A Bitmap-Centric Environmental Model for Mobile Navigation Inside Buildings

Martin Werner and Moritz Kessel
Mobile and Distributed Systems Group
Ludwig-Maximilians-University Munich, Germany
Email: martin.werner@ifi.lmu.de, moritz.kessel@ifi.lmu.de

***Abstract*—Most indoor navigation applications need a realistic description of ways that is easy to understand, remember, and follow. In this paper, we propose a bitmap-centric model that enables the on-demand computation of such ways on smartphones. We explain the creation of the model with the help of common bitmap floorplans by identifying rooms, doors, and their topological relationship, and we describe the generation of navigation graphs from our model. Those navigations graphs are efficiently calculated, as they are formulated as sequences of bitmap operations. Smartphones are well-optmized for such operations, as they are needed for the graphical user interface, too. The final results are realistic and easy to follow, which is achieved with the help of an algorithm for the identification of relevant landmarks. We also show the results of a performance analysis of the algorithms implemented on a standard smartphone, demonstrating the feasibility of our approach.**

***Keywords*-Indoor Navigation; Environmental Models; Image Processing; Indoor Positioning;**

I. Introduction

In the last years, many new multimedia services have been designed. Furthermore, the ongoing trend towards mobile computing and the immense development in the field of cellular phones lead to more and more context-information that can actually be used in numerous applications from the field of location based services. While it is comparably easy to provide location based services for the outdoor area, it is much more difficult to do the same for the indoor area. In the field of indoor positioning and navigation, there are many problems, which have been solved for the outside. The first problem is the availability of digital map data. While for most outdoor location based services the map functionality offered by major Internet companies (e.g., Google Maps) is enough, there is not yet a comparable service for the indoor area. There are several reasons behind: The complexity of indoor maps would be much higher than outdoors (different floors, differences in the treatment of free space). Moreover the content of indoor maps is often protected by intellectual property rights of architects. To enable more users to generate map data for indoor navigation purposes and to enable crowdsourcing approaches to the lack of environmental information, some algorithms were proposed in [1], which enable simple bitmap floorplans to be used for navigation. In this publication, these concepts are extended to a sufficiently detailed environmental model for indoor navigation applications.

Indoor navigation is a very promising technology. People want to have technological support for indoor orientation, which is comparable to the outdoor situation, especially in very complex buildings (industrial buildings), buildings not known to the majority of guests (foreign airports, exhibitions), or buildings not known exactly enough for safety services (firefighter, police, etc.). The most difficult task for indoor navigation is of course the positioning of the users. But in the last decade many promising technologies are under development, which will solve this problem completely in the near future. These include classical signal-strength methods based on Wireless LAN [2]–[8] or more specialised approaches based on UWB [9]–[11], as well as advanced statistical treatments of this measurement data [12]–[15]. Beside these propagation-based systems there have been proposals, which use GPS pseudolites and similar dedicated infrastructure [16]–[18], the variation of the magnetic field [19]–[21], or the variation of the acoustic background spectrum [22].

Furthermore, cellular phones are becoming more and more powerful in terms of calculational power as well as in terms of sensing capabilities. A complete integration of all data that a modern cellular phone can sense will lead to indoor positioning with acceptable accuracy in the future.

Often, indoor navigation has been implemented in a relatively small area due to the focus on positioning technology. In such small settings it is not really important to have efficient algorithms for several tasks. However, large-scale indoor navigation is an upcoming topic [23], [24].

With this paper, we want to introduce a bitmap-centric environmental model for use within indoor navigation applications. Using bitmaps, we follow a minimal modelling effort approach, as bitmaps can be generated from CAD-data or be scanned from a blueprint or even generated by hand by non-specialists using an image manipulation program. The drawback of such an approach is that bitmaps basically do not provide a method to store semantic models, such as the interconnection of rooms. However, this is easy to overcome, if we define a symbolism. In this paper, we provide a bitmap symbolism very near to the typical symbolism of floorplans (e.g., any floorplan can easily be completed and checked by non-specialists using a drawing program).

This symbolism allows us to find rooms and doors,

generate an interconnection graph, calculate shortest routes based on this navigation graph and, by removing the door symbols and calculating a shortest path on a per-pixel basis, we get a candidate for a shortest path inside the floorplan. Unfortunately, such a path will keep near walls. To remedy this effect, we provide two algorithms, which have already been presented in [1]. The first one allows for the transformation of a shortest way into an augmented form, which is easy to understand, remember, and follow.

The second algorithm allows to enhance the visualisability of a given way. Navigation in general is often based on a waypoint graph [25], [26]. A waypoint graph consists of a set of waypoints and an edge inbetween those waypoints, which are directly reachable from each other (usually on a straight line of walking). As the impact of the graph size on the performance of shortest path algorithms is high, people have studied algorithms generating small graphs. However the reduction of the number of waypoints in a graph leads to fewer ways inside the graph. Hence, the shortest way inside the graph is not a way that a human being would choose. One of the smallest and most efficient graphs is for example the corner graph. The corner graph is a graph containing all corners of the building (map) as vertices and an edge between two corners if and only if the direct line inbetween these two points is inside the building and free (walkable) space. The resulting ways tend to scrape along walls as can be seen in Figure 1(a). Another well-behaved form of a navigation graph consists of a subset of a grid of waypoints in navigation space defined by some properties (such as a minimal distance to a wall and being inside the building) and edges between adjacent waypoints if there is free space inbetween. A shortest way inside such a graph can contain some flaws as can be seen in Figure 1(b). In this example the relatively small grid size leads to the effect that the chosen way is too near to the upper wall. This type of problem is exactly what we want to remedy with the second algorithm in this paper.

When it comes to indoor navigation it is essential to fix a model of the surroundings. It is important to define exactly, in terms of available data, what is meant when talking for instance about a room or a door. In the following chapter, we give possible definitions for this in terms of images. Of course there are other views (especially concerning GIS databases), which can be more flexible, but our choice is made on the background that good GIS data does seldom exist for the indoor area, while a basic floorplan (possibly scanned from a building blueprint) is almost always available. Furthermore, cellular phones have more difficulties with handling vector GIS-data and the associated queries than with handling and manipulating bitmaps. Moreover, the standardisation of bitmap file formats allows for flexible and sustainable treatment of indoor navigation data.

