations consist of uniform polar or rectangular sectors. Typically, a sector then contains the following information:

- polar sector: angle and distance to the closest obstacle in the area the sector covers
- rectangular sector: x- and y-coordinates of the closest obstacle in the area the sector covers

While this is information about the presence of obstacles (and used for collision avoidance), a sector map can also contain other information, such as the slope of the ground or overall terrain traversability.

In case of polar sector maps, space in front of an autonomous vehicle is divided into sectors of some degree; e. g. $\Delta\varphi = 5°$ from $-120°$ to $+120°$. Let the distance r_i to an obstacle be measured from the kinematic center at an angle φ_i and the distance to the border of the vehicle be k_i. Then the distance between vehicle and obstacle is $d_i = r_i - k_i$.

Figure 5.26 shows the sectors and figure 5.27 a typical sector map made up of 24 sectors of 5° each, spanning 120°.

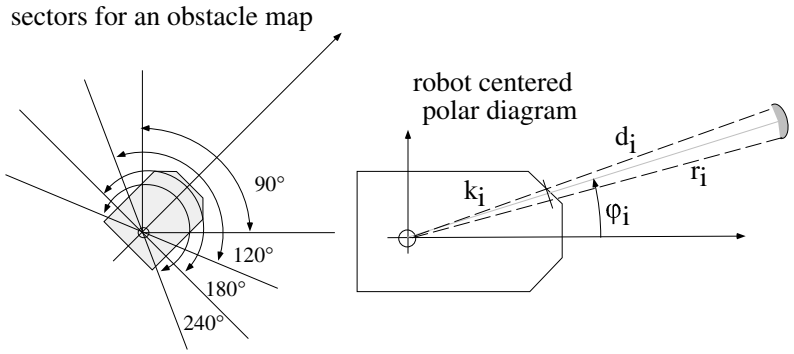


Figure 5.26 Typical angle range in front of the mobile robot, used for sector maps (left). Obstacle representation inside a sector (right).

A sector map can be transferred into a grid map. Figure 5.27 shows a typical sector map and its transfer to a robot-centered grid map. Its transformation into a world-centered grid map is presented in figure 5.28. In both pictures free space and obstacles are marked in the tiles; blank tiles mark unknown regions.

The transformation from a robot-centered sector map to world-centered grid map is shown in figure 5.29.

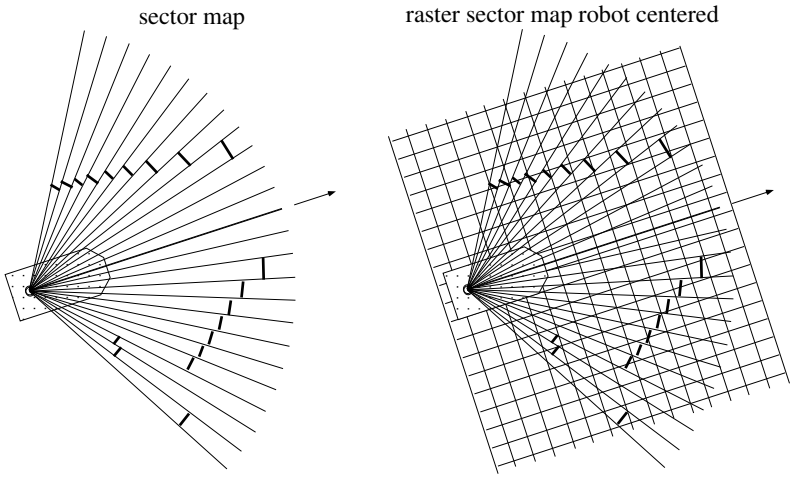


Figure 5.27 Transformation of a sector map to a grid map

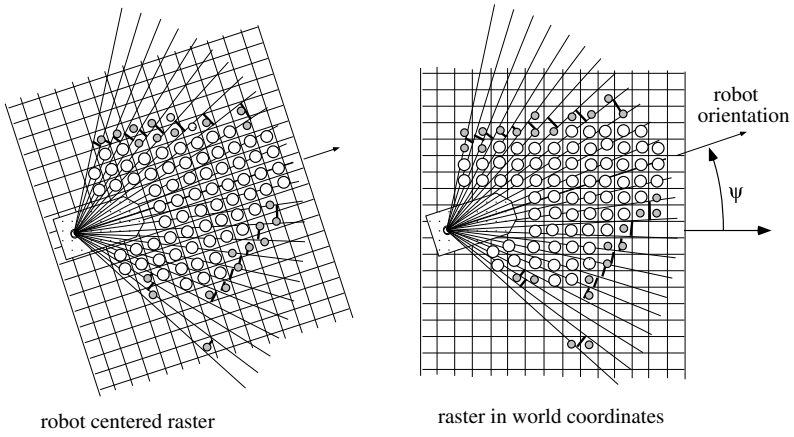


Figure 5.28 Robot-centered vs. world-centered grid map

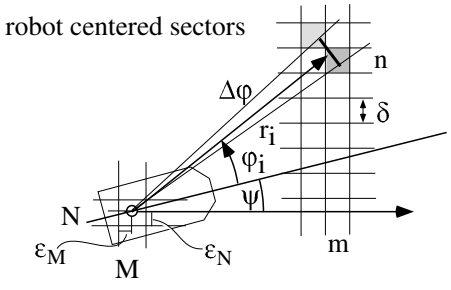


Figure 5.29 Transformation of a robot-centered sector map to a world-centered grid map

