Automatic Generation of Topological Indoor Maps for Real-Time Map-Based Localization and Tracking

Martin Schäfer, Christian Knapp and Samarjit Chakraborty
martin.schaefer@tum.de, knapp_christian@mytum.de, samarjit@tum.de
Institute for Real-Time Computer Systems,TU-Munich, Germany.

Abstract—**Personal location information is regarded as the most important contextual information transmitted in ubiquitous systems. Many pedestrian indoor localization systems rely on map-matching to constrain sensor errors. The maps required for computer aided localization and tracking need to incorporate a semantic structure. Such maps are not readily available and therefore most groups working on localization solutions manually create the required maps for specific testing scenarios. To provide a solution for map generation on a larger scale, we have developed a map generation toolkit that parses standard CAD-plans, to automatically generate topological maps for indoor environments. We propose a heuristic parser that separates superfluous data from the information depicting semantic building entities, e.g. rooms and doors. In our experiments approximately 95% of all structures were detected successfully. After the extraction we transform the extracted building information into an object-based building model designed for the application of fast particle-filter-based map-matching algorithms. A performance test with a typical filter implementation demonstrates that the model is sufficiently optimized to achieve pedestrian tracking and localization in real-time.**

I. Introduction

Every personal indoor positioning system needs an underlying map as reference. An absolute position is of no use without relation to the surrounding building. Furthermore, many pedestrian indoor localization and tracking concepts rely to a certain extent on map-based filtering algorithms to bound drift and noise induced errors. These algorithms are most commonly based on particle filters. The users trajectory is described by a set of particles. The particle distribution models the measured trajectory as well as the errors of the measurement systems. To limit the accumulation of measurement errors, the set is filtered using map induced constraints. In other words: if a particle traverses a wall, its weight is reduced to zero and the particle is deleted from the filter set. To filter the trajectories of all particles, a topological map structure with fast look-up times is imperative. In contrast to semantically enhanced outdoor maps, e.g used for car navigation, information on indoor environments is usually available in form of architectural CAD files. These files contain all types of building information like walls, stairs, windows and appliances. However, their structure is focused on human readability and does not contain information that permits a computer to distinguish the different object types. Since a semantic classification of obstacles and accessible areas is necessary for map-based indoor localization and tracking, most research groups in this domain have resorted to manual construction of suitable plans [1], [2], [3]. While this approach is valid to demonstrate the performance of a filtering concept, the application of the proposed algorithms in global or urban scenarios usually remains infeasible, due to the lack of time required for the generation of appropriate maps. As also stated in [4], we argue that the lack of maps that are suited for both, visualization and map-based filtering, is one of the main obstacles for the mass-market deployment of indoor positioning systems. Our work therefore aims to provide the missing link between architectural maps in CAD format and semantic mapping information. We present a parser that analyses standard CAD files to extract topological map information. This information is used to create an object-based map optimized for localization and tracking applications. To illustrate the model's properties, we additionally propose an adapted map-matching algorithm and analyze its performance.

The remainder of this paper is structured as follows:

In section II we cite related work. Section III describes the parsing algorithms necessary to extract information from the CAD files. Section IV introduces the object-based map and section V illustrates the proposed map-matching algorithm and analyses the framework's performance. Section VI concludes the paper.

II. Related Work

As CAD plans do not provide any topological information, several standards on building models have been formulated to extend the geometric information of CAD plans with semantic content. The most prominent among them are the Industry Foundation Classes (IFC) [5] and the CityGML framework [6]. Both are focused on visualization in the architectural domain but they already offer detailed descriptions of internal structures including semantic links. Unfortunately for most public buildings there are no CityGml models available. In [7] and [8] concepts to convert CAD data into CityGml models are proposed, however both approaches require considerable user interaction and only provide good results with high quality CAD drawings. For instance rooms must be depicted as closed polygons, doors must be encoded as an arc and each structure type must reside in a dedicated layer, which is not the case for many available CAD data. In [9] a variation of the CityGML model is described. The model named BIGML provides a semantic map suited for location-based services. The authors also propose a parser for DXF files, but the parser is once more limited to few specific CAD files that encode rooms as polygons and group information in a predefined layer structure. Besides the extraction of topological plans from CAD files several research groups also suggest the refinement of coarsely digitalized floor-plans with user generated data. In [10] map users are encouraged to indicate appliance and furniture positions to improve the accuracy of a map. However, the basic floor-plan must still be derived manually. The authors of [11] propose the creation of maps during the localization phase (SLAM). While this approach does necessitate only minor manual post-processing to generate navigable maps, the data acquisition process itself remains laborous. Several users need to thoroughly explore every entity in a building before a complete map can be obtained. To the best of our knowledge none of the mentioned frameworks provides automatic map-creation from standard 2D CAD files and none of the maps is designed to provide map-based filtering.

III. Parsing CAD Files

Since we aim at providing a general map extraction tool, our focus lies on analyzing CAD files in the Drawing Interchange Format (DXF) published by AutoDesk [12]. DXF is an open specification specially designed to provide an interchange format between proprietary CAD formats of commercially available CAD tools. Since almost every CAD file can be converted into a DXF file, authorities wanting to equip their facility with an indoor navigation system, can easily generate the required format from their building plans.

The CAD data encoded in DXF files consist of several unconnected lines, arcs and poly-lines (several line segments concatenated) spread across several drawing layers. Lines depicting doors are typically grouped in one or two layers. The outlines of rooms are often grouped together with labels, pillars and other line information and spread over several layers. Additionally the desired entities are not distinguished and the lines delimiting physical rooms are often superimposed with drawing information concerning floor, ceiling and accessory labels. For the scope of this paper, we assume the CAD data to be structured as one file per each floor of a building. The current parser is restricted to the extraction of 2D structures like rooms and doors and the 2D projections of stairs. Thus the resulting maps only contain the projections of three dimensional structures like stairs. Although this is a shortcoming, the impact on the suitability for map-matching is only minor. While a three dimensional map can be useful to describe complex rooms as pointed out in [2] localization and tracking in a normal building can easily be achieved on the basis of several two dimensional floor plans and the according heights. The main purpose of map-based filters is to bound noise and drift of a 2D position and the associated heading. The change of a floor via stairs or an elevator can be easily detected via a change of the measured altitude. The measurement of the altitude is generally much more accurate and does not necessarily require sophisticated filtering.

A. Data Pre-Processing

Before the parser analyses the CAD data, the user has to indicate the floor depicted by the opened file and mark the most promising of the visualized drawing-layers. CAD designs are never completely free of errors. Furthermore, conversion from proprietary formats into the DXF format can introduce additional conversion errors like numerical inaccuracies or logical errors. To take this into account, the proposed extraction algorithms use error tolerant calculation and not an exact algebra. A point is defined as a regular 2D vector $p \in \mathbb{R}^2$. For brevity, the vector connecting two points p_1 and p_2 is written as p_{12}. A line is defined as an ordered set of two points $\mathbb{L} := \{(p_1, p_2)| \; p_1, p_2 \in \mathbb{R}^2\}$. On these types we define the following relations:

a) Aproximately Equal Points: Two points are approximately equal iff their distance is smaller than a predefined threshold ϵ.

$$\forall p_1, p_2 \in \mathbb{R}^2 : p_1 \; adj \; p_2 \Leftrightarrow | \; p_{12}| < \epsilon \tag{1}$$

b) Aproximately Equal Lines: Two lines are defined as approximately equal iff their points are approximately equal. Start- and end-point can be switched.

$$\forall L_1, L_2 \in \mathbb{L} : L_1 \approx L_2 \Leftrightarrow \{\forall p \in L_1, \exists! q \in L_2 | \; p \approx q\} \tag{2}$$

c) Point Adjacent to Line: A point p is adjacent (adj) to a line L iff one of L's endpoints is approximately equal to p.

$$\forall p \in \mathbb{R}^2, \forall L \in \mathbb{L} : p \; adj \; L \Leftrightarrow \{\exists! q \in L | \; p \approx q\} \tag{3}$$

d) Adjacent lines: Two lines L_1, L_2 are adjacent (adj) iff L_1 contains one endpoint adjacent to L_2.

$$\forall L_1, L_2 \in \mathbb{L} : L_1 \; adj \; L_2 \Leftrightarrow \{\exists! q \in L_1 | \; q \; adj \; L_2\} \tag{4}$$

e) Paralell Lines: Lines L_1 and L_2 are considered as approximately parallel $||$ iff the dot product of their vectors almost equals the product of their lengths ($\pm\epsilon$).

$$\begin{aligned} &\forall L_1(p_1, p_2), L_2(p_3, p_4) \in \mathbb{L} : L_1 || L_2 \Leftrightarrow \\ &|(p_2 - p_1) * (p_4 - p_3)| - |p_{12}| * |p_{34}|| < \epsilon \end{aligned} \tag{5}$$

f) Orthogonal Lines: Lines L_1 and L_2 are considered as approximately orthogonal $\perp$ iff the dot product of their vectors is approximately zero ($\pm\epsilon$).

$$\begin{aligned} &\forall L_1(p_1, p_2), L_2(p_3, p_4) \in \mathbb{L} : L_1 || L_2 \Leftrightarrow \\ &|(p_{12}) * (p_{34})| < \epsilon \end{aligned} \tag{6}$$

g) Line 1 Contains Line 2: A line $L_1(p_1, p_2)$ contains the line $L_2(p_3, p_4)$ iff both points p_3 and p_4 are almost covered by the line L_1 and not exactly matching L_1's points p_1 and p_2.

$$\begin{aligned} &\forall L_1(p_1, p_2), L_2(p_3, p_4) \in \mathbb{L} : L_2 \subset L_1 \Leftrightarrow \\ &\exists u \in \mathbb{R}^2, \; \exists a \in \mathbb{R}, 0 < a < 1| \\ &u = p_1 + (p_{12}) \cdot a, \; u \approx p_3 \wedge \\ &\exists v \in \mathbb{R}^2, \; \exists b \in \mathbb{R}, 0 < b < 1| \\ &v = p_1 + (p_{12}) \cdot b, \; v \approx p_4 \end{aligned} \tag{7}$$

