ulm university universität ulm

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Bachelorarbeit
im Studiengang Informatik

Konzeption und Realisierung von Algorithmen zur POI-Clusterbehandlung in einem Location-based Augmented Reality Szenario

vorgelegt von

Johann Listunov

Dezember 2014

Gutachter	Prof. Dr. Manfred Reichert
Betreuer:	Rüdiger Pryss
Matrikelnummer	722638
Arbeit vorgelegt am:	12.12.2014

Kurzfassung

Die Augmented-Reality-Technologie (AR-Technologie) findet heutzutage in vielen Bereichen ihre Anwendung, doch durch die Verbreitung von immer leistungsfähigeren mobilen Endgeräten und durch das immer besser werdende mobile Internet, findet diese Technologie immer mehr Einsatz auf privaten Smartphones.

Diese Arbeit beschäftigt sich mit einer solchen mobilen Augmented-Reality-Anwendung. In Echtzeit wird hier der Blick durch die Geräte-Kamera um Informationen über in der Nähe liegende Interessenspunkte (POI = Point of Interest) erweitert.

Die Entwicklung einer solchen Anwendung ist sehr aufwändig und baut auf vielen, schon vorangegangenen Arbeiten auf: [PMLR14], [SSP+14], [PLRH12] und [PTKR10] sind dabei wichtige Referenzen. Eine weitere Hürde, in einem mobilen Szenario stellt, die Benutzung von Sensoren da [SSP+13].

Im genaueren aber beschäftigt sich diese Arbeit mit dem Problem von überlappenden POIs, die die Interaktion mit dem gewünschten POI erschweren oder ganz unmöglich machen. Im Verlauf dieser Arbeit werden zwei mögliche Lösungen für das Problem vorgestellt und auch implementiert.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sinngemäße Übernahmen aus anderen Werken sind als solche kenntlich gemacht und mit genauer Quellenangabe (auch aus elektronischen Medien) versehen.

Ulm, den 12.12.2014 Johann Listunov

Inhaltsverzeichnis

Abbildungsverzeichnis

Zu einem guten Ende gehört auch ein guter Beginn.

Konfuzius, (551 - 479 v. Chr.)

1 Einleitung

1.1 Motivation und Problemstellung

Unter Augmented-Reality (auf Deutsch: Erweiterte Realität) versteht man die Erweiterung der Wahrnehmung durch virtuelle Informationen. Diese Arbeit beschäftigt sich mit einer Smartphone-Anwendung (AREA, siehe 1.2), die in Echtzeit das Kamerabild um den Standort und die Entfernung von in der Nähe liegenden POIs (wie z.B. Sehenswürdigkeiten) erweitert. Doch wie man sich vorstellen kann, ist das nur ein sehr kleiner Anwendungsbereich für diese Technologie. Von Navigationssystemen, Einparkhilfen, Spielen, sogar bis hin zu virtuellen Kleideranproben, die Möglichkeiten sind vor allem durch die Massenverbreitung an leistungsstarken mobilen Endgeräten grenzenlos. Im genaueren beschäftigt sich diese Arbeit aber mit einem Problem, das bei diesem Anwendungsszenario entsteht. Es handelt sich dabei um die Clusterbildung bei vielen und vor allem überlappenden POIs, die das Bedienen der Anwendung erschweren oder unmöglich machen.

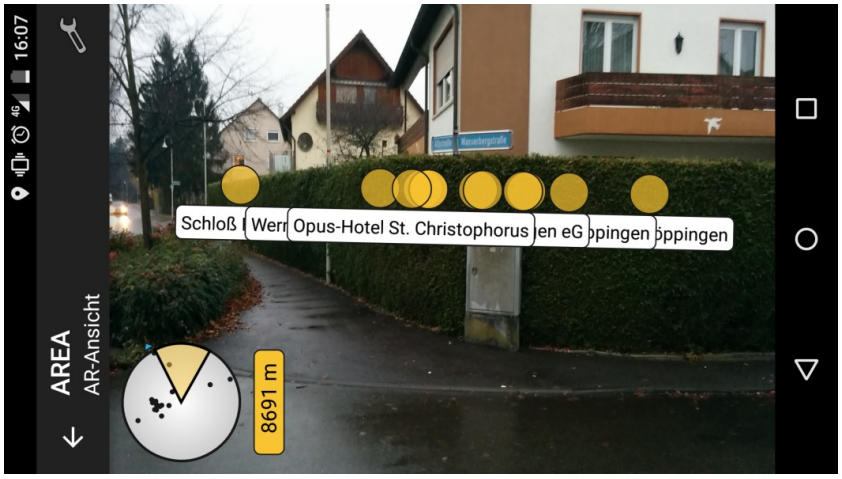


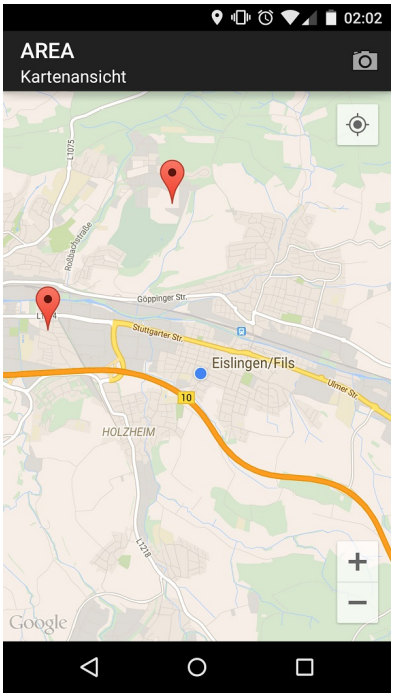
Abbildung 1.1: Viele überlappende POIs

Wie man auf der Abbildung 1.1 sieht, ist es bei vielen POIs fast unmöglich, einen bestimmten zu finden oder gar anzuklicken. Im Rahmen dieser Arbeit werden deswegen zwei mögliche Lösungen für das Problem vorgestellt und auch implementiert.

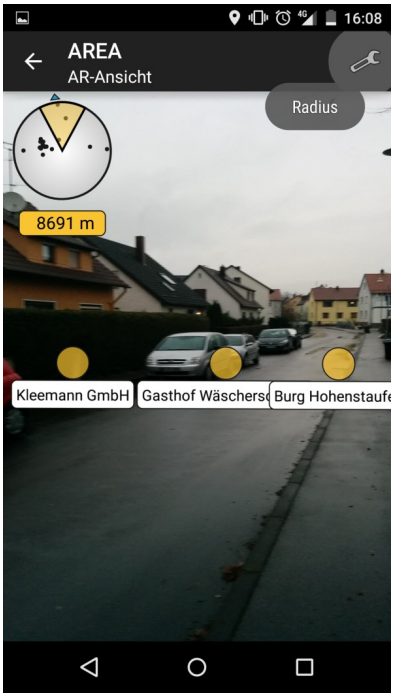
1.2 AREA-Projekt

Augmented Reality Engine Application oder kurz AREA ist eine, im Rahmen einer Bachelorarbeit [Gei12] [GSP+14] [GPSR13], für das Apple iOS [Inc14a] entwickelte mobile Augmented Reality Anwendung. Mittlerweile ist die AREA Anwendung auch für das Androidsystem vorhanden und genau diese Version von AREA wird in dieser Arbeit verwendet.

Startet der Benutzer die Anwendung, findet er sich in der Kartenansicht (Abbildung 1.2a) wieder.



(a) AREA Kartenansicht



(b) AREA AR-Ansicht

Abbildung 1.2: AREA

Auf der Karte wird nun die eigene Position angezeigt und mit der Hilfe der Google Maps API [Inc14d] werden Interessenspunkte aus der Umgebung geladen. Beim Anklicken des Kamerasymbols in der oberen rechten Ecke (siehe Abbildung 1.2a), wechselt die Anwendung in den AR (Augmented Reality) Modus (siehe Abbildung 1.2b). Die Kameraansicht wird dabei in Echtzeit, mit den davor in der Kartenansicht geladenen POIs (die gelblichen "Punkte" in Abbildung 1.2b), erweitert. Das Radar in der oberen linken Ecke zeigt an. in welcher Richtung sich die POIs befinden.

(a) AREA Radiuseinstellungen

(b) AREA POI Informationen

Abbildung 1.3: AREA Einstellungen

Klickt der Benutzer nun auf einen der POI, öffnet sich ein Dialog mit zusätzlichen Informationen(siehe Abbildung 1.3b) zum gewählten POI. Weiterhin kann man per Klick auf den

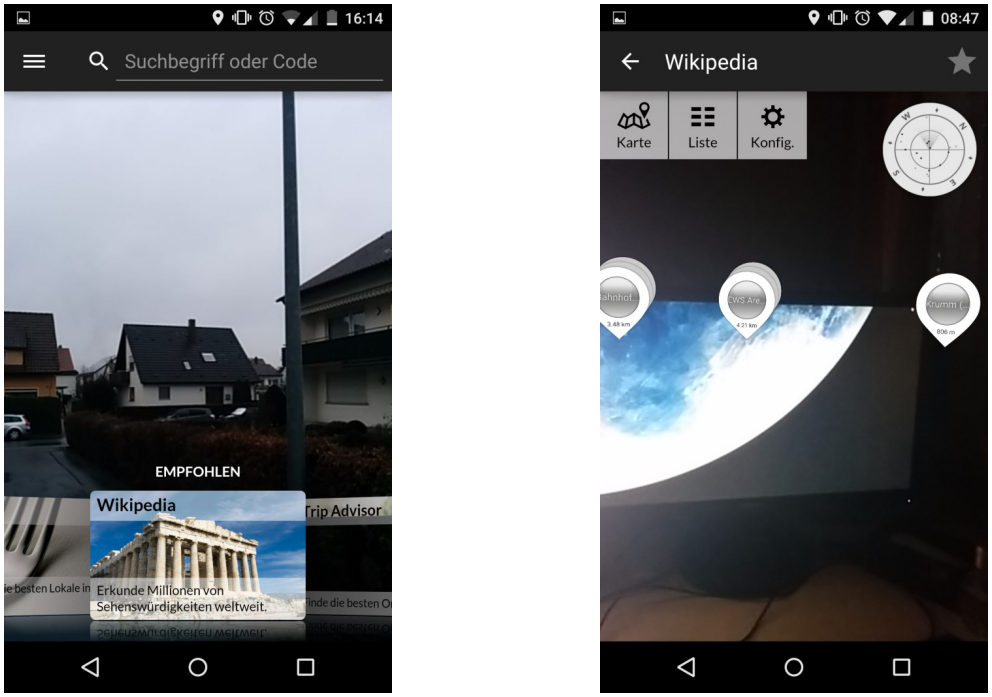
‘Schraubenschlüssel", in der oberen rechten Ecke (siehe Abbildung 1.2b), die maximale Distanz (1000m - 15000m) der angezeigten POIs einstellen (siehe Abbildung 1.3a). Und schlussendlich kommt man durch einen Klick auf den Pfeil, in der oberen linken Ecke (siehe Abbildung 1.2b), wieder zurück zu der Kartenansicht.

2 Theorie

2.1 Verwandte Arbeiten

Dieser Abschnitt beschäftigt sich mit anderen mobilen Anwendungen, die einen ähnlichen oder sogar gleichen Verwendungszweck haben wie AREA. Dabei wird die Anwendung kurz mit Screenshots vorgestellt und die Lösung des Clusterproblems wird, falls vorhanden, auch dargelegt. Wegen des Fehlens eines iOS Gerätes, werden im weiteren Verlauf nur Android Anwendungen vorgestellt.

2.1.1 wikitude

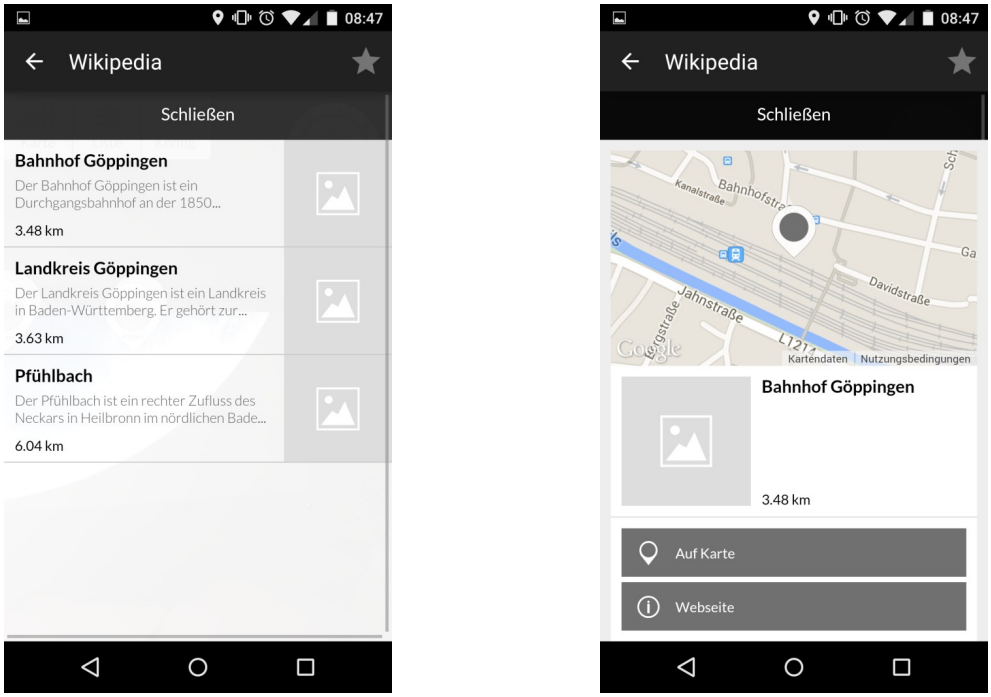


(a) wikitude Aswahl der Kategorie **(b)** wikitude AR-Ansicht

Abbildung 2.1: wikitude Start

Wikitude ist eine auf Basis des kommerziellen Wikitude SDK aufbauende AR-Anwendung, die für das Finden der POIs hauptsächlich Wikipedia [Fou14b] als Quelle benutzt. Beim Start dieser Anwendung wir der Benutzer gefragt aus welcher Kategorie (siehe Abbildung 2.1a) die zu ladenden POIs sein sollen (wie z.B. in der Nähe liegende POIs mit einem Wikipedia-Eintrag).

Nachdem der Benutzer seine Wahl getroffen hat, wird in die AR-Ansicht gewechselt und die POIs werden geladen. Die geladenen POIs werden als tropfenförmige Symbole dargestellt (siehe Abbildung 2.1b). Dabei werden überlappende POIs in Gruppen (drei hintereinander liegende Symbole) zusammengefasst. Klickt der Benutzer nun auf eine dieser Gruppen, öffnet sich eine Liste (siehe Abbildung 2.2a) mit allen POIs, die in der Gruppe enthalten sind. Wird nun einer der POIs in der Liste vom Benutzer angeklickt so öffnet sich eine neue Ansicht mit den genaueren Informationen über den gewählten POI (siehe Abbild 2.2b).



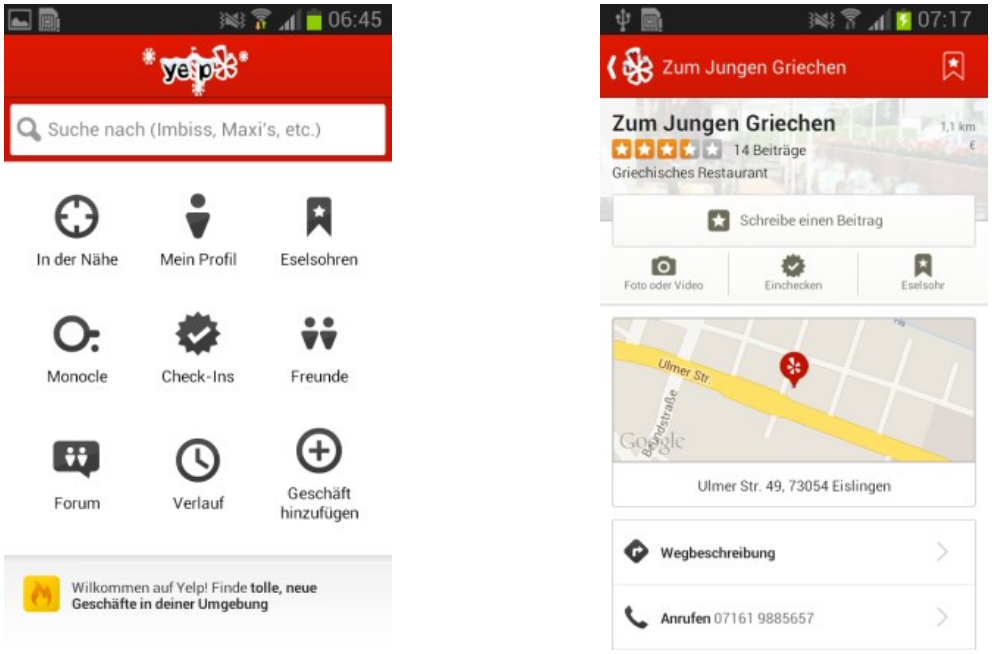
(a) wikitude POI Liste **(b)** Weitere Informationen zu einem POI

Abbildung 2.2: wikitude POI

Das Problem der überlappenden POIs wird hier damit gelöst, dass diese in Gruppen zusammengefasst werden (siehe Abbildung 2.1b). Wie das aber genau geschieht, ist ein Geheimnis des Wikitude SDKs. Der Algorithmus kann also hier nicht bewertet werden, dafür aber die Methode an sich: POIs in Gruppen zu fassen, ist eine einfache, aber effektive Idee. Der Bildschirm wirkt aufgeräumt, aber an der Bedienung der Anwendung wird nicht viel verändert. Diese wird sogar erleichtert, da überlappende POIs nun durch einen Cluster ersetzt werden. Es bleibt die Frage wie man die POIs effizient in Gruppen zusammenfasst. Eine mögliche Lösung dafür wird später in der Arbeit vorgestellt und implementiert.

2.1.2 yelp

Yelp ist eine Anwendung vom gleichnamigen Unternehmen yelp und hilft dem Benutzer dabei, die Geschäfte/Restaurants/Bars/Tankstellen/usw. mit den meisten positiven Bewertungen zu finden. Von all den Funktionen, die diese Anwendung anbietet, ist im Rahmen dieser Arbeit nur der "Monocle"- Modus interessant. Dieser ermöglicht die Suche nach Restaurants und Bars in der AR-Ansicht.



(a) yelp Menü **(b)** Weitere Informationen zu einem POI

Abbildung 2.3: yelp Bedienung

Im Hauptmenü der Anwendung kommt der Benutzer über den dazu gehörigen Knopf in den Monocle-Modus und damit in die AR-Ansicht (siehe Abbildung 2.4). Die POIs beschränken sich bei der Anwendung auf Restaurants oder Bars und werden hier in einem schwarz-transparenten Kasten angezeigt (siehe Abbildung 2.4), in dem sich neben dem Namen und der Entfernung auch die durchschnittliche Bewertung finden lässt. Klickt man nun auf einen dieser POIs, öffnet sich eine neue Ansicht mit zusätzlichen Informationen.

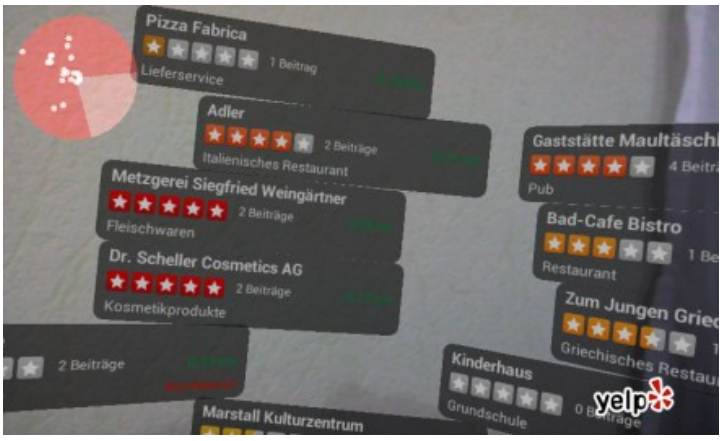


Abbildung 2.4: Monocle-Modus (AR-Ansicht)

Das Problem der überlappenden POIs wurde hier mit der Verschiebung der Punkte gelöst. Alle POIs werden so auf dem Bildschirm verschoben, dass sie direkt nebeneinander gezeichnet werden und sich deswegen nicht gravierend überlappen. Das heißt, bei vielen POIs auf einem

Punkt muss der Benutzer sein Handy gegebenenfalls herumschwenken, um alle POIs zu sehen. Dieser Lösungsansatz funktioniert bei einer kategorisierten AR-Anwendung wohl am besten. Beschränkt man die Art der POIs nur auf eine Kategorie (wie z.B. auf Restaurants), so sind das in den meisten Fällen nicht so viele POIs und die AR-Ansicht wird nicht zu voll. Für AREA wäre dieser Lösungsansatz so nicht zu empfehlen, denn in einer größeren Stadt mit vielen POIs kann es durchaus vorkommen, dass auf dem Bildschirm 50 und mehr POIs angezeigt werden müssen. Mit dieser Methode wäre der gesamte Bildschirm voller POIs und die Suche nach einem bestimmten würde sich als schwer und lästig erweisen.

2.1.3 Junaio AR Browser

Der Junaio AR Browser [Gmb14a] ist eine Anwendung auf der Basis der Metaio 6 Plattform [Gmb14c]. Die Metaio 6 Plattform von Metaio GmbH [Gmb14b] ist ein kommerzielles SDK für AR-Anwendungen. Von den vielen Features des Junaio AR Browsers wird im Rahmen dieser Arbeit nur der "Junaio Around You"-Modus betrachtet. Dieser Modus zeigt dem Benutzer die in der Nähe liegenden POIs durch eine AR-Ansicht an. (Abbildung 2.5 zeigt, rot markiert, den Knopf, den man anklicken muss, um in den "Junaio Around You"-Modus zu kommen.)

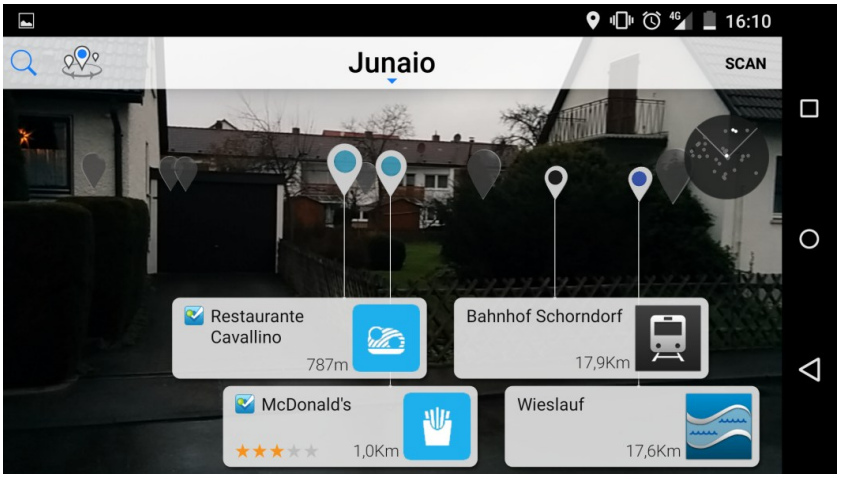
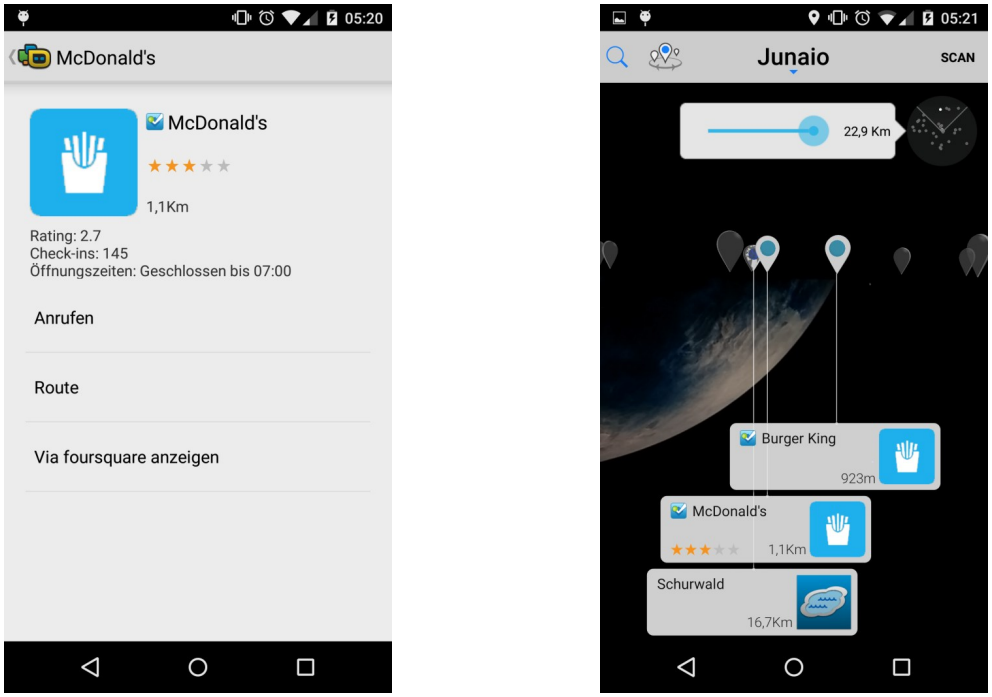


Abbildung 2.5: Junaio Around You

Beim Anklicken des Radars kann man den Radius auswählen, in dem die POIs geladen werden sollen. Mit einem Slider (siehe Abbildung 2.6b) lassen sich alle Distanzen zwischen 0m und 23km einstellen. Jeder POI wird mit einem grauen, tropfenförmigen Symbol dargestellt und lässt sich anklicken. Dies führt zu einer weiteren Ansicht mit genaueren Informationen (siehe Abbild 2.6a) über den gewählten POI.



(a) Weitere Informationen eines POI **(b)** Eisntellung des Radius

Abbildung 2.6: Junaio Bedienung

Der Junaio AR Browser hat dabei eine interessante Art das Clusterproblem zu lösen, wie man in den Abbildungen 2.6b und 2.5 sehen kann, werden nur drei bis vier POIs mit Namen und Entfernung angezeigt und ihre Informationen sind am unteren Rand des Bildschirms zu finden. Diese Punkte werden aus der Mitte des Sichtfeldes anhand der Entfernung ausgewählt, wobei Punkte, die näher liegen, favorisiert werden.

Diese Methode aber löst das Problem der überlappenden Punkte nur teilweise. Spaziert man einfach so durch eine Stadt, wird diese Methode dem Benutzer anzeigen, welche POIs sich in der Nähe befinden und man kann was Neues entdecken. Sucht man aber nach einem bestimmten POI, ist diese Lösung nicht dafür geeignet. Sollte der gesuchte POI nicht zu den drei bis vier naheliegenden Punkten gehören, so muss man sich durch die unbeschrifteten Symbole klicken. Sind dann auch noch viele Symbole auf einem Punkt, lässt sich nur das anwählen, das am nächsten liegt.

2.1.4 mixare

Der Abschnitt zu mixare (mix Augmented Reality Engine) [Srl14a] wird sich nur auf eine kleine Vorstellung beschränken, da in mixare überlappende POIs nicht behandelt werden, wie man in Abbildung 2.7 sehen kann. Daher ist mixare für diese Arbeit nur von kleinem Interesse.

Abbildung 2.7: Mixare

Mixare ist eine open-source-Anwendung, erhältlich für beide Systeme Android und iOS. Die Informationen zu den POIs erhält mixare unter anderem von Wikipedia, Twitter [Inc14f] und OpenStreetmap [Fou14a]. Bei einem Klick auf einen POI wird dem Benutzer sofort die dazugehörige Wikipedia-Seite angezeigt (siehe Abbildung 2.8).

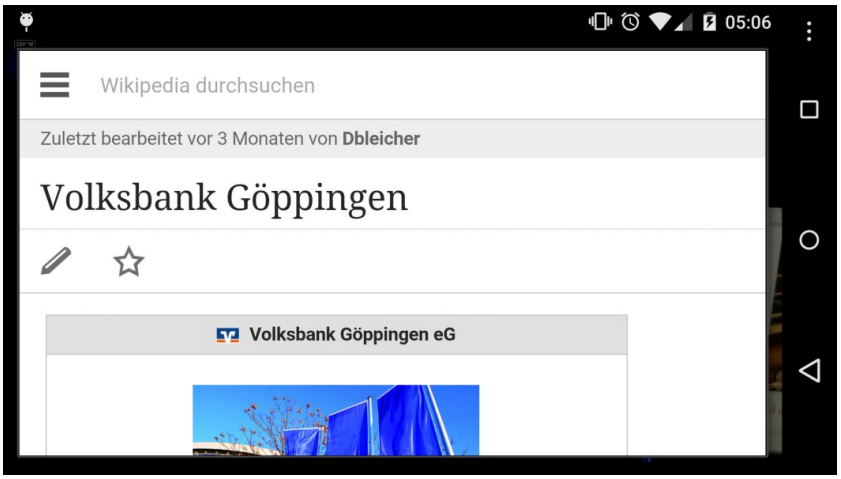


Abbildung 2.8: Mixare - POI Information

Die Tatsache, dass mixare aber Open-Source ist und ein interessierter Benutzer fehlende Features theoretisch selber hinzufügen könnte, macht mixare erwähnenswert. Es sei auch gesagt, dass in der Roadmap [Srl14b] von mixare die Rede von Clustering ist und die Anwendung vielleicht in der Zukunft relevanter wird, was das Thema angeht.

2.2 Anforderungen

Die Anforderungen an die Lösung für das Clusterproblem sind im Grunde die gleichen, die an jede mobile Anwendung gestellt werden: Der Algorithmus, der sich um die Erkennung und

Bildung der Cluster kümmert, darf nicht zu viele Ressourcen belegen, muss aber trotzdem alles in Echtzeit berechnen. Die Anwendung darf also nicht ins Stottern kommen, wenn die POIs zu einem Cluster zusammengefasst werden.

Eine weitere Anforderung wird an die Benutzerfreundlichkeit gestellt: Die Anzeige der POIs darf sich nicht so stark verändern, so dass die Bedienung der Anwendung unnötig erschwert wird.

2.3 Architektur

Im Rahmen dieser Arbeit werden die beiden vorgestellten Lösungen nur für die Android Version des AREA-Projektes implementiert.

2.3.1 Android OS und SDK

Android [All14] ist ein auf Linux basierendes, von Google [Inc14c] mitentwickeltes Betriebssystem für eine Vielzahl an mobilen Endgeräten (Tablets,Smartphones, usw.). Das Betriebssystem ist seit Ende 2008 offiziell verfügbar und ist mittlerweile (Dezember 2014) in der Version 5.0.1 vertreten. Die Entwicklung von Android-Anwendungen findet in den meisten Fällen in Java (2.3.2) statt. Für Anwendungen bei denen die Geschwindigkeit eine kritische Rolle spielt gibt es aber auch die nativen C/C++ Bibliotheken. Weiter benötigt man für die Entwicklung von Anwendungen noch das Android SDK [Inc14b]. Dabei handelt es sich um die Sammlung an Entwicklungsprogrammen für Android wie z.B. adb (Android Debug Bridge). Implementiert wird alles für die API-Version 14 (Android 4.0, API = Application Programming Interface, auf Deutsch: Programmierschnittstelle) und funktioniert auch für die späteren Versionen.

2.3.2 Java

Java [Mic14] ist in dieser Arbeit die einzige Programmiersprache, die bei der Implementierung verwendet wird. Java ist eine von Sun Microsystems [Ora14] entwickelte Objektorientierte Programmiersprache. Die hier verwendete Version ist 1.7.

2.3.3 Testgeräte

Zum Testen der Anwendungen wurden zwei Testgeräte verwendet. Zum einen ein Samsung Galaxy S2 [Sam14] mit der Android Version 4.1.2 und ein Google Nexus 5 [Inc14e] mit der Android Version 5.0. Als Anmerkung sei noch gesagt: Alle in dieser Arbeit verwendeten Screenshots sind mit dem Nexus 5 gemacht, einfach der höheren Auflösung wegen.

3 Implementierung

Diese Kapitel beschäftigt sich mit den Implementierungen der beiden Lösungsansätze und es wird auf Probleme (Bugs) eingegangen die beim Implementieren entstanden sind.

3.1 Clustering

Die Grundidee dieses Lösungsansatzes war es naheliegende und dadurch überlappende Punkte in eine Gruppe (Cluster) zusammenzufassen. Das eigentliche Problem bestand also eher darin diese Gruppen effizient zu erkennen. Die endgültige Lösung wurde dann noch ein wenig vom k-Means-Algorithmus [Wik14] inspiriert, deswegen wird hier kurz darauf eingegangen, wie dieser funktioniert.

3.1.1 k-Means

Anmerkung: Der k-Means-Algorithmus hat nur ein kleinen Teil zu der Endidee beigetragen und wird deshalb hier nur sehr oberflächlich erklärt.
Der k-Means-Algorithmus wird dazu verwendet, um eine beliebige Anzahl an Punkten in k viele Gruppen zu teilen. Dazu werden nach einer bestimmten Strategie, von denen es mehrere gibt, k viele Punkte ausgewählt. Zwischen den k Punkten wird nun in jeweils der Mitte eine "Linie" gezogen und so die Anzahl der gesamten Punkte in k Gruppen geteilt. In Abbildung 3.1 kann man ein mögliches Ergebnis des Algorithmus sehen. Die drei dunkelfarbigen Punkte sind die Punkte, die aus einer zufälligen Strategie, bei der das k gleich drei ist, entstanden sind und als Mittelpunkte der Gruppen bestimmt wurden. Danach werden die restlichen Punkte dem markierten Punkt zugewiesen, der ihnen am nächsten ist. So werden aus den Punkten genau drei Gruppen.

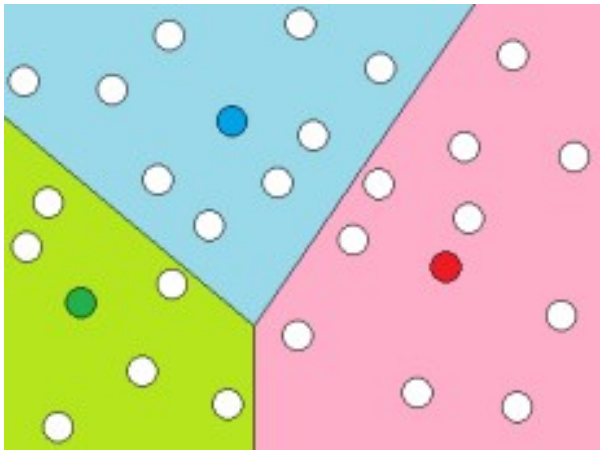


Abbildung 3.1: k-Means-Algorithmus

Der reine k-Means-Algoritmus ist aber für den Zweck dieser Arbeit in seiner Grundform so eher nicht zu gebrauchen. Es treten an einigen Stellen Probleme auf:

Der Algorithmus kann erst ausgeführt werden, wenn alle Punkte zur Verfügung stehen, da ansonsten für jeden nachfolgenden Punkt der Algorithmus komplett neu ausgeführt werden müsste um bessere Mittelpunkte für die Gruppen zu finden. Dies ist, bei vielen Punkten, ein Problem für die Echtzeitausführung, da hier sehr viel Rechenzeit gebraucht wird. Des Weiteren tritt im Grunde das selbe Problem beim Bestimmen der Anzahl der Gruppen auf, also beim Bestimmen des k, auch hier muss darauf gewartet,bis alle Punkte geladen worden sind. Das dritte Problem tritt bei einem schlecht gewählten k auf. Wird der Bildschirm in genau k Gruppen aufgeteilt, kann passieren, dass POIs zu einer Gruppe (Cluster) zusammengefasst werden, obwohl sie sich nicht wie auf Abbildung 1.1 überlappen.

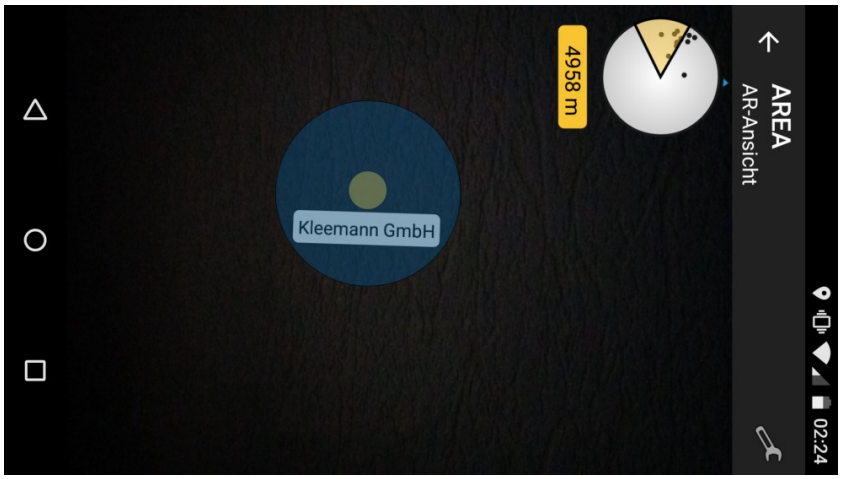


Abbildung 3.2: Neuer Interessenpunkt (POI)

Deswegen wurde bei dieser Lösung der Algorithmus nur als Anregung genommen und eine angepasste Version wurde implementiert. Das statische k wird dynamisch und automatisch vom Algorithmus angepasst. Die Auswahl der Punkte, um die danach ein Cluster entsteht,

geschieht fast vollkommen zufällig. Es wird der Einfachheit halber angenommen, dass jeder POI der Mittelpunkt eines Cluster sein könnte, d.h. wenn der erste POI auf den Bildschirm gezeichnet wird, wird dieser als der Mittelpunkt eines Clusters angesehen und ein unsichtbarer Kreis wird um dessen Mittelpunkt gezogen (der blau ausgemalte Kreis in Abbildung 3.2). Wenn nun der nächste POI geladen wird, so wird zuerst getestet, ob sich dieser innerhalb des blauen Kreises (Abbildung 3.2) befindet, wenn dies der Fall ist, wird aus dem vorherigen POI nun ein Cluster aus zwei POIs (wie auf Abbildung 3.3).

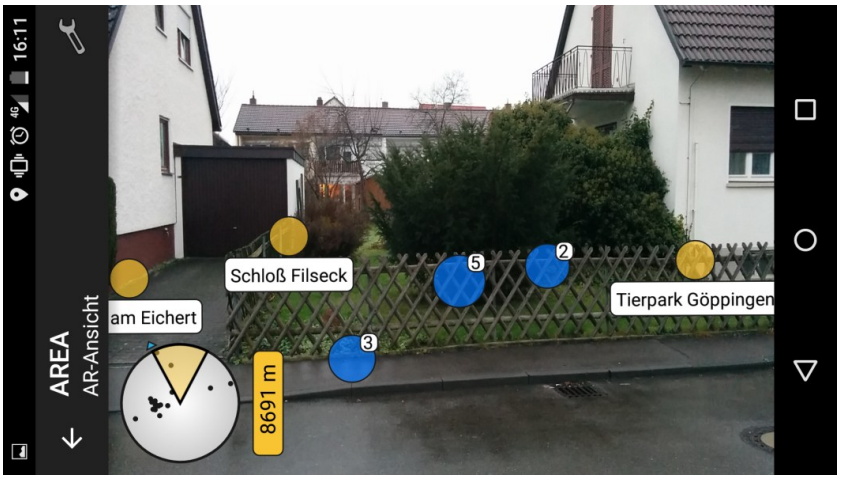


Abbildung 3.3: AREA Cluster

Wenn das nicht der Fall ist, wird der neue Punkt einfach wieder als ein möglicher Mittelpunkt eines Clusters angesehen und der dritte geladene Punkt wird zuerst darauf getestet, ob er in einem der zwei möglichen Cluster liegt. Dabei wird der Punkt immer in den ersten passenden Cluster eingefügt und dadurch das die Liste der POIs immer in der gleichen Reihenfolge (Eigenheit von AREA selbst) geladen werden, entstehen immer die gleichen Cluster.

3.1.2 Implementierung

Für die Implementierung war es zuerst notwendig, die Methode zu verändern, die die POIs erhält und auf dem Bildschirm anzeigt. Die gesuchte Methode ist die *onHeadingWithLocationsChanged* in der *AREAMainActivity*-Klasse. Die Methode erhält eine Liste (*rawPoiList*) der POIs, die zum *locationView* (der Inhalt davon wird auf dem Bildschirm gezeichnet) hinzugefügt werden sollen. Zuerst durchlaufen die ankommenden POIs alle bereits vorhandene Cluster und es wird geprüft ob diese innerhalb eines vorhandenen Clusterradius liegen (Codeausschnitt 1).

Diese Überprüfung der POIs wird von der Methode *updateAndCheckPoi* (Codeausschnitt 1 Zeile 8) der *AREAClusterView*-Klasse übernommen. Die *AREAClusterView*-Klasse ist dabei eine neue Klasse, die nach dem Vorbild der *AREAPointOfInterestView*-Klasse erstellt wurde und nun als einziges Objekt auf dem *locationView* gezeichnet wird. Die *AREAPointOfInterestView*-Klasse war davor vollkommen dafür verantwortlich POIs auf dem Bildschirm anzuzeigen, siehe Abbildung 1.1. Jeder der angezeigten POIs ist dabei vom Typ *AREAPointOfInterestView*. Für das Clusteringverfahren wird diese komplett durch die *AREAClusterView*-Klasse ersetzt. Die

Codeausschnitt 1 onHeadingWithLocationsChanged-Methode

```
for (int i = 0; i < locationView.getChildCount(); i++) {
 if (locationView.getChildAt(i) instanceof AREAClusterView) {
  AREAClusterView cv = (AREAClusterView)locationView.getChildAt(i);
  boolean clusterViewIsValid = false;
  cv.startUpdate();
  for (int j = 0; j < rawPoiList.size(); j++) {
   AREAPointOfInterest poi = rawPoiList.get(j);
   if(poi != null && cv.updateAndCheckPoi(poi)){
    rawPoiList.remove(j);
    rawPoiList.add(j, null);
    clusterViewIsValid = true;
   }
  }
  if(!clusterViewIsValid || !cv.stopUpdate()){
   locationView.removeView(cv);
   cv.clearContent();
   cv = null;
  }else{
   clusterList.add(cv);
}}}
```

AREAClusterView-Klasse passt dabei ihr Aussehen (*updateClusterAppearance*-Methode), an die Anzahl der in ihr enthaltenen POIs, an. Ist nur der POI selber im Cluster, so ist das Aussehen das selbe wie bei *AREAPointOfInterestView*. Sind im Cluster zwei oder mehr POIs enthalten, dann ändert sich die Farbe des Punktes zu einem Blauton und es wird ein Label mit der Anzahl der im Cluster enthaltenen POIs (siehe Abbildung 3.3) angezeigt.

Codeausschnitt 2 updateAndCheckPoi-Methode

```
public boolean updateAndCheckPoi(AREAPointOfInterest poi){
 if(clusterContent.contains(poi)){
  if(rootPoi == poi){
   updatePosition();
  }else if(!isWithinClusterRadius(poi)){
   return false;
  }
  tempClusterContent.add(poi);
  return true;
 }else if(poi != null && isWithinClusterRadius(poi)){
  tempClusterContent.add(poi);
  clusterContent.add(poi);
 }
 return false;
}
```

Die *updateAndCheckPoi*-Methode (Codeausschnitt 2) prüft, ob ein POI bereits im Cluster vorhanden ist und/oder ob dieser sich im Clusterradius befindet. Befindet sich der POI nicht im Clusterradius (*isWithinClusterRadius*-Methode gibt ein *false* zurück), so wird der POI nicht in den Cluster aufgenommen oder falls der POI davor schon im Cluster war und seine Position sich geändert hat, so wird der POI aus dem Cluster entfernt. Die *isWithinClusterRadius*-Methode

benutzt dabei einfach den Satz des Pythagoras, um die Entfernung zwischen dem Clustermittelpunkt und dem neuen POI auszurechnen und vergleicht die errechnete Entfernung mit einem festgelegtem Wert. Handelt es sich bei dem zu prüfenden POI um den Ausgangspunkt der als Clustermittelpunkt verwendet wird, so wird nur die Position des Clusters auf dem Bildschirm aktualisiert. Wird der POI einem Cluster zugewiesen oder wurde der POI in einem Cluster gefunden, wird dieser in der *rawPoiList* durch *null* ersetzt (Codeausschnitt 1 Zeile 10) und damit für die weitere Verarbeitung entfernt.

Aus allen POIs, die nach der Überprüfung, immer noch in der *rawPoiList* stehen werden nun nach der Reihe Clustermittelpunkte, bei denen wieder der Rest der *rawPoiList* getestet wird„ ob dieser in dem neu entstandenen Cluster liegt (siehe Codeausschnitt 3). Außerdem erhält jeder Cluster beim erstellen einen *OnClickListener* (Codeausschnitt 3 Zeile 17).

Codeausschnitt 3 onHeadingWithLocationsChanged-Methode

```
1for (AREAPointOfInterest poi : rawPoiList) {
2 if(poi == null){
3  continue;
4 }
5 boolean addedPoiToACluster = false;
6 for(AREAClusterView cv : clusterList){
7  if(cv != null && cv.addPoi(poi)){
8   addedPoiToACluster = true;
9   break;
10  }
11 }
12 if(!addedPoiToACluster){
13  AREAClusterView cv = new AREAClusterView(this,poi);
14  cv.setOnClickListener(new OnClickListener() {
15  @Override
16  public void onClick(View v) {
17   //...
18  }});
19  clusterList.add(cv);
20  locationView.addView(cv);
21}}
```

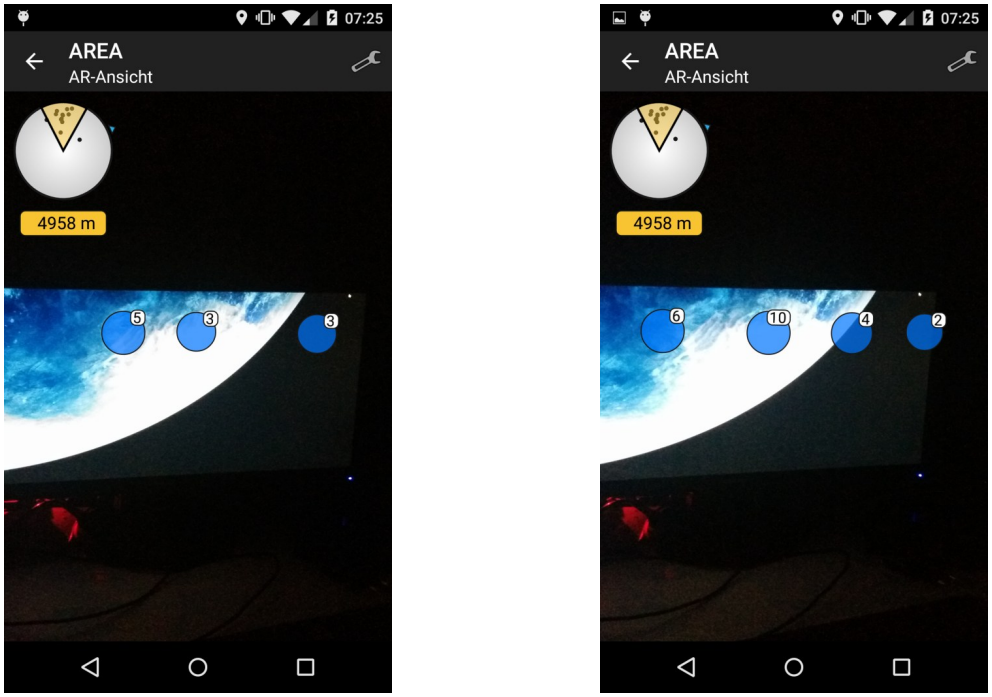
Dieser *OnClickListener* wird aufgerufen, wenn der Benutzer auf den POI (oder Cluster) tippt. Er erzeugt beim Cluster eine Liste aus den POI, die im Cluster liegen. Handelt es sich beim gewählten Objekt noch um einen POI, so werden genauere Informationen zum POI angezeigt (siehe dazu Abschnitt 4.1).

3.1.3 Probleme bei der Implementierung

In den weiteren Abschnitten werden Probleme und falls vorhanden Lösungen vorgestellt, die beim Implementieren auftraten.

3.1.3.1 Multiplizierende Punkte

Wenn man die Anwendung ganz normal per Zurück-Taste beendet und sie wieder startet, verdoppeln sich die zuvor geladenen POIs (siehe Abbildung 3.4). Dieses Problem ist, wie sich später herausgestellt hat, ein Problem von AREA selber.



(a) Beim erststart insgesammt 11 POIs **(b)** Nach dem Neustart 22 POIs

Abbildung 3.4: Multiplizierende Punkte

Wie sich nach einigen Tagen des Testens herausgestellt hat, dieses Problem seinen Ursprung in der *AREAStore*-Klasse, diese speichert alle POIs zwischen, wird aber nie geleert und besitzt auch keine Abfrage, ob die Punkte schon abgespeichert wurden und erlaubt deswegen multiple Einträge des gleichen POI. Eine einfache Lösung ergab sich durch das Hinzufügen einer *onBackPressed*-Methode in die *MainActivity*-Klasse (siehe Codeausschnitt 4).

Codeausschnitt 4 onBackPressed-Methode

```
1@Override
2public void onBackPressed(){
3 AREAStore.getInstance().getStore().clear();
4 super.onBackPressed();
5}
```

Diese *onBackPressed*-Methode wird jedes mal aufgerufen wenn das Programm beendet wird und leert den gesamten Inhalt des *AREAStore*, so dass beim nächsten Start der Anwendung keine Duplikate im *AREAStore* entstehen.

3.1.3.2 Springender Punkt

Befindet sich ein Punkt genau am Rand des Clusterradius (siehe Abbildung 3.5), kann es durch die Berechnung (Fließkommaberechnungen und die Umwandlung von Fließkommazahlen in ganze Zahlen führen unweigerlich zu kleinen Fehlern) der Position des POI und durch die noch vorhandene Ungenauigkeit des eigenen Standorts passieren, dass dieser Punkt bei einer Aktualisierung in dem Radius liegt und bei der nächsten wieder nicht. Dies führt zum Flackern weil, der Punkt immer wieder auftaucht und verschwindet.

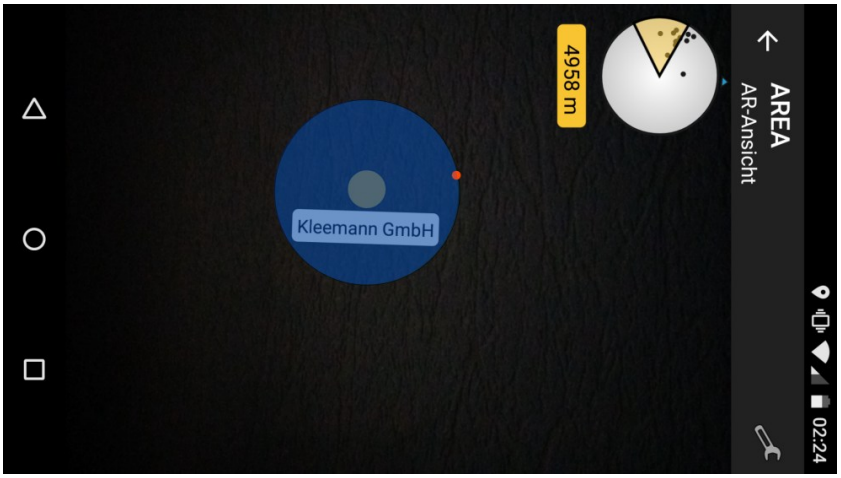


Abbildung 3.5: POI am Rande des Clusterradius

Die Lösung dafür war es, den Radius für die Punkte, die bereits im Cluster liegen, zu erweitern. Das heißt die Punkte, die in den Cluster gekommen sind (blauer Kreis in Abbildung 3.6), müssen eine größere Strecke "zurücklegen", um wieder aus dem Cluster zu kommen (roter Kreis in Abbildung 3.6).

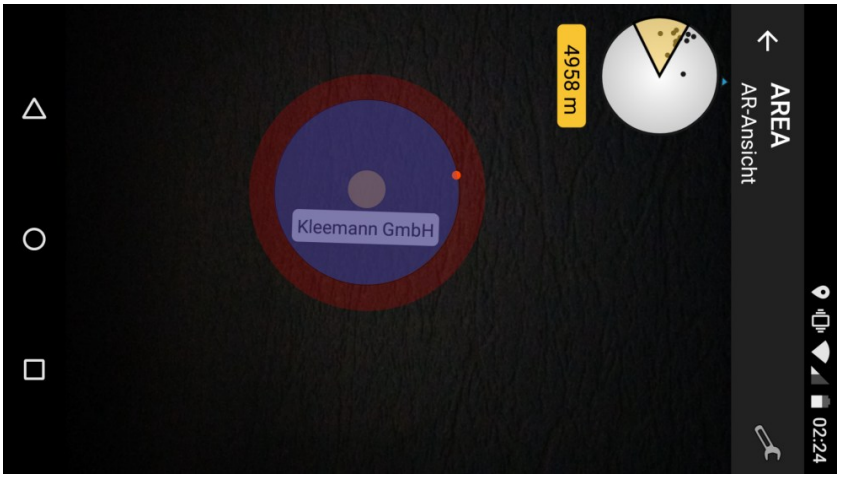


Abbildung 3.6: Erhöhter Radius für die Punkte im Cluster

So wird sichergestellt, dass kleine willkürliche Änderungen der POI-Position nicht dazu führen können, dass der POI den Cluster wieder verlässt. Implementiert wurde die Lösung in der *isWithinClusterRadius*-Methode, die in der *AREAClusterView*-Klasse liegt. Bei der Überprüfung, ob ein Punkt sich innerhalb eines Clusters befindet, wird auch darauf getestet, ob der Punkt sich schon davor im Cluster befand (Codeausschnitt 5 Zeile 9). Wenn das der Fall ist so wird der Clusterradius um *INNER_POI_CLUSTER_RADIUS_PERCENTAGE*-Prozent erweitert (z.B. 15%)

Codeausschnitt 5 isWithinClusterRadius-Methode

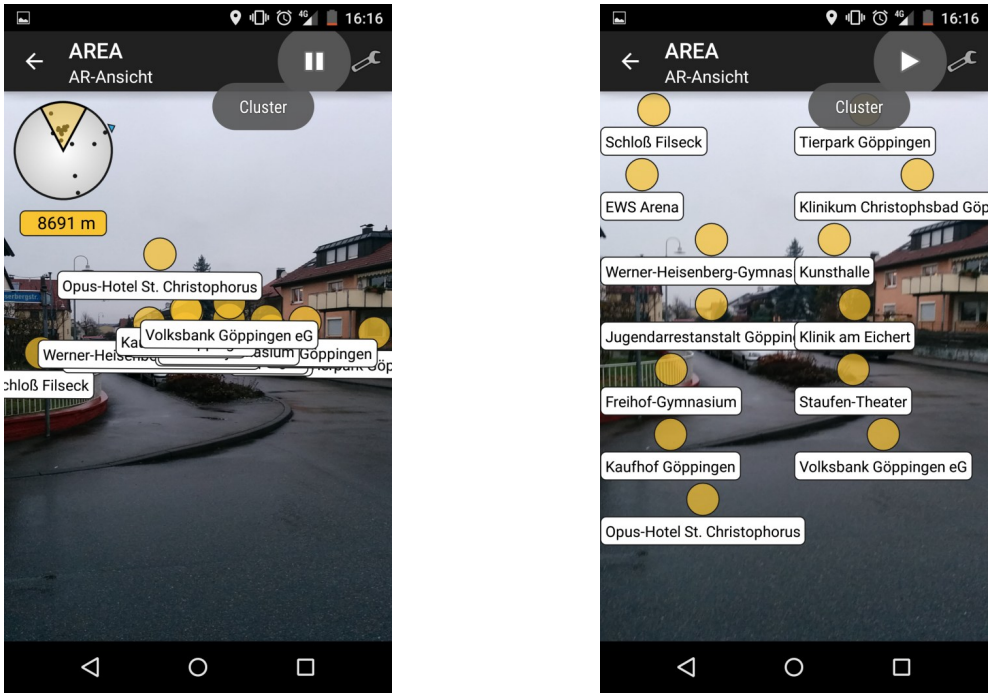
```
1private boolean isWithinClusterRadius(AREAPointOfInterest poi){
2 int x1 = rootPoi.getPoint().x;
3 int y1 = rootPoi.getPoint().y;
4
5 int x2 = poi.getPoint().x;
6 int y2 = poi.getPoint().y;
7 float distance = (float) Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
8
9 if(clusterContent.contains(poi)){
10  return (distance <= (clusterRadius*(1+INNER_POI_CLUSTER_RADIUS_PERCENTAGE)));
11 }
12 return (distance <= clusterRadius);
13}
```

3.1.3.3 Gleicher onClickListener

Hierbei handelt es sich um ein Problem für das zurzeit noch keine Lösung gefunden wurde. Wenn der Benutzer beim Starten der AR-Ansicht das Smartphone bereits in die Richtung von vielen POIs hält, werden vielen Cluster der selbe *OnClickListener* zugewiesen, d.h. unabhängig davon auf welchen Cluster der Benutzer klickt, er bekommt immer die gleiche Liste an POIs angezeigt. Schaut man nun mit der Kamera in eine andere Richtung und schwenkt wieder in die Ausgangsstellung zurück, bekommt jeder Cluster seinen eigenen *OnClickListener* und alles funktioniert wie vorgesehen.

3.2 Play/Pause-Methode

Die Idee hinter dieser Lösung war es dem Benutzer zu überlassen, wann er nicht mehr mit den überlappenden Punkte zurechtkommt. Tritt dieser Fall auf, kann der Benutzer per Drücken auf den Pauseknopf (siehe Abbildung 3.7a) dafür sorgen, dass alle zur Zeit angezeigten POIs in einen GridView angezeigt werden. Der Pauseknopf wird zu dem Menü hinzugefügt und ist soweit die einzige Änderung auf den ersten Blick.



(a) Der Pauseknopf **(b)** Der Playknopf

Abbildung 3.7: Die beiden möglichen Zustände der Anwendung

Beim Drücken des Pauseknopfes werden alle POIs im Sichtfeld zusammengefasst und in einem GridView aufgefächert (siehe Abbildung 3.7b).

3.2.1 Implementierung

Der größte Eingriff wird dazu in der *AREAMainActivity* Klasse gemacht. Eine klassenglobale Variable vom Typ *boolean* (namens *isPaused*) stellt dabei den Zustand dar, in welchem sich die AR-Ansicht gerade befindet: Bei *true* ist sie pausiert, bei *false* nicht.

Zu der Menüleiste wird in der *onCreateOptionsMenu*-Methode ein Pauseknopf hinzugefügt. Für den Pauseknopf wird nun in der *onOptionsItemSelected*-Methode ein Fall hinzugefügt, der dafür sorgt, dass beim Drücken des Pause- oder des Playknopfs die neu hinzugefügte *handlePausePlayButton* Methode (Codeausschnitt 6)aufgerufen wird. Diese kümmert sich intern, abhängig vom Zustand der *isPaused* Variable, um das Aussehen des Play/Pause-Knopfes und ruft dabei entweder die *pause* (Codeausschnitt 6 Zeile 5) oder die *unpause* (Codeausschnitt 6 Zeile 9) Methode auf.

Codeausschnitt 6 handlePausePlayButton-Methode

```
1private void handlePausePlayButton(MenuItem playPauseButton){
2 if(!isPaused){
3  //Pause:
4  playPauseButton.setIcon(android.R.drawable.ic_media_play);
5  pause();
6 }else{
7  //Unpause:
8  playPauseButton.setIcon(android.R.drawable.ic_media_pause);
9  unpause();
10}}
```

Diese *pause* und *unpause* Methoden kümmert sich dann um die gesamte Anzeige.

Codeausschnitt 7 pause-Methode

```
1private void pause(){
2 if(!isPaused){
3  isPaused = true;
4  mainLayout.removeView(userinterfaceLayout);
5  mainLayout.removeView(locationView);
6  createPOIGrid();
7}}
```

Die *pause* Methode setzt die *isPaused* Variable auf *true* (Codeausschnitt 7 Zeile 3). In den Zeilen Vier und Fünf werden zuerst das Radar entfernt und danach das *locationView*. Das *locationView* stellt die Anzeige dar, auf der alle POIs zuvor gezeichnet wurden. Die Veränderung an der *isPaused* Variable führen bei der *onHeadingWithLocationsChanged*-Methode bei jedem Aufruf zum sofortigem Ende. Dies hat zu Folge, dass der Inhalt der *locationView* Variable nicht mehr aktualisiert wird und so pausiert wirkt. Die *createPOIGrid*-Methode lädt also alle POIs aus der *locationView*-Variable und macht daraus mit der Hilfe eines *AREAGridPOIAdapter* einen GridView und zeigt diesen an. Die *AREAGridPOIAdapter* ist dabei nur eine einfache Schnittstelle für das GridView, um die Liste von POIs verwenden zu können.

Codeausschnitt 8 unpause-Methode

```
1private void unpause(){
2 if(isPaused){
3  isPaused = false;
4  mainLayout.removeView(poiGrid);
5  mainLayout.addView(userinterfaceLayout);
6  mainLayout.addView(locationView);
7  locationView.bringToFront();
8  userinterfaceLayout.bringToFront();
9}}
```

Die *unpause* Methode macht dann dafür genau das Gegenteil. Die *isPaused*-Variable wird wieder auf *false* (Codeausschnitt 8 Zeile 3) gesetzt. Weiterhin werden in den Zeilen Vier bis Sechs

der GridView wieder entfernt und das Radar und die POI-Anzeige werden wieder auf dem Bildschirm angezeigt.

Die beiden Zeilen Sieben und Acht stellen sicher, dass das Radar immer als letztes gezeichnet wird, so dass es immer im Vordergrund ist.

3.2.2 Probleme

Auch dieser Lösungsansatz war vom Problem aus Abschnitt 3.1.3.1 betroffen, die dort beschriebene Lösung funktionierte auch hier und wird deshalb nicht ein zweites mal aufgeführt.

3.2.2.1 Pause

Durch die Vorarbeit aus dem ersten Lösungsansatz, traten hier kaum mehr unbekannte Probleme auf. Das einzig wirklich neue Problem war: Verlässt man die AR-Ansicht im pausierten Zustand, kehrt also wieder zurück zur Kartenansicht, so stürzt die Anwendung bei dem erneuten Betreten der AR-Ansicht ab. Es muss also sichergestellt werden, dass die AR-Ansicht beim Verlassen den pausierten Zustand auch verlässt.

Dazu ist eine Anpassung der *AREAMainActivity* Klasse notwendig. In der *onOptionsItemSelected* Methode muss das *super.onBackPressed()* zu einem *onBackPressed()* geändert werden und die Klasse muss um eine *onBackPressed* Methode erweitert werden (siehe Codeausschnitt 9).

Codeausschnitt 9 onBackPressed-Methode

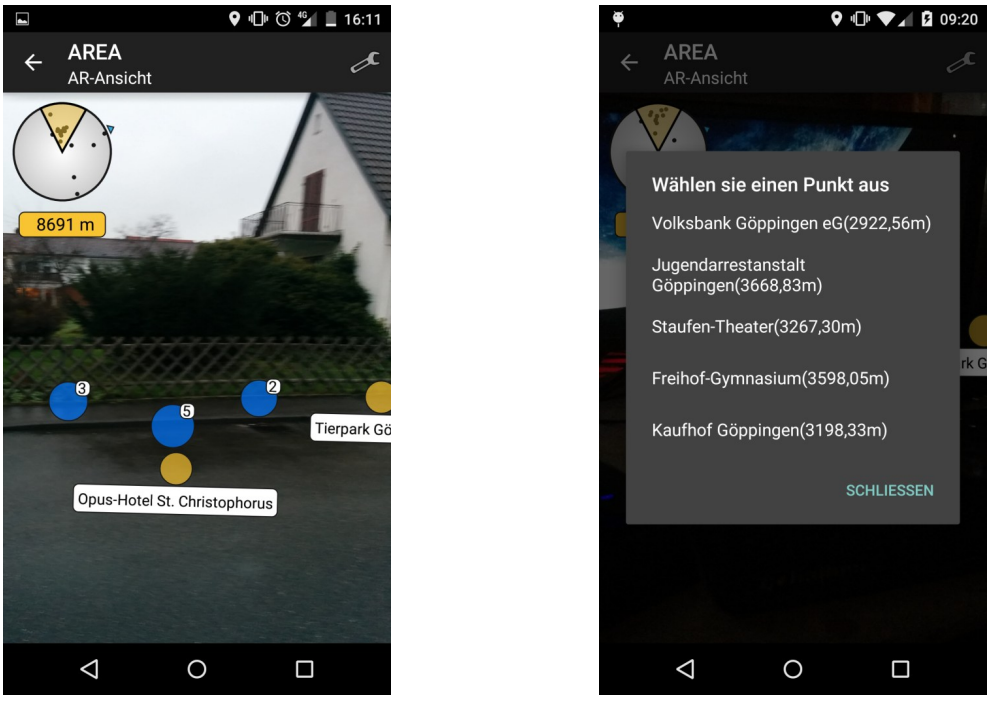
```
1@Override
2public void onBackPressed(){
3 unpause();
4 locationView.removeAllViews();
5 super.onBackPressed();
6}
```

So wird jedes Mal, wenn der Benutzer die AR-Ansicht verlässt, sichergestellt, dass der pausierte Zustand beendet wird (Codeausschnitt 9 Zeile 3).

4 Vorstellung der Anwendung

4.1 Clustering

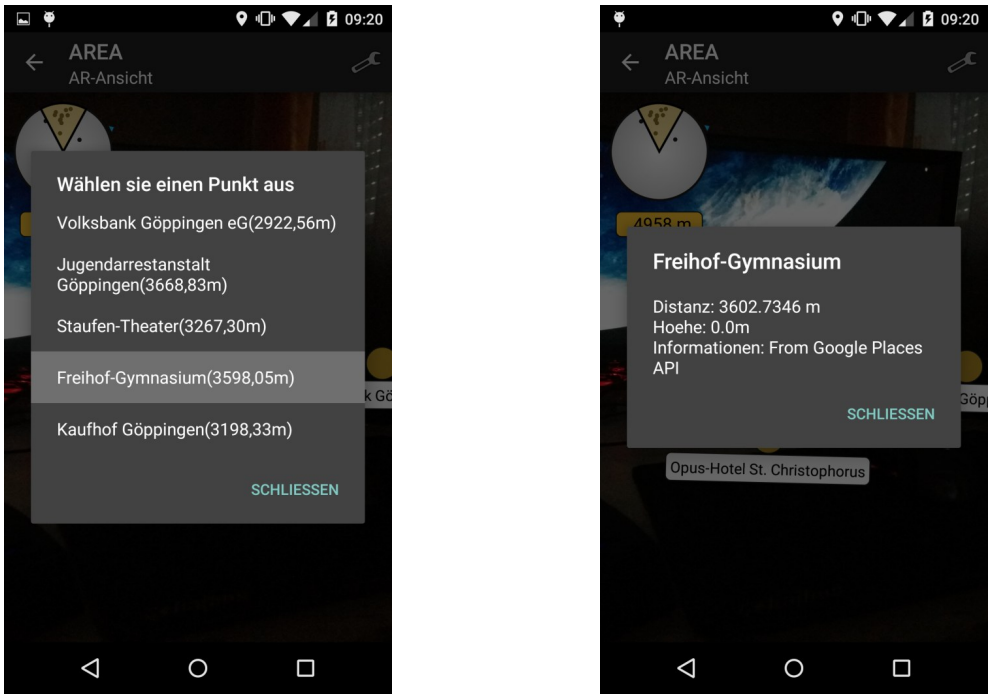
Da die Clustering Lösung kaum die Art und Weise der Bedienung im Vergleich zur unveränderten AREA-Anwendung ändert, muss nicht viel Neues vorgestellt werden.



(a) AR-Ansicht mit Cluster **(b)** Cluster mit 5 POIs angeklickt

Abbildung 4.1: Clusteransicht

Die AR-Ansicht wird nun anstatt viele überlappende POIs nur noch wenige Cluster, wie auf Abbildung 4.1a zu sehen ist, anzeigen. Jeder dieser Cluster erhält ein Label mit der Anzahl der POIs in dem Cluster. Beim Anklicken eines solchen Clusters wird dem Benutzer eine Liste der POIs, wie in Abbildung 4.1b, aus dem ausgewählten Cluster angezeigt.



(a) Auswahl eines POI aus der Liste **(b)** Informationen über den ausgewählten POI

Abbildung 4.2: Cluster Bedienung

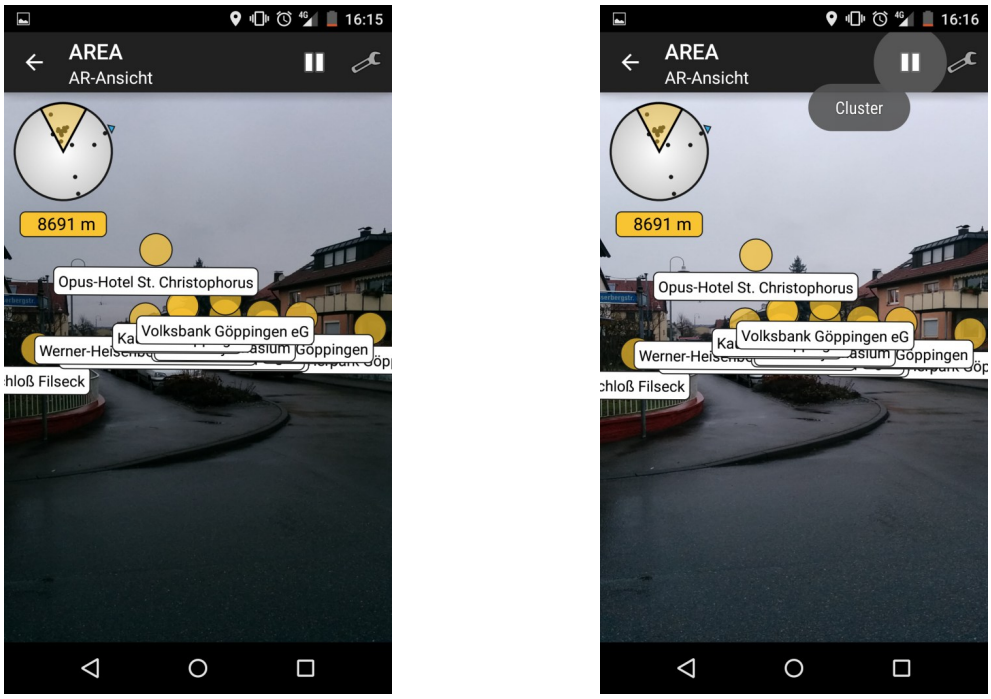
Aus der Liste wählt nun der Benutzer einen POI aus (siehe Abbildung 4.2a) zu dem er mehr Informationen erhalten möchte. Ein weiteres Popup zeigt (siehe Abbildung 4.2b) dem Benutzer dann die Informationen zum gewünschten POI an.

4.1.1 Abgleich mit den Anforderungen

Die Anforderung an die Bedienung wurde erfüllt, es wurde lediglich ein weiterer Dialog eingefügt, der nach dem Anklicken eines Clusters auftaucht und die Liste von POIs anzeigt, die sich im Cluster befinden. Im Bezug auf die Echtzeitberechnung, gab es auf beiden Testgeräten zu keinem Zeitpunkt Probleme, weil die Berechnungen zu lange gedauert haben. Es muss zwar bei jedem ankommenden POI geprüft werden, ob sich dieser in einem Cluster befindet, aber dafür müssen weniger Punkte (POIs) gezeichnet werden. Bei wie vielen Punkten dieser Algorithmus in die Knie gezwungen wird, hängt vom Gerät selber ab.

4.2 Play/Pause-Methode

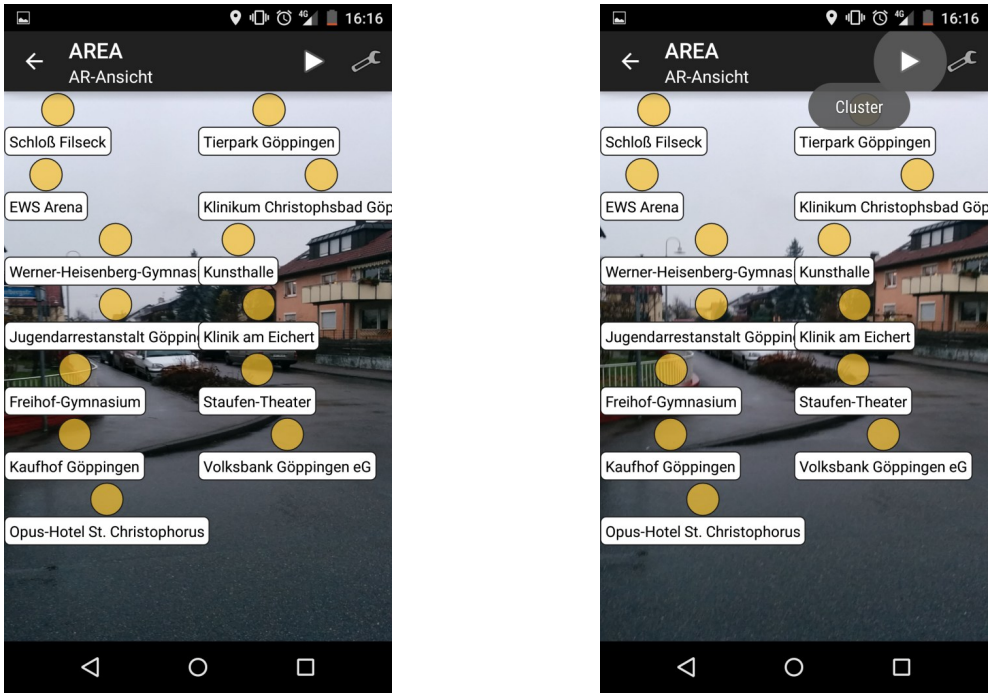
Der Play/Pause-Lösungsansatz greift da schon mehr in die Art der Bedienung ein. Hier wird auf den ersten Blick kaum was an der AR-Ansicht verändert, lediglich ein Pauseknopf wird dem Menü hinzugefügt (siehe Abbildung 4.3a). Beim Drücken des Pauseknopfs (siehe Abbildung 4.3b) wird das aktuelle Bild angehalten, das Radar ausgeblendet und alle sichtbaren POIs werden in einem GridView aufgefächert (siehe Abbildung 4.4a). Auch der Pauseknopf ändert sich zu einem Playknopf.



(a) Die normale AR-Ansicht mit Pauseknopf **(b)** Pauseknopf gedrückt

Abbildung 4.3: Pause

In diesem pausierten Modus sind alle POIs einfach zu finden und auch einfach anzuklicken. Falls die Anzahl der Punkte den Rahmen des Bildschirms überschreiten würden, lässt sich das GridView auch scrollen.



(a) Pausiertes Bild **(b)** Playknopf gedrückt

Abbildung 4.4: Play

Um wieder in die normale, nicht pausierte AR-Ansicht zu kommen, muss man einfach nur den Playknopf drücken (siehe Abbildung 4.4b) und das GridView wird wieder entfernt, Radar wird wieder eingeblendet und der Playknopf wird wieder zum Pauseknopf.

4.2.1 Abgleich mit den Anforderungen

Die Anforderungen an die Bedienung wurden erfüllt. Auf den ersten Blick scheint die Bedienung zwar nicht üblich für eine AR-Anwendung, aber sie ist auf keinen Fall kompliziert, eine kurze Erklärung sollte jedem genügen, um die Bedienung zu verstehen. Auch die Anforderung an die Ressourcen sind erfüllt: Es muss nichts extra berechnet werden, die Anwendung arbeitet genau gleich wie die unveränderte AREA-Anwendung, bis der Pauseknopf gedrückt wird. Danach wir das gesamte Aktualisieren der POIs angehalten, was nur noch mehr Ressourcen freigibt. Das kurze Iterieren über alle bereits angezeigten POIs und das Erstellen des GridView ist kaum ein Aufwand und geschah auf beiden Testgeräten ohne jegliche Verzögerung.

5
Zusammenfassung und Ausblick

Die Aufgabe dieser Arbeit bestand darin, mögliche Lösungen für das Problem der überlappenden POIs in der AR-Ansicht zu finden und diese prototypisch zu implementieren. Im Verlauf der Arbeiten wurden zwei mögliche Methoden vorgestellt und implementiert. Beide Ansätze haben die gestellten Ziele erreicht, aber sicher bestehen an einigen Stellen noch Verbesserungsmöglichkeiten. Beim ersten Ansatz könnte man einen anderen, vielleicht auch effizienteren Algorithmus suchen, der überlappende POI findet und zu Cluster zusammenfasst. Für den zweiten Ansatz wäre eine optische Verbesserung denkbar. Das Problem am GridView ist, dass man nicht sehen kann, wo der POI vor der Auffächerung war, denkbar wäre hier, die POIs im GridView transparent zu machen und vielleicht Linien zu ihren alten Positionen zu ziehen.

Ein paar weitere Ansätze wurden noch durch andere AR-Anwendungen vorgestellt. Was aus der Arbeit aber klar wird, so etwas wie eine Universallösung wird es nicht geben, für jede Anwendung muss fast immer eine eigene oder eine angepasste Lösung entwickelt werden. Bei AREA zum Beispiel sind alle POIs ohne Kategorie, was zur Folge hat, dass immer viele POIs auf dem Bildschirm dargestellt werden. Dies wiederum heißt, dass sich für AREA besser die Lösungen eignen, die den Bildschirm "aufräumen", um die Bedienung der Anwendung zu erleichtern. Als eine Idee für die Zukunft wäre also vielleicht die Implementierung eines Kategoriensystems für die POIs zu realisieren. Damit würden sich noch weiter angepasste Lösungen finden und Implementieren lassen.

Literaturverzeichnis

[All14] Open Handset Alliance. *Android*, 2014. http://www.android.com/.

[Fou14a] OpenStreetMap Foundation. *OpenStreetMap*, 2014. http://www.openstreetmap.org/about.

[Fou14b] Wikimedia Foundation. *Wikipedia*, 2014. http://de.wikipedia.org/wiki/Wikipedia.

[Gei12] P. Geiger. Entwicklung einer Augmented Reality Engine am Beispiel des iOS. *Bachelor thesis, Ulm University*, 2012.

[Gmb14a] Metaio GmbH. *Junaio*, 2014. http://www.junaio.com/.

[Gmb14b] Metaio GmbH. *Metaio GmbH*, 2014. http://www.metaio.com/home/.

[Gmb14c] Metaio GmbH. *Metaio SDK*, 2014. http://www.metaio.com/products/sdk/.

[GPSR13] P. Geiger, R. Pryss, M. Schickler, and M. Reichert. Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices. *Technical Report (UIB-2013-09), Ulm University*, 2013.

[GSP+14] P. Geiger, M. Schickler, R. Pryss, J. Schobel, and M. Reichert. Location-based Mobile Augmented Reality Applications: Challenges, Examples, Lessons Learned. In *10th Int'l Conf on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*, pages 383–394, 2014.

[Inc14a] Apple Inc. *Apple iOS*, 2014. https://www.apple.com/de/ios/what-is/.

[Inc14b] Google Inc. *Android SDK*, 2014. http://developer.android.com/sdk/index.html.

[Inc14c] Google Inc. *Google Inc*, 2014. https://www.google.com/intl/de_de/about/company/.

[Inc14d] Google Inc. *Google Maps API*, 2014. https://developers.google.com/maps/?hl=de.

[Inc14e] Google Inc. *Google Nexus 5*, 2014. http://www.google.com/nexus/5/.

[Inc14f] Twitter Inc. *Twitter*, 2014. https://about.twitter.com/.

[Mic14] Sun Microsystems/Oracle. *Java*, 2014. https://www.java.com/de/download/whatis_java.jsp.

[Ora14] Oracle. *Sun Microsystems*, 2014. http://www.oracle.com/us/sun/index.html.

[PLRH12] R. Pryss, D. Langer, M. Reichert, and A. Hallerbach. Mobile Task Management for Medical Ward Rounds - the MEDo Approach. In *1st Int'l Workshop on Adaptive*

Case Management (ACM'12), BPM'12 Workshops, LNBIP, pages 43–54. Springer, 2012.

[PMLR14] R. Pryss, N. Mundbrod, D. Langer, and M. Reichert. Supporting Medical Ward Rounds Through Mobile Task and Process Management. *Information Systems and e-Business Management*, 2014.

[PTKR10] R. Pryss, J. Tiedeken, U. Kreher, and M. Reichert. Towards Flexible Process Support on Mobile Devices. In *Proc CAiSE'10 Forum - Information Systems Evolution*, number 72 in LNBIP, pages 150–165. Springer, 2010.

[Sam14] Samsung. *Samsung Galaxy S2*, 2014. `http://www.samsung.com/de/consumer/mobile-device/mobilephones/archive-mobile-phones/GT-I9100LKADBT`.

[Srl14a] Peer Srl. *mixare*, 2014. `http://www.mixare.org/`.

[Srl14b] Peer Srl. *mixare*, 2014. `https://code.google.com/p/mixare/wiki/Roadmap`.

[SSP+13] J. Schobel, M. Schickler, R. Pryss, H. Nienhaus, and M. Reichert. Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned. In *9th Int'l Conf on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps*, pages 509–518, 2013.

[SSP+14] J. Schobel, M. Schickler, R. Pryss, F. Maier, and M. Reichert. Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In *10th Int'l Conf on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*, pages 371–382, 2014.

[Wik14] Wikipedia. *k-Means-Algorithmus*, 2014. `http://de.wikipedia.org/wiki/K-Means-Algorithmus`.