TECHNISCHE UNIVERSITÄT DRESDEN

PROFESSUR FÜR MENSCH-
COMPUTER INTERAKTION

Dokumentation des Forschungsprojekts "Loomo Karten"

Michaela Loumová

Dresden, 6. August 2018
Betreuer: M.Sc. David Gollasch

Inhaltsverzeichnis

1 Theoretische Grundlagen

Im Kapitel Theoretische Grundlagen wird eine Übersicht über die Grundlagen zu den Themen Fluchtpläne, Vektorisierung und möglichen Repräsentationen einer Umgebung gegeben.

1.1 Fluchtpläne

Da sich Fluchtpläne in allen öffentlichen Gebäuden befinden und der Norm DIN ISO 23601 entsprechen, eignen sie sich gut als universelles Kartenmaterial für das Kennenlernen einer Umgebung für Loomo. Die Norm beschreibt folgende Gestaltungsregeln:

> a) der genaue Standort des Betrachters muss angegeben werden,
> b) die Fluchtpläne müssen farbig angelegt sein,
> c) einen von der Größe der baulichen Anlage abhängiger Maßstab: 1:100, 1:250, 1:350 haben,
> d) innerhalb der baulichen Anlage einheitlich im Layout sein,
> e) gut sichtbar und lesbar sein,
> f) Mindestwert für das Lichtspektrum Ra ist größer gleich 40,
> g) Hintergrund der Flucht- und Rettungspläne sind weiß oder nachleuchtend weiß,
> h) Mindestgröße beträgt 297 mm x 420 mm (A3),
> i) die Fluthcpläne müssen immer auf dem neuesten Stand sein,
> j) Ausrichtung des angebrachten Planes muss aus der Sicht des Betrachters lagerichtig sein,
> k) es sollen Sicherheitszeichen verwendet weden, die denen in der baulichen Anlage und der ISO 7010 entsprechen,
> l) müssen eine Legende haben,
> m) haben eine Standardüberschrift: „Flucht- und Rettungsplan“,
> n) enthalten eine Darstellung von Sammelstellen [V09]

Dieser Auszug aus der Norm ist ebenfalls relevant:

> Der Maßstab für den Flucht- und Rettungsplan ist abhängig von der Größe der baulichen Anlage, des darzustellenden Detaillierungsgrades und des vorgesehenen Anbringungsortes des Flucht- und Rettungsplanes. Die folgenden Mindestmaßstäbe sind anzuwenden:
> - 1:250 für große bauliche Anlagen;
> - 1:100 für kleine und mittlere bauliche Anlagen;
> - 1:350 für Pläne, die in einzelnen Räumen angebracht werden.
>
> Für detaillierte Elemente wie Treppen oder Flure darf ein größerer Maßstab gewählt werden, um diese Elemente hervorzuheben oder um die Platzierung von Sicherheitszeichen auf dem Flucht- und Rettungsplan zu ermöglichen. Bei mehreren Flucht- und Rettungsplänen für dieselbe bauliche Anlage sollte für alle Pläne der gleiche Maßstab gewählt werden. Für spezielle Bereiche der baulichen Anlage, z. B. Parkplätze oder technische Bereiche, können andere Maßstäbe gewählt werden, um freie Flächen zu berücksichtigen. [V09]

Die Fluchtpläne entsprechen teilweise maßgetreu den Bauplänen, wichtige Elemente dürfen aber vergrößert und irrelevante rausgenommen werden. Leider ist in der Norm nicht definiert, was ein relevanter Inhalt ist. Im Rahmen einer Studie wurden in [Tsc13] 120 Fluchtpläne aus 41 verschiedenen Gebäuden analysiert. Dabei wurde festgestellt, dass es viele verschiedene Notationen im Gebrauch gibt. In jedem Fluchtplan wurden andere Elemente als irrelevanter Inhalt rausgenommen. Außerdem wurden drei verschieden Arten identifiziert, um eine Tür darzustellen. Die vorhandene Norm fixiert viele Charakteristiken eines Fluchtplans, lässt aber auch viel Freiraum bei der Darstellung. [Tsc13]

1.2 Tools zur Vektorisierung

Im Rahmen des Praktikums wurden mehrere Tools zur Vektorisierung der Bildern eines Fluchtplans verwendet: vectormagic, pngtosvg.com, drawsvg.org oder vectorizer.io. Um sich die generierten SVG-

Dateien anzuschauen und zu editieren, wurde Inkscape benutzt. Mit einem Originalfluchtplan aus dem Internet (kein Foto) hat das gut geklappt, alle Linien und Räume wurden erkannt. Ähnliche Objekte wurden einem Layer hinzugefügt.

Ein Foto von einem Fluchtplan wurde zuerst bearbeitet: höherer Kontrast und Anpassung des Weißabgleichs. Trotzdem wurde der Hindergrund nicht richtig weiß und die Schatten wurden mitvektorisiert. Daher probierte ich es noch mit einem anderen Tool: Vectormagic. Dieses arbeitet online oder auch als Desktopanwendung. Damit ließ sich der Fluchtplan vektorisieren und auch bearbeiten.

1.3 Repräsentation der Umgebung

Repräsentation der Umgebug soll laut der Aufgabenstellung auf Graphen basieren.

1.3.1 Diskrete und kontinuierliche Repräsentation

Laut [Cor14] gibt es zwei Ansätze, um die Umgebung zu repräsentieren: diskret und kontinuierlich.

In der diskreten Approximierung wird der Raum in mehrere gleich oder unterschiedlich große Teile dividiert, die beispielsweise die Räume darstellen können. In diesem Ansatz lässt sich die Karte als Graph anzeigen, indem die Teile einen Knoten und die Verbindungen die Kanten darstellen. Dieser Graph kann als eine Adjazenz- oder Inzidenzmatrix gespeichert werden.

Im kontinuierlichen Ansatz werden die inneren Hindernisse und die äußeren Grenzen definiert. Die Pfade werden als Zahlensequenzen kodiert.

Da die Aufgabenstellung besagt, dass die Karten als Graphen gespeichert werden sollen, wird bei dieser Aufgabe der diskrete Ansatz eingesetzt.