To reduce the amount of redundant data the following pre-processing steps are automatically performed on each superset of layers selected for one of the extraction routines:

- If several layers contain information required in the same extraction routine, these layers are merged into one superset.
- A line that is *approximately equal* to another line (a duplicate) is removed from the superset.
- A line L_1 contained in another line L_2, that has no other adjacent lines than L_1, is removed from the superset.
- *Adjacent* lines that are *parallel* and have no other neighbors, i.e that could also be formulated as one single line without loosing information, are concatenated.
  $\exists L_1, L_2 \in \mathbb{L} |\ L_1\ adj\ L_2\ \wedge\ L_1 || L_2$
  $\exists p \in \epsilon L_1 | p\ adj\ L_2\ \wedge\ p\ !adj\ L_{i|i\neq 2} \Rightarrow$ fuse L_1, L_2.

B. Extracting Doors

In well drawn CAD plans, the walls show an open passage at the door sill and the door is normally encoded as a line depicting the open door. The opening path usually touches the wall of a room indicating the position of the closed door. The origin of the arc indicates the hinge. The information important for map-matching is the actual door sill. As already pointed out in [13] the extraction of a door from a drawing showing all these features is rather trivial. The left sketch in Fig. 1 shows such an ideal door. In contrast to this representation, the right image in Fig 1 shows a detail from CAD data. Four doors are shown, none of which has a clearly visible door sill. Two doors seem to open into a wall. Further the opening path of a door is sometimes encoded as an arc and sometimes as a polyline approximating an arc. To take these circumstances into account we focus on the extraction of characteristic arcs and polylines to identify doors. As depicted in Fig. 2 up to two lines eligible as door sill can be identified per arc.
If one line is found this usually depicts the open door. In some sources two lines per door are depicted, one for the open door, the other one representing the actual door sill. To isolate the desired door sills our parser performs the following sequence.

- Find consecutive lines that roughly depict a quarter circle with a radius ≈ the length of a door step.
- Convert these lines to arcs. An arc is defined by its center p_c, and a start p_s and end point p_e.

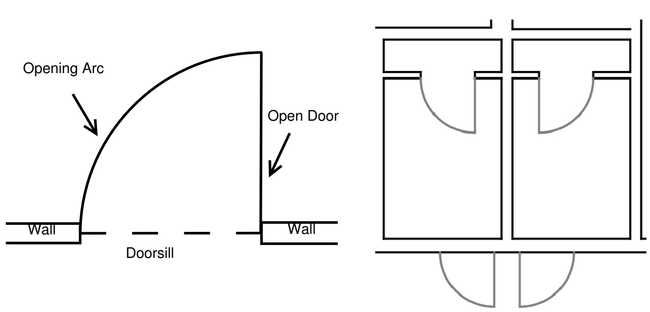

Fig. 1. Comparison of an ideal door with a detail taken from a CAD-file

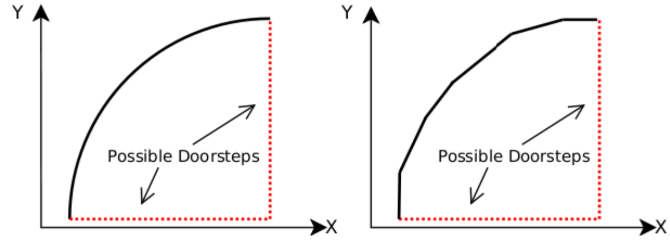

Fig. 2. Examples of alternative door representations, often only one of the dotted lines (the open door) is present in the CAD data

- Find arcs that roughly depict a quarter circle with a radius ≈ the length of a door step.
- For each identified quarter circle, search for lines that are adjacent to the arc's center and *adjacent* to one of the arc's endpoints p_s/p_e . If such a line is found save the line between p_c and the opposing endpoint p_e/p_s as door sill candidate.
- For all arcs with more than one door sill candidate: For each candidate L_i try to find a wall line *containing* L_i. If found, mark L_i as valid candidate. If no candidate remains or more than one candidate remains, mark the arc for user interaction.

The algorithm implementing the described steps reads as follows:

Algorithm Door-Extraction

```
foreach(arc in rawArcs){
  //sort out arcs that do not fit
  if ((arc.rad ≤ dMin) || (dMax ≤ arc.rad)){
    continue;
  }
  if (arc.angleDiff() !≈ 90.0°)){
    continue;
  }

  vector doorCandidates;
  foreach(line in rawDoorLines){
    //sort out lines that are
    //too short or too long
    if ((line.length ≤ dMin) ||
       (dMax ≤ line.length)) continue;

    //sort out lines not adj to arcs center
    if ((line !adj arc.center) continue;

    //if a line from arcs center to one end
    //point is found the opposing line
    //is a candidate for closed door
    if ((line adj arc.end){
      newLine = Line(arc.start,arc.center);
      doorCandidates ← newLine;
    }else if (line adj arc.start){
      newLine = Line(arc.end,arc.center);
      doorCandidates ← newLine;
    }
  }
```

```
  if(doorCandidates.size()==1){
    doorSteps ← doorCandidates[1];
  }else if  (doorCandidates.size()>1){
    foreach(cand in doorCandidates){
      cand.setInValid();
      foreach(wall in wallLines){
        if (wall contains cand){
          cand.setValid;
          break;
        }
      }
      if (!cand.isValid()){
        doorCandidates.remove(cand);
      }
    }
    if(doorCandidates.size()>1){
      problems ← arc;
    }
  }
}
treatInteractively(problems);
```

C. Extracting Rooms

For room extraction a more sophisticated approach is necessary, since the variation of room representations used in CAD files is significantly higher. For instance, in the available data we encountered problems like incomplete room boundaries, lines delimiting several rooms and lines depicting only half a wall. We therefore propose an iterative algorithm that isolates closed and almost closed line-sequences that are candidates for rooms. These candidates are stored and filtered for duplicates and erroneous detections using several additional constraints. The algorithm is structured into four stages (a,b,c,d). Fig. 3 gives an example for each stage for better understanding.

(a) Iterate all lines L_i for the generated superset and search for lines adjacent to each L_i's **Endpoint** (not their starting point). The lines found are called successors of L_i.

(b) For each successor L_j build a poly-line P_{ij} consisting of L_i and L_j. Make a copy of the superset $\mathbb{L}$ containing all lines dedicated to room extraction, remove line L_i and its successors from the copy forming the new subset $\mathbb{L}_i$.

(c) Grow each poly-line by finding new successors in the reduced set $\mathbb{L}_i$ and by building new poly-lines $P_{ijk\ldots}$. The set of unused lines is reduced for each split ($\mathbb{L}_{ij\ldots}$).

(d) The propagation of a poly-line is complete when the poly-line forms a closed polygon with its starting point. This polygon is inserted into the list of room candidates. If a polygon with another point is found, it is discarded. The propagation is canceled if no adjacent lines can be found. This criterion leads to robust termination because the set of possible lines is reduced with each step.

When the algorithm has finished iterating over all lines, some polygons have been detected multiple times and are redundant. This is due to the fact that several lines belonging to the same room all lead to the generation of a duplicated room. In addition, some falsely detected rooms can occur. For instance, closed polygons depicting a large pillar can be detected as a room and overlapping polygons can be created due to erroneously parsed annotation lines. We therefore use the following post-processing rules to clean the list from errors and complete the map's raw data.

- Delete all polygons with surfaces below a predefined threshold a_{th} (usually $1 - 2m^2$).
- Delete all polygons that have no edge superimposing a door or a stair (every room has an entry).
- Delete all polygons intersecting two other polygons.
- Delete all polygons that are inside bigger ones.

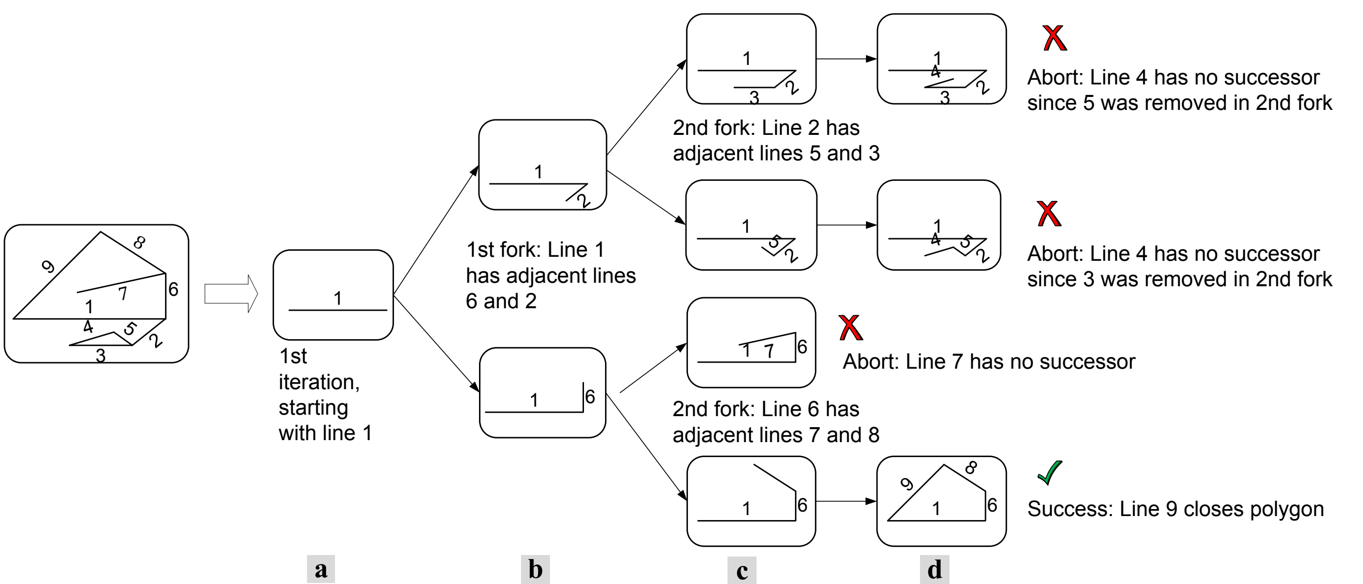

Fig. 3. Schematic Description of the Room Extraction Process

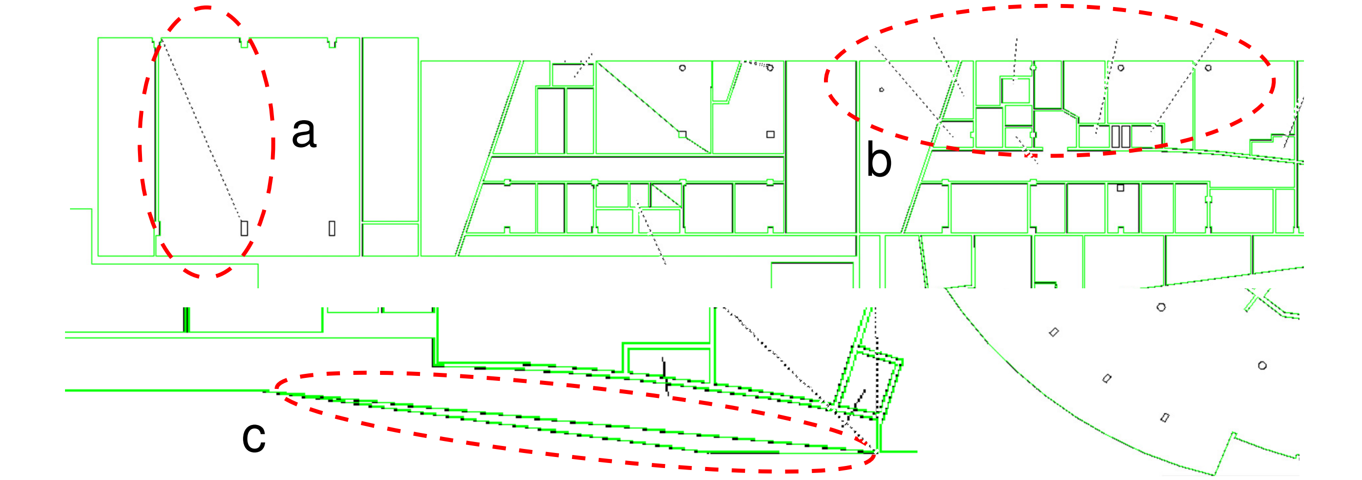

Fig. 4. Extraction Examples: Region a) and b) show corrected source errors, region c) gives an example for the algorithm's limitations

Fig. 4 shows a section of an analyzed floor containing several problematic regions. The extracted rooms are shown in green continuous lines while the raw lines are depicted using dotted black lines. Region (a) and region (b) are examples for the error correction functionality of the proposed algorithm. Although several intersecting lines perturb the plan, the rooms are correctly identified. Region (c), on the other hand, is an example for the algorithm's limitations. It is unclear, which of the lower lines depicts the room's boundary.

D. Extracting Stairs and Elevators

Straight stairs are usually depicted as rectangles partitioned by several equidistant parallel lines, sometimes superimposed by an arrow that indicates the stairs' direction. Turns or corkscrew stairs are depicted as equally partitioned arcs (see Fig. 5). By determining the bounding shape of the partitioning lines the outlines of a stair can be determined. Elevators are often drawn in the same layer as stairs and are visualized as barred or crossed box. If possible outlines for stairs and elevators are found on one floor, both adjacent floors are parsed for stairs and elevators with similar outlines at similar places. If the floor contains a matching region, this region is defined as portal between the two floors that contain it.

E. Parsing Results

To evaluate the average detection quality, we compare original building plans with the extracted representations. Table I lists the results for one typical university building.

Floor	Doors	Detected	Ratio	Rooms	Detected	Ratio
-1	98	98	100%	73	69	94.5%
0	80	79	97.5%	59	56	94.9%
1	76	76	100%	57	57	100%
2	73	73	100%	54	51	94.4%
3	68	68	100%	50	50	100%
4	70	70	100%	48	46	95.8%
5	50	47	94%	33	32	96.7%
Average:			98.8%			96.6%

TABLE I
EXTRACTION RESULTS BUILDING 1

The door detection algorithm is observed to be quite robust. On average, 98.8% are of the depicted doors were detected. No false positives were found. The remaining outliers are mostly caused by inaccurate sources, e.g. a door being literally drawn beside its room. The detection ratio obtained for rooms is similar on average but the algorithm is not as robust due to the different drawing alternatives. However, failures can be corrected via a user interaction interface that visualizes the differences between the raw data and the extracted information. The user can thus identify inaccuracies and remedy them with an integrated drawing GUI. The detection of stairs is up to now the most error prone part of our parser. For regular stairs good results are obtained but minor irregularities can already lead to the algorithm's failure.