Let $\psi \neq 0$ be the orientation and $(x,y) = (M \cdot \delta + \varepsilon_M, N \cdot \delta + \varepsilon_N)$ be the position of the kinematic center of the vehicle in world coordinates. (M,N) is the index of the tile in which the kinematic center is located. ε_M and ε_N is the offset of the kinematic center in x- and y-direction due to the origin of the tile. δ is the edge length of the tiles. Suppose there is an obstacle at distance r_i in the sector i with an opening angle $\Delta\varphi$. Then all tiles with index (n,m), which fulfill one of the 3 pairs of inequations listed below have to be marked as obstacles and the space covered by the cone up r_i as free space.

$$(M+m)\cdot\delta - \varepsilon_M \leq r_i \cos(\varphi_i + \psi) \leq (M+m+1)\delta - \varepsilon_M \quad (5.47)$$
$$(N+n)\cdot\delta - \varepsilon_N \leq r_i \sin(\varphi_i + \psi) \leq (N+n+1)\delta - \varepsilon_N \quad (5.48)$$

$$(M+m)\cdot\delta - \varepsilon_M \leq r_i \cos(\varphi_i + \psi + \Delta\varphi/2) \leq (M+m+1)\delta - \varepsilon_M \quad (5.49)$$
$$(N+n)\cdot\delta - \varepsilon_N \leq r_i \sin(\varphi_i + \psi + \Delta\varphi/2) \leq (N+n+1)\delta - \varepsilon_N \quad (5.50)$$

$$(M+m)\cdot\delta - \varepsilon_M \leq r_i \cos(\varphi_i + \psi - \Delta\varphi/2) \leq (M+m+1)\delta - \varepsilon_M \quad (5.51)$$
$$(N+n)\cdot\delta - \varepsilon_N \leq r_i \sin(\varphi_i + \psi - \Delta\varphi/2) \leq (N+n+1)\delta - \varepsilon_N \quad (5.52)$$

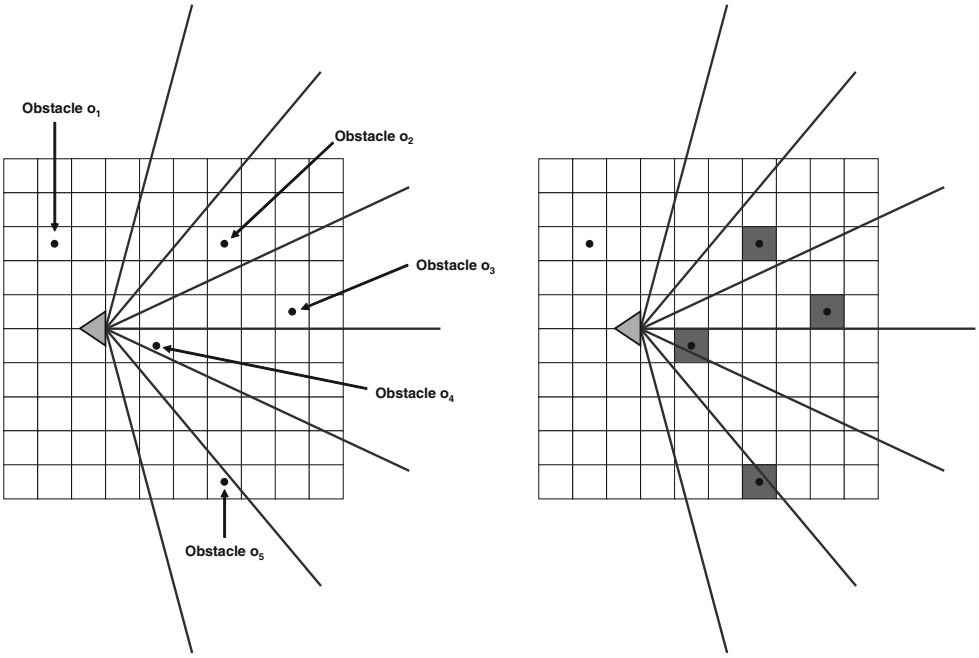
For a sector up to 3 tiles may be marked as obstacles in the world-centered grid map.

A sector map has an arbitrary position and orientation in terms of the robot coordinate system. It is important to note that although the content of a sector map is usually generated by the data of a specific sensor, a sector map does not have to be aligned with this sensor. Instead, it can be placed anywhere in the area around the robot as a so-called "virtual sensor".

On the autonomous mobile robot ARTOS [AKSB07] of the TU Kaiserslautern sector maps are not only used for collision avoidance, but also as data source for an occupancy grid map built by the mapping system. The data of a laser range finder and two chains of ultrasonic sensors is put into three polar sector maps with uniform sectors. The two-step algorithm described in the following is used to extract the information from the sector maps and alter the occupancy counters of the grid map.