In the following Section, we describe an environmental model in terms of bitmap floorplans. In Section III we explain, how to extract the needed room and navigation information out of said floorplans, in Section IV we explain the problem of finding relevant landmarks and describe an algorithm solving this problem. In Section V we describe an algorithm to enhance the visualizability of ways, which also clarifies turning points useful for textual description of the way. We then explain an implementation of this algorithm for mobile phones and give experimental results on the performance. Section VII finalizes this paper with a outlook.

II. The Indoor Environmental Model and its Association with Bitmaps

In the following paragraphs, we want to describe exactly our environmental model and how it is setup with different bitmaps. Starting with a building (or site) in some reference system, we define the bitmap projection by first projecting the building information into an orthogonal coordinate system and map the bounding box of the building to a bitmap by a choice of pixel size in terms of the orthogonal coordinate system. This pixel size can be used to mediate between a small bitmap using large pixel sizes and fine-grained geometry representation using small pixels.

For this bitmap floorplan we start with some definitions, which do not exactly resemble the definitions of common agreement.

Definition 1: An **area** is a subset of available pixels.

Note that an area need not be connected or otherwise have properties, which the term area describes in other contexts.

Definition 2: The **4-neighbour-floodfill** operation at (x, y) is defined to fill the pixel at (x, y), if it is of the same colour as the initial pixel and continues with the neighbouring pixels (above, left, right and below) of the same color.

Definition 3: The **floodfill-closure** of a point (x, y) is the area inside the bitmap that would be filled by a 4-neighbour-floodfill operation at (x, y) with a colour that is not used elsewhere in the bitmap.

Definition 4: An area is called **floodfill-connected**, if and only if it is the floodfill-closure of one (and hence any) of its points.

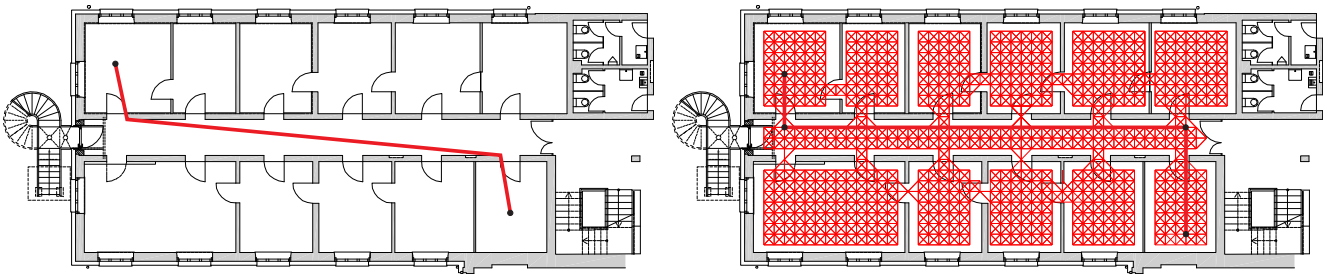
Definition 5: A **room** is a floodfill-connected walkable (non-black) area.

Definition 6: The **outside-space** is the floodfill-closure of the pixel coordinate (0,0).

Definition 7: A room R_1 is **inside** another room R_2 if and only if R_1 is in the convex closure of R_2.

Of course, this definition does not recover the usual term of a room being inside another room. But as it is unlikely that a room that is contained in another room in our sense is not reachable easily this discrepancy to the real world is not severe.

With these definitions in place, we formulate some properties that a floorplan should have for the application of our algorithms below:



(a) A shortest way inside a corner graph

(b) A shortest way inside an accessibility graph

Figure 1. Examples of waypoint graphs and problems

- *Closed-Building-Property*: The building is closed by black lines and completely surrounded by white space. This essentially means that the outside space resembles the usual definition of what is outside a building.
- *Doors-Are-Rooms-Property*: A door inside the building is drawn such that it itself is a room in the sense of Definition 5. An example of a legal door and an illegal door is given in Figure 2.
- *Walkable-Space-Is-Known*: There is a means of telling whether a straight line between two points lies in walkable space (which is defined to be the space where a human being can walk).

For the rest of the paper, we assume that we are given a bitmap - simply called floorplan in the sequel - where these properties hold. Especially, we assume, that the walkable space is provided as a marker color (say white) in the map. Then the property "Walkable-Space-Is-Known" is given by a simple line painting algorithm, which checks that all pixels in the line are of said marker color.

A. Map Symbol Recognition in Mobile Navigation Systems

The environmental model described so far is really minimalistic. A typical environmental model is a representation of data such that a given set of operations is possible. For the GIS-community working with large server infrastructures and desktop computing, this is interpreted in the sense that the set of operations should be as big as possible, but with minimal cost for the individual operation. The size of the model and the complexity of generating such a model are widely ignored. This can be best subsumed by saying that traditional development of location models for ubiquitous computing is based on the question "How should it be?".

A classical discussion of features a location model should provide is given in [27]. Essentially, the authors discuss some examples and queries and conclude that a location model should provide

- *Object Positions:* Positions of objects have to be modelled (either in form of geometric coordinates or in form of symbolic coordinates).
- *Distance Function:* Positions should be interrelated by a well-defined distance function.

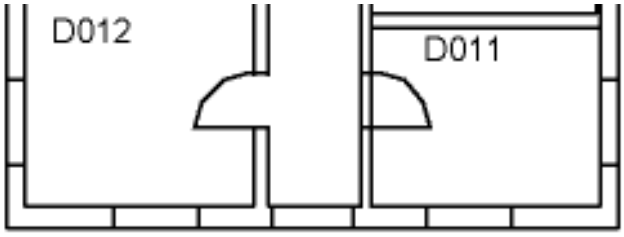


Figure 2. Examples of different doors. The left door is not legal with respect to our properties while the right door is legal

- *Topological Relations:* Spatial containment and spatial connection should be modelled.

Furthermore, they propose to regard these requirements in conjunction with the requirement of minimal modeling effort. The paper then explains some types of basic location models (e.g., Set-based models, Hierarchical models, Graph-based models, etc.) and how they can be structured to fulfill the requirements above.