1.3.2 Roadmaps und Cell-Decomposition

Der Ansatz der Roadmaps basiert auf dem Erfassen der Konnektivität vom Freiraum des Roboters in einem Netzwerk von eindimensionalen Kurven. [Lat12] nennt vier Methoden, um Roadmaps zu realisieren: Visibility-Graph, Voronoi-Diagramm, Freeway-Net oder Silhouette.

Visibility-Graph war einer der ersten Ansätzen, um Pfade zu repräsentieren und beschreibt einen zweidimensionalen Konfigurationsraum mit polygonalen Regionen, die Hindernisse darstellen. Es handelt sich um einen ungerichteten Graphen und die Links sind Teile von Linien, die zwei Knoten verbinden und durch kein Hindernis durchlaufen.

Der zweite übliche Ansatz ist neben den Roadmaps die Cell-Decomposition, die den freien Raum des Roboters in einfache Regionen unterteilt (Zellen). Ein ungerichteter Graph repräsentiert die Adjazenzmatrix der Verbindungen und wird Konnektivitätsgraph genannt. Laut [Lat12] ist der zweite Ansatz einfacher zu implementieren und außerdem garantiert, dass der Algorithmus einen Pfad findet, wenn es einen gibt, sonst gibt er einen Fehler zurück.

1.4 Formate zur Repäsentation einer Umgebung

In diesem Abschnitt werden die bekanntesten Methoden zum Speichern und zur Darstellung von Graphen beschrieben.

1.4.1 SVG

Scalable Vector Graphics (SVG) ist eine XML Grammatik, die eine zweidimensionale Vektorgrafik beschreibt. Sie gilt als ein Formatstandard für Vektorgrafiken, weswegen ich sie auch für Loomo Karten einsetzen würde. [FJJ00] Für diese Arbeit ist das Element *Path*, sowie die Gruppierung der Elemente relevant. Ein *Path* wird durch den Tag <*path*> definiert und beschreibt jegliche Formen mit Hilfe von absoluten oder relativen Punkten, die mit verschiedenen Kommandos zu weiteren Punkten geführt werden. Es ist möglich Linien, aber auch Kurven zu definieren. Diese Pfad-Information wird unter dem Attribut *d* gespeichert. Es besteht die Möglichkeit, einem SVG *Path* das Attribut *id* hinzuzufügen, was beispielsweise benutzt werden kann, um die Namen der Räume direkt in der SVG-Datei zu speichern.

1.4.2 GraphML

GraphML basiert ebenfalls auf XML und ist ein Format für die Beschreibung von Graphen. Neben den bereits vorhandenen Tags wie <*node*>, <*edge*>, <*position*> oder <*size*> hat der Benutzer auch die Möglichkeit eigene Erweiterungen hinzuzufügen. Diese Eigenschaft ist nützlich und könnte für Loomo Karten eingesetzt werden. [BP04]

1.4.3 GML

Geography Markup Language (GML) ist eine XML-Grammatik für das Modellieren und Speichern von geographischen Informationen. Der Benutzer hat die Möglichkeit eigene Tags zu definieren. GML-Daten lassen sich durch Extensible Stylesheet Language Transformations (XSLT) zu Graphikformaten transformieren, beispielsweise SVG, um eine Karte anzuzeigen. [PZ04] GML könnte ebenfalls in Betracht für Loomo gezogen werden.

1.5 Related Work

Die Idee, aus einem Fluchtplan eine Karte zu erstellen, ist nicht neu und wurde bereits in den letzten Jahren bearbeitet.

[Tsc13] beschäftigt sich damit, wie Indoor Karten aus öffentlichen Fluchtplänen extrahiert werden können. In dieser Arbeit wurde ein Prototyp gebaut, der selbstgemachte Handy-Fotos mithilfe von Computer Vision Algorithmen in eine OSM-XML-Datei transformiert. Es wurde gezeigt, dass der Prototyp unter unterschiedlichen Bedingungen funktioniert.

Laut [Tsc13] wurden in den ersten Ansätzen der Digitalisierung der Fluchtpläne in [PHSO10], [PHF11] und [HFPK11] folgende Schritte durchgeführt, um aus einem Bild von einem Fluchtplan eine Karte zu erzeugen: Das Bild wurde zuerst mithilfe von einfachem Thresholding binariziert, dann mit Connected-Component-Analysis in verschiedene Regionen geteilt. Um die Symbole zu identifizieren, wurden die entstandenen Regionen klassifiziert und Template-Matching angewandt. Nach der Identifizierung und dem Entfernen aller Symbole werden die restlichen Umgrenzungen skelettiert und die Endpunkte identifiziert. Die Connected-Component-Analysis wird nochmal durchgeführt und die Endregionen werden mit Thresholding nach der Größe klassifiziert.

In [PHSO10] werden zuerst die Eckpunkte vom Gebäude erkannt, um ein Koordinatensystem zu erstellen. [LLKM97] präsentiert ein komplettes System zur Analyse der Fluchtpläne. Die grundlegenden Elemente werden mit Erkennungsalgorithmen identifiziert und diese Methode wird mit einem menschlichen Feedback kombiniert.

2 Workflow zur Digitalisierung von Fluchtplänen

In diesem Kapitel wird die Konzeption eines Workflows zur Digitalisierung von Fluchtplänen vorgestellt. Es ist nicht vorgesehen, möglichst viele Fluchtpläne digitalisieren zu lassen und somit den Vorgang voll automatisiert durchzuführen, deswegen wird auf manuelle Zwischenschritte nicht verzichtet. Wie im vorherigen Kapitel erwähnt, wurden auch automatisierte Systeme mit einem menschlichen Feedback kombiniert, wodurch die Genauigkeit der Ergebnisse verbessert werden konnte.

Der vorgeschlagene Bearbeitungsprozess dauert je nach Erfahrung des Bearbeitenden und Komplexität des Fluchtplans ungefähr eine bis zwei Stunden.