The first step of the algorithm deals with the grid cells whose counters have to be increased and the second one deals with the grid cells whose counters have to be decreased. The algorithm will be explained with the following example. Figure 5.30(a) depicts four obstacles o_1–o_4, the grid of a grid map and a sensor together with the sector boundaries of a sector map belonging to it.

In the first step, all sectors are traversed and the occupancy counters of the grid cells in which a sector's obstacle lies are incremented. The sector in which o_1 lies is not marked as occupied as o_1 is not in any of the sectors. The resulting grid map is shown in figure 5.30(b).

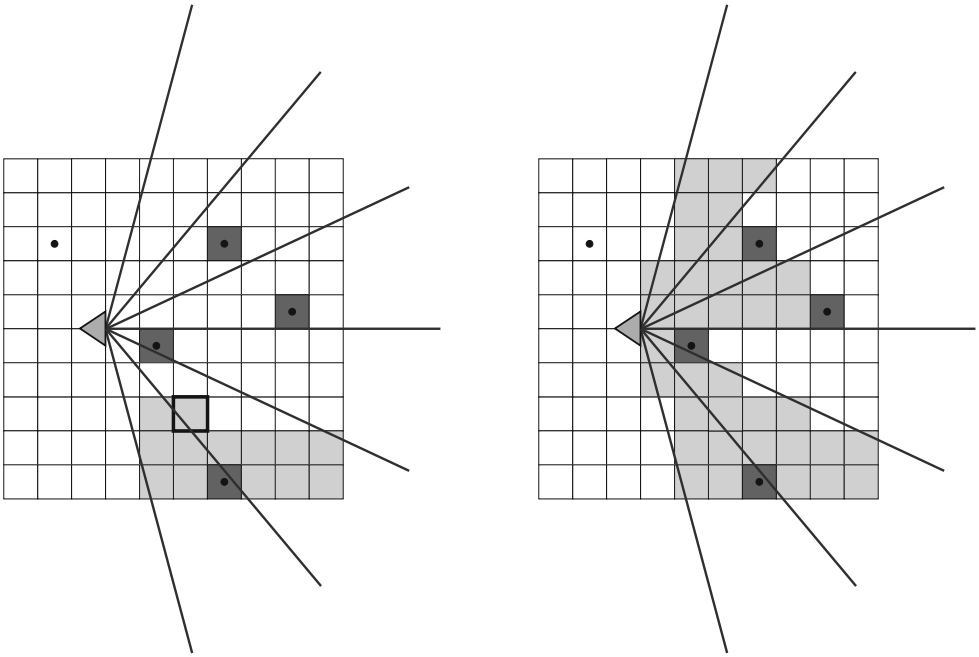


(a) Five obstacles lie in the area around the sensor. One of them, o_1, is currently not seen. The grid map is in its initial state.

(b) The cells in which visible obstacles lie have been marked as occupied.

Figure 5.30 First part of the creation of a grid map using a polar sector map as data source.

In the second step, the counters of the cells which are covered by a sector and are closer to the sensor than the obstacle in this sector are decremented. The problem is to determine the cells to which this applies. A comfortable way would be to have a list of these cells. But then the problem would be how to update this list when the robot's pose changes. So instead of calculating for each sector a list of the cells it covers, all cells in a user-specified rectangular area around the sensor are processed. For each of them the containing sector is determined. If the center of the grid cell is closer to the sensor than this sector's obstacle, its counter is decremented. Figure 5.31(a) shows an intermediate result of the map creation process, and figure 5.31(b) depicts the final grid map.



(a) The second step of the grid map creation algorithm is being executed. The cell with the bold frame has just been processed and marked as free.

(b) The second step of the grid map creation algorithm has been completed. All cells between the sensor and visible obstacles have been marked as free.

Figure 5.31 Second part of the creation of a grid map using a polar sector map as data source.

Note that in the second step, the counter decrementation is not limited to cells that are traversed by a sensor beam. Instead, the fact that the relevant obstacle in a sector is also the closest one is utilized to update the counters of more cells.

5.4 Topological maps

Compared to metrical maps, topological maps are more abstract descriptions of large-scale structures of the environment. Topological maps are typically represented as graphs in which navigation-relevant places are modeled as graph nodes and connections between places are indicated by graph edges. Often, some metrical information is also stored in a topological map, such as the coordinates of a place or the metrical length of the topological edge. Even

if topological maps also contain such information, processing topological maps (generation, path finding) need less computation than metrical maps. Typical examples of topological maps are:

- bus lines and bus stops in a town,
- a highway network of a country,
- a network of stations and railway lines for a subway or railway system,
- a grid of the high voltage transmission lines of a country,
- the sewage system of a town.

Typical questions to be answered with help of a topological map are:

- Where do I change buses between stations A and B?
- Can I also drive from A to B via C or via D?
- How many stops are there on the way from A to B?
- If one transmission line fails, are there lines to circumvent the failed one?
- Where is the entrance to a main sewage line?

These questions are difficult to answer using geometric maps only. Therefore, for the solution of many mapping problems the transformation of a geometric map into a topological one is necessary, as shown in figure 5.32 and figure 5.33. In figure 5.32 three rooms are presented which are connected to each other by a passageway. Connecting the corners with edges which are not crossing any objects is shown in figure 5.33. This decomposes the map into regions free from objects.

This graph can easily be generated by visibility graph algorithms. The emerging regions can be represented by a graph as shown in figure 5.33. The nodes in this graph represent regions of the metrical map. The edges between nodes denote a passageway between the regions. This graph can be transformed into a **decomposition tree**. Here, all nodes belonging to one room are mapped to subgraphs; interconnection graphs represent the passageways between rooms. The decomposition tree algorithm divides the whole graph into subgraphs, in which nodes exist with one connection to another subgraph.

A possible interpretation of the generated **decomposition tree** is shown in figure 5.34.

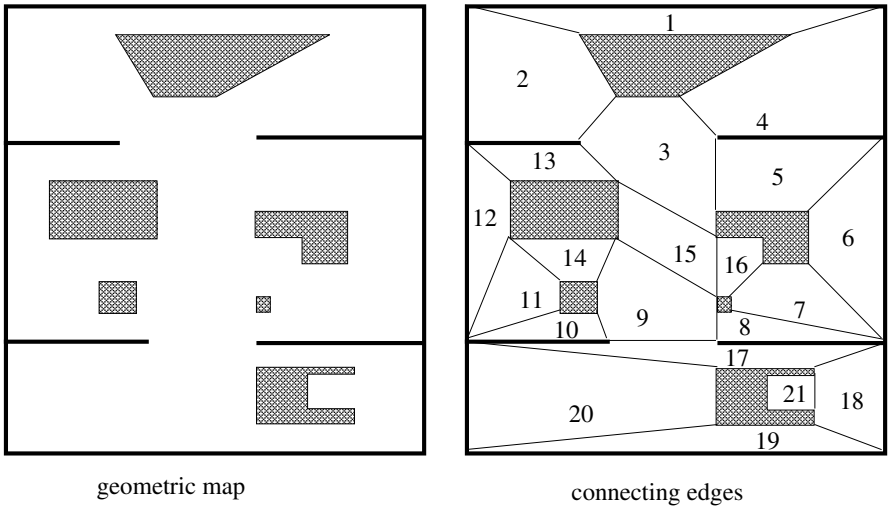


Figure 5.32 Interconnecting edges in a map of 3 rooms

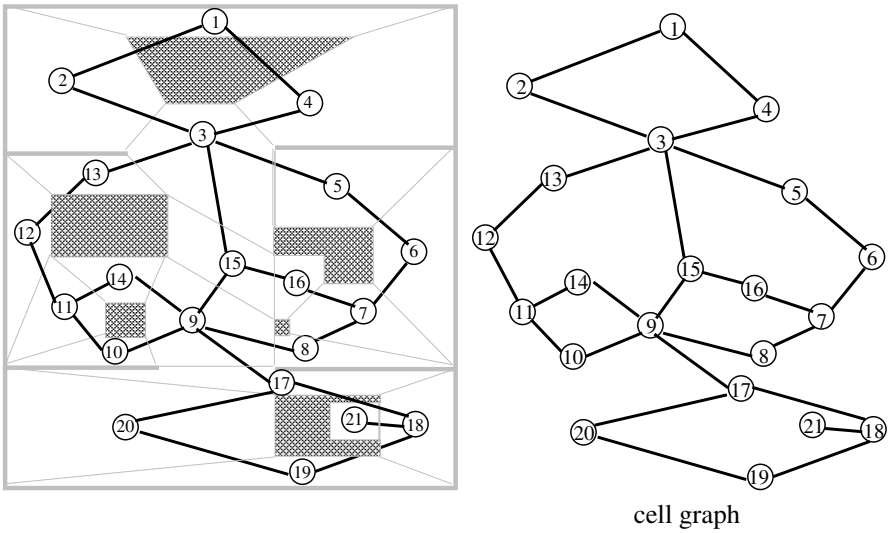


Figure 5.33 Interconnecting regions and the corresponding graph

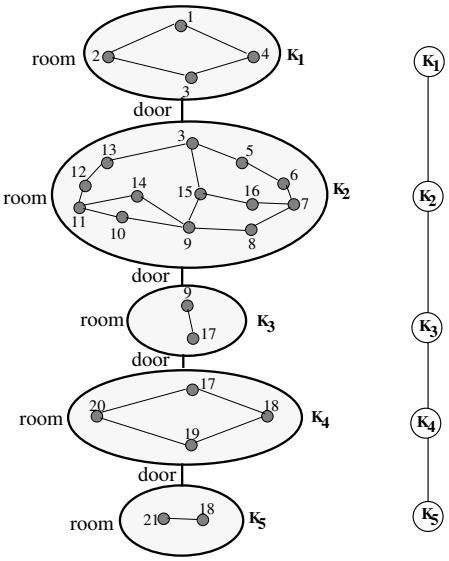


Figure 5.34 Decomposition tree

5.4.1 Growing neural gas net

Fritzke [Fri96] gave an algorithm to build a topological map of a scene from multiple measurements by different sensors.

Let the representation be a net with m nodes K_i made up from n-dimensional vectors $K_i = (b_{n-1,i}, \ldots, b_{0,i})$ with normalized components $b_{r,i} \in [0,1]$. The representation will be a net with m nodes K_i, $i = 1, \ldots, m$. In order to keep the number of nodes manageable, relevant nodes K_i will have to be calculated from many scene vectors $S = (s_{n-1}, \ldots, s_0)$. Each scene vector describes one set of normalized measured values $s_r \in [0,1]$. If the components of S are independent from each other – their covariance $c_{r,q} = \delta r,q$ – then the Euclidean distance between a node K_i and S is well defined:

$$\|K_i,S\| = \sqrt{\sum_{r=0}^{n-1} (b_{r,i} - s_r)^2} \tag{5.53}$$

Let a visiting counter z_m be attached to each node K_m and let D be a critical distance. Building up a growing neural gas net runs as shown in algorithm 5.5.

Having read in G scene vectors, then $G = \sum_{i=1}^{m} z_i$ and $m =$ number of nodes. There is still a difficulty to be solved: looking for the bmu in a large net given a scene vector S.

Algorithm 5.5 Growing Neural Gas Net

initialize: $m := 1$; take a first measurement $K_m = S$; $z_m := 0$
repeat
  read S
  for all $j = 1,\ldots,m$ **do**
    $\|K_j,S\|$
  end for
  for all $j \neq i$ **do**
    $\|K_i,S\| \leq \|K_j,S\|$ {– K_i is the node with minimum distance to S – }
  end for
  if $\|K_i,S\| > D$ **then**
    {a new node is inserted}
    $m := m + 1$
    $K_m := S$
    $z_m := 0$
  else
    {– K_i is the best matching unit – **bmu** – }
    if $z_i < Z$ **then**
      $z_i := z_i + 1$ {– the visiting counter is increased – }
      for all $r = 1,\ldots,n$ **do**
        $b_{r,i} := b_{r,i} + \varepsilon(s_r - b_{r,i})/z_i$ {– the components of K_i are shifted into the direction of S while the weight of K_i increased – }
      end for
    else if $z_i = Z$ **then**
      {– the region around K_i is under represented and S gets a new node – }
      $m := m + 1$; $z_i := Z/2$; $K_m := S$; $z_m := Z/2$
    end if
  end if
until FOREVER

The number of nodes to be checked should be kept manageable. The scene vectors as well as the nodes describe points in an n-dimensional unit cube. The direct approach as described in the algorithm to calculate $\|K_j,S\|$ for all $j = 1,\ldots,m$ and look for the minimum has an effort of $(2 \cdot m \cdot n)$. The idea is to preselect the nodes K_i, see figure 5.35.

- By multiplying the values of the components of the nodes by 2^p, the components of $K_i = (b_{(n-1),i},\ldots,b_{0,i})$ become integer numbers $b_{r,i} \in (0,\ldots,2^p - 1)$.

- Chop the interval $0,\ldots,2^p-1$ into 2^k parts t_q with $q=0,\ldots,2^k-1$. The interval t_q contains the numbers $q2^{p-k} \le b < (q+1)2^{p-k}$.
- The uppermost k bits of the values of b_r and $b_{r,i}$ form the indices q_r or $q_{r,i}$.
- Choose k such that $D = 2^{p-k}$.
- Form a list at index q in dimension r of all K_i with $(q-1)2^{p-k} \le b_{ri} < (q+2)2^{p-k}$ under their number i. The list is ordered with respect to i. The storage effort is $(n2^k)$ lists with indices j and the values $b_{r,j}$.
- Candidates for the bmu are only those K_j whose number is to be found at **all** indices q_r, belonging to b_r.
- If the search fails, then S is a new node; there is no node K_j at a distance less than D around S.
- Let the indices $j_1,j_2,\ldots,j_l$ be found in all boxes q_r then only the distances from S to $K_{j1},K_{j2},\ldots,K_{jl}$ have to be calculated to find the minimum, the bmu.

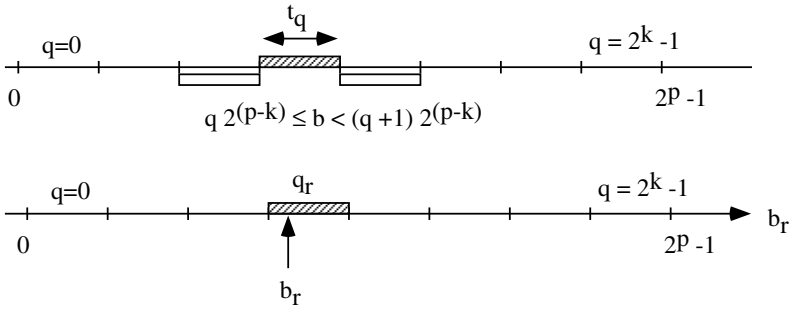


Figure 5.35 Splitting the realm of possible values

There are three parameters to be adapted to the scene at hand:

- The distance D, describing the influence of a node,
- the maximum value Z of a visiting counter z_j describing the number of matches before a split at the node K_j occurs,
- and the shift ε describing how much the current scene vector affects the bmu.

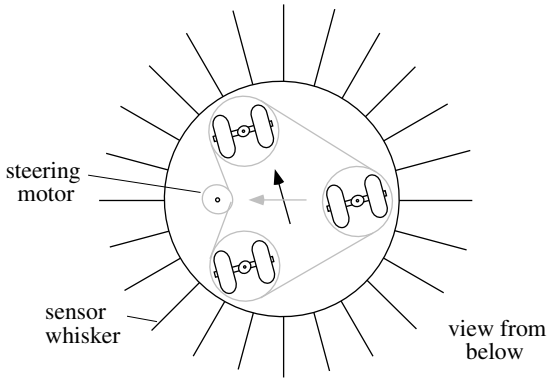


Figure 5.36 Sketch of ALICE

An example of building a topological map using a very simple vehicle, ALICE is described in [ZvP94]. Figure 5.36 shows the structure of the vehicle.

The robot has a circular shape with a diameter of 30 cm and is actuated by a synchro drive. ALICE is equipped with 24 photo sensors with an angle of beam spread of 15° each. The same number of touch sensors detect contact with walls as shown in figure 5.37.

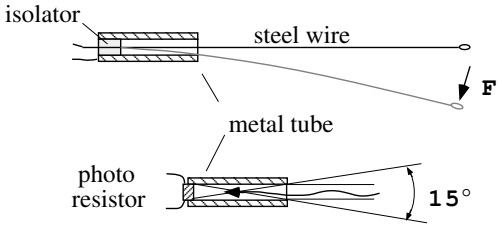


Figure 5.37 The touch sensor and the photo sensor of ALICE. The touch sensor is realized as a simple electrical switch. If the steel wire hits an obstacle the wire is bended and closes the contacts of the switch.

The measurements of 24 photo sensor $(I_0,\ldots,I_{23})$ are normalized to $\widetilde{I}_i = I_i/I_{\max} \in [0,1]$. The 24 touch sensor values $(T_1,\ldots,T_{24}) \in [0,1]$ are smoothed to cope with the rather large uncertainty of these sensors: $\widetilde{T}_i = (2T_i+T_{i-1}+T_{i+1})/4, i = i \mod 24$. Only these 48 measurements describe the environment around ALICE. For the calculation of position and orientation of the vehicle the movement of the chain of the synchro drive is detected by light barriers and the wheel velocity measured by optical encoders.

The ALICE scene vector has 48 components. They provide input for a neural gas net modified slightly bit for the task of building a representation of the environment ALICE finds itself in.

- A first modification is to use the position information to build a graph: the vehicle actually has been driven from one best matching unit to the next, so there a line is drawn. The node is annotated with the position.
- The line is annotated with a visiting counter. It is used to cope with wrong measurements: once in a while a measurement gives wrong values. The sensor situation becomes unique and becomes a new best matching unit (**bmu**). From the last bmu to this new one a line is drawn and a visiting counter counted up. All the other counters attached to edges from that last bmu are diminished. It is highly improbable that a false measurement will be repeated. So the line will never be driven again. If the value of a visiting counter drops below the limit, the line is taken from the graph. This eliminates wrong measurements after a while.
- In order to get a reliable representation of the environment, the inevitable drift in direction has to be corrected. To this end the light impression of the photo sensors is used to correct a drift in direction.

Figure 5.38 shows a typical sensor situation of ALICE. Aside from the sensor values, the position and an identifier for the bmu part of the sensor situation. Figure 5.39 shows a test environment for ALICE and figure 5.40 a neural net formed after a while. Despite the primitive sensors, the environment can be recognized in form of the free space for ALICE.

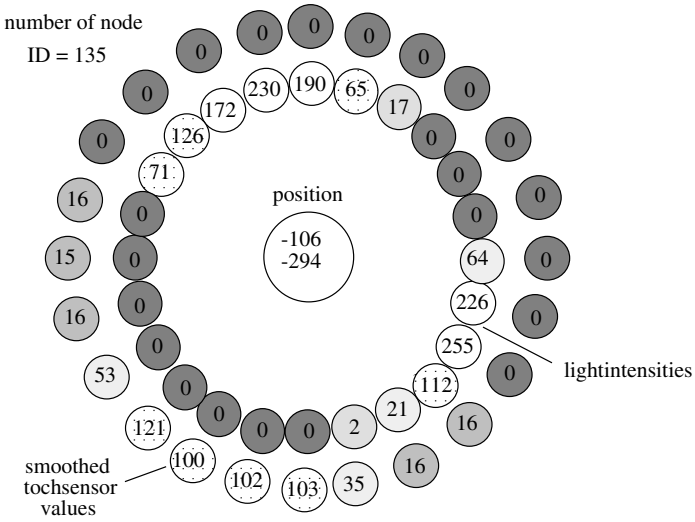


Figure 5.38 Example of sensor measurements (touch sensor values and light intensities) at an estimated position