Our basic "Indoor Environmental Model" does not fulfill the requirements above directly, but we will show how easy it is to come up with complete support for these queries by using some combination of algorithms, and that it is possible to embed all additional information into the floorplan images EXIF-tags.

1) Object Positions and Orientation: Object positions are easily modeled with our approach. A global coordinate consists of a unique identifier of the file (possibly an URL) together with a pixel coordinate (x, y) inside the image. It is of course possible to fix another scale (e.g., one unit on the axis is one meter in the real world) at the expense of saving the scale somewhere or defining it globally for the complete system.

2) Distance functions: As a distance function, we can use the Euclidian distance between pixel coordinates. This basic distance function can be used to calculate the distance between two individual places (e.g., two pixel coordinates (x_1, y_1) and (x_2, y_2)) or even as the distance between two rooms (compare Definition 5).

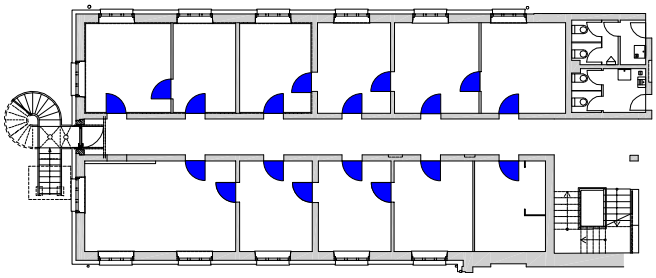


Figure 3. The training image (excerpt)

3) Topological Relations: The relation of a room being contained in another room is immediately available by using the convex closure (see Definition 7). This can be easily extended to any area, including the union of some rooms, a room and its neighbours et cetera. The topological relation of being connected to something is not modelled explicitly, but in the next Section, it will be explained how a navigation graph with one vertex per room from the given floorplan is constructed.

III. Automated Construction of a Navigation Graph from a Floorplan

A navigation graph is a graph consisting of vertices and edges. A vertex has a coordinate and an edge connects two vertices, if a direct walking connection is available inbetween. Such a graph structure can be used to easily and efficiently calculate ways between two points. To keep the graph small enough, we decided to use the rooms as symbolic coordinates. So the navigation graph consists of one vertex per room. If we now discuss the creation of edges, we need two functions based on two given vertices: First of all, we want to detect whether a given vertex constitutes a room and secondly, we need to decide whether two rooms are adjacent to each other. Then the edges in the navigation graphs are exactly between those vertices, whose corresponding rooms are neighboring and where exactly one of them is a door. For an automated construction of a navigation graph, we only need a stable and fast method for detecting doors (e.g., rooms, which have the shape of a door) and a fast method for detecting all neighboring rooms.

For enhanced performance, we will only give the graph in its natural implicit form and generate it on the fly while a graph search algorithm (e.g., Dijkstra or A^*, see [28]) is examining vertices and their neighbours. In this way, we do not have to explode the floorplan into a list of rooms, which could be quite large using our definitions. Think of a lattice drawn somewhere on the floorplan. This will consist of many rooms, which are all irrelevant for the connection graph.

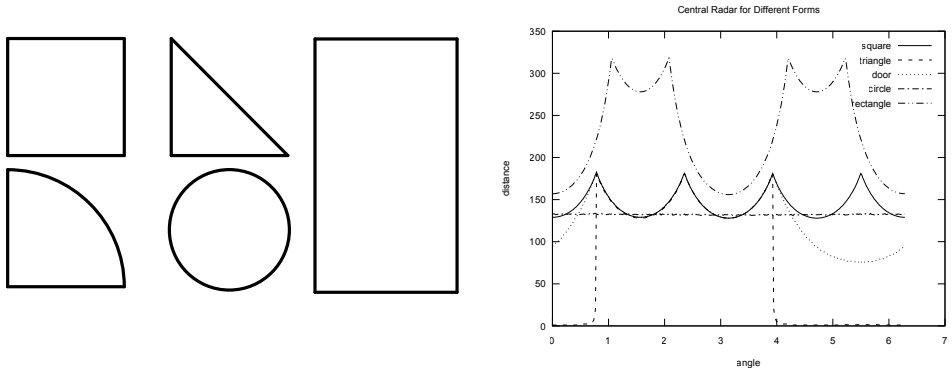
A. Detecting Door Symbols

For the detection of door symbols, we wanted to construct a condensed model with a simple classification system, which is then used to classify a room. Therefore, we set up a list of features, which can be easily extracted from a room (e.g., a floodfill-connected region). These feature vectors are then used together with a manually classified image, where all doors have been marked in blue color for the construction of a training set (see Figure 3). The strength and generalization abilities of the models created from this training data have been tested against rotation transformations. They are not scale-invariant by intention, as we will save the model into the image file itself. Hence, there is no need to recapture scaled versions of an object.

The features are basic features used to distinct between different convex areas, which are easy to compute even on a mobile device. The features are all numeric values describing some aspect of the form. Two very basic features used are given below.

- *Area:* The area is one of the simplest features, but still it is the strongest one. The area of a room is defined as the number of pixels that would be filled by a regular 4-neighbour-floodfill operation. As we want to embed the classification model into the graphics file, it is no problem that this measure is not scale-invariant.
- *Bounding Box:* The width and height of the bounding box of a room used. Here the bounding box is not a minimal enclosing box (which would be more complicated to compute) but is an axis-parallel box containing all pixels that belong to the room. We use the width and height of this box as features.

These simple features did not suffice to classify successfully between doors and other rooms inside a floorplan. The main reason is that the bounding box is not very distinctive for cases, where a door is rotated. It is difficult to differentiate between a square and a door by using a bounding box. Moreover, the bounding box is not rotation invariant. Hence, we constructed a pretty simple, yet efficient algorithm, which calculates a sequence of values based on the floodfill-area, which can be used to derive pretty distinctive features at least for convex forms. The general technique is a variant of a line sweeping mechanism, which we will call *Central Radar*. A ray casting is done in several equally stepped directions (say 1°) recording the radius of the room measured from the middlepoint of the minimal enclosing box. This measure can be zero for nonconvex rooms. Figure 4 shows the central radar of some convex forms and gives a hint, why the central radar is well able to classify between doors and other rooms. For the classification task of detecting doors, we found out that it suffices to use the following simple features derived from a central radar: The *maximum radius*, the *minimum radius* and as a measure for the deviation of the shape from a square their quotient called *squarity*. If the minimal radius is zero and hence the value of squarity is not defined, we declared squarity to be a missing value for the machine learning task.



(a) Some Forms (b) The central radar of these forms

Figure 4. Central radar as a method of obtaining distinctive features for automic recognition of door symbols

Algorithm	Datset 1.)	Dataset 2.)	Dataset 3.)
RIPPER [29]	99.85%	97.47%	99.91 %
Naive Bayes [30]	99.37%	99.15%	99.67 %
C4.5 [31]	99.85%	97.89%	99.90 %

Table I
SUCCESS-RATE OF DIFFERENT MACHINE-LEARNING ALGORITHMS DATASETS 1.) - 3.) DIFFERENTIATING BETWEEN DOORS AND ROOMS AS DETERMINED BY TENFOLD, STRATIFIED CROSS-VALIDATION

Using this feature set, we trained several simple classification algorithms. So we are not using very complex classification algorithms such as neuronal networks as long as simple algorithms provide comparable performance. In essence, all major, simple classification algorithms performed more or less equally well, see Table I. We used three datasets where all rooms where extracted from a floorplan pixel-by-pixel and classified according to a training image, where the doors have been coloured blue:

- *Dataset 1.):* A wing of the building as depicted in Figure 4 (238 rooms including 36 doors).
- *Dataset 2.):* The same wing of the building rotated by 20° (2447 rooms including 36 doors).
  Note that by rotating and the associated blur a lot of small-area "rooms" with non-black pixels appeared, which could have been ruled out by rejecting area below a threshold. But we wanted to see the influence of such typical image noise, which is fortunately void. Removing all rooms below an area of 30 pixels would leave only 198 rooms.
- *Dataset 3.):* A complete floor of the same building. (1755 rooms including 263 doors)
  Note that the building is not axis-parallel, but has some rotated wings as you can see in Figure 5.

As there is no significant performance difference in the classification algorithms, we can choose the right classification type by taking into account, which secondary properties the model does have. For example, it is easy to use RIPPER [29] to induce a set of rules and store this set of rules in an ASCII-string. It is user-readable, pretty easy to understand, and has good performance. Moreover, the rulesets generated with RIPPER were very small. For the complete map, RIPPPER used only three rules containing seven inequalities.Using these rules, only ten doors were overlooked. It could be easy to complete this model by storing ten coordinate pairs as a correction to the model giving another 40 bytes overhead for the classification model. As we want to bind the model to the image file itself, it is also possible to use unpruned and highly overfitted models. Though the models loose their generalization capabilities, the success rate gets almost perfect. A RIPPER model containing ten rules was able to reach a performance of 99.85% with only three misclassifications and uses only 230 bytes in a human-readable text format.

B. Detecting Neighboring Rooms

The only thing left, before a complete navigationg graph can recursively be constructed, is given by the need for a method to find the neighbours of a room. We decided to implement this also as a variant of the central radar algorithm such that the features for detecting doors can be calculated in one step together with the neighbours of a given room. Since it is sufficient to calculate only the neighbors of doors in order to derive a complete navigation graph, problems with non-convex rooms can be avoided. For calculation of neighbours, we use a marker color and find all neighbours floodfilling them to the colour. Then we can enumerate all neighbour rooms by finding the for example first topleft pixel having the marker color, storing this as a room-defining pixel, and floodfilling with white to remove the marker colour from the connected region.

For the colouring of the neighbours, we first fill the room with a marker color (say yellow). Then we use a rotating line around the center point of this room (e.g., the middlepoint of the bounding box), and find the first pixel along each line, which is not black or the marker color of the base

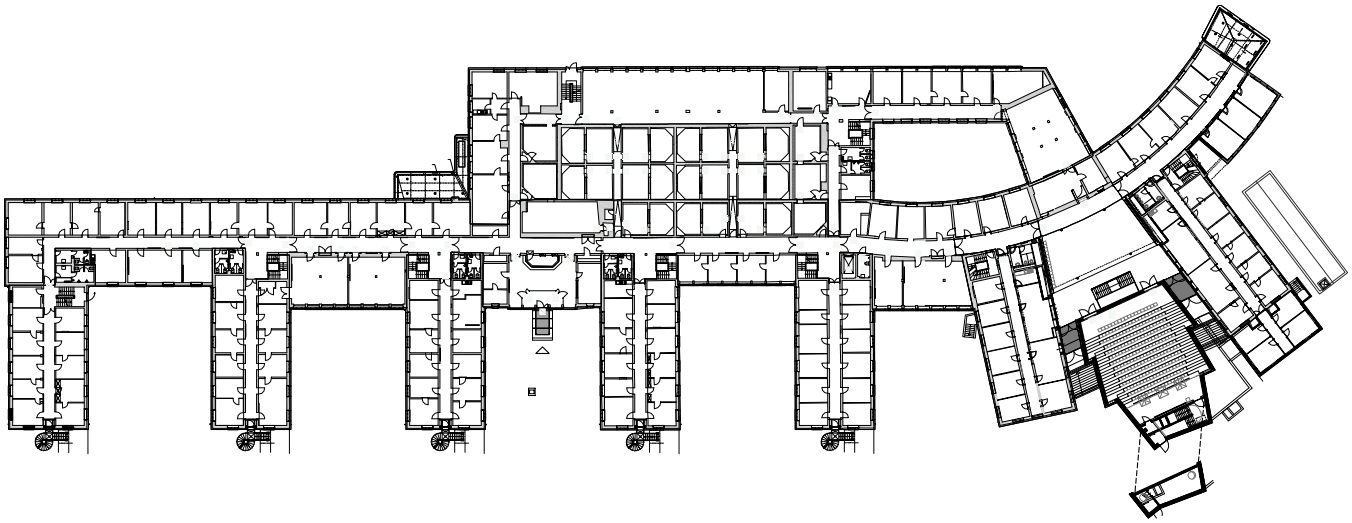


Figure 5. The complete building floorplan

room (yellow). We then floodfill at this pixel coordinate with the marker color for the neighbours (say magenta). Having done this, the base room is yellow and the neighbours are coloured magenta. Note that this algorithm does only work for convex rooms and that the neighbourship relation can still go through walls. But as we stated earlier, our navigation graph will only contain an edge between two neighbours if one of these neighbours is a door, which we can quickly decide using the algorithm of Section III-A.

C. Generating a Navigation Graph

Combining both algorithms from the previous two Sections, it is easy to calculate a navigation graph for the given floorplan as described in the following pseudo code:

```
void calculateGraph(int x, int y){
  if isReady(x,y) return
  for each (nx,ny) in Neighbours(x,y){
    if (isDoor(nx,ny) || isDoor(x,y)){
      AddVertex(nx,ny)
      AddEdge((x,y),(nx,ny))
      calculateGraph(nx,ny)
    }
  }
  isReady(x,y) = true
}
```

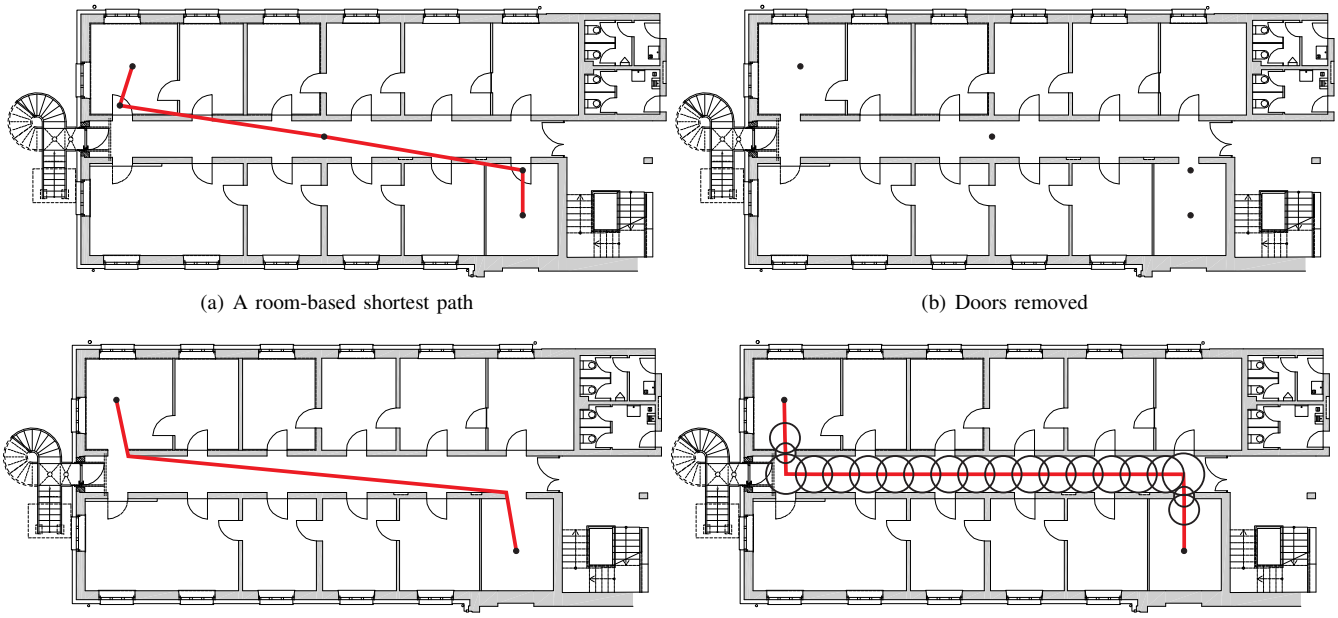
Starting with a specific pixel coordinate (x, y) given by the users position possibly estimated with a positioning system, we first check whether the room specified by (x, y) has already been done. The information of finished rooms can either be coded into the image using another marker color or be stored in a global table. If the room has not been done, we enumerate all neighboring rooms using the algorithm of Section III-B. We then add a vertex for each neighbour and an edge connecting this neighbour to the room specified by the current pixel coordinate (x, y) if one of both rooms is a door. For this task, we use the algorithm given in III-A. We can integrate this process with a graph search algorithm such as Dijkstra or A^*. Then the loop over all neighbours is invoked each time a specific room vertex is popped from the central priority queue. For A^* it is then very likely that not the complete map will be explored, but only a small portion of the given graph for navigation. The shortest path can then be readily given on a per-room-basis but there is no reason to believe that the edges lie inside the rooms.

D. Removing doors

Once we have constructed the navigation graph, we can calculate a shortest route on a per room basis. Once we have calculated such a shortest path inside the graph, we have a sequence of rooms, which leads from the given start position to the end position on a short route. The problem is now, that the shortest path might not lie inside the walkable space, as the graph consists of some points inside the rooms and doors. Even choosing the middlepoints as depicted in Figure 6 as room reference points does not solve this problem. In this situation, we do the following as a preparation for the visualisation algorithm presented in Section V below:

For each door in the shortest path, we calculate a minimum enclosing box, enlarge it a bit (such that the bounding lines of a door get inside the enclosing box) and fill it with white color. Thus, we retrieve one single walkable floodfill-connected region, which connects the startpoint and the endpoint.

Using this representation, we can calculate again a shortest route between the starting point and the endpoint on the basis of a implicit navigation graph consisting of white



(a) A room-based shortest path

(b) Doors removed

(c) The implicit graph structure

(d) The result of the PPW algorithm

Figure 6. Preparing a room-based shortest path for visualisation

pixels as vertices and up to eight edges connecting each pixel with all white neighbour pixels. A weighting for this graph is given by a constant distance of 1 for horizontal and vertical edges and a distance of $\sqrt{2}$ for diagonal edges. Figure 6(c) illustrates this step.

IV. Landmark Search

Pedestrian navigation results need a different presentation form as compared to vehicle navigation results. While for a vehicle navigation system based on GPS the typical errors of the positioning system do not have much influence on the identification of a turn and the orientation of the vehicle is known by the orientation of the street, this is not true for the indoor area. We usually have positioning systems with low accuracy and different possible turns within this accuracy. Typically the positioning errors do not allow the distinction of two doors, which are directly next to each other. In this situation, we want to augment a navigation solution with semantical information such that it is easy to remember and follow. For this case the concept of a landmark is often used. Landmarks are objects in the surroundings having a local uniqueness and being eye-catching. All classical signs are landmarks in this sense. But even shops and plants can serve as landmarks. With landmark information the problems of coarse positioning can be reduced. If landmarks are drawn into a map in form of a pictogram (e.g., the logos of the shops) the relation between the proposed way and those shops can be easily remembered and used for difficult way decisions. This role of a landmark is best explained by the following textual instruction, which can be generated if good landmark information is available: ”Before the post office turn left. Then you will see a red sculpture in 300m distance.” As you can see from this sentence, landmark information can be used to make explicit the position of a turn in relation to semantical information (as opposed to geometric information). Landmark information can also be used to enhance the confidence in having a good orientation as you can see from the second sentence.

The difficult task is now to reduce the set of landmarks (which is usually very big) to the set of landmarks that are visible from the way to facilitate more complex election algorithms for the actual integration of landmark information into visual and textual representations of the way.

The following algorithm is a good symbiosis of a search technology with a geometric enhancement technology. The results of this algorithms is an image containing a set of visible landmarks identified by a colour convention. It is possible to very quickly extract this information as a list of visible landmarks or to directly integrate the graphical result into the visualisation pipeline. One could for example highlight the visible space, draw a pictogram over visible landmark positions and so on.

A. Landmark Search By Image Processing

With the following algorithm we solve the problem of finding relevant landmarks out of a list of landmarks visible from within a way. Common algorithms to solve this problem are more or less searching for landmarks by checking whether a given landmark is visible. As there is no good geometric ordering of landmarks (i.e., reducing the search space by a distance limit will miss good landmarks in long

rooms) it is not easy to do such a search efficiently. This type of search problem also shows up in other problems of ubiquitous and context-aware computing.

A landmark in the sense of the following algorithm consists of a pixel coordinate and a connected information (identification in a database, etc.). The input of the algorithm consists of

- A set of landmarks
- A floorplan
- A way (given as a list of points forming a line-strip)

The configurable parameters influencing this algorithms are

- The maximal viewing distance
- The length threshold used during tessellation of the way

1) Step 1: Prepare Landmark Map: The very first step is to overlay our floorplan with drawn landmark locations. Therefore we use a colour palette mapping a colour (that is not used in the floorplan) to the identification data. This mapping is symbolised in the following pseudo-codes by the function `landmark_to_colour(landmark l)`.

```
void draw_landmarks(){
  copy (floorplan, landmark_map)
  for each landmark l in landmark_set{
    putpixel(landmark_map, position,
         landmark_to_colour(l))
  }
}
```

This step is of course general and the result can be cached for subsequent applications of this algorithm.

2) Step 2: Tessellation of the Way: As it is computationally very expensive to calculate the set of pixels, which are visible from a line-strip, we approximate this set of pixels by the set of pixels visible from the points of a tessellation of the line-strip. To obtain this tessellation, we keep inserting middlepoints between two subsequent points until the distance between all points is shorter than the *tessellation length*.

```
global tessellation_length
void tessellate(linestrip l){
  for each segment s of l{
    if (s.length() > tessellation_length){
      s.split()
      return tessellate(l)
    }
  }
}
```

3) Step 3: Calculate the mask bitmap: In this step we calculate a mask, which resembles the set of pixels visible from the way. This is done by preparing the mask bitmap to be of the same size as the floorplan and filled with black. We then use a radial floodfill operation starting at each point of the tessellated way and copying every examined pixel into the mask bitmap.

```
void calculate_mask(){
  for p in tessellated_way{
    radial_flood_fill(p);
  }
}
```

The radial floodfill algorithm fills out the area surrounding a point as long as there is a direct line between each point and the starting point. Note that the resulting area need not be convex (see Figure 7(b) for an example).

4) Step 4: Multiply the landmark map with the mask: In this step, we stamp the visible area out of the landmark map. For each black pixel in the mask bitmap, we black out the same pixel in the landmark map.

```
void mask_out(){
  for each (x,y) in landmark_map{
  if (mask_map(x,y) != black)
    result(x,y) = landmark_map(x,y)
  }
}
```

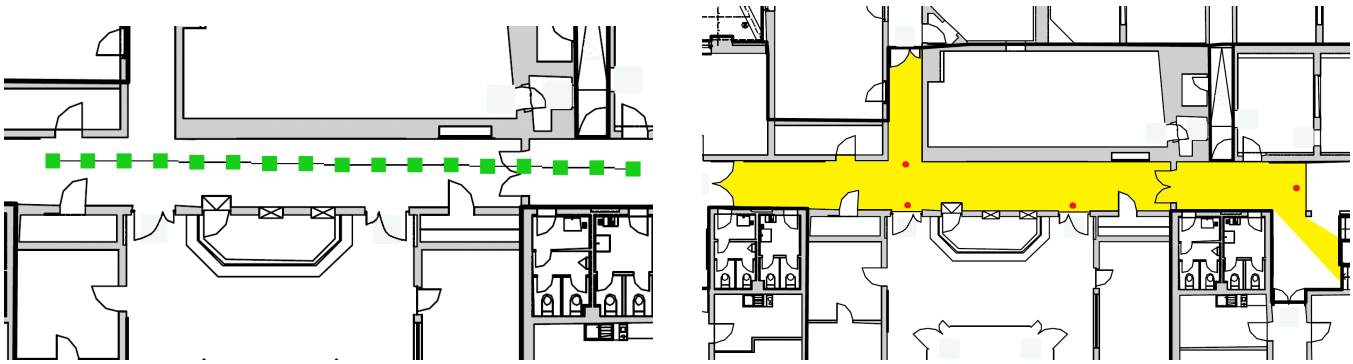
This results in a bitmap containing exactly the visible landmarks together with their geometric location. It is now up to the rest of the visualisation pipeline how to work further. One could quickly scan the image and get a list of visible landmarks or one could just overlay all landmark pixels with a pictogram assigned to the landmark.

Though this algorithm seems to be very complex, it has some beneficial properties. First of all, it is a formulation of the landmark search problem in terms of basic image processing. Secondly, its result is near to a complete visualisation of the landmarks and thirdly, it is highly parallelisable and designed to be run on dedicated graphics hardware. Examples for an application of this algorithm are given in Figure 7.

V. Post-Processing Ways (PPW-Algorithm)

Waypoint graphs in general impose visualisation problems. If the graph is too coarse, then the shortest ways will appear unnatural, e.g. moving to middlepoints of rooms. Even if this is not the case, the best natural-looking ways are usually not the shortest, but have some relation to the surrounding geometry making them easily visible. If the graph is too fine, then shortest ways tend to scrape along walls and cross large halls in diagonals. Hence, for an indoor navigation system it is difficult to find ways, which are short and at the same time have a specific quality with respect to visualisation. As the efficiency of search algorithms is tightly coupled with the number of vertices and edges inside the graph, it is common that people try to have small waypoint graphs.

With the following algorithm, we want to post-process such ways to obtain a relatively nice visualisation with acceptable computational overhead.



(a) The tessellation of a way

(b) The mask and landmarks (large dots) found

Figure 7. The results of the Landmark Search Algorithm

A. The PPW-Algorithm for Better Visualisation

Therefore, we propose the following algorithm, which essentially is a series of image processing operations. The input of the algorithm consists of

- A way (given as a list of points forming a linestrip)
- A collision map

The configurable parameters influencing this algorithms are

- The maximal length a point may move during the algorithm
- The length threshold used during tessellation of the way
- The valuation choosing the best movement in a set of possible movements

1) Step 1: Tessellation of the Way: For this algorithm we need a tessellation of the way just as for the previous algorithm. The reader is referred to Section IV-A2 for details.

2) Step 2: Move the tessellation points: For each point in the tessellation of the way determine a set of points, where we could move this point. Therefore we grow a circle until this circle collides and if this circle collides, we move the middlepoints away from the collision points until we get stuck. Hence we find out the position where - in a limited neighbourhood of each tessellation point - the biggest circle fits into free space and move this tessellation point to this position (see Figure 6(d)).

Number of Points	Running Time
5	0.759s
9	0.985s
17	1.633s
33	2.955s
65	5.623s

Table II
PERFORMANCE OF THE PPW ALGORITHM RUNNING ON HTC DESIRE

The following pseudo code illustrates this step. We found out in experiments, that a maximal movement distance of four times the *tessellation length* makes sense for relatively fine tessellations.

```
global tessellation_length
for each point p {
  res = grow_a_circle(p,
          tessellation_length * 4)
}
```

The method `grow_a_circle` is just scanning possible positions and calculating the maximal circle in free space centered around these positions. Each position is then assigned a valuation composed out of the radius of the circle and the distance of the movement. In the examples throughout this paper, we just used a valuation prefering the biggest circle in the allowed space and inbetween all those circles with maximal radius the one nearest to the original point.

```
void grow_a_circle(point p, double d){
  last_result = p;
  for  q in box (p-(d,d), p+(d,d)) {
    r = maximal_radius(q);
    if (is_better_than_last_result(q,r))
      last_result = p;
  }
}
```

The results of this algorithm are given in Figure 6(d). As you can see, this algorithm leads to a fairly good way. What is not obvious but has been tested with a multitude of other ways, is the fact that the algorithm has the beneficial side effect of having a relatively clear turn (e.g., a turn of merely exactly 90 degree in the Figure above) exactly where a turn should be indicated by a text generation engine. Furthermore, due to using a rotation-invariant definition of good way, the orientation of the rooms inside the bitmap is not relevant.

VI. An Implementation of PPW for Mobile Phones

The algorithms presented in this paper are relatively complex. If we apply these algorithms to long ways, the uniform tessellation algorithm leads to many points in the tessellation and for each of those points another complex operation is needed. To enhance clarity, we decided to explain the algorithms in the most simple form given above. Of course the growing circle algorithm can gain a real performance boost from not growing the radius one pixel at a time. Starting with an exponential growth of the radius and correcting the first collision by a nested interval algorithm in comparison to the last non-colliding circle will gain much speed. Moreover, for plans where the magnitude of rooms is constructed from parallel lines and the number of randomly placed obstructions is small, the circle can be replaced by a square without any harm. Using integral images in this case allows to answer the question, whether a square collides with geometry, in constant time. If we can afford the memory and the map does not change too often, we can even compute the maximal radius for each pixel and store it as a color component value inside another bitmap. In this case (ignoring the time of constructing this map), we can omit the process of growing a circle and concentrate on the movement of the center point.

We implemented the PPW algorithms for modern smartphones running Android OS 2.2 and above. The Post-Processing Ways algorithm (Section V) is running fast enough. The system is rendering a map into a screen buffer (a Java bitmap of the exact pixel size of the screen), which is then passed to our implementation of this algorithm via Java natives. In Java natives, we are performing the image processing as described on a 16-bit-per-pixel bitmap (using essentially 5 bits per color). For this task, we implemented a fast and stable bitmap manipulation library. The tessellation is carried out in Java. Experimental performance results for the Post-Processing Ways algorithm are given in Figure V-A2.

For moderate numbers of tessellation points, the running time of the algorithm is quite acceptable. The algorithm has to be run only once for each navigation result. As the effective screen resolution of a full-screen application (not drawing over the status bar) on the device is 480x725, the number of tessellation points to consider will not exceed 20 points. As mobile devices are able to run this type of algorithms natively, we are able to provide full navigation functionality with navigation graphs, which have relatively bad visualisation properties such as corner graphs.

VII. Outlook

With this paper we have presented an extension of our previous work [1]. We have shown that a bitmap can provide enough environmental information for high-quality location-based services by applying some simple image precessing tasks. Furthermore, a technique for the generation pretty general symbol recognition is provided and the possibility of embedding this information into the bitmaps EXIF tags in ASCII format has been demonstated. We applied this technique to differentiate between doors and rooms using highly overfitted models of how a door is drawn on the plan and were able to automatically generate a suitable navigation graph hierarchy. The first level consists of a navigation graph containing one vertex per room (and hence also a vertex per door, as we expect doors to be rooms). The second level consists of a free-space pixel-based graph, which connects adjacent pixels of the same colour. Having all this in place, we have a perfect starting point for the visualisation and augmentation algorithms presented before in [1].

This work allows us to extend the philosophy that the visualization properties of navigation graphs are not important. Furthermore, we showed that simple bitmaps conform to some conventions are feasible alternatives to the design and use of complex environmental models. For a navigation application on a smartphone, some simple rules and some good algorithms are enough to provide a complete and high-quality indoor navigation application. Finally, our integration of non-trivial functionality into the EXIF-tags of the floorplans makes it easy to implement our environmental model inside a mobile browser. Those browsers not supporting our new technology will still be able to show the map as a basic navigation aid.

References

[1] M. Werner, “Efficiently using bitmap floorplans for indoor navigation on mobile phones,” in *Proceedings of the Seventh International Conference on Wireless and Mobile Communications (ICWMC 2011)*, 2011, pp. 225–230.

[2] Y. Chen and H. Kobayashi, “Signal strength based indoor geolocation,” in *Proceedings of IEEE International Conference on Communications (ICC 2002)*, 2002, pp. 436–439.

[3] F. Evennou and F. Marx, “Advanced integration of wifi and inertial navigation systems for indoor mobile positioning,” *EURASIP Journal of Applied Signal Processing*, 2006.

[4] P. Bahl and V. N. Padmanabhan, “Radar: an in-building rf-based user location and tracking system,” in *Proceedings of the 19th IEEE Conference on Computer Communications (INFOCOM 2000)*, 2000, pp. 775–784.

[5] T. King, S. Kopf, T. Haenselmann, C. Lubberger, and W. Effelsberg, “Compass: A probabilistic indoor positioning system based on 802.11 and digital compasses,” in *Proceedings of the First International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH 2006)*, 2006, pp. 34–40.

[6] M. L. Zehner, K. Bannicke, and R. Bill, “Positionierungsansätze mittels WLAN-Ausbreitungsmodellen,” 2005.

[7] M. Kessel and M. Werner, "Smartpos: Accurate and precise indoor positioning on mobile phones," in *Proceedings of the First International Conference on Mobile Services, Resources, and Users (MOBILITY 2011)*, 2011, pp. 158–163.

[8] W. Xiao, W. Ni, and Y. Toh, "Integrated wi-fi fingerprinting and inertial sensing for indoor positioning," in *Proceedings of the International Conference on Indoor Positioning and Indoor Navigation (IPIN 2011)*, 2011.

[9] C. Falsi, D. Dardari, L. Mucchi, and M. Z. Win, "Time of arrival estimation for uwb localizers in realistic environments," *EURASIP Journal of Applied Signal Processing*, 2006.

[10] B. Waldmann, *Design of a Pulsed Frequency Modulated Ultra-Wideband System for High Precision Local Positioning*. Logos Verlag Berlin, 2011.

[11] P. Steggles and S. Gschwind, "The ubisense smart space platform," in *Adjunct Proceedings of the Third International Conference on Pervasive Computing*, 2005, pp. 73–76.

[12] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174 – 188, 2002.

[13] P. Davidson, J. Collin, and J. Takala, "Application of particle filters for indoor positioning using floor plans," in *Ubiquitous Positioning, Indoor Navigation, and Location Based Service*, 2010.

[14] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P. J. Nordlund, "Particle filters for positioning, navigation and tracking," in *IEEE Transactions on Signal Processing*, vol. 50, no. 2, 2002, pp. 425 – 437.

[15] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman filter: Particle filters for tracking applications*. Artech House Publishers, 2004.

[16] H. S. Cobb, "Gps pseudolites: theory, design, and applications," *Ph. D. Thesis, Stanford University*, 1997.

[17] C. Kee, D. Yun, H. Jun, B. Parkinson, S. Pullen, and T. Lagenstein, "Centimeter-accuracy indoor navigation using GPS-like pseudolites," in *GPSWorld*, 2001.

[18] C. Rizos, G. Roberts, J. Barnes, and G. N., "Locata: A new high accuracy indoor positioning system," in *Proceedings of the International Conference on Indoor Positioning and Indoor Navigation (IPIN 2010)*, 2010.

[19] Ascension Technologies, "Motionstar magnetic tracker," 2001, http://www.ascension-tech.com/.

[20] W. Storms, J. Shockley, and J. Raquet, "Magnetic field navigation in an indoor environment," in *Ubiquitous Positioning, Indoor Navigation, and Location Based Service*, 2010.

[21] F. Raab, E. B. Blood, T. O. Steiner, and H. R. Jones, "Magnetic position and orientation tracking system," in *IEEE Transactions on Aerospace and Electronic Systems*, 1979, pp. 709–718.

[22] S. P. Tarzia, P. A. Dinda, R. P. Dick, and G. Memik, "Indoor localization without infrastructure using the acoustic background spectrum," in *9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011)*, 2011, pp. 155–168.

[23] P. Ruppel, F. Gschwandtner, C. K. Schindhelm, and C. Linnhoff-Popien, "Indoor navigation on distributed stationary display systems," in *Proceedings of the 33rd International Conference Computer Software and Applications Conference (COMPSAC 2009)*, vol. 1, 2009, pp. 37–44.

[24] W. Chung, G. Kim, M. Kim, and C. Lee, "Integrated navigation system for indoor service robots in large-scale environments," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2004)*, 2004, pp. 5099–5104.

[25] P. Tozour, "Search space representations," *AI Game Programming Wisdom 2*, vol. 2, pp. 85–102, 2004.

[26] K. Yu, "Finding a natural-looking path by using generalized visibility graphs," in *PRICAI 2006: Trends in Artificial Intelligence*, 2006, pp. 170–179.

[27] C. Becker and F. Dürr, "On location models for ubiquitous computing," *Personal and Ubiquitous Computing*, vol. 9, no. 1, pp. 20–31, 2005.

[28] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag Berlin Heidelberg, 2008.

[29] W. W. Cohen, "Fast effective rule induction," in *Proceedings of the Twelfth International Conference on Machine Learning*, 1995, pp. 115–123.

[30] G. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 1995, pp. 338–345.

[31] R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.