Ein Workflow zur Digitalisierung von Fluchtplänen wird in folgenden Schritten beschrieben:

2.1 Foto machen

Es wird davon ausgegangen, dass das Foto mit einer durchschnittlichen Handykamera gemacht wird. Die Kamera soll sich dabei direkt vor dem Fluchtplan befinden, um Verzerrungen zu minimieren. Der ganze Fluchtplan befindet sich auf dem Foto und gleichzeitig ist er das größte Objekt darauf.

2.2 Foto bearbeiten

Das Foto muss zuerst manuell bearbeitet werden, um es auf die Vektorisierung optimal vorzubereiten. Es ist notwendig je nach Belichtung des Bildes den Kontrast zu erhöhen und einen Weißabgleich durchzuführen. Optimaler Weise kann auch das Rauschen teilweise entfernt werden.

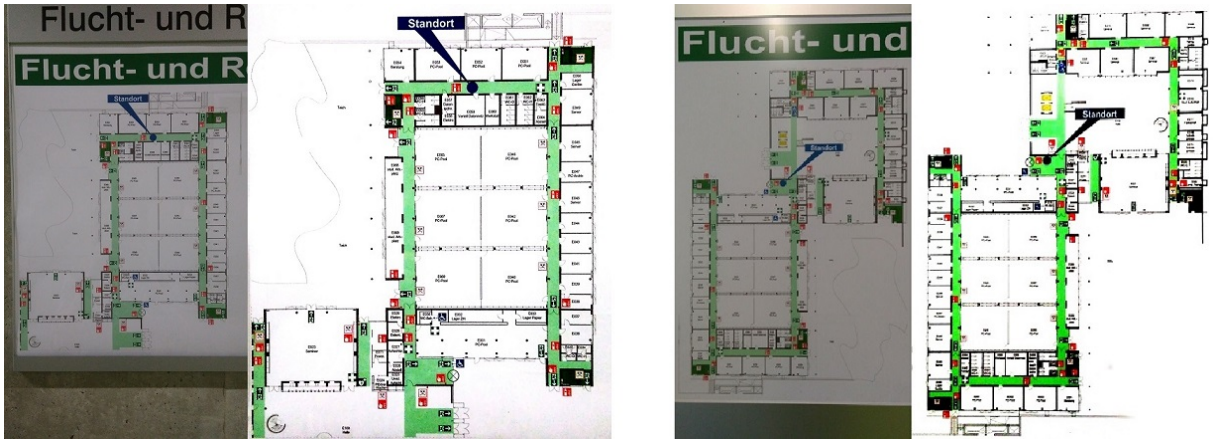

Abbildung 1: Schritt 2: Beispiel der Bearbeitung

Dabei kann ein beliebiges Bildbearbeitungstool verwendet werden, beispielsweise Photoshop oder als kostenlose Alternative Photoscape.

2.3 Foto vektorisieren und bearbeiten

Das bearbeitete Bild wird mit einem der vorher genannten Vektorisierungstools vektorisiert. Dabei ist ein Tool auszuwählen, das einen guten Detailierungsgrad und gleichzeitig die Möglichkeit bietet, das vektorisierte Bild zu bearbeiten, was der Vorteil von drawsvg.org ist. Das online Tool pngtosvg.com bietet auch zusätzlich die Möglichkeit, die Anzahl der vorhandenen Farben auszuwählen, was signifikant hilft, Rauschen auf dem weißen Hintergrund zu minimieren. Mit Vectorizer.io ist es möglich, die Farben zu Farbgruppen einzuteilen, was ebenfalls benötigt wird, damit durch die Anwendung die Farben richtig erkannt werden können. Die Bearbeitung kann in folgenden Schritten erfolgen:

Abbildung 2: Schritt 3: Auswahl von Anzahl der Farben in pngtosvg.com

1. Rauschen entfernen
2. Irrelevante grüne und rote Symbole löschen (Feuerlöscher, Brandmelder...)
3. Sicherstellen, dass nur Wände, Türen, Treppen und Aufzüge übrig bleiben
4. Hintergrund (Umgebung des Gebäudes) pink markieren
5. Alle Räume weiß lassen/markieren
6. Alle Flure grün markieren (nicht nur Fluchtflure)
7. Alle Türen rot markieren
8. Alle Aufzüge blau markieren

Das nachbearbeitete Bild wird als SVG-Graphik gespeichert.

2.4 Graphik auf Loomo übertragen

Die SVG-Graphik wird auf Loomo übertragen und später als Eingabe bei der App erwartet.

2.5 Transformation der Graphik in eine interne Karte

Ein Algorithmus wandelt die Informationen aus der SVG-Graphik in eine interne Datenstruktur um, die die Karte darstellt.