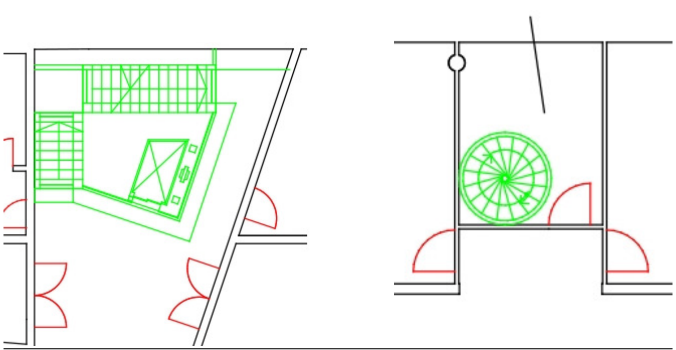

Fig. 5. Different Stair Types and Elevator

F. Creation of the 3D Topology

When the parsing process is terminated, the algorithm starts to construct the map bottom-up. First, the rooms are created. The lines of a room's outline are compared to the room's doors and both are used to create an edge vector that is later used in the filtering algorithm. When a door is inserted, the room is added to the door object, since the door will serve as gateway during the filtering process. When all rooms of one floor are parsed, the *BoundingBox* of the floor is calculated as conjunction of all rooms' outlines. Equally, a building's *BoundingBox* is determined using the floors' outlines and their altitude values. When all floors have been extracted and the according 2D topologies have been built, the separate topologies need to be merged into a connected 3D topology. The coordinate systems of the floor plans are not absolutely coherent for all plans. Therefore, it is necessary to match the separate coordinate systems with each other. For a building with a strictly vertical facade the floors' outlines can be used to match the coordinate systems. Unfortunately, not all buildings have strictly vertical facades and subterranean levels may even have a completely different outline than the upper floors of a building. We therefore additionally match specific structures like elevators to obtain coherent coordinate systems. Since an elevator traverses a building on a strictly vertical path, the outline of the elevator is always situated at the same position on each floor. Hence, we define the coordinate system of the ground floor as reference and then propagate this coordinate system throughout the other floors. Elevator outlines and identical floor outlines are used as anchor references. In the end we obtain a three dimensional semantic topology model.

IV. Semantic Topology Model

The model shown in Fig. 6 is similar to the one proposed in [9] and contains the physical entities *Building*, *Floor*, *Stairs*, *Elevator*, *Room* and *Door*. Additionally several abstract entities are introduced, namely a general container, a portal and an edge.

A. Container

The majority of the the proposed structures inherit from a generic container. Every container defines a reference point as a relation between a Cartesian coordinate system and the coordinates of this point in a global coordinate system (currently the World Geodetic System 1984 -WGS84). Thus every position in the local coordinate system can be matched to the global coordinate system. Further every container incorporates information about its extension above and below surface relative to its reference point and stores information about its outline. Hence, it is easy to determine whether the container includes a given position or not.

B. Building

A building serves as container for one or several floors and their interconnecting portals. An approximation of the buildings 3D outline is obtained by interconnecting the outlines of all included floors.

C. Floor

A floor serves as container for all structures on one level. Its geometric outlines are defined by a bounding polygon, the floor's height and the height above ground relative to the ground floor of the incorporating building. Further a floor incorporates an ID as level identifier.

D. Room

A room is stored as a polygon. Each edge of this polygon can be either traversable (a door or a portal entry) or not traversable