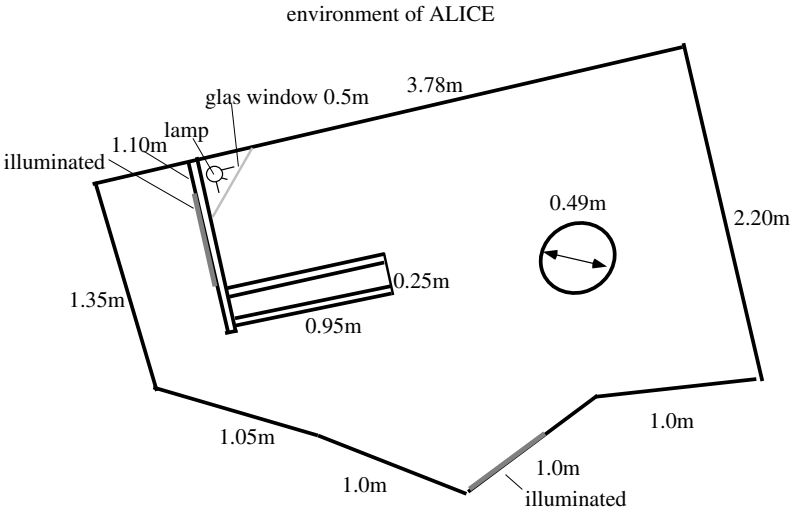


Figure 5.39 ALICE test environment

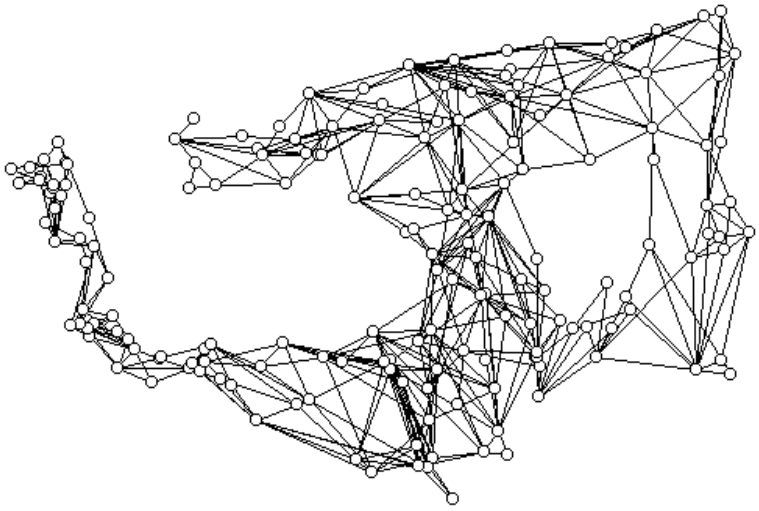


Figure 5.40 Topological graph of environment

5.5 Hybrid maps

As discussed before, metrical and topological mapping algorithms feature different characteristics (see figure 5.41). Hybrid approaches try to combine both types, allowing localization and map building with the high precision of metrical maps while retaining the compactness of topological maps.

	Topological Maps	Metrical Maps
Scale	Large-scale space	Small-scale space
Sensor inputs	Abstracts sensor inputs	Stores sensor inputs
Computational power	Low	High
Memory consumption	Low	High
Sensitive to noise	Less	More
Real-time mapping	Yes	Depends on computational power

Figure 5.41 Comparison of features of metrical and topological maps

Hybrid maps found in literature can be classified as two main types: abstraction-based and hierarchical hybrid maps. In abstraction-based hybrid maps, a metrical map of the environment is typically constructed as a basis, and an abstraction of the map is performed in order to create a compact topological representation. The benefit of this abstraction is more efficient planning of an approximate path to a given goal location than a detailed metrical map. However, the metrical map must often be kept for relocalization and obstacle avoidance purposes.

An example of a hybrid mapping strategy that derives a Voronoi-graph based topological map through abstraction of a long-lived, complete metrical map is presented in [Thr98]. Here, topological map building is initiated by thresholding an occupancy grid map built using Bayesian probability techniques. Then, the Voronoi diagram is built by selecting all free grid cells, with two nearest occupied grid cells (the *base points*) being equidistant. Cells on the Voronoi diagram are termed critical points if the base point distance is a local minimum. Lines between critical points and their base points divide metrical space into separate topological regions at locally narrow passages. From this regional decomposition, a topological graph can be generated. Figure 5.42 shows an example of the procedure.

With the help of this graph, fast path planning is possible without resorting to the detailed, underlying metrical map. During travel, however, the arrival at a topological node can only be detected by using localization techniques based on the detailed map, since the topological nodes are

only characterized by their spatial position in the metrical map's frame of reference.

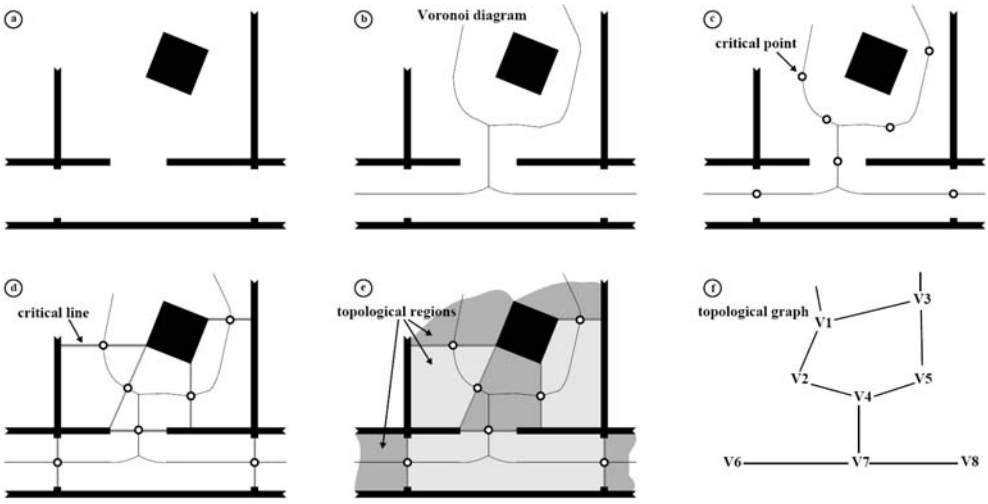


Figure 5.42 Steps in the topological abstraction scheme of [Thr98]. (a) Original thresholded occupancy map (b) Overlaid Voronoi diagram (c) critical points (d) critical lines (e) topological regions (f) resulting topological graph

Hierarchical hybrid maps on the other hand try to arrange the two map methodologies in a *hierarchical* fashion. This is accomplished by creating several local metrical maps with a limited scale and tying them together using a global topological map (figure 5.42). This approach uses the classical divide-and-conquer paradigm to address the scalability problems that are inherent in large metrical maps. Also, it prevents errors incurred during metrical mapping from spreading over the entire mapped area. However, one also has to pay a price for the segmented map structure, as information contained in partially overlapping local maps cannot be used to enforce global consistency. This can lead to increased uncertainty for each local map, especially if they are closely spaced.

In [TNS03], a hierarchical hybrid map of an indoor environment is presented which combines a global topological map with local metrical maps. On the global level, a topological map is constructed which contains intermediate *nodes* spanning the topological graph, leafs that signify metrical *places* and *corner lists* on the links between nodes (see figure 5.43). The corners can be extracted easily from the data of a 360° laser scanner and serve as landmarks for localization on the topological scale.

While the landmark-based localization strategy is used during robot travel along edges, the approach switches to a metrical method once the

robot has arrived at a leaf node marked 'place'. The local map stored in this leaf is a line-based metrical map. For each line segment a 'line feature' is generated that is described by the angle of the perpendicular to the line and its length. The local feature map can be used for localization by modeling all feature parameters and the current robot pose as the state vector of an extended Kalman filter. With the use of this local metrical representation, the robot can travel freely in the vicinity of the leaf node, performing navigational tasks with high metrical precision.

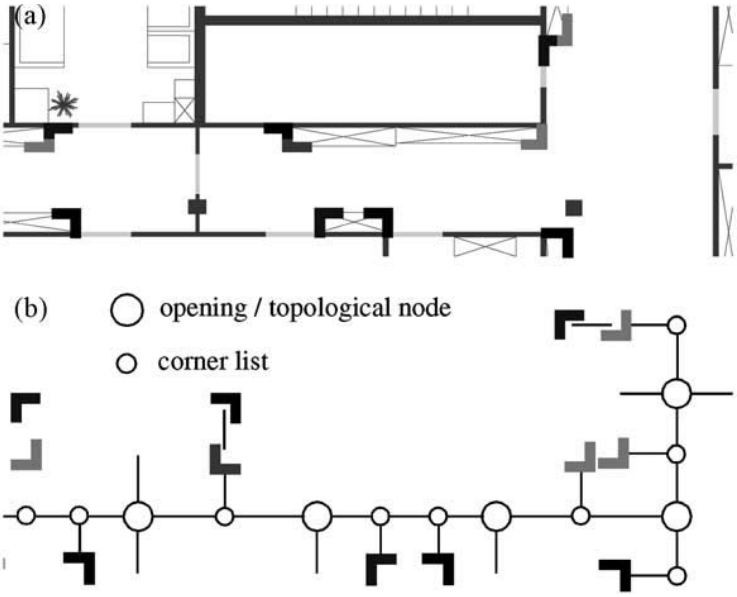


Figure 5.43 Hierarchical hybrid map of [TNS03]. (a) shows a portion of a hallway with extracted corner and opening features. (b) represents the resulting topological map with nodes and corner landmarks. Open arches either lead to metrical places or other hallways.