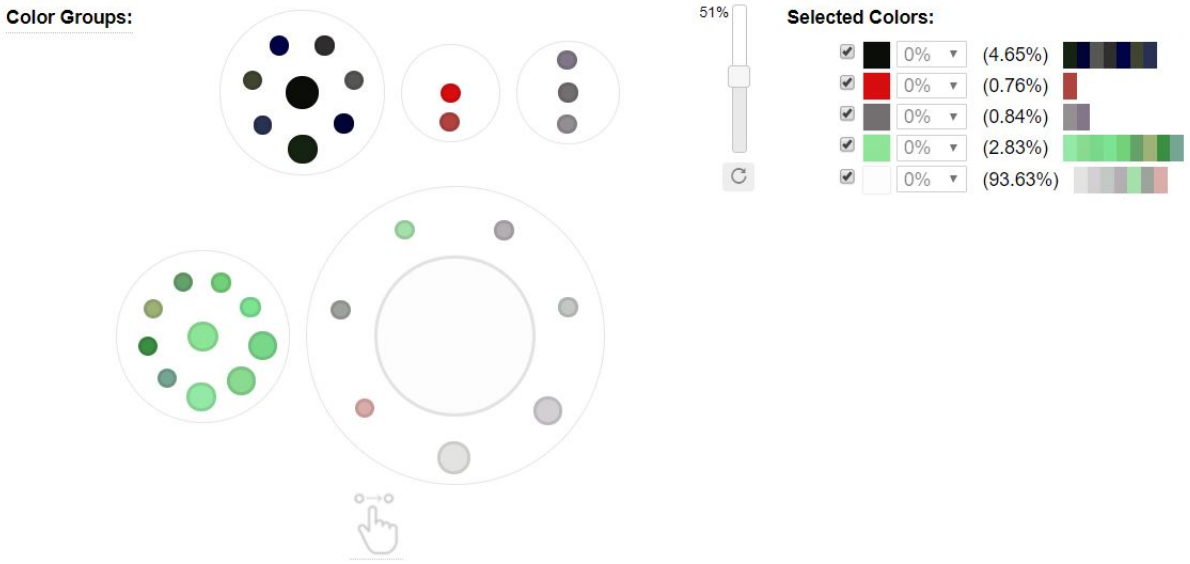

Abbildung 3: Schritt 3: Gruppieren von Farben in vectorizer.io

3 Praktische Umsetzung

In diesem Kapitel werden die einzelnen Schritte der Implementierung dokumentiert, die Funktionsweise erklärt und der Verlauf beschrieben.

3.1 Rahmenbedingungen

Das Szenario soll in einer einzelnen Android App bereitgestellt werden. Die API-Version der App für Loomo ist 22 (Android 5.1 Lollipop).

3.2 Herangehensweise

Am Anfang des Projektes wurde ein Zeitplan erstellt, in dem ein Zeitraum für die Implementierung vorgesehen wurde. In den ersten Treffen wurde die grobe algorithmische Umsetzung entworfen und diskutiert. Die Implementierung erfolgte in Sprints, die zwei Wochen dauerten. Nach jedem Sprint wurde der Fortschritt dem Betreuer vorgestellt und die weiteren Schritte für den nächsten Sprint besprochen.

3.3 Die ersten Ansätze der Umsetzung

Bei der Implementierung wurden zwei verschiedene Ansätze umgesetzt und verglichen.

3.3.1 Ansatz Rastergraphik

Der Ansatz, eine Rastergraphik zu benutzen, besteht darin, die Pixel eines Bildes durchzugehen und den Farbwert auszulesen. In diesem Projekt wurde zuerst dieser Ansatz umgesetzt, da er einfacher zu implementieren war. Leider wurde festgestellt, dass er ungenaue Ergebnisse liefern kann. Je nach Größe des Bildes kann es zum Verlust der Daten kommen, der vermieden werden sollte. Nach einer Konsultation mit dem Betreuer des Projektes wurde entschieden den Ansatz Vektorgraphik zu implementieren.

3.3.2 Ansatz Vektorgraphik

In einer SVG-Vektorgraphik werden gleichfarbige Punkte einem sogenannten Layer zugeordnet. Dieser Ansatz besteht darin, die einzelnen Punkte im beliebigen Maßstab durchzugehen und zu überprüfen, zu

welchem Layer sie gehören. Damit kann erkannt werden, zu welchen Objekten (Raum, Tür) diese Punkte gehören. Leider war dieser Ansatz nicht so einfach umzusetzen und es ergaben sich mehrere Herausforderungen, die bewältigt werden mussten.

Die erste Herausforderung war, die SVG-Pfade, die als Strings rausgelesen werden, in Android zu benutzen. Das habe ich gelöst, indem diese Strings zu Android Paths transformiert werden.

Die weitere Herausforderung war, zu überprüfen, ob sich ein Punkt innerhalb eines Pfades befindet. Dafür habe ich den Test nach Jordan implementiert, der überprüft, ob sich ein Punkt in einem Polygon befindet. Leider stellen aber SVG und Android Paths keine Polygone dar, da sie unter Anderem auch Kurven beinhalten. Damit wir aus einem Pfad ein Polygon erhalten, muss der Pfad approximiert werden. Ein Ansatz wäre, diesen Pfad entlangzulaufen und nach einer Längeneinheit den Punkt zu speichern. Dieser Ansatz würde aber beispielsweise bei langen Linien zu viele Punkte erzeugen. Daher habe ich mich entschieden, nach einem weiteren Ansatz zu suchen. Die Suche war erfolgreich bei der Funktion *approximate*, die von Android zur Verfügung gestellt wird. Mit Hilfe dieser Funktion wird ein Pfad zu einem Array von Punkten transformiert, indem Linien gleich definiert bleiben und nur die Kurven approximiert werden. Damit werden alle Pfade zu Polygonen transformiert. Beim Durchgehen aller Punkte wird dabei überprüft, in welchen Polygonen dieser Punkt liegt und dementsprechend zu welchem Layer und damit zu welcher Kategorie dieser Punkt gehört.

3.4 Probleme im Verlauf der Implementierung

Im Verlauf der Implementierung sind Probleme bei der Performance der App aufgetreten. Die Berechnung einer Adjazenzliste für eine SVG-Graphik sollte voraussichtlich mehrere Stunden in Anspruch nehmen. Die App wurde anschließend auf mehreren Laptops mit mehreren Android Emulatoren oder Smartphones getestet, es konnte aber kein Unterschied in der Performance festgestellt werden. Das algorithmische Vorgehen wurde diskutiert und durchgegangen, wobei kein Fehler entdeckt werden konnte. Da kein Fehler auch innerhalb den nächsten zwei Wochen gefunden wurde, wurde vom Betreuer ein anderer Ansatz zur Berechnung der Adjazenzliste vorgeschlagen, und zwar nicht alle Punkte der Graphik durchzugehen und daraus eine Adjazenzliste zu berechnen, sondern erstmal nur alle Räume, Flure und Türen. Dadurch können viele irrelevante Punkte gleich am Anfang ausgelassen werden und die Zeit für die Berechnung verringert sich.

Nach der Umsetzung dieses Ansatzes wurde beim Testen des Algorithmus herausgefunden, dass sich ein Fehler in der Implementierung der Strahl-Methode nach Wikipedia befindet und diese Methode in Einzelfällen falsche Ergebnisse liefert. Danach wurde ein anderer Algorithmus genutzt, der auf einer ähnlichen Methode basiert. Neben den richtigen Ergebnissen wurde auch die Performance deutlich besser.