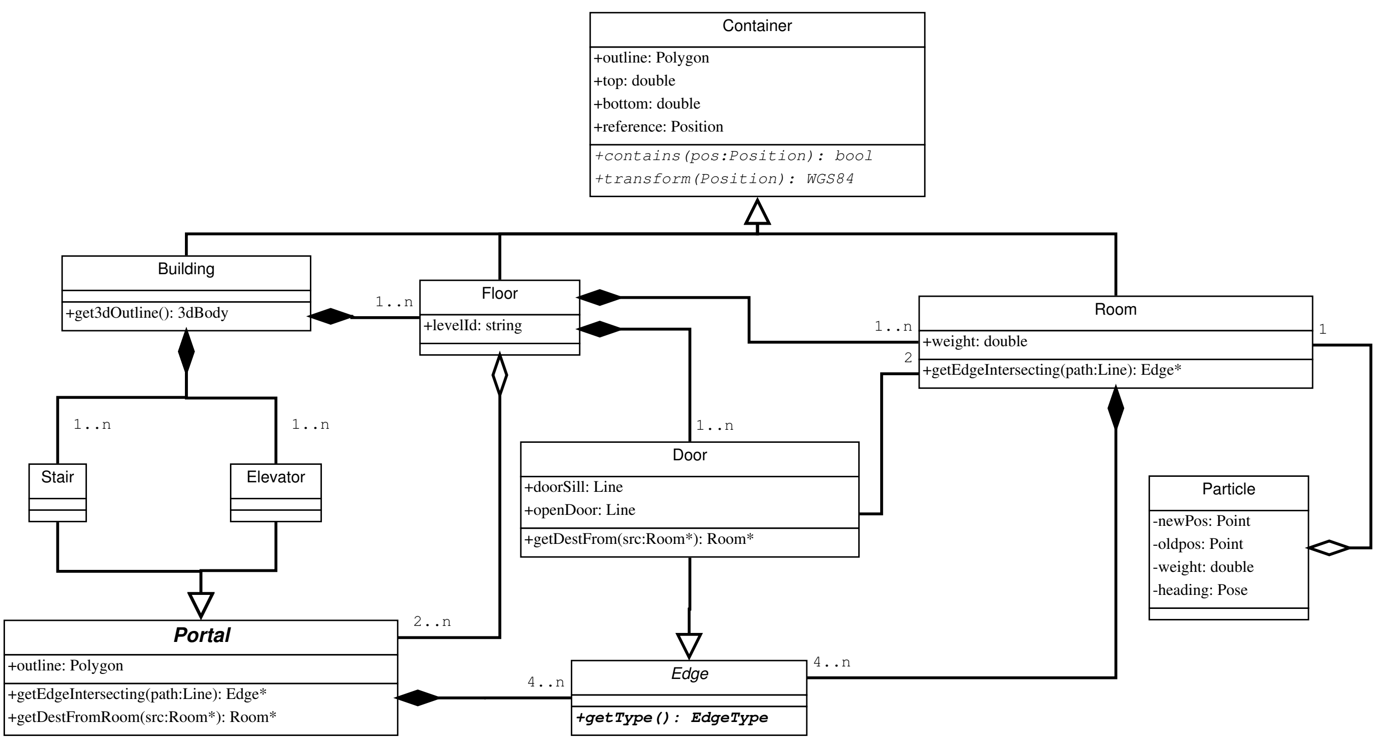

Fig. 6. UML-Model of the Semantic Map Including Particle Representation

(a wall). Every Room provides a test function to determine whether a path from a position inside the room leads to a position outside the room. If this is the case, the traversed edge indicates whether the path is plausible or not. Additionally, a weight attribute can be used to assign a usage indicator to the room which can be used for advanced filtering techniques (see also V).

E. Edge

An edge is the elementary segment of each polygon. It can be either a non traversable wall (default) or it can be extended to model the passage from one room or portal into another room.

F. Door

A door is a traversable edge that interconnects rooms on the same floor. Each door stores its door sill, the open door and associations to the rooms interconnected by the door. It further provides direct access to the destination of a path traversing the door.

G. Portal

A portal is the second type of container that can serve as physical location. It models a connection between two or several floors. Similar to a room, it has an outline and provides a test function to determine whether a path from a position inside the portal to a position outside the portal is plausible.

H. Stairs

A stair object is a special portal that interconnects two floors with each other. Stairs are modeled as the two dimensional projection of the real stairs. This projection, which is approximated as a polygon, contains traversable edges and rigid edges, similar to a room. Additionally it includes an access function that provides the destination of a given path similar to a door. The difference is the location of the associated rooms. They must lie on adjacent floors.

I. Elevator

The structure of the elevator class resembles to the stairs class. The main difference is the preservation of three dimensional information and the number of associated rooms. The structural information of the elevator is encoded as a polygon which contains up to two traversable edges (the elevator's doors). When an edge is traversed leaving the elevator, the height of its current position is used to determine the destination room.

V. Filtering Algorithm

For localization and tracking, we implement a map-matching particle filter, which basically consists of three well-known phases: re-sampling, propagation and correction (see e.g., [14] or [15] for details on particle filters). Similar to the approaches used in [2] and [16], we use measurements from a 3D accelerometer, 2 2D gyrometers and a 3D magnetic field sensor as input for the propagation phase of the filter and apply the map constraints in the correction phase. The detailed structure of the filter's re-sampling and propagation phase is out of scope of this work and does not differ fundamentally from the cited works. What differs is the retrieval of legal paths necessary for every particle during the filters correction phase. As depicted in Fig. 6, we define a particle incorporating its old and new position, its heading, its weight and a pointer to its room. Using this type and the proposed map structure the filter's correction algorithm can be formulated as follows:

Map-based particle filter correction phase

```
foreach(p in particleSet){
  path = p.newPos - p.oldPos;
  *edge = p.room->getEdgeIntersecting(path);
  if (edge==null){
    p.weight = p.weight;
  }else if (edge->getType()== Wall){
    p.weight = 0;
  } else if (edge->getType()== Door){
    *door = edge;
    p.room = door->getDestFrom(p.room);
    p.weight *= p.room->getWeight(newPos);
  }
}
```

A. Filter Performance Evaluation

To deduce an upper bound for the proposed filter's complexity it is necessary to describe the steps of the proposed filtering algorithm in more detail: In the beginning of the propagation phase each particle object contains a pointer to the room it is residing in. When a particle is propagated, an iterative containment query is launched to determine its new location. Using a variant of the so called "winding number" method described in [17] we determine whether the particle has left the room and if so which edge it has traversed. When the edge provides a legal path, the traversed edge directly indicates the new room and the particle weight is readjusted according to the weight of this new room. Hence, no search iterations across adjacent entities are necessary and the complexity of the described algorithm only depends on the edge number of the polygon a particle is residing in. Thus an upper bound for the overall complexity of the filtering algorithm can be given as $O(N_{part} * max(n_{poly_i}))|i = [1..rooms]$ where N_{part} is the number of particles currently used and n_{poly_i} indicates the number of edges of polygon $poly_i$. Most room-outlines consist of relatively few edges and hence n_{poly_i} is relatively small. The number of particles N_{part} depends on the filter's application. In tracking-scenarios, less than 1,000 particles often suffice to model the position's uncertainty. On the other hand in a localization-scenario a significantly higher particle number is necessary. As a testing scenario for the latter case, we uniformly distributed 45,400 particles on an office-floor that covers $1{,}050m^2$ to perform localization. Run on a 1.6 GHz Mobile-CPU, the entire particle filtering process, including re-sampling, propagation and correction, terminated in less than $150ms$. Hence it is significantly faster than required to provide real-time position updates for pedestrian movement. This will obviously also hold true for the tracking scenario where less particles are required.

VI. Conclusion and Future Work

In this paper paper we have presented a novel parser that automatically extracts semantic building information from architectural CAD floor plans. Since these plans are available for the majority of public and official buildings, our work provides a practical low-cost solution to create topological maps of large-scale urban environments. The created map-model can be used for efficient particle-filter-based map-matching algorithms. Our performance evaluation with a standard implementation has proven that real-time indoor tracking and localization, based on the proposed model, are easily achievable. Our next steps are to investigate the possibilities of weighted maps that incorporate different likelihoods for different rooms. Those likelihood values could, for instance, reflect the usage of certain rooms or structural changes not incorporated in the often slightly dated CAD-files. Finally we plan to extend the map-model towards a navigable map.

References

[1] Widyawan, M. Klepal, and D. Pesch, "A bayesian approach for rf-based indoor localisation," in *4th International Symposium on Wireless Communication Systems (ISWCS)*, 2007.

[2] O. Woodman and R. Harle, "Pedestrian localisation for indoor environments," in *ACM International Conference on Ubiquitous Computing (UbiComp)*, 2008.

[3] S. Beauregard, Widyawan, and M. Klepal, "Indoor PDR performance enhancement using minimal map information and particle filters," in *IEEE/ION Position Location and Navigation System Conference (PLANS)*, 2008.

[4] L. Wirola, T. Laine, and J. Syrjä Andrinne, "Mass-market requirements for indoor positioning and indoor navigation," in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2010.

[5] *The Industry Foundation Classes (IFC) specification*, Buildingsmart International Std., Rev. IFC2x3-TC1, 2007.

[6] *OpenGIS® City Geography Markup Language (CityGML) Encoding Standard*, Open Geospatial Consortium Inc. Std., Rev. Version: 1.0.0, 2008.

[7] S. Z. Annet Groneman, "Toposcopy: a modeling tool for citygml," in *Proceedings of the GSDI 11 World Conference*, 2009.

[8] T. Becker and M. Butz-Bonczyk, "Interpretation von 2D-CAD Plänen zur automatisierten Erstellung eines CityGML Gebäudemodells," in *Entwicklerforum Geoinformationstechnik 2007 - Junge Wissenschaftler forschen*, 2007.

[9] M. Kessel, P. Ruppel, and F. Gschwandtner, "BIGML: A location model with individual waypoint graphs for indoor location-based services," *PIK - Praxis der Informationsverarbeitung und Kommunikation*, vol. 13, p. 260–266, 2010.

[10] A. Rice and O. Woodman, "Crowd-sourcing world models with openroommap," in *8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2010.

[11] P. Robertson, M. Angermann, and B. Krach, "Simultaneous localization and mapping for pedestrians using only foot-mounted inertial sensors," in *ACM International Conference on Ubiquitous Computing (UbiComp)*, Wessling, Germany, 2009.

[12] Autodesk, *DXF Reference*, AutoCAD® Std., Rev. v.u.26.1.01, 2012.

[13] A. K. L. Miu, "Design and implementation of an indoor mobile navigation system," Ph.D. dissertation, Massachusetts Institute of Technology, 2002.

[14] I. M. Rekleitis, "A particle filter tutorial for mobile robot localization," Centre for Intelligent Machines, McGill University, Tech. Rep. TR-CIM-04-02, 2004.

[15] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, 2004.

[16] B. Krach and P. Robertson, "Integration of foot-mounted inertial sensors into a bayesian location estimation framework," in *5th Workshop on Positioning, Navigation and Communication, (WPNC)*, 2008.

[17] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker, "A characterization of ten hidden-surface algorithms," *ACM Computing Surveys*, vol. 6, no. 1, pp. 1–55, 1974.