Inferring Position Knowledge from Location Predicates

Jörg Roth

Univ. of Applied Sciences Nuremberg
Kesslerplatz 12, 90489 Nuremberg, Germany
Joerg.Roth@FH-Nuernberg.de

Abstract. Many context- and location-aware applications request high accuracy and availability of positioning systems. In reality however, knowledge about the current position may be incomplete or inaccurate as a result of, e.g., limited coverage. Often, position data is thus merged from a set of systems, each contributing a piece of position knowledge. Traditional sensor fusion approaches such as Kalman or Particle filters have certain demands concerning the statistical distribution and relation between position and sensor output. Negated position statements ("I'm *not* at home"), cell-based information or external spatial data are difficult to incorporate into existing mechanisms. In this paper, we introduce a new approach to deal with different types of position data which typically appear in context- or location-aware application scenarios.

Keywords: Location inference, probability model, position data fusion.

1 Introduction

To detect context from the user's current location, position information ideally has a high precision and is constantly available. Many positioning systems, however, have a limited coverage and availability. E.g., GPS does not work indoors and often fails in city centres. Indoor positioning systems only cover some rooms or buildings. Systems with a higher coverage (e.g. based on mobile phone cells) are often inaccurate. In reality, we thus receive a number of incomplete pieces of position knowledge such as:

- I currently receive WLAN cell ABC;
- checking my IP address, I know, I'm *not* at home;
- one minute ago, I received GPS position XY, and I have a maximum speed of 5 km/h as pedestrian.

We call such statements *location predicates*. We often still have uncertainty about the actual position, but we can at least mark some positions as *probable*. A location-aware application could request the most probable positions(s) based on incomplete knowledge to detect the context.

One research field that traditionally deals with different position information is robotics [1, 22]. Based on odometers, ultrasonic distance measurement and cameras, a mobile robot computes the most probable location (often indoors). Established approaches to compute a position from these sensors are Kalman or Particle filters.

J. Hightower, B. Schiele, and T. Strang (Eds.): LoCA 2007, LNCS 4718, pp. 245–262, 2007.

Even though these approaches are widely used, they often cannot be applied to mobile user scenarios due to significant differences: First, potential positions cover the entire Earth's surface and not only rooms or buildings. Second, position data are more complex from the viewpoint of statistics; especially we have non-Gaussian distributed sensor values. Third, we have to access external spatial data such as road maps. These differences highly complicate the task of position inference.

In this paper, we introduce a new mechanism that deals with these issues. As a first idea, every piece of position information is modelled by a standardized data structure that reflects the corresponding knowledge. Second, a mechanism processes this information and constructs a structure that represents the position probability for every location. Third, most probable positions (local maxima or centroids) are derived from these result structures.

2 General Considerations and Related Work

Any piece of information about the position such as

- not to be at home,
- a GPS receiver measured position XY,
- to drive on a road or
- to reside anywhere in the GSM cell ABC

affect the *probability* to reside at a certain position. Actually, the current position is not a random variable in the traditional sense, as it is fix but unknown. We thus formulate the problem as follows: given a position; what is the probability to get the specific list of predicates? In the case of continuous random variables, the probability of any single discrete event is in fact 0. Thus, the probabilities of all positions are represented by a *probability density function* (*pdf*). Strongly related to a pdf *f*, there exists a *cumulative distribution function* (*cdf*) *F* with

$$P(X \le x,\, Y \le y) = F(x, y) = \int_{-\infty}^{x} \int_{-\infty}^{y} f(a,b)dbda \tag{1}$$

for a position (x, y) and random variables X, Y. In principle, we could express the information about the location by a cdf. But as pdfs in contrast to cdfs have small values (or even zero) for improbable positions, approximations that precisely reflect the knowledge require far less memory. Most approaches to model position information by probabilities thus use pdfs.

We first consider a fix point in time and further assume that we collect position information represented by a set of independent predicates z_i. According to the equation of conditional probabilities (Bayes rule) we get

$$f(x \mid z_1..z_n) = \frac{\prod f(z_i \mid x) f(x)}{\prod f(z_i)} \tag{2}$$

which describes the position density according to the new information. As $f(z_i)$ does not depend on x, we can consider the denominator as constant. Thus, we do not actually compute $f(z_i)$ and instead normalize the numerator to fulfil

$$\int_{-\infty}^{+\infty} f(x \mid z_1..z_n) = 1 \tag{3}$$

Here, $f(z_i \mid x)$ describes the predicate's general character, e.g., the error probability of a GPS sensor. This distribution is *predefined* and does not change over time, thus can be precomputed for each predicate.

If we had zero position knowledge (uninformative prior), but only rely on the given predicates, the equation simplifies to

$$f(x \mid z_1..z_n) = c \cdot \prod f(z_i \mid x) \text{ where } c = \frac{1}{\int \prod f(z_i \mid x)} \tag{4}$$

To consider multiple predicates at a single point in time, we thus mainly need to multiply densities.

To model motion over time, we have to consider a density that represents the position knowledge for position p at a time t_1, say $f(t_1, p)$, and a second density that represents *relative* movement by p_Δ, say $g(t_1, t_2, p_\Delta)$. The resulting density at time t_2 can be computed using the convolution equation

$$f(t_2, p) = \int f(t_1, p - p_\Delta) g(t_1, t_2, p_\Delta) dp_\Delta \tag{5}$$

Note that the integral is already one, thus no normalization is required.

Equations (4) and (5) form the basic toolset for any probability computation of position data. To compute densities at runtime, we have to approximate densities or assume simplifications. There exist two basic approaches:

- We assume only Gaussian densities and a linear dependency between states. As the two basic equations then significantly can be simplified, we get closed formulas. The Kalman filter is based on this idea.
- We approximate complex densities with the help of simple densities. The most popular example, the Particle filter, approximates any density by a sum of so-called *particles* which actually are Dirac densities. Dirac densities have an infinite value at the given point, 0 elsewhere and an integral of 1.

At this point, we briefly present these approaches.

2.1 Kalman Filters

The Kalman filter [13] is considered as one of the most important mathematical formalisms that deal with positioning. Detailed descriptions can be found in [9, 20]. The Kalman filter assumes a state vector x with arbitrary dimensions. For our scenarios, the state contains typical spatial state information, i.e. the position, but also orientation, speed and acceleration [2, 7, 10]. The state is unknown, but Gaussian distributed measurements y indirectly reflect information about the state. Further, two states at different points in time are linearly related, expressed by a matrix.

A resulting probability density for x is Gaussian distributed expressed by a mean (the most probably state) and an error, expressed by an error covariance matrix. A computation step contains a *time update phase* based on equation (5) and a *measurement update phase* based on equation (4). For the assumptions made for

Kalman filters, the equations can be simplified to a few matrix multiplications and one matrix inversion. These computations can be performed efficiently, even on small computers or embedded systems.

2.2 Particle Filters

Particle filters [5, 8, 11] use a set of particles; each presents a specific potential state. A particle contains a state vector p_i and a weight w_i which reflects the probability density for this state. A probability density f can be approximated by

$$f(p) \approx \frac{1}{N} \sum_{i=1}^{N} w_i \delta(p - p_i) \tag{6}$$

where δ is the Dirac delta density. In principle, a particle can have multiple dimensions, but in contrast to Kalman filters, too many dimensions dilute to result. Typical state vectors only have three dimensions (e.g. 2D position and orientation).

The so-called *Motion Model* that follows equation (5) moves all particles according to the relative movement density. This indirectly computes a convolution. The *Perceptual Model* that follows equation (4) assigns new weights according to multiple measurements at a point in time. This indirectly computes multiplying densities.

Particle filters support a huge variety of densities. Increasing numbers of particles improve the precision, but also increase the required memory and processing time. Particle filters also have to face the *degeneracy problem* [4] where all but one particle have a weight near 0. Approaches such as *Sequential Importance Sampling* (*SIS*) [6] and *Sampling Importance Resampling* (*SIR*) [19] counteract this problem applying a resampling step to particles.

2.3 Further Representations

Further representations of position knowledge are *grids*, *points* and *areas*. The *Position Probability Grid* [3] stores the density values for equidistant positions. Grids require a huge amount of memory space and the potential positions are limited by the initial grid border. Thus, this approach is inappropriate for our intended scenarios.

The *Area model* [15, 18] could be viewed as a simple statistical representation where the area border separates two regions with a uniform distributed position probability – inside the area the integral of probabilities is 1, outside it is 0. The computations based on this idea are quite simple (using the geometric intersection), but this simplification significantly dilutes the position knowledge, especially for Gaussian distributed sensor information.

The *Point model* is the simplest approach to model location knowledge. For any sensor input it only stores the most probably corresponding point in space. This approach is only reasonable, if all positioning systems have a high accuracy and measured values are distributed with a certain mean (i.e. this prohibits COO). Even though it is very unlikely that the measured and the true location are identical, this approach sometimes is useful: multiple pieces of position information can be processed using a weighted average. To promote more accurate sensors, the reciprocal

value of the measurement variance can be used as a weight. For our intended scenarios, this approach does not provide sufficient expressiveness.

2.4 Discussion

A system that infers knowledge about the positions in order to detect context information or to support location-based services has to meet the following requirements [18]:

- As many approaches to determine the current position based on the *cell of origin* (*COO*), such information has to be considered. Beyond a certain distance to an access point (especially outdoors), signal strengths do not significantly extend knowledge about the position [21], thus we often use the entire cell as set of potential positions.
- A reasonable approach must be able to consider multiple *alternative* potential positions. E.g., consider a user is at the crossroads to two streets going in nearly the same direction. Further assume that the position data is not precise enough to detect the choice. For a certain time, we thus have to consider both paths for potential positions until further information determines the choice.
- *Negated* information should be modelled. E.g., the current IP address may indicate that an end-user device does not reside a home. *Not* to reside somewhere means to reside nearly anywhere in the world and a nearly infinite space of potential positions has to be considered.
- *External spatial data* should be integrated. A huge amount of spatial information (e.g. roadmaps, information about places) may only be accessible over network. An appropriate algorithm must be able to download additional spatial data at runtime. This especially means to limit the search space for data lookup as not an entire spatial database can be downloaded.

We now check how existing approaches meet these requirements.

If position data meet assumptions described in section 2.1, the Kalman filter extracts the maximum information about the position. Unfortunately, in some typical scenarios these presumptions are not valid:

- COO measurements do not provide a Gaussian density with a certain mean;
- negated information cannot be modelled at all;
- alternative potential paths cannot be modelled as the system always assumes a single probable position;

Particle filters relax some of the assumptions. Especially, they can follow alternative paths as some particles model one alternative and further particles a second one. However, if the number of parallel alternatives increases, the average number of particles decreases and thus the overall precision. Note that at least one particle has to be selected that follows the actual (but unknown) path of real positions. Thus, an appropriate number of particles has to be chosen *beforehand* to anticipate potential alternatives.

With a sufficient (usually high) number of particles, COO measurements can in principle be modelled. The negation, however, is difficult to express if we do not limit the potential space. Finally, both types of filters have their problems with external data as they expect to access all spatial information locally.

Surprisingly, from the approaches described above, the Area model meets a high number of requirements:

- we can model COO measurements using the cell border as modelled area;
- we approximate Gaussian distributions by, e.g., the range of 95% measured positions (the so-called *2dRMS* area for GPS [14]);
- we can describe negations using borders that mark the outer area;
- we can easily load additional data using the area as spatial index.

The main drawback of the Area model is that after a few steps, the precision decreases dramatically and important information is destroyed. We thus looked for a new approach that on the one hand combines the benefits of the Area approach and on the other hand provides sufficient precision.

3 The MAP^3 Approach

MAP^3 (*Multi-Area Probability-based Positioning by Predicates*) introduces a new approach that deals with problems of comparable approaches described above. The main idea:

- Any piece of information about the location at any point in time is mapped to a *location predicate*. As predicates form a kind of universal interface to any positioning system, they easily can be integrated into existing location driver structures such as the Nimbus VPS [17, 18] or the Location Stack [12].
- A predicate is mapped either to a probability *density representation* (for a predicate that describes a location of a single point in time) or a *convolution operation* (for a predicate that describes movement between two points in time).
- If all predicates are processed, we get a set of densities – one for each considered time stamp. The application now can select a specific point in time.
- Depending on the application, the most probable location (the *centroid*) or a set of local maximum values is computed.

Fig. 1 presents the corresponding data flow. Important: we execute density operations (e.g. multiplication, convolution) with the help of geometric operations widely known in the area of spatial modelling. For this, so-called *simple features* [16] are used for which efficient software libraries are available. From the variety of geometric objects (e.g. points, line strings) we only require the *multipolygon with holes* (*mph*) that represents the most common approximating two-dimensional structure. An mph contains a number of polygons representing the surface. Each of it in turn contains a number of polygons that represent the holes in the surface.

We assume that spatial information is represented in two dimensions. Table 1 presents a selection of considered predicates. We can easily extend this list in the future.

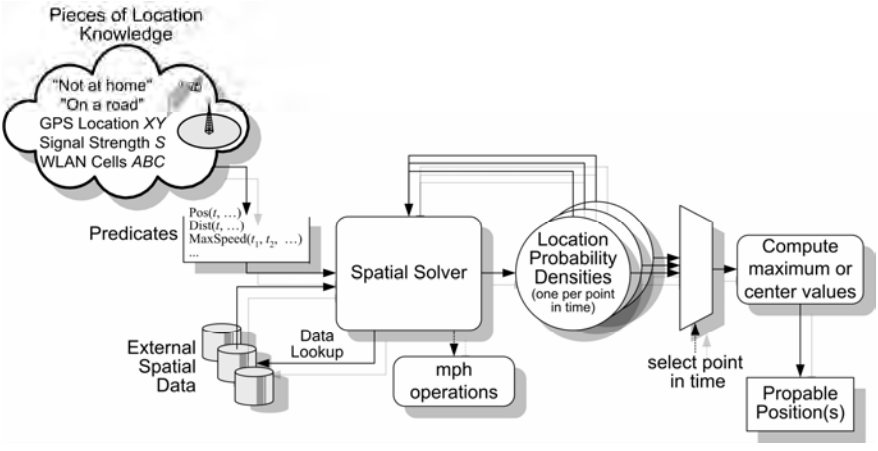

Fig. 1. Data flow in MAP[3]

Table 1. List of predicates

Predicate	Meaning	Example
Pos(t, x, y, σ_x, σ_y, ρ)	Gaussian distributed 2D-position	GPS
Dist(t, p_x, p_y, d, σ)	Gaussian distributed distance d to a fix point (p_x, p_y)	Runtime or signal strength measurement to a base station
Nearer(t, p_x, p_y, d)	The distance to a fix point (p_x, p_y) is below d	Circular cell
InPoly(t, *poly*)	The location is inside a polygon.	Map matching – car must reside on roads
Dir(t, p_x, p_y, α_1, α_2, m)	From the viewpoint of (p_x, p_y) the position has a direction inside [α_1, α_2], maximum distance m	Segmented antenna
MaxSpeed(t_1, t_2, v)	Maximum speed in this time interval	Pedestrian with a maximum walking speed

Some remarks:

- σ_x, σ_y, σ are standard deviations of the respective Gaussian distribution, ρ the correlation coefficient.
- t denotes the predicate's time (i.e. of the underlying measurement), t_1, t_2 describes a time interval.
- *MaxSpeed* and *InPoly* can be defined without any point in time. In this case, they are considered as always valid.
- Some predicates (details see section 3.2) can be modified using the *Not* modifier, which means that the opposite fact is true. E.g., *Not*(*Nearer*(t, p_x, p_y, d)) means, the distance to the fix position is equal or more than d.

Fig. 2 illustrates the respective densities. As *MaxSpeed* leads to a special case relating two points in time, it is not shown here (see later). According to the considerations in section 2 we need at least the following operations on densities: computing the centroid and maximum values; multiplying two densities (see equation (4)) and convolving two densities (see equation (5)).

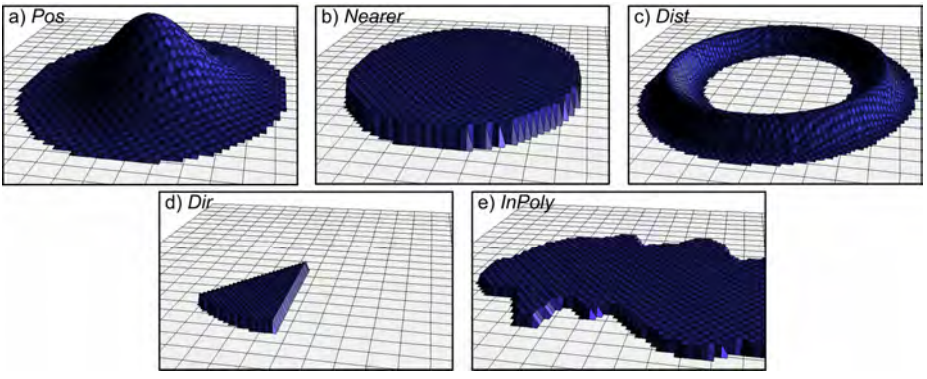

Fig. 2. Predicate densities

A density representation should allow the efficient execution of these operations. In addition, the input densities and resulting densities for (4) and (5) should precisely be presented. The MAP^3 approach represents a density f with the help of *areas* as follows

$$f(p) \approx \hat{f}(p) = \sum_{i=1}^{n} w_i \Lambda(p, mph_i) \tag{7}$$

In this equation

- p describes a point in space (x, y),
- n denotes the number of areas that approximate the density,
- w_i denotes a constant weight of an area,
- mph_i denotes the geometric description of an area and
- Λ the function $\Lambda(p, mph) = \begin{cases} 1 & \text{if } p \in mph \\ 0 & \text{otherwise} \end{cases}$

  (we can consider Λ as the characteristic function of *mph*).

To be a density, $\hat{f}$ has to cover a volume of one, i.e. $\sum_{i=1}^{n} w_i \cdot |mph_i| = 1$. Here $|mph_i|$ denotes the surface area of the multipolygon, i.e. $|mph_i| := \int \Lambda(p, mph_i) dp$.

Two different variations fulfil equation (7):

- Variation 1: $mph_{i-1} \subset mph_i$ for every i>1, i.e. we have an ordered list of areas by their size and each area is fully embedded into the next area.
- Variation 2: $mph_i \cap mph_j = \varnothing$ for every $i \neq j$. We have an ordered list of areas by their weights (from low to high).

The two variations and their influence on the weights are illustrated in fig. 3. Both variations have their pros and cons that we discuss from the viewpoint of variation 2. The advantages of variation 2 are:

- Each point p is enclosed by zero or one polygon, thus computing $\hat{f}(p)$ is very easy. That is why the computation of maximum values also is simple.
- Multiplying densities, two areas that do not overlap do not contribute to the result. Thus, the multiplication has an efficient realization.

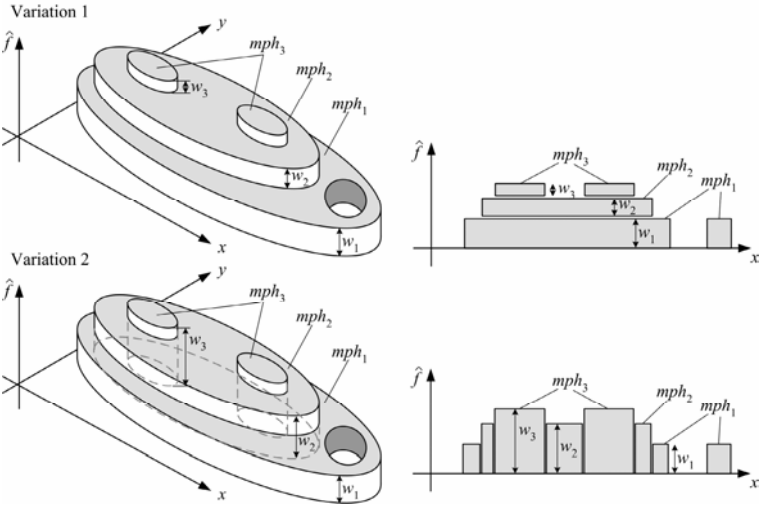

Fig. 3. Variations to present densities with the help of $\hat{f}$

Drawbacks of variation 2:

- Convolving two densities is more complex (see section 3.4).
- Comparing a variation 1 representation for the same density, variation 2 has to encode additional holes. Thus, the memory space to store the mphs nearly doubles compared to variation 1.

As these arguments nearly counterbalance, we introduce a further property into the discussion: the resampling operation. Typical multiplication and convolution operations increase the number of areas, thus, we need a resampling step that joins similar areas to a single area. We know an analogous operation for Particle filters.

We conducted a number of experiments with different resampling operations, coded for the two variations. The only suitable resample operation that is both efficient and does not remove too much information is based on variation 2. Its idea is to unite areas that have similar weights:

```
resample(density d):
as long as d.n>maxArea
   look up area k with |d.w_k − d.w_{k+areaDecr−1}| minimal
   create new area with mph_new ← ⋃_{i=k}^{k+areaDecr−1} d.mph_i
                        and w_new ← ( Σ_{i=k}^{k+areaDecr−1} w_i · |d.mph_i| ) / |mph_new|
   replace area k by the new area
   remove areas k+1 ... k+areaDecr-1 from d
```

In this algorithm $d.n$ denotes number of areas in the density, $d.mph_i$ the multipolygon for area i, and $d.w_i$ the weight of area i. This algorithm further requires two constants: *maxArea* – the maximum numbers of areas allowed in a density and *areaDecr* – the number of areas that are united in a single resampling step.

Based on the considerations above, especially an effective resampling mechanism, we chose variation 2 for the density representation.

3.1 Multiple Predicates at a Single Point in Time

According to equation (4), we have to multiply all densities representing the same time. After normalizing, we then get the result density. For two approximated densities $\hat{f}_1$, $\hat{f}_2$ we use the equation

$$\begin{aligned}\hat{f}_1 \cdot \hat{f}_2 &= \hat{c} \cdot \left(\sum_{i=1}^{n_1} w_{1i} \Lambda(mph_{1i}) \right) \cdot \left(\sum_{j=1}^{n_2} w_{2j} \Lambda(mph_{2j}) \right) \\ &= \hat{c} \cdot \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} w_{1i} w_{2j} \Lambda(mph_{1i} \cap mph_{2j})\end{aligned} \tag{8}$$

where $mph_{1i} \cap mph_{2j}$ is the geometric intersection of two areas and $\hat{c}$ the normalization factor.

Some remarks on efficiency: First, only those mph_{1i}, mph_{2j} that overlap contribute to the result, thus an efficient algorithm first tests this, before the actual intersection is computed. As the overlapping test knows efficient implementations (e.g. using bounding boxes), this approach is reasonable.

Second, we have predicates that only contain a single area (e.g. *Nearer*). Such predicates can efficiently be multiplied by another density as the overall number of areas does not increase.

Third, even though we get $O(n_1 \cdot n_2)$ resulting areas, the actual number of overlapping mphs is far less than $n_1 \cdot n_2$. Note that low numbers of areas, e.g. 10, lead to sufficiently precise results, thus typically 20-50 areas are included in the result. But as the result may be input for further multiplications, a resampling step is required.

The multiply algorithm can be sketched as follows:

```
multiply(density d1, density d2):
result ← ∅
for i=1 to d1.n do
   for j=1 to d2.n
      if  d1.mph_i ∩ d2.mph_j ≠ ∅
          create new area with  mph_new ← d1.mph_i ∩ d2.mph_j
                            and  w_new ← d1.w_i · d2.w_j
          add new area to result
normalize result
resample result
```

To normalize the result, we sum up the surface areas multiplied by their weights, i.e. compute $I \leftarrow \sum w_i |mph_i|$. We then adapt all weights $w_i \leftarrow w_i / I$.

3.2 Modelling Negations

The *Not* modifier negates a predicate. Usually, this means to specify a very large area of possible positions by a single density. E.g., *Not*(*Nearer*(*t*, p_x, p_y, *d*)) specifies all positions outside a given circle. We can consider this area as infinite, even though in reality, the respective area is limited by the Earth's surface. This means that the actual probability density at any given point is in fact 0, i.e. meaningless.

More formally: if F is the cdf and f the pdf of a predicate, the negation can be expressed by

$$1 - P(X \le x, Y \le y) = 1 - F(x, y) = 1 - \int_{-\infty}^{x}\int_{-\infty}^{y} f(a,b)dbda \tag{9}$$

To express the negated predicate by a density, we need a density $\bar{f}$ that fulfils

$$1 - \int_{-\infty}^{x}\int_{-\infty}^{y} f(a,b)dbda = \int_{-\infty}^{x}\int_{-\infty}^{y} \bar{f}(a,b)dbda \text{ and } \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \bar{f}(a,b)dbda = 1 \tag{10}$$

Unfortunately, it is not possible to provide a close description for $\bar{f}$ or $\hat{\bar{f}}$. A possible solution to avoid zero densities would be to presume a limited area of potential positions. To be useful, this maximum area has to be small enough to avoid densities near to 0. The important drawback of this solution is the lack of *a priori* knowledge about potential positions. This is an important difference to the robotic (indoor) scenario.

To achieve a solution for negated predicates, we propose the following approach:

- Only predicates that have a unique density inside a finite area and a zero density outside can be negated. Such predicates are *Nearer*, *InPoly*, and *Dir*. Only such densities are reasonable densities for negations.
- The negation can only be applied inside a multiplication of two densities, where the second density has to be non-negated. This especially means, at least one non-negated predicate is required for a certain point in time.

To give a realistic example that conflicts with the second assumption: If we had three predicates stating "not at home", "not at work" and "not on any road", we still had a virtually infinite space to consider. Only with an additional *positive* predicate (e.g. "inside GSM cell XY"), we can construct an area of probable positions.

According to these considerations, we now can compute negations: let f_1 be a density and f_2 a density that should be negated. Then

$$f_{result} = c \cdot f_1 \cdot \bar{f}_2 \text{ where } \bar{f}_2(p) = \begin{cases} 1 & f_2(p) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Again, c is the normalizing factor. Note that $\bar{f}_2$ is not a density, as it does not produce an integral of one. Thus, we explicitly mark a density as negated and store the original non-negated density.

To actually perform this operation, we simply subtract the mph of f_2 from all areas of f_1 and normalize the result. This means, the multiply operation with a negated density knows an efficient geometric realization. We present the algorithm together with an extension to model uncertainty in the next section.

3.3 Modelling Uncertainty

In contrast to positive predicates, negative statements about the position can be uncertain. If I got sensor information stating a certain position (e.g. a *Pos* predicate), the only uncertainty can be a measurement error that is already modelled by the Gaussian distribution. In contrast, negative predicates may be uncertain in a more general meaning. If, e.g., I do not receive my home WLAN, I may be outside my home area, but also, with a small probability, I *am* at home and my WLAN router is switched off. To model these characteristics, we can append a general probability *pr* to the *Not* modifier. E.g., *Not*(*Nearer*(t, p_x, p_y, d), 0.9) means:

- with a probability of 90%, the position is outside the specified circle;
- with a probability of 10%, the position may be anywhere (i.e. inside *or* outside the circle).

We could use this predicate to, e.g., model a circular WLAN cell, where the access point's uptime is 90% on average.

For pr=1 we get the definite negative statement as introduced above. We can modify equation (11) as follows

$$f_{result} = c \cdot (pr \cdot f_1 \cdot \bar{f}_2 + (1 - pr) \cdot f_1) \tag{12}$$

$$f_{result}(p) = c \cdot \begin{cases} pr \cdot f_1(p) + (1 - pr) \cdot f_1(p) = f_1(p) & \text{if } \bar{f}_2(p) = 1 \\ (1 - pr) \cdot f_1(p) & \text{if } \bar{f}_2(p) = 0 \end{cases} \tag{13}$$

Based on these equations, we can easily derive an algorithm that is fully built on geometric mph operations:

```
multiplyNegation(density d1, density d2, pr):
result ← ∅
for i=1 to d1.n do
   create new area with mph_new ← d1.mph_i \ d2.mph_1
      and w_new ← d1.w_i                    // case f̄2(p) = 1
   if not empty add new area to result
   create new area with mph_new ← d1.mph_i ∩ d2.mph_1
      and w_new ← d1.w_i · (1 − pr)         // case f̄2(p) = 0
   if not empty add new area to result
normalize result
```

For every area of the first density up to two areas are created: the first area represents the case $\bar{f}_2(p) = 1$ of equation (13), the second area represents the second case.

3.4 Modelling Motion over Time

Until now, we only considered multiple predicates at a single point in time. But what, if we had multiple predicates at different times. This is the usual case for position measurements: we do not only get current sensor input, but can consider all recent sensor information to improve the current estimation. For this, we have to model spatial movement between two points in time.

Kalman and Particle filters often assume precise movement sensors such as odometers. In our intended scenarios, however, such sensors are often not available. E.g., for a pedestrian it is not possible to *explicitly* measure the direction and distance she or he walked in the last 10 seconds. Usually, the only mechanism to detect movement is to build vectors between last measured absolute positions. But these positions are already considered by the mechanisms presented before, thus they would not increase the overall position knowledge.

If no relative movement sensors are available, we only can define a maximum speed, derived from knowledge about the movement context (e.g., to be a pedestrian, or to drive a car). A maximum movement distance for a certain time is defined by a circular density with centre at the zero point, with a unique density inside the circle and 0 outside, i.e.

$$c(p,r) = \begin{cases} 1/\left(\pi r^2\right) & \text{if } x^2 + y^2 \le r^2 \text{ for } p = (x,y) \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

In the following we describe how to convolve an arbitrary density with such a density c. The convolution (equation (5)) then simplifies as follows:

$$\begin{aligned} \int \hat{f}(t_1, p - p_\Delta) \cdot c(p_\Delta, r) dp_\Delta &= \int \left(\sum_{i=1}^{n} w_i \Lambda(p - p_\Delta, mph_i) \right) \cdot c(p_\Delta, r) dp_\Delta \\ &= \sum_{i=1}^{n} w_i \left(\int \Lambda(p - p_\Delta, mph_i) \cdot c(p_\Delta, r) dp_\Delta \right) \end{aligned} \tag{15}$$

This means, we have to approximate the inner integral and create a sum. As the convolution operation is very complex, we can due to space limitations of this paper only provide the idea here.

We have to consider two cases as illustrated in fig. 4. If the area of an mph is large compared to the circle, the area that covers all points with a non-zero integral can be computed using the so-called *buffer* operation [16]. The buffer contains all points that do not exceed a certain distance to an mph. The buffer operation is available in typical geometric software libraries and can efficiently be executed.

Fig. 4 (upper right) shows the buffers for three integrals: the maximum integral (inner buffer), 50% if the maximum (second buffer) and an integral of 0 (outer buffer). With a linear approximation (that produces a maximum error of 5.8%) we can easily compute any degree between 0 and the maximum integral.

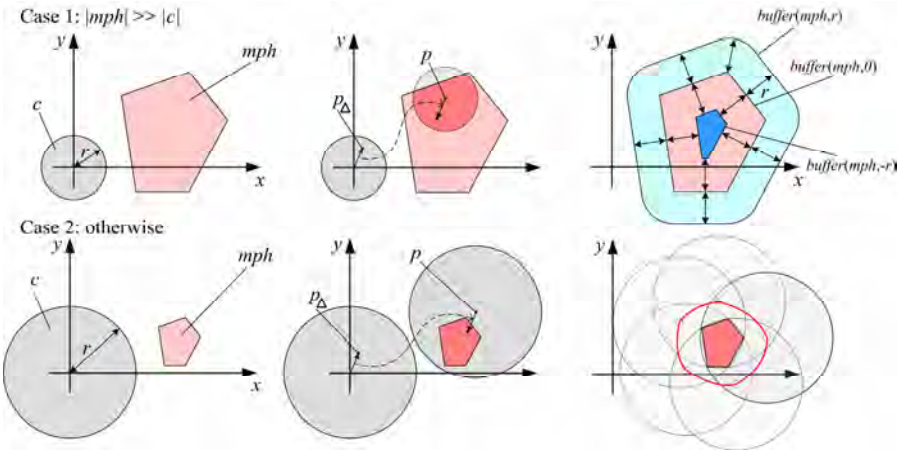

Fig. 4. Cases to convolve densities

The second case is more difficult. We can easily see that for very large circles, the area of non-zero integrals converts itself to a circle. For smaller circles we get a bulky shape as shown in fig. 4 lower right. To compute this shape, we would need a new mph operation that we call *inverse buffer*: similar to the buffer, it contains all points that do not exceed a certain distance, but in contrast to the traditional buffer, we do not use the distance with the *smallest* Euclidean value, but the *largest* one.

Unfortunately, the inverse buffer operation is usually not available as library function. Thus, we conducted some experiments using circular approximations. Usually, these approximations provide sufficient precision for our intended scenarios.

Independent from the case, we can produce a number of areas for every mph of the second density. The number defines the convolution's precision. As the overall number of areas increases, a final resampling step is required.

3.5 Generating Results

With our density representation, it is very easy to compute the *centroid* which represents the most probable position based on all processed predicates. Obviously, the centroid can be computed according to the equation

$$centroid = \frac{\sum\left(centroid(mph)\cdot\left|mph_i\right|\cdot w_i\right)}{\sum\left(\left|mph_i\right|\cdot w_i\right)} = \sum\left(centroid(mph)\cdot\left|mph_i\right|\cdot w_i\right) \qquad (16)$$

Note that for mphs there exist efficient centroid functions.

Often, the centroid does not reflect the intension, especially if we follow multiple alternative paths as introduced in section 2.4. The centroid can, e.g., reside inside a hole with low probabilities. Thus, a further algorithm computes the set of local maxima. At this point we only give the idea of this algorithm:

- As the areas are sorted by their w_i, it is easy to select the absolute maximum value.
- For the selected maximum, we can delete areas that belong to this maximum going "downhill", i.e. we delete such *neighbouring* areas with smaller weights.
- From the remaining areas, we select the next maximum and so forth.

This algorithm is a typical hill climbing algorithm (even though we actually walk downhill). This approach is efficient: First, $\hat{f}(p)$ can easily be computed for every position p. Second, the neighbourhood relation can efficiently be tested using an mph *distance* operation. Note that for variation 1 (see fig. 3) this approach would be much more difficult, as "hills" would be modelled by multiple areas.

Finally, we have to argue, why MAP^3 is able to deal with external data as required in section 2.4. Consider a user with a GPS receiver. The last measured position indicates a subway station before reception fails inside the subway. We apply an *InPoly* predicate that defines all conceivable connected subway stations. As we only have *external* access to subway coordinates, MAP^3 has to generate a look-up request to the subway's spatial database.

In contrast to comparable approaches, it is easy for MAP^3 to compute all conceivable positions (i.e. those with $f(p)>0$), which is simply the geometric union of all mph areas. Kalman filters do not explicitly identify improbable positions. Affirmed knowledge about probabilities in Particle filters is only available for those positions represented by particles. Thus, a complex heuristic would be required to compute an area with non-zero probabilities.

As in our case the area of potential position can easily be computed, we get a simple mechanism to access external data: we use this area as spatial index to external databases. To get a most useful index, we process external data very late in the chain of predicates.

Considering the characteristics of all involved predicate types, we now can define the procedure to process location predicates as follows:

1. We identify the earliest time t for which non-processed predicates exist. We then multiply predicate densities for t according to the priority (1) available convolution results (2) non-negated, local predicates; (3) negated, local predicates (4) external predicates, using the prior result as spatial index.
2. For any time t_2 defined by further predicates, we produce a convolution for (t, t_2).
3. We go back to 1 until all predicates are processed.

After terminating the loop, all predicates are processed and we get a list of densities according to fig. 1. We finally can compute centroids or maximum values for a desired time stamp.

4 Experimental Results

We conducted several experiments to verify the approach. Not surprisingly, MAP^3 works well, if the position input provides a high precision such as position data from GPS. Compared to Kalman and Particle filters, our new approach leads to nearly the same results in such scenarios.

The benefit becomes apparent, if we process typical pieces of information available in location-based service scenarios. Here, we have to consider COO input, especially with negations. Kalman and Particle filters have significant problems with these data as discussed above, but our approach still works properly.

We consider a scenario as presented in fig. 5.

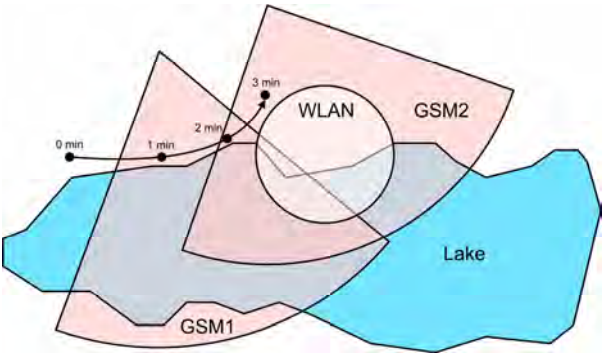

Fig. 5. The sample scenario

A walker promenades at a lakeside. Every minute, his mobile device tries to receive GSM and WLAN cell information. We assume the device can receive up to two GSM cells and one WLAN. The track in fig. 5 first passes cell GSM1, then the overlapping area and finally only cell GSM2. We further know that inside GSM2 there is a WLAN cell that we do not receive. The WLAN cell has a 90% uptime. According to these data, we derive the location predicates as presented in table 2.

Table 2. Input predicates

Time	Predicate	Meaning
-	*MaxSpeed*(5km/h)	Pedestrians are not faster than 5km/h
-	*Not*(*InPoly*(LAKE_POLY))	Pedestrians cannot walk on water
1 min	*InPoly*(1min, GSM1_POLY)	Receiving cell GSM1
2 min	*InPoly*(1min, GSM1_POLY) *InPoly*(1min, GSM2_POLY)	Receiving both GSM cells
3 min	*InPoly*(1min, GSM2_POLY) *Not*(*InPoly*(1min, GSM1_POLY)) *Not*(*Nearer*(2min, WLAN_CENTER, WLAN_RADIUS),0.9)	Receiving only cell GSM2 Not GSM1 WLAN cell is not in range and the WLAN has an uptime of 90%

These predicates undergo the MAP[3] process as presented in section 3. In summary, the process executed 2 convolutions, 3 multiplications and 5 multiplications with negation. Fig. 6 shows the results.

We get the first result after one minute. As we know not to be inside the lake, the remaining cell GSM1 covers the potential positions. In this case, our maximum value computation replies two most probable positions (fig. 6b, indicated by the arrows). This is a typical case of two alternatives as discussed in section 2.4.

The next step (2 min) takes into account the convolution (based on the *MaxSpeed* predicate). Looking at the last step (3 min), we see a plateau with low probability and a peak that indicates the most probable position. The plateau is a result of the WLAN's 10% downtime. This plateau does not significantly contribute to the result for t = 3 min, but as further measurements may indicate that this area covers the only possible positions, it is important to preserve this information.

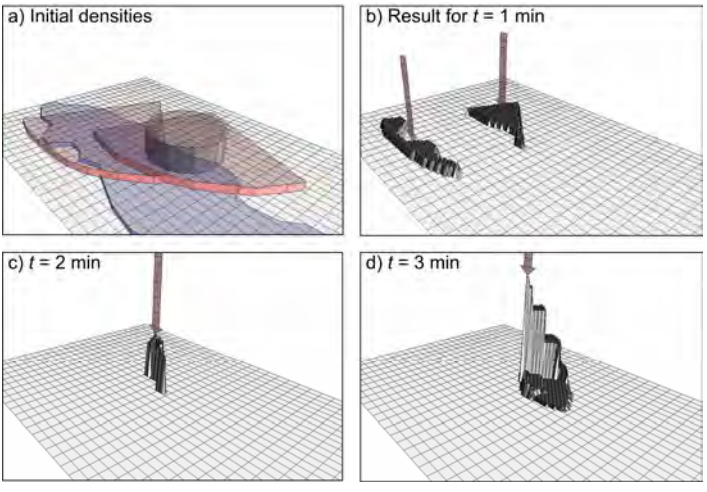

Fig. 6. Computed densities

In this example, the final position estimation considered *all* given information: the position is outside the lake, it considered the received (and not received) cells and it considered the maximum speed. The density presented in fig. 5d thus contains much more information than outputs of comparable approaches.

We chose this simple example due to presentation reasons. We conducted much more complex evaluations that in principle show similar results. With the MAP^3 approach we effectively are able to derive probable positions even from uncertain input data such as presented in this example.

5 Conclusion and Future Work

In this paper we presented the new MAP^3 approach that infers knowledge about the current position from sets of location predicates. Whereas many current approaches assume precise and reliable information, our approach is also able to derive probable positions from widely used positioning systems based on the cell of origin paradigm.

As main benefits, MAP^3 is able to deal with negated and uncertain predicates, alternative paths and non-Gaussian densities that often appear in location- or context-based scenarios. Especially, MAP^3 is able to specify spatial indices to access external spatial databases. Our approach heavily makes use of geometric operations widely available and efficiently implemented in many tool environments, software libraries and spatial databases.

Our current approach processes each unknown position separately. In the future we want to introduce predicates that relate multiple unknown positions to each others. This scenario becomes more and more important. Users may exchange their respective probabilities inside ad-hoc networks and improve their own position knowledge. If they are connected by, e.g., Bluetooth, they know to reside at the nearly same location. To incorporate this knowledge, we have to introduce a new predicate type and the respective processing mechanism.

References

1. Borenstein, J., Everett, B., Feng, L.: Navigating Mobile Robots: Systems and Techniques. A. K. Peters Ltd, Wellesley MA (1996)
2. Bruch, M.H., Gilbreath, G.A., Muelhauser, J.W., Lum, J.Q.: Accurate Waypoint Navigation Using Non-differential GPS. AUVSI Unmanned Systems. Lake Buena Vista, FL (2002)
3. Burgard, W., Fox, D., Hennig, D., Schmidt, T.: Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids. IAAI 2, 896–901 (1996)
4. Doucet, A.: On Sequential Monte Carlo Methods for Bayesian Filtering. Technical Report University of Cambridge, UK Dept. of Engineering (1998)
5. Doucet, A., de Freitas, N., Gordon, N. (eds.) Sequential Monte Carlo in Practice. Springer, New York (2001)
6. Doucet, A., Godsill, S., Andrieu, C.: On Sequential Monte Carlo Sampling Methods for Bayesian Filtering. Statistics and Computing 10(3), 197–208
7. Drolet, L., Michaud, F., Côté, J.: Adaptable sensor fusion using multiple Kalman filters. In: Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), Takamatsu, Japan (2000)
8. Fox, D., Thrun, S., Burgard, W., Dellaert, F.: Particle Filters for mobile robot localization. Sequential Monte Carlo Methods in Practice. Springer, New York (2001)
9. Grewal, M., Andrews, A.: Kalman Filtering: theory and practice. Prentice-Hall, Inc, Englewood Cliffs, New Jersey (1993)
10. Hide, C.D., Moore, T., Smith, M.J.: Multiple model Kalman filtering for GPS and low-cost INS integration. In: Proceedings of ION GNSS 2004, Long Beach, CA, USA (2004)
11. Hightower, J., Borriello, G.: Particle Filters for Location Estimation in Ubiquitous Computing: A Case Study. In: Davies, N., Mynatt, E.D., Siio, I. (eds.) UbiComp 2004. LNCS, vol. 3205, pp. 88–106. Springer, Heidelberg (2004)
12. Hightower, J., Brummit, B., Borriello, G.: The Location Stack: A layered model for location in ubiquitous computing. In: Proc. of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002), June, pp. 22–28. Callicoon, New York (2002)
13. Kalman, R.: A new approach to linear Filtering and prediction problems. Transactions ASME Journal of Basic Engineering 82, 35–44 (1960)
14. Küpper, A.: Location-based Services. John Wiley & Sons, Chichester (2005)
15. Leonhardi, A., Kubach, U.: An Architecture for a Distributed Universal Location Service. In: Proc. of the European Wireless '99 Conference, pp. 351–355 (1999)
16. Open Geospatial Consortium Inc.: OpenGIS. In: Herring, J.R. (ed.) Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture & Part 2: SQL option. (2006)
17. Roth, J.: Flexible Positioning for Location-based Services. IADIS Journal on WWW/Internet 1(2), 18–32 (2003)
18. Roth, J.: A Decentralized Location Service Providing Semantic Locations. Computer Science Report 323, Habilitation thesis, University of Hagen (January 2005)
19. Rubin, D.B.: Using the sir algorithm to simulation posterior distributions. In: Bernado, J.M., DeGroot, M.H., Lindley, D.V., Smith, A.F.M. (eds.) Bayesian Statistics, vol. 3, pp. 395–402. Oxford University Press, Oxford (1988)
20. Sorenson, H.W.: Least-Squares estimation: from Gauss to Kalman. IEEE Spectrum, 7, 63–68 (1970)
21. Wang, Y., Jia, X., Lee, H.K., Li, G.Y.: An indoor wireless positioning system based on WLAN infrastructure. 6th Int. Symp. on Satellite Navigation Technology Including Mobile Positioning & Location Services, Melbourne, Australia, (22-25 July) (2003)
22. Weiß G., Wetzler, C., von Puttkamer, E.: Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans. In: Proc. of the Intl. Conf. on Intelligent Robots and Systems, pp. 595–601 (1994)