3.5 Ursprüngliche Funktionsweise

Die App erwartet als Input eine SVG-Datei mit dem Fluchtplan. Diese Datei wurde eingelesen, alle Punkte wurden durchgegangen und es wurde überprüft, welche Farbe dieser Punkt hat, womit erkannt wurde, zu welcher Kategorie (Raum, Tür) dieser Punkt gehört. Dabei wurde eine Matrix erstellt, wo die Indices x und y den Koordinaten des Punktes entsprechen und der Wert die Kategorie beschreibt, zu der der Punkt gehört. In dieser Matrix wurde erkannt, welche Nachbarn ein Punkt hat und damit auch, ob es da einen Durchgang gibt. Damit wird ein Graph erstellt, der in Form einer Adjzenzliste gespeichert wird. Diese Adjazenzliste wird als Matrix gespeichert, wo x und y die Koordinaten eines Punktes sind und der Wert ist ein Array von erreichbaren Nachbarn. Dabei werden 8 Nachbarn um den Punkt herum betrachtet.

3.6 Optimierte Funktionsweise

Die App erwartet als Input eine SVG-Datei mit dem Fluchtplan und diese Datei wird eingelesen. Es werden nicht alle Punkte durchgegangen, sondern einzelne Räume. In jedem Raum-Pfad gibt es ein Startpunkt. Von diesem Startpunkt wird ausgegangen und nacheinander bei allen Nachbarn überprüft, ob sie sich auch in diesem Raum befinden. Dieser Algorithmus basiert auf der Breitensuche. Nachdem alle Punkte gefunden wurden, die sich in diesem Raum befinden, werden alle Daten in die Datenbank reingeschrieben - der Raum als ein neuer Subgraph, alle Punkte im Raum als Nodes und die Verbindungen zwischen benachbarten Punkten als Edges.

3.7 Format

Wie bereits in diesem Kapitel beschrieben wurde, wird der endliche ungerichtete Graph, der das Gebäude beschreibt, in Form von Punkten und Kanten in die SQLite Datenbank reingeschrieben. Eine Adjazenzliste kann in der Datenbank simuliert werden, indem man ein SQL Query an die Tabelle mit Kanten stellt, der alle Einträge zu einem bestimmten Ausgangspunkt liefert.

3.8 Graphische Nutzerschnittstelle

Für die App wurde eine einfache graphische Nutzerschnittstelle entwickelt, um die Ergebnisse der Anwendung zu präsentieren. Die Oberfläche wurde nur zum Zweck des Testens und der Präsentation der Ergebnisse entwickelt und deswegen wird dabei nicht auf das Desgin oder die Bedienungsfreundlichkeit geachtet. Die Oberfläche besteht aus drei Activities - MainActivity, MapActivity und FileChooserActivity. MainActivity ist die erste Activity, die der Benutzer zu sehen bekommt und übernimmt alle grundlegenen Eingaben. Falls der Benutzer eine SVG Datei auf dem lokalen Speicher auswählen möchte, klickt er auf den Button *Select SVG* und geht damit zu der Activity FileChooserActivity über. Falls er sich entscheidet den generierten Graphen anzuschauen, klickt er auf *Show Graph*. Damit wird er auf die MapActivity weitergeleitet, wo er den Graphen als Android Drawable zu sehen bekommt. Der Benutzer hat die Möglichkeit auf zwei Punkte in der Karte zu klicken, um die Suche nach dem kurzesten Weg zu starten. Das Ergebnis wird dann auf dem Canvas gezeigt.

3.9 Backend

Die Java Klassen der Funktionalität im Hintergrund der App wurden nach deren Funktionalität in folgende Packages unterteilt: activities, database, graph.logic, local.storage, polygon.geometry und svg.processing.

3.9.1 Package activities

Dieses Package enthält Klassen, die die Hintergrundlogik der einzelnen Activites definieren.

MainActivity
MainActivity ist die Activity, die der Benutzer als erste zu sehen bekommt und über die er die App steuern kann. Dementsprechend sind in dieser Klasse alle Aufrufe von einzelnen Funktionalitäten beispielhaft implementiert.

MapActivity
Diese Klasse bekommt als Eingabe Arrays von Punkten, die die einzelnen Objekte darstellen und zeichnet diese mit den entsprechenden Farben, wie ursprünglich in der SVG-Graphik dargstellt, auf den Android Canvas. Damit kann der Benutzer überprüfen ob die Karten richtig erstellt werden konnte. Es ist möglich durch händische Auswahl zweier Punkte auf der Karte die Suchalgorithmen nach dem kürzesten Weg zu testen.

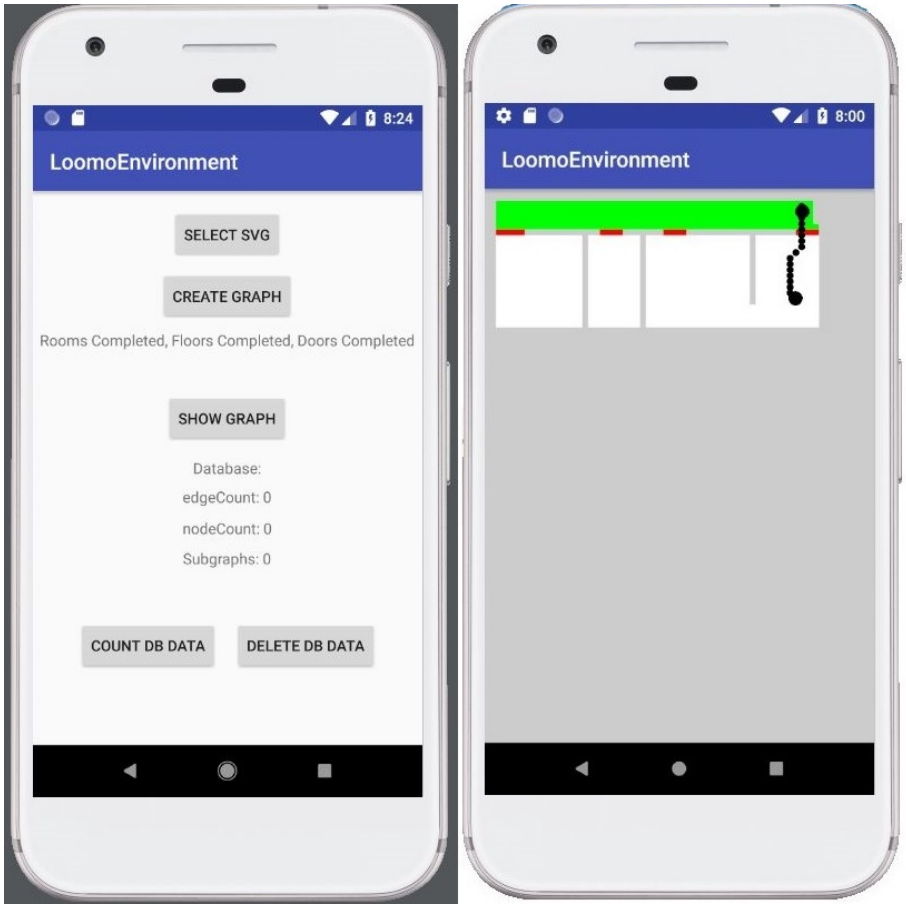

Abbildung 4: MainActivity und MapActivity mit erfolgten Suche

3.9.2 Package database

Im Package database befinden sich alle Klassen, die relevant für die Arbeit mit der Datenbank sind.

Database-Klassen
Als Database werden die Klassen NodeDatabase, EdgeDatabase, SubgraphDatabase und GraphDatabase bezeichnet. Diese Klassen sind abstrakt, definieren die jeweilige Datenbank und verweisen auf die dazgehörige DaoAccess-Klasse.

DaoAccess-Klassen
Zu DaoAccess-Klassen gehören DaoAccessNode, DaoAccessEdge, DaoAccessSubgraph und DaoAccessGraph. Diese Klassen beinhalten die Funktionen, mit denen auf die Datenbank zugegriffen werden kann. DaoAccessNode besitzt die Funktionen getNodeById, getNodeByPoint, getNodesByPoint, getAll und getFirstNodeBySubgraphId. Dao AccessEdge beinhaltet die Funktionen getEdgeById und get EdgeByStartNodeId. In DaoAccessSubgraph befinden sich die Funktionen getSubgraphById und getSubgraphByName und in DaoAccessGraph ebenfalls getGraphById, getGateBySubgraphId, getGateByBothSubgraphIds sowie getGateByPortalId.

Entity-Klassen
Entity-Klassen sind Node, Edge, Subgraph und Graph Klassen. In Node werden die Attribute node_id, x, y, type und subgraph_id definiert. Edge definiert die Attribute edge_id, from_node_id, to_node_id und subgraph_id. Subgraph weißt subgraph_id, type und name als Attribute auf. In Graph werden als Eigen-

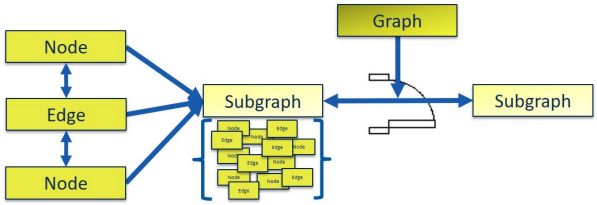

Abbildung 5: Die Datenbank nach eigener Darstellung

schaften gate_id, area_one_id, are_one_node_id, are_two_id, are_two_node_id und portal_id festgelegt.

3.9.3 Package graph.logic

Das Package graph.logic enthält Klassen, die den Kern der App beschreiben, und zwar die Funktionen, die den Graphen generieren.

Die Klasse *AreaManagement* verwaltet die gemeinsame Funktionalität von *RoomManagement* und *FloorManagement* und schreibt den dazugehörigen Subgraph in die Datenbank. Die Klassen *RoomManagement* und *FloorManagement* erben von *AreaManagement* und definieren die Funktionen, die spezifisch für Räume oder Gänge sind, einer der Unterschiede besteht beispielsweise darin, dass Räume benannt werden dürfen.

Analog dazu verwaltet die Klasse *PortalManagement* die gemeinsame Funktionalität von *DoorManagement* und *LiftManagement*.

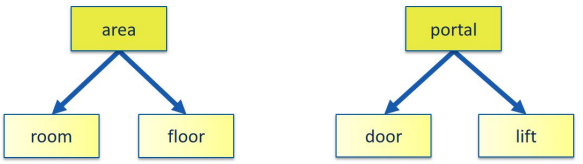

Abbildung 6: Klassen im Package graph.logic

Die Klassen *Room*, *Floor*, *Door* und *Lift* stellen die entsprechenden Objekte dar und schreiben die einzelnen Knoten und Kanten in die Datenbank.

3.9.4 Package polygon.geometry

Dieses Package beinhaltet Klassen, die sich um die Umwandlung und Approximierung von einem SVG-Pfad in der Form eines Strings in ein Android Pfad und anschließend in ein Polygon in der Form eines Arrays mit Eckpunkten dieses Polygons kümmern.

Die Klasse *DifferenceUtil* dient zu Berechnungen bezüglich der Android Paths. Da SVG-Dateien in einzelnen Layern aufgebaut sind, die nacheinander gemalt werden, entsteht ein Bedarf, diese Pfade voneinander zu subtrahieren. Diese Funktionalität ist in dieser Klasse implementiert.

Die Klassen *MyPoint*, *Line* und *Polygon* beschreiben die zur Berechnung benötigten Datenstrukturen und liefern Funktionen, die in der Transformation aus Strings in Polygone von Gebrauch sind.

Die Klasse *PathPolygonPointsConverter* dient als Parser zwischen Paths und Polygonen in Form eines Arrays von Punkten.

3.9.5 Package search

Package search enthält einfache Suchalgorithmen, um den kürzesten Pfad zwischen zwei Punkten zu finden. Dieses Package wird nicht als Teil der Anwendung wahrgenommen, da es nur zum Zweck der Präsentation der Ergebnisse implementiert wurde. Die Funktionen bilden nur einfache Suchalgorithmen ab, die nicht alle relevanten Faktoren für Loomo in Betracht ziehen. In diesem Package sind die Klassen BFS, Dijkstra und SearchPath enthalten.

Die Klasse *BFS* enthält ein Algorithmus, der auf der Breitensuche basiert und den kürzesten Pfad anhand der Anzahl der Türen berechnet. Damit können später für die wirkliche Suche nach dem kürzesten Pfad nur die relevanten Punkte geladen werden.

Die Klasse *Dijkstra* bildet den Dijkstra-Algorithmus ab. Dieser wurde nach dem Pseudocode aus Wikipedia implementiert. Hier passiert die eigentliche Suche nach dem kürzesten Weg zwischen zwei Punkten. Dafür werden die relevanten Punkte aus der Datenbank geladen und die Suche gestartet.

Die Klasse *SearchPath* kümmert sich um das Laden des Start- und Zielpunktes aus der Datenbank und das Aufrufen des Dijkstra-Algorithmus.

3.9.6 Package svg.processing

Im Package svg.processing kann man Klassen finden, die sich mit dem Rauslesen der Informationen aus der SVG-Datei beschäftigen und Hilfsfunktionen anbieten, wie Farben im Hexformat richtig zu identifizieren.

Die Klasse *ColorUtil* enthält die Funktionalität, um dem Hexformat einer Farbe die Farbe zuzuordnen. Dies wird für die Identifikation der einzelnen Elemente im Fluchtplan gebraucht. Der Autor dieser Funktionen ist Ryan Mast und die Funktionalität wurde von mir auf wenige Farben beschränkt.

Die Klasse *SvgProcessingUtil* erwartet als Eingabe einen Pfad zu der SVG-Datei. Falls keine URI angegeben wurde, wird ein internes Biespiel aus dem Asset-Ordner genommen. Danach werden die Farben der einzelnen Layer interpretiert und für die relevanten Pfade die Transformation in Polygone aufgerufen, eventuell auch die Subtraktion zwischen zwei Layern, falls benötigt. Bei Räumen werden aus der SVG-Datei ebenfalls die Labels rausgelesen.

3.9.7 Datenbank

In der App wird SQLite als Datenbank eingesetzt, was als Standard in der Android Entwicklung zählt. Um die Datenbank zu erstellen und mit Daten zu füllen, wird die Room Persistence Library benutzt.

Die Datenbank enthält vier grundlegende Tabellen: Node, Edge, Subgraph und Graph.

Node Die Tabelle *Node* enthält alle Knoten der Karte. Jeder Knoten hat eine eindeutige Id und enhält Informationen zu den x und y Koordinaten, zu dem Typ (Raum, Tür...) und zur Id des zugehörigen Subgraphs.

Edge Die Tabelle Edge enhält Informationen zu allen Kanten. Die Kanten haben eine eindeutige Id und enhalten Ids der zwei Knoten.

Subgraph Die Tabelle Subgraph enthält die Informationen zum Namen bei Räumen, zum Typ und eine eindeutige Id.

Graph Die Tabelle Graph beschreibt den Zusammenhang zwischen einzelnen Subgraphen, die durch Türen verbunden sind. Dadurch lässt sich auch eine Suche realisieren, die zuerst einen Pfad durch einzelne Räume sucht und erst danach die relevanten Knoten aus der Datenbank lädt und den kürzesten Pfad innerhalb der Räume berechnet. Diese Suche ist relevant für den Anwendungsfall von Loomo, da die Anzahl der Türen auf dem Weg ein wichtiger Faktor bei der Auswahl der Wege ist.

3.10 Testen

Die Anwendung wurde in Android Studio mit drei Emulatoren mit verschiedenen API Versionen getestet (22, 24, 27). Für die Tests wurden mehrere Fotos von Fluchtplänen bearbeitet und in Karten transformiert. Mit einem Minimalbeispiel konnte auch die Oberfläche und die Suche getestet werden.

4 Zusammenfassung und Ausblick

In diesem Kapitel wird ein Fazit über das Gesamtprojekt gezogen. Abschließend wird ein Ausblick gegeben.

4.1 Zusammenfassung

In der Arbeit wurden theoretische Grundlagen bezüglich Fluchtplänen und Repräsentationen einer Umgebung zusammengefasst. Die erworbenen Kenntnise wurden auf ein konkretes Anwendungsszenario für Loomo übertragen und eine neuartige Umsetzung vorgestellt. Es wurde ein Prototyp entworfen, implementiert und validiert. Die Lösung ist eine Android App für Loomo Segway Robotics, die zum Kennenlernen einer Umgebung mittels Fluchtplänen dient.

4.2 Erfüllung der Anforderungen

Im Rahmen dieses Forschungsprojekts wurde ein Workflow zur Digitalisierung von Fluchtplänen konzipiert, eine graphische Nutzerschittstelle für die zu entwickelnde App, ein geeignetes Format zur Navigation im Raum auf Basis von Graphen entworfen und die Anwendung implementiert und dokumentiert, wodurch alle Anforderungen erfüllt werden konnten.

Es wurde bewiesen, dass die Anwendung funktionsfähig ist. Die Ergebnisse sind aber sehr stark vom guten SVG-Input abhängig.

4.3 Ausblick

Die Lösung ist ein exemplarischer funktionsfähiger Prototyp, der den vollständigen Prozess abbildet, eine Karte aus einem bearbeiteten Foto eines Fluchtplan zu generieren. Falls weiterhin Interesse an der Anwendung besteht, könnte sie mit anderen Anwendungen für Loomo verbunden werden, so dass Loomo auch selbstständig nach diesen Karten fahren kann. Dazu wäre eine Verbindung mit einer Anwendung nötig, die die Bewegung des Roboters steuert, und einer Anwendung, die eine Hindernis-Erkennung realisiert.

Ein essentielles Verbesserungspotential besteht auch in der Suche nach den kürzesten Wegen. Um die Ergebnisse der Anwendung zu präsentieren, wurde ein Algorithmus nach Dijkstra implementiert. Dieser bildet aber nur eine einfache Suche ab und zieht nicht solche Faktoren in Betracht, so wie die Größe des Roboters und eine Entfernung zu einer Wand einzuhalten. In der Anwendung ist auch ein Graph abgebildet, der die einzelnen Räume und die Beziehungen zwischen denen beschreibt, hiermit ist es auch möglich, nach Pfaden zu suchen, die die wenigsten Türen enthalten. Die Suche könnte auch in der Ansicht verbessert werden, sodass zuerst eine Suche nach einem kürzesten Pfad mit Räumen durchgeführt wird, und erst dann nur die relevanten Nodes aus der Datenbank geladen werden, um die genauen Pfade durch diese Räume zu finden.

Eine weitere Möglichkeit, die Funktionalität zu verbessern, wäre eine automatische Erkennung des Maßstabs. In der implementierten Anwendung erfolgt diese Umrechnung manuell, indem der Benutzer den Maßstab von SVG-Punkten zu Metern angibt. Da es aber laut des Standards drei mögliche Maßstäbe gibt, um Fluchtpläne darzustellen, wäre es machbar, beispielsweise anhand der Türen den Maßstab automatisch zu erkennen.

Eine zusätzliche Erweiterung würde darin bestehen, das Matching der Aufzüge über mehrere Etagen zu implementieren. Momentan ist es in dieser Anwendung nötig, nach einem Weg zu einem Aufzug zu suchen und aus dem Auszug dann eine neue Suche zu starten. Da aber die Fotos von Fluchtplänen nicht

professionell gemacht werden, können die Aufzüge auf den einzelnen Bildern von verschiedenen Etagen verschoben werden. Deswegen wäre es nötig, eine Lösung zu finden, um diese Aufzüge zueinander richtig zu mappen.

Literatur

[BP04] BRANDES, Ulrik ; PICH, Christian: GraphML transformation. In: *International Symposium on Graph Drawing* Springer, 2004, S. 89–99

[Cor14] CORRELL, Nikolaus: *Introduction to autonomous robots*. CreateSpace Independent Publishing Platform, 2014

[FJJ00] FERRAIOLO, Jon ; JUN, Fujisawa ; JACKSON, Dean: *Scalable vector graphics (SVG) 1.0 specification*. iuniverse, 2000

[HFPK11] HAALA, Norbert ; FRITSCH, Dieter ; PETER, Michael ; KHOSRAVANI, A: Pedestrian mobile mapping system for indoor environments based on MEMS IMU and range camera. In: *Archiwum Fotogrametrii, Kartografii i Teledetekcji* 22 (2011)

[Lat12] LATOMBE, Jean-Claude: *Robot motion planning*. Bd. 124. Springer Science and Business Media, 2012

[LLKM97] LLADS, J. ; LPEZ-KRAHE, J. ; MART, E.: A system to understand hand-drawn floor plans using subgraph isomorphism and hough transform. In: *Machine Vision and Applications* 10 (1997)

[PHF11] PETER, Michael ; HAALA, Norbert ; FRITSCH, Dieter: Using photographed evacuation plans to support MEMS IMU navigation. In: *Proceedings of the 2011 International Conference on Indoor Positioning and Indoor Navigation (IPIN2011), Guimaraes, Portugal*, 2011, S. 30–81

[PHSO10] PETER, Michael ; HAALA, Norbert ; SCHENK, Markus ; OTTO, Tim: Indoor navigation and modeling using photographed evacuation plans and MEMS IMU. In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* (2010), S. 15–19

[PZ04] PENG, Zhong-Ren ; ZHANG, Chuanrong: The roles of geography markup language (GML), scalable vector graphics (SVG), and Web feature service (WFS) specifications in the development of Internet geographic information systems (GIS). In: *Journal of Geographical Systems* 6 (2004), Nr. 2, S. 95–116

[Tsc13] TSCHORN, Georg: Extracting Indoor Map Data From Public Escape Plans On Mobile Devices. (2013)

[V09] FÜR NORMUNG E. V, DIN Deutsches I.: Sicherheitskennzeichnung – Flucht- und Rettungspläne (ISO 23601:2009). (2009)

A Protokolle

A.1 Treffen 26.04.2018

- Aufgabenstellung erhalten, Gruppenaufteilung
- Einleitung in die Aufgabe, Kontext, Hintergrund
- Vorstellung von Loomo
- Aufgaben bis zum nächsten Treffen: Aufgabe verstehen, Zeitplan erstellen

A.2 Treffen 03.05.2018

- Fragen zur Aufgabenstellung geklärt, Aufgabe abgegrenzt
- Zeitplan besprochen und angepasst
- Aufgaben bis zum nächsten Treffen: Konzeption eines Workflows zur Digitalisierung von Fluchtplänen, Gedanken zu einem geeigneten Format für Loomo zur Navigation im Raum auf Basis von Graphen

A.3 Treffen 17.05.2018

- Workflow und Format vorgestellt und diskutiert
- Erste Gedanken zum Implementierungsteil besprochen und abgestimmt
- Aufgaben bis zum nächsten Treffen: Algorithmus, der aus der Graphik einen Graphen baut

A.4 Treffen 31.05.2018

- Algorithmus für Rastergraphik implementiert
- Lösungsansatz für Vektorgraphiken diskutiert
- Aufgaben bis zum nächsten Treffen: Algorithmus, der aus der Vektorgraphik einen Graphen baut

A.5 Treffen 18.06.2018

- Algorithmus für Vektorgraphik implementiert
- Probleme bei der Performance
- Aufgaben bis zum nächsten Treffen: Code kommentieren und Debuggen (Performance Test)

A.6 Treffen 28.06.2018

- Performance Probleme diskutiert, kein Fehler entdeckt
- Aufgaben bis zum nächsten Treffen: Algorithmus raumweise implementieren

A.7 Treffen 12.06.2018

- Aufgaben bis zum nächsten Treffen: Graphische Nutzerschnittstelle, Dokumentation, Folien