Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 50

Gestensteuerung von Routenplanern mittels Kinect

Martin Scholz

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Stefan Funke
Betreuer:	Dipl.-Inf. Jochen Eisner
Beginn am:	10. April 2013
Beendet am:	10. Oktober 2013
CR-Nummer:	J.7

Kurzfassung

In dieser Arbeit wird ein System zur Steuerung einer Landkarte mittels Gesten vorgestellt. Die Gestenerkennung findet dabei mittel der Kinect und dem OpenNI SDK statt. Die Steuerung der Karte umfasst das Bewegen der Karte, das Hinein- beziehungsweise Herauszoomen, sowie das Setzen von Routenpunkten und damit eine Routenberechnung. Für das Berechnen der Routen wird ToureNPlaner, ein Projekt, das im Rahmen eines Studienprojekts an der Universität Stuttgart entwickelt wurde, verwendet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Seit der Einführung der Kinect, einem Gerät, mit dem Gestensteuerung auf der Spielekonsole Xbox 360 ermöglicht wurde, hat diese Art der Steuerung stark an Popularität gewonnen. Die Idee der Gestensteuerung wurde bisher gern als futuristische Steuerung von Geräten angesehen. So wurde beispielsweise auch schon im Film "Minority Report"mittels Gestensteuerung auf einem großen, neuartigen Bildschirm nach Hinweisen zu Verbrechen gesucht. Mit der Kinect wurde diese Art der Steuerung für jedermann zugänglich. Allerdings war sie primär fürs Spielen gedacht. So gibt es beispielsweise Tanzspiele für die Xbox 360 (*Dance Central*), bei denen man den eigenen Körper synchron zu einer virtuellen Figur bewegen muss, oder Spiele, bei denen Kinder durch Gesten mit einem animierten Tier interagieren können. Microsoft, der Entwickler der Kinect, sorgte jedoch zeitnah dafür, dass die Kinect auch auf PCs betrieben werden konnte. So wurden schnell auch Anwendungen entwickelt, die die Kinect auf ganz andere Art nutzen. Es gibt beispielsweise ein Projekt, bei dem ein motorbetriebener Einkaufswagen Rollstuhlfahrern im Supermarkt folgt, um deren Einkauf zu erleichtern. Dabei wird die Kinect dazu genutzt, die Bewegung des Rollstuhls zu erfassen, damit der Einkaufswagen folgen kann [kin13b]. Ein anderes Projekt, benutzt die Kinect, um virtuell Kleidung anprobieren zu können, indem das Kamerabild des Benutzers mit Bildern von Kleidungsgegenständen überlagert wird. Dabei wird die Kinect genutzt, um die Bewegung des Benutzers zu verfolgen und dadurch auch die Ansicht der virtuelle Kleidung entsprechend anzupassen [kin13a].
All diese Projekte gibt es unter Anderem auch deshalb, da die Kinect trotz der komplexen Technik bezahlbar ist. Mit rund 100 Euro ist sie günstig genug, um beispielsweise von Schulen an Schüler ausgegeben zu werden, um neue Anwendungsfälle zu erforschen.

In dieser Arbeit wird die Kinect dazu genutzt, mittels Gesten eine Landkarte zu steuern. Dabei ist es möglich, die Karte zu bewegen, hinein- und herauszuzoomen, sowie Routen berechnen zu können. Dieses System wurde *MapKin* genannt und soll im Rahmen von Veranstaltungen wie dem Tag der Wissenschaft an der Universität Stuttgart vorgeführt werden.

1.1 Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Beschreibung: Im zweiten Kapitel werden die verschiedenen Komponenten des Systems erklärt und die erworbenen Erkenntnisse vom Tag der Wissenschaft erläutert.

Kapitel 3 – Implementierung Das dritte Kapitel beschreibt, wie bei der Implementierung vorgegangen wurde.

Kapitel 4 – Inbetriebnahme Im vierten Kapitel wird erklärt, wie MapKin ausgeführt und benutzt werden kann. Zusätzlich beinhaltet es Anleitungen zur Installation der benötigten Softwarekomponenten.

Kapitel 5 – Zusammenfassung und Ausblick: Im letzten Kapitel wird eine Aussicht über mögliche Erweiterungen von MapKin gegeben.

2 Beschreibung

2.1 Externe Abhängingkeiten

Für die Ausführung von MapKin werden verschiedene Komponenten benötigt, die in diesem Kapitel beschreiben werden. Dazu gehört sowohl die Kinect, als auch diverse Softwarekomponenten. Hier wird jedoch nur die Funktion der Softwarekomponenten erläutert – die Integration der Komponenten in MapKin wird im Kapitel *3.1* auf Seite 25 beleuchtet.

2.1.1 Hardware

Kinect

Die Kinect ist eine Erweiterung für die Spielekonsole Xbox 360, die von Microsoft in Zusammenarbeit mit PrimeSense entwickelt wurde, um das Spielen durch den Einsatz des ganzen Körpers interessanter zu gestalten. Sie ist seit November 2010 erhältlich.

Um die technische Seite der Kinect zu erläutern, zeigt Abbildung 2.1 eine Übersicht über die verbauten Elemente.

Abbildung 2.1: Bild der Kinect mit den wichtigsten Komponenten. Zu erkennen sind die Infrarotkamera, die Farbkamera, sowie der Infrarotprojektor. Die restlichen Elemente befinden sich im Inneren der Kinect. Die entsprechenden Positionen sind vermerkt. Quelle: in Anlehnung an [kin13c]

Die *Infrarotkamera* und der der *Infrarotprojektor* sind für die Erkennung der Gesten erforderlich. Der Projektor projiziert Infrarotpunkte in den Raum vor der Kinect. Diese Punkte sind für das menschliche Auge nicht sichtbar. Die Infrarotkamera kann sie jedoch erfassen und mithilfe von Software ein dreidimensionales Bild aus diesen Daten errechnen. Die Infrarotkamera besitzt eine Auflösung von 640 x 480 Pixel. Die Tiefeninformationen werden mit 11 Bit übertragen, was 2048 verschiedene Tiefenstufen ermöglicht. Für die Kühlung des Infrarotprojektors sorgen das *Peltierelement* und der *Lüfter*. Die Wärme im Innenraum wird vom Lüfter, der an der Seite der Kinect angebracht ist, nach außen abgeführt [iFi13].
Die *Farbkamera* liefert ein gewöhnliches Farbbild in einer Auflösung von 640 x 480 Pixel bei einer Widerholungsrate von 30Hz. Dieses Farbbild wird unter anderem dazu genutzt, dem Benutzer vor der Kamera zu zeigen, wie ihn die Kinect erfasst. Dadurch ist es möglich, einzuschätzen, ob der Benutzer für die Kinect vollständig sichtbar ist, oder beispielsweise zu nah vor der Kamera steht. Die Xbox 360 verwendet die Farbkamera auch oft dazu, ein Bild vom Spieler zu machen, um es beispielsweise als Avatar zu verwenden. MapKin nutzt die Farbkamera nicht.
Die vier, in der Kinect verbauten, *Mikrofone* können zur Sprachsteuerung genutzt werden. Eine Software kann durch die an verschiedenen Stellen positionierten Mikrofone ermitteln, aus welcher Richtung gesprochen wird. Dadurch ist es möglich, das Gesprochene bei mehreren Benutzern einer bestimmten Person zuzuordnen. Auch die Mikrofone finden in diesem Projekt keine Verwendung. Es ist jedoch denkbar, sie später für die Suche nach Städtenamen zu verwenden (Siehe dazu Kapitel *5.3.2* auf Seite 49).
Mithilfe des *Neigungsmotors* kann die Kinect horizontal geneigt werden. Dies hilft dabei, den Benutzer stets vollständig erfassen zu können. Die Xbox 360 ist in der Lage, den Winkel der Kinect selbstständig anzupassen um Personen zu folgen, wenn sich deren Abstand zur Kinect ändert. Dieses Projekt würde von der Nutzung des Motors profitieren – gerade weil im Rahmen von Präsentationen des Projekts an der Universität Stuttgart verschieden große Personen das Projekt ausprobieren wollen. Da die Ansteuerung des Motors jedoch nicht ohne weiteres möglich ist, wurde darauf verzichtet, den Motor zu nutzen.

Da der Motor mehr als die für USB spezifizierten 500 mA benötigt, muss die Kinect bei der Nutzung an einem PC zusätzlich mit einem Netzteil versorgt werden. Deshalb besitzt die Kinect keinen USB-Stecker, sondern einen proprietären Anschluss (siehe Abbildung 2.2). Dieser wird an ein Netzteil angeschlossen, welches einen USB-Stecker besitzt. Beim Anschluss an die Xbox 360 der ersten Generation muss dieses Netzteil ebenfalls verwendet werden. Erst spätere Generationen bieten einen Anschluss, der die Kinect direkt mit der Xbox 360 verbindet.

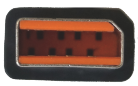


Abbildung 2.2: Bild des propriertären Steckers der Kinect

2.1.2 Software

Microsoft Kinect SDK

Im Jahr 2011 veröffentlichte Microsoft ein Software Development Kit (SDK), um die Kinect auch auf einem Windows PC nutzen und eigene Programme in C++, C# und Visual Basic dafür schreiben zu können [eng13]. Dieses SDK beinhaltete auch Treiber, um die Kinect auf Windows-PCs betreiben zu können. Vorerst durfte das SDK nur für nicht-kommerzielle Anwendungen genutzt werden. Später wurde eine weitere Version veröffentlicht, die die Entwicklung kommerzieller Software erlaubte.

OpenNI und NiTE

Da MapKin in Java entwickelt werden sollte und das SDK von Microsoft dieses nicht unterstützt, wurde OpenNI für die Implementierung gewählt. Dabei handelt es sich um ein SDK, das von PrimeSense, dem Mitentwickler der Kinect, veröffentlicht wurde. Um die Kinect mittels dieses SDKs in Java nutzen zu können, sind folgende Softwarekomponenten erforderlich, deren Funktionen und Aufgaben im weiteren Verlauf genauer erklärt werden:

- OpenNI SDK
- NiTE Middleware
- Treiber für die Kinect

Beim OpenNI SDK muss zwischen zweierlei Versionen unterschieden werden. Die neuste Version des SDKs trägt die Versionsnummer 2. Zum Zeitpunkt der Erstellung dieser Arbeit befand sich Version 2 jedoch noch im Beta-Stadium. Da Verwendung von Software im Beta-Stadium jedoch vermieden werden sollte, um nicht mit unvorhersehbaren Problemen konfrontiert zu werden, wurde die letzte stabile Version, Version 1.5, gewählt. Version 2 wurde im Vergleich zu Version 1.5 stark vereinfacht. Dadurch hat sich auch das API grundlegend verändert und ist nicht abwärtskompatibel zu Version 1.5 geblieben. Eine graphische Darstellung der Unterschiede zwischen den beiden Versionen findet sich in Abbildung 2.4. Um MapKin auch mit der aktuellen Version von OpenNI verwenden zu können, wurde zusätzlich eine zweite Version entwickelt, die mit der neueren Version von OpenNI kompatibel ist. Der Funktionsumfang und die Bedienung der beiden Versionen sind absolut identisch. Es musste lediglich die Logik angepasst werden.
Leider reicht es nicht aus, nur das jeweilige OpenNI SDK auszuwechseln. Auch der Treiber für die Kinect, sowie die NiTE Middleware müssen zur OpenNI-Version passen. Je nach Version ändern sich auch die Aufgaben der Softwarekomponenten. Deshalb wird die folgende Erklärung aufgeteilt. Eine Beschreibung zur Installation der Komponenten findet sich in Kapitel *4.1* auf Seite 35.

OpenNI Version 1.5

Der *Treiber* für die Kinect stammt in diesem Fall von PrimeSense. Es handelt sich hierbei jedoch um eine angepasste Version. Er kann ausschließlich von einem GitHub-Repository [Sen13] heruntergeladen werden und steht für Linux, sowie Windows 7 zur Verfügung. Da der Treiber nach der Zertifizierung verändert wurde, lässt er sich nicht problemlos unter Windows 8 installieren. Die Funktion des Treibers besteht darin, mit der Kinect zu kommunizieren. Das bedeutet, dass einerseits Steuerbefehle an die Kinect gesendet, andererseits die Rohdaten von der Kinect ausgelesen werden können.
Das *OpenNI SDK* bietet die Möglichkeit, auf einer hohen Abstraktionsebene mit standardisierten Tiefensensoren zu kommunizieren. Da dieser Standard von PrimeSense definiert wurde, ist die Kinect kompatibel dazu. OpenNI ermöglicht es beispielsweise, die Kinect durch das API zu initialisieren, ohne dabei direkt den Treiber ansprechen zu müssen. Außerdem erhält man unter anderem einfachen Zugriff auf die Videostreams des Tiefensensors und der Farbkamera.
Des Weiteren können Plug-Ins mit dem API kommunizieren und OpenNI um Funktionen erweitern, welche jedoch weiterhin über die API von OpenNI angesprochen werden. Zu diesen Plug-Ins gehört das ebenfalls von PrimeSense entwickelte *NiTE*. Dieses erweitert OpenNI um eine Personen- und Gestenerkennung. Die rohen Videostreams der Kinect enthalten noch keine semantischen Informationen. NiTE erkennt in dem Tiefenbild Personen und die Positionen ihrer Gliedmaßen, also z.B. die der Hände, der Schultern oder des Kopfes. Diese Positionen werden Joints genannt und lassen sich über die API als dreidimensionale Punkte abrufen. Die Summe dieser Punkte ergibt ein sogenanntes „Skelett“ (siehe Abbildung 2.3). Die Informationen, die aus diesem Skelett ausgelesen werden können, reichen aus, um verschiedene Gesten, die der Benutzer ausführt, erkennen zu können.

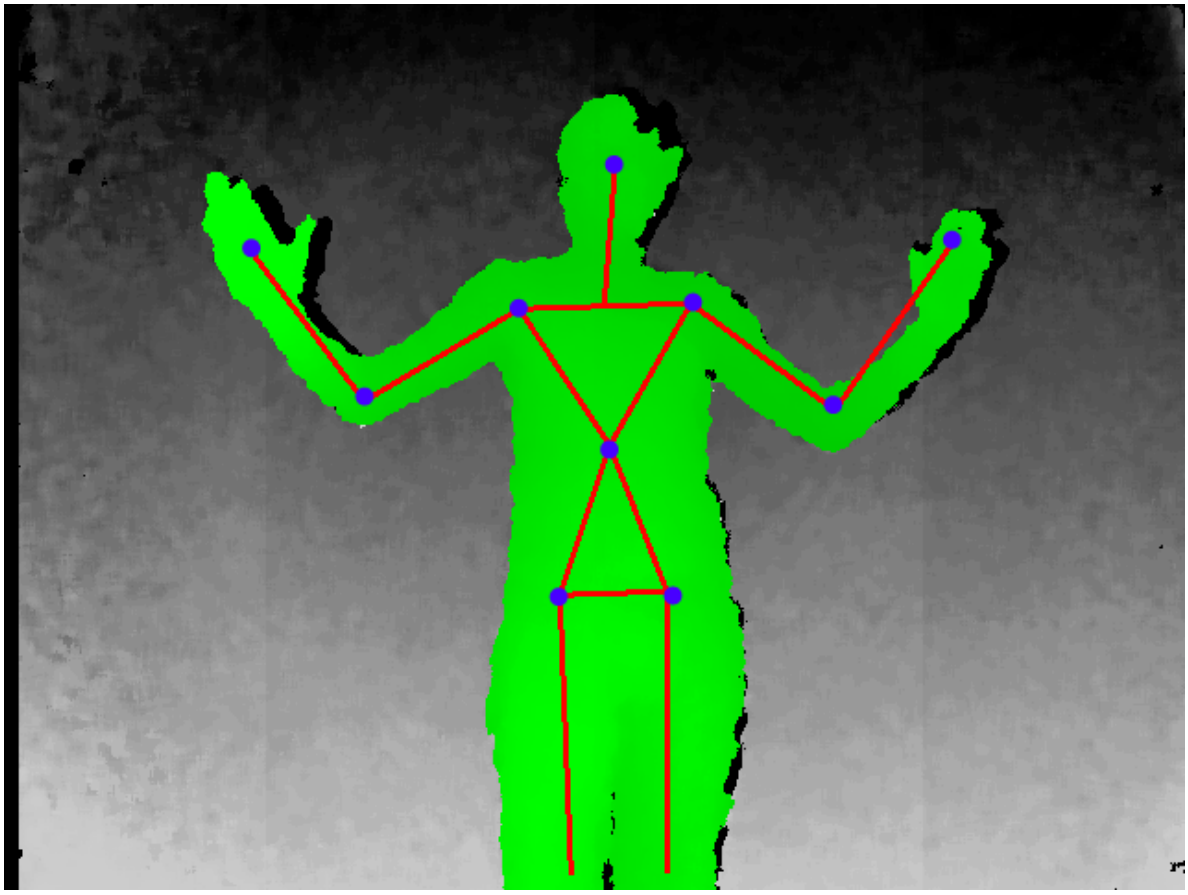


Abbildung 2.3: Bild eines Skeletts. Ein Skelett ist die Summe, der von NiTE zur Verfügung gestellten Punkten. Die Punkte repräsentieren Gliedmaßen oder Gelenke und werden Joints genannt. Sie sind hier als blaue Punkte dargestellt. In diesem Bild sind folgende Joints zu erkennen: Kopf, Torso, beide Hände, beide Ellenbogen, beide Schultern, beide Hüftgelenke. Die Informationen des Skeletts werden verwendet, um Gesten zu erkennen.

OpenNI Version 2

Da der Treiber umständlich zu beziehen ist und er auf dem aktuellen Windows Betriebssystem, Windows 8, nur über Umwege zu installieren ist, fiel die Entscheidung, die zweite Version zu entwickeln, die diese Defizite nicht aufweist.
Für OpenNI 2 muss der Microsoft Treiber für die Kinect verwendet werden. Dieser ist Teil des Kinect SDKs von Microsoft und dementsprechend einfach über die Internetseiten von Microsoft zu beziehen [Mic13a]. Um Version 2 mit Linux verwenden zu können, muss auf den Freenect zurückgegriffen werden [Cup13]. Freenect ist ein Treiber, der libnect, eine Opensource-Bibliothek für die Verwendung der Kinect, verwendet.
Das OpenNI SDK wurde in dieser Version grundlegend verändert. Es fungiert nur noch als

reines API und bietet keine Möglichkeit mehr Plug-Ins einzubinden. Trotz der Vereinfachung des APIs von OpenNI sind die grundlegenden Funktionen identisch geblieben.
Da OpenNI kein Plug-In System mehr anbietet, musste auch NiTE angepasst werden. Obwohl sich auch hier die Funktionalität nicht geändert hat, musste dessen API angepasst werden. NiTE erweitert OpenNI nicht mehr als Plug-In, sondern wird direkt durch eine API angesprochen. Es kommuniziert also „nach unten“ mit OpenNI um an Daten zu gelangen, während es „nach oben“ Informationen in einem hohen Abstraktionslevel an Anwendungen weitergibt. In OpenNI 1.5 muss MapKin ausschließlich das API von OpenNI einsetzen, wohingegen in OpenNI 2 zusätzlich die API von NiTE eingebunden werden muss.
Eine Kombination von OpenNI 1.5 mit NiTE 2 oder umgekehrt ist nicht möglich. Auch die Treiber müssen zur jeweiligen OpenNI-Version passen.

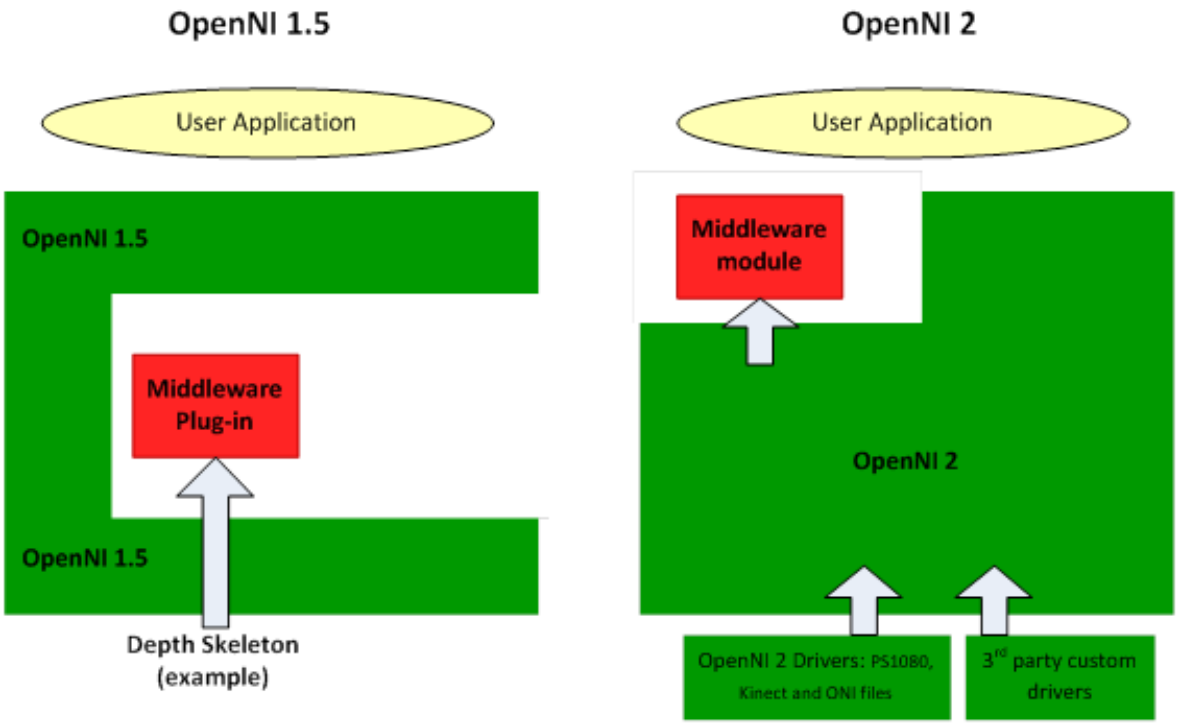


Abbildung 2.4: Veranschaulichung der Unterschiede bei der Einbindung von NiTE in Version 1.5 und Version 2 von OpenNI. Links ist die OpenNI Version 1.5 zu sehen. Die Middleware-Bibliotheken (darunter NiTE) werden durch das API von OpenNI angesprochen, welches die Middleware „umgibt“. Rechts, in Version 2, ist zu erkennen, dass das API der Middleware nun direkt von Anwendungen benutzt werden kann, ohne über das API von OpenNI zu kommunizieren. Quelle [ope13b]

JXMapViewer

JXMapViewer ist eine Swing-Komponente, die es auf einfache Weise erlaubt, Karten darzustellen. Es ist dabei möglich, verschiedene Kartenquellen zu nutzen. Im Fall von MapKin stammen die Kartendaten von OpenStreetMap (siehe OpenStreetMap Homepage [osm13]) und werden bei Bedarf vom entsprechenden Server heruntergeladen. Die Kartendaten werden als vorgerenderte Tiles, kleine viereckige Kartenausschnitte, übertragen. Es werden jeweils nur die aktuell benötigten Tiles geladen. JXMapViewer cached sie jedoch, wodurch bereits geladene Tiles nicht erneut übertragen werden müssen. Somit kann MapKin nicht offline betrieben werden. Sollte keine Internetverbindung bestehen, zeigt JXMapViewer nur graue Platzhalter-Tiles an.
Die Steuermöglichkeiten von JXMapViewer beschränken sich hauptsächlich auf das Zoomen und Verschieben der Karte. Außerdem ist es möglich, eine Minikarte zur Übersicht, sowie einen Schieberegler zum Einstellen des Zoomlevels einzublenden.

TourenPlaner

TourenPlaner ist eine Anwendung, die es erlaubt im Browser oder mittels einer Android-App Routen zu berechnen [Tou13]. Diese Routen können durch zusätzliche Bedingungen, wie beispielsweise einen minimalen Höhenunterschied, genauer definiert werden. Da diese Bedingungen für MapKin nicht relevant sind, wird an dieser Stelle nicht näher darauf eingegangen. TourenPlaner wurde im Rahmen eines Studienprojekts 2012 an der Universität Stuttgart entwickelt. Gerade deshalb bot es sich an, TourenPlaner zur Routenberechnung für MapKin zu verwenden. Die Integration von TourenPlaner gestaltet sich sehr einfach, da es ausreicht, dem API von TourenPlaner zwei Koordinaten zu übergeben. Daraus wird dann eine Route berechnet, die direkt über die Karte gezeichnet wird. TourenPlaner benutzt, wie JXMapViewer, OpenStreetMap-Daten für die Routenberechnung. Diese Daten werden von der Universität Stuttgart auf einem Server gehostet.

2.2 Die Gesten

Zu Beginn musste festgelegt werden, welche Gesten für die verschiedenen Aktionen benutzt werden sollten. Nachdem feststand, dass es möglich sein sollte die Karte zu verschieben, hinein- und herauszuzoomen, sowie Routen berechnen zu können, musste für jede dieser Aktionen eine Geste festgelegt werden. Die Gesten sollten intuitiv erscheinen und gleichzeitig möglichst einfach mittels NiTE zu erkennen sein. Da es dieselben Aktionen auch in modernen Karten-Applikationen auf Geräten mit Touchscreen gibt, wurde entschieden, die Gesten daran anzulehnen, da diese bereits allgemein bekannt sind. Hierdurch wird das Erlernen der Benutzung von MapKin erleichtert.
Ein Problem bei der Gestensteuerung besteht darin, dass der Benutzer keine unmittelbare Rückmeldung erhält, da er im eigentlichen Sinne mit nichts interagiert. Auf Touchscreens

hingegen spürt man, wann man den Bildschirm berührt und man kann sehen, auf welche Position der Finger zeigt. Aus diesem Grund musste eine Möglichkeit geschaffen werden, dem Benutzer zu zeigen, wie ihn die Kinect wahrnimmt. Dies wurde durch sogenannte Joint-Indikatoren umgesetzt. Dabei handelt es sich um farbige Punkte, welche die Positionen der Joints, insbesondere die der Hände, auf dem Bildschirm anzeigen. Während des Kalibrierungsprozesses werden diese Indikatoren direkt in das Tiefenbild gezeichnet. Bei der Kartenansicht erscheinen die Punkte direkt auf der Karte. Somit ist es für den Benutzer möglich, zu sehen, was er auf der Karte auswählt.
In der Kalibrierungsansicht werden der Kopf, der Torso und beide Hände durch die zuvor genannten Indikatoren dargestellt. Dies dient einerseits dazu, den Kalibrierungsprozess einfacher abschließen zu können, andererseits erlernt der Benutzer dadurch die Funktion der Indikatoren. In der Kartenansicht sind nur die beiden Hände durch Joint-Indikatoren visualisiert. Grundsätzlich werden von der Gestenerkennung in MapKin pro Hand drei verschiedene Status erkannt, welche von der Position der Hand abhängen. Je nach Status der Hand des Benutzers, erhält der zugehörige Indikator eine andere Farbe:

- *vorne (grün)*
- *hängend (gelb)*
- *sonstiges (rot)*

Der Status *vorne* hängt von der Armlänge ab. Wird der Arm vor dem Körper ausgestreckt, wird er ab einem bestimmten Prozentsatz der Gesamtarmlänge als *vorne* erkannt. Um die Entfernung, ab der die Hand als *vorne* erkannt wird, festzulegen, wird die Armlänge während der Kalibrierung gemessen. (Eine genaue Beschreibung des des Kalibrierungsprozesses, sowie eine Übersicht über die dadurch gewonnenen Parameter findet sich in Kapitel 3.3 auf Seite 30). Der Prozentsatz der Gesamtarmlänge wurde durch Versuche mit Personen mit unterschiedlichen Armlängen festgelegt. Im Laufe dieser Versuche stellte sich heraus, dass die meisten Benutzer 60% der Armlänge für die *Vorne*-Erkennung als angenehm empfanden. Deshalb wurde dieser Prozentsatz für das Programm festgelegt. Die meisten Gesten für MapKin beinhalten das Bewegen einer oder beider Hände, während sie den Status *vorne* haben. Wird die Hand als *vorne* erkannt, wird der Indikator dieser Hand **grün** gezeichnet.
Der Status *hängend* findet beim Verschieben der Karte, sowie beim Setzen eines Routenpunktes Verwendung. Er wird erkannt, wenn sich die Hand etwa unterhalb des Bauchnabels befindet. Diese Bedingung reicht für die Erkennung aus. Hat die Hand den Status *hängend*, wird der Indikator dieser Hand **gelb** eingefärbt.
Ist der Status der Hand weder *vorne*, noch *hängend*, befindet sie sich automatisch im Status *sonstiges*. Dieser Status ist nur für das Zurücksetzen der Routenpunkte relevant. Ansonsten kann in diesem Status nicht mit der Karte interagiert werden. Hat die Hand den Status *sonstiges*, wird der Indikator dieser Hand **rot** eingefärbt.
Im Folgenden werden die verschiedenen Gesten beschrieben. Technische Details, sowie weitere Informationen zur Gestenerkennung finden sich im Kapitel 3.2 auf Seite 28. Tabelle 4.1 gibt eine Übersicht über die verschiedenen Gesten.

2.2.1 Verschieben der Karte

Auf Touchscreens werden Karten meist mit einem Finger bewegt. Einfach ausgedrückt, folgt die Karte bei Berührung dem Finger. Da die Auflösung des Tiefenbildes der Kinect nicht ausreicht, um einzelne Finger präzise genug erkennen zu können, musste die „nächst größere Stufe“ gewählt werden—die Hand. Das Ziel war es, die Mechanik dabei möglichst wenig zu verändern. Die Karte soll also der Hand folgen. Beim Touchscreen ist es hierbei möglich, den Finger abzusetzen, um die Karte in der aktuellen Position zu belassen. Da die Hand bei der Benutzung von MapKin jedoch zu keinem Zeitpunkt ein festes Objekt berührt, muss das „Absetzen“ auf eine andere Art und Weise auf die Gestensteuerung übertragen werden. Die Mechanik wurde auch hier identisch belassen. Das „Absetzen“ wird simuliert, indem die Hand näher zum Körper bewegt wird. Die Analogie zum Berühren des Touchscreens ist das Ausstrecken des Arms. Um den Punkt zu finden, ab dem die Hand als *vorne*, also in der Touchscreen-Analogie „berührt den Bildschirm“ interpretiert werden soll, ist eine Kalibrierung notwendig. Genaueres dazu wird im Kapitel *3.3* auf Seite 30 beschrieben. Der Raum vor dem Körper kann weiterhin für die natürliche Gestikulation verwendet werden, ohne dass MapKin darauf anspricht.
Da der Großteil der Bevölkerung Rechtshänder und das Verschieben der Karte eine der am häufigsten benutzen Gesten ist, wurde die rechte Hand für das Verschieben gewählt. Um die Geste deutlich von der Zoom-Geste (siehe im nächsten Abschnitt) unterscheiden zu können, funktioniert das Verschieben der Karte mittels dieser Geste nur, wenn die linke Hand während des Verschiebens den Status *hängend* hat.

2.2.2 Zoomen der Karte

Für die Zoomgeste wurde ebenfalls die Mechanik der Steuerung auf Touchscreens übernommen. Hierbei ist es üblich, dass Zoomen mittels „Pinch to zoom“ umgesetzt wird. Bei dieser Geste werden zwei Finger auf dem Touchscreen aufgesetzt und dann gespreizt oder zusammengeführt. Durch das Spreizen wird der Ausschnitt vergrößert, beim Zusammenführen verkleinert. Diese Geste ist sehr anschaulich, da auch hier, wie beim Verschieben, die berührten Punkte auf der Karte den Fingern folgen. Um in MapKin zu zoomen, müssen beide Hände ausgestreckt werden. Anschließend können die Hände entweder auseinandergezogen oder zusammengeführt werden. Auf diese Weise kann in die Karte hinein- oder aus der Karte heraus-gezoomt werden.
Es gibt jedoch einen entscheidenden Unterschied zwischen dem Zoomen auf Touchscreens und dem mittels MapKin. Das Zoomen in Programmen, die für Touchscreens ausgelegt sind, funktioniert meist stufenlos. Das bedeutet, die Größe des zu zoomenden Ausschnitts kann frei gewählt werden. Da JXMapViewer jedoch lediglich vorgerenderte Kartentiles anzeigt, steht in MapKin nur eine bestimmte Anzahl von Zoomstufen zur Verfügung. Ein stufenloses Zoomen ist somit nicht möglich. Deshalb weicht die Steuerung des Zooms von der auf Touchscreens ab. Um bei MapKin ein Zoomen auszulösen, muss der Abstand zwischen den Händen bei der Ausführung der zuvor genannten Geste einen gewissen Abstand über- bzw. unterschreiten. Da dieser Abstand abhängig von der Armweite des Benutzers ist, muss

anfangs eine Kalibrierung vorgenommen werden (siehe Kapitel *3.3* auf Seite 30).
Das komplette Hinein- oder Herauszoomen unter Verwendung dieser Geste kann aufgrund der vielen Zoomstufen recht aufwendig werden. Deshalb wurde die Geste dahingehend abgeändert, dass pro Zoomgeste zwei Zoomstufen hinein- oder herausgezoomt werden können. Dazu müssen zwei verschiedene Handabstände für das Auslösen des Zoomens festgelegt werden. Auf diese Weise ist es sehr schnell möglich, die größte oder kleinste Zoomstufe zu erreichen. Gleichzeitig kann aber auch noch präzise genug in die nächst höhere oder niedrigere Zoomstufe gewechselt werden.
Die Geste wurde auch mit drei Zoomstufen pro Zoombewegung getestet. Dabei müssen die Abstände zwischen den Punkten, die ein Zoomen auslösen, jedoch so gering gewählt werden, dass das präzise Wählen einer Zoomstufe nicht mehr möglich ist. Deshalb wurde dies so nicht übernommen.

2.2.3 Setzen von Routenpunkten

Beim Setzen von Routenpunkten finden sich ebenfalls Analogien zur Touchscreenbedienung. Im ToureNPlaner, aber auch bei anderen Anwendungen, werden Routenpunkte durch langes Berühren der entsprechende Position auf der Karte gesetzt. Da festgelegt wurde, dass es ausreicht, einen Start- und einen Zielpunkt zu setzen, gab es anfangs die Überlegung, für das Setzen dieser Punkte unterschiedliche Gesten zu verwenden. Diese Idee wurde jedoch wieder verworfen, da es die Steuerung unnötig verkomplizieren würde. Stattdessen wird der erste gesetzte Routenpunkt als Startpunkt definiert, während jeder weitere einen Zielpunkt setzt. Bereits vorhandene Zielpunkte werden automatisch überschrieben. Das Löschen aller Routenpunkte ist mittels einer separaten Geste möglich, die später erläutert wird.
Da der linken Hand bisher keine Funktion zugeordnet wurde, wird sie für das Setzen der Routenpunkte benutzt. Die Geste, die hier verwendet wird, ist analog zu der, mit welcher die Karte verschoben wird. Der rechte Arm muss den Status *hängend* haben, während mit der ausgestreckten linken Hand ein Punkt auf der Karte ausgewählt wird. Um nicht direkt beim Ausstrecken des Arms einen Punkt zu setzen, sondern zielen zu können, muss die Hand zum Setzen für eine gewisse Zeit möglichst unbewegt auf derselben Position auf der Karte bleiben.

2.2.4 Löschen von Routenpunkten

Auf Touchscreens geschieht das Löschen gesetzter Routenpunkte meist durch Berühren eines Buttons. Es gibt keine allgemein bekannte Geste dafür. Aus diesem Grund musste eine komplett neue Geste gefunden werden, an die nur zwei Anforderungen gestellt werden mussten. Zum einen sollte sie nicht versehentlich ausgelöst werden können – also möglichst unnatürlich sein. Ferner sollte sie von Seiten der Software leicht zu erkennen sein. Da auf nichts gezielt werden muss, kann die Geste statisch sein. Sie besteht deshalb darin, beide Arme nah am Körper für eine gewisse Zeit auf Kopfhöhe zu halten. Damit erfüllt sie die beiden gestellten Ansprüche. Sie ist nicht Bestandteil normaler Gestikulation und gleichzeitig leicht zu erkennen. Um die Geste dennoch nicht versehentlich auszulösen,

muss diese Position, wie beim Setzen von Routenpunkten, für eine bestimmte Zeit gehalten werden.

2.3 Erkenntnisse vom Tag der Wissensschaft

Am Tag der Wissenschaft, der in diesem Jahr am 22. Juni stattfand, ergab sich zum ersten Mal die Gelegenheit, MapKin mit vielen verschiedenen Benutzern zu testen. Dabei wurden verschiedene Erkenntnisse gewonnen, die wesentlich zur Verbesserung von MapKin beitrugen. Sie waren deshalb besonders hilfreich, da MapKin für die Verwendung am Tag der Wissenschaft oder ähnlichen Veranstaltungen entwickelt wurde.
Der Tag der Wissenschaft findet jährlich an der Universität Stuttgart statt und soll Einblicke in die Forschungsarbeit der verschiedenen Instituten geben. Am Stand des Instituts FMI (Institut für Formale Methoden der Informatik) wurde der ToureNPlaner und MapKin vorgestellt. Der Großteil der Interessenten waren Kinder im Alter von 8 bis 14 Jahren. Dennoch haben auch viele Erwachsene aller Altersklassen MapKin getestet. An diesem Tag kam die Version von MapKin zum Einsatz, die auf OpenNI 1.5 basiert und zu diesem Zeitpunkt noch nicht vollständig entwickelt war. Vor allem die Kalibrierung unterschied sich noch stark von der endgültigen Version.
Die erste Erkenntnis war, dass NiTE Probleme damit hat, kleinere Kinder zuverlässig zu erkennen. Es dauerte zum Teil mehr als eine Minute bis das Kind erkannt wurde und mit der Kalibrierung beginnen konnte, obwohl das Kind mittig vor der Kamera stand und vollständig auf dem Tiefenbild zu sehen war. Außerdem wurden Kinder häufiger wieder „verloren“. Das bedeutet, NiTE erkannte sie plötzlich nicht mehr, obwohl sie nach wie vor an derselben Stelle standen. Da für die Erkennung jedoch ausschließlich NiTE verantwortlich ist, kann MapKin nicht dahingehend verändert werden, diese Situation zu verbessern.
Des Weiteren haben die Benutzer angemerkt, dass der Kalibrierungsvorgang nicht genau genug beschrieben sei und sie mehr Rückmeldung von MapKin wünschten. Zu diesem Zeitpunkt gab es von MapKin tatsächlich noch kaum Rückmeldung. Es wurde nur ein Text angezeigt, der dazu aufforderte, die Arme nach vorne zu nehmen, beziehungsweise sie zu öffnen. Zusätzlich wurde eine graphische Darstellung der Geste präsentiert. Aufgrund dessen wurde die Kalibrierung dahingehend verbessert, dass Ziele eingeführt wurden. In der endgültigen Version kann der Benutzer nun sehen, wie er die Hände halten muss und in welchem Bereich die Pose erkannt wird. Außerdem wurde die grüne Färbung der Joint-Indikatoren aus der Kartenansicht übernommen. Zuvor waren die Indikatoren während der Kalibrierung grundsätzlich rot.
Diese Funktion ist vor allem deshalb hilfreich, da die Kalibrierung zuvor häufig daran gescheitert ist, dass die Benutzer die Arme im zweiten Schritt der Kalibrierung so weit öffneten, dass die Hände nicht mehr den Status *vorne* hatten. Durch die farbigen Indikatoren erkennen die Benutzer nun, wenn die Arme zu weit hinten sind. Dadurch wird die Kalibrierung einfacher.
Weiterhin wurde festgestellt, dass es sinnvoll ist, einen weiteren Schritt am Ende des Kalibrierungsprozesses einzuführen. Bis zu diesem Zeitpunkt wurde nach der Kalibrierung der Distanz zwischen den Händen sofort die Karte angezeigt. Dies hatte den Nachteil, dass sich

die Hände direkt vorne befanden, wenn der Kalibrierungsmodus beendet und die Karte eingeblendet wurde. Somit wurde zwangsläufig direkt mit der Karte interagiert, obwohl die Testpersonen noch nicht genau wussten, wie die Steuerung der Karte funktionierte. Deshalb wurde ein Schritt eingeführt, der überprüft, ob beide Arme den Status *hängend* haben, bevor die Kalibrierung abgeschlossen wird. Es findet hierbei keine Kalibrierung im eigentlichen Sinne statt. Vielmehr diente diese Maßnahme ausschließliche dazu, dass sich die Hände der Testpersonen im Status *hängend* befinden, wenn die Karte eingeblendet wird.
Eine weitere Erkenntnis war, dass die Einschränkung, Gesten unterhalb des Bauchnabels grundsätzlich zu ignorieren, die damals noch bestand, unnötig und verwirrend ist. Diese Restriktion wurde ursprünglich eingeführt, um die natürliche Gestikulation, die beim Sprechen oft unterhalb des Bauchnabels stattfindet, nicht fälschlicherweise als Geste zur Steuerung zu interpretieren. Da es jedoch auch ohne die Restriktion keinerlei Fehlbedienungen diesbezüglich gab, konnte diese Einschränkung aufgehoben werden.
Ein weiterer Punkt, der am Tag der Wissenschaft festgestellt wurde, ist, dass die Benutzer meist als erstes ihren Heimatort auf der Karte suchen. Da die Karte anfangs auf London zentriert war, der Standardpositionierung von JXMapViewer, mussten sie zu diesem Zeitpunkt noch lange danach suchen. Um die Suche nach Orten in Deutschland zu vereinfachen, ist die Karte nun über Stuttgart zentriert. Außerdem kehrt die Ansicht immer wieder nach Stuttgart zurück, wenn die Kalibrierung für eine neue Person zurückgesetzt wird.
Auch das Erklären der Zoomgeste konnte vereinfacht werden, nachdem festgestellt wurde, dass vor allem Kinder die Erklärung der Zoomgeste mit der Pinch-To-Zoom-Analogie auf Touchscreens meist nicht verstanden. Deshalb wurde dazu übergegangen, das Zoomen mit einer „Schwimmbewegung“ zu erklären -– die Kinder sollten in die Karte „hinein- oder herausschwimmen“. Dies wurde meist sofort verstanden und das Zoomen fiel ihnen leichter.

Zusammenfassend lässt sich sagen, dass dem Großteil der Testpersonen das Steuern der Karte und das Berechnen von Routen mittels Gesten sichtlich Spaß gemacht hat. Sowohl Kinder als auch Erwachsene konnten sich schnell mit der Benutzung vertraut machen und die Karte präzise steuern. Dies zeigte, dass die Gesten grundsätzlich nicht verändert werden mussten. Das Konzept der Bedienung wurde verstanden und den meisten Benutzern als interessant und intuitiv aufgefasst.

3 Implementierung

3.1 Übersicht

Die Implementierung orientierte sich anfangs stark an den von OpenNI zur Verfügung gestellten Beispielen. Diese zeigen, wie die Kinect initialisiert wird und wie ein Tiefenbild dargestellt werden kann. So war es vergleichsweise einfach, die Grundfunktionalität, nämlich das Erkennen des Skeletts zu implementieren. Auch das Erkennen der Gesten erwies sich als unkompliziert. Aufwendiger gestaltete sich hingegen das Finden von geeigneten Parameter für die präzise und angenehme Steuerung der Karte. Schnell fiel auf, dass das Festlegen von ausschließlich festen Parametern nie für alle Benutzer funktionieren wird. Deshalb wurde die Kalibrierung entwickelt. Dadurch können verschiedene Parameter so gewählt werden, dass diese genau zu den Proportionen des Körpers des Benutzers und auf dessen Abstand zur Kinect passen.
MapKin hat zwei Ansichten – die Kalibrierungsansicht (gennant *CameraView*, weil diese auch das Tiefenbild des Benutzers darstellt) und die Kartenansicht (*MapView* genannt). Diese Zweiteilung spiegelt sich auch in der Architektur des Programms wider (siehe Abbildung 3.1).

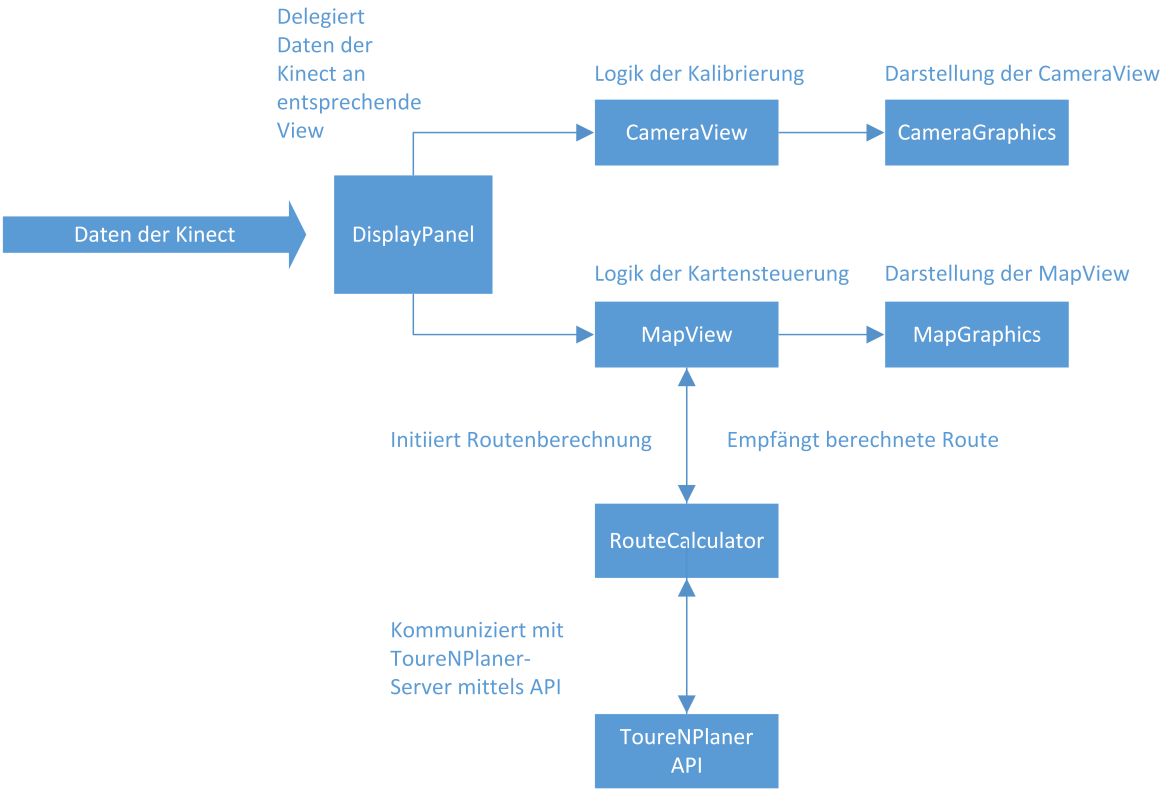


Abbildung 3.1: Übersicht über die Aufgaben der Klassen. Die Daten der Kinect werden von der *DisplayPanel*-Klasse empfangen. Diese leitet die Daten während der Kalibrierung an die *CameraView*-Klasse und während der Steuerung der Karte an die *MapView*-Klasse weiter. In diesen *Views* findet die Verarbeitung der Daten statt. Für die Darstellung sind die *Graphics*-Klassen zuständig. Die Routenberechnung wird von der *MapView*-Klasse initiiert. Die Koordinaten für die Berechnung werden an die *RouteCalculator*-Klasse weitergereicht, welche mit dem API des ToureNPlaners kommuniziert. Die berechnete Route wird an die *MapView*-Klasse weitergegeben, welche die Daten wiederum an die *MapGraphics*-Klasse weiterleitet, wo die Darstellung der Route und der zugehörigen Informationen geschieht.

MapKin beginnt beim Start mit der Ausführung der *Controller*-Klasse, welche das Fenster initialisiert, das Menü am oberen Fensterrand erstellt und ein sogenanntes *DisplayPanel* startet. Dieses ist für die Kommunikation mit der Kinect zuständig. Die eigentliche Verarbeitung der Daten von der Kinect findet jedoch in den jeweiligen *Views* statt. Deshalb ist das *DisplayPanel* unter anderem dafür zuständig, zu entscheiden, welche der beiden *Views* angezeigt werden muss. Während der Kalibrierung und wenn ein Benutzer verloren wurde wird die *CameraView* angezeigt, ansonsten die *MapView*. Je nachdem, welche Ansicht gerade aktiv ist, delegiert das *DisplayPanel* die Daten der Kinect an die entsprechenden *Views*.

In der *CameraView*-Klasse werden diese Daten dazu genutzt, die Kalibrierung durchzuführen. Die eigentliche Darstellung erfolgt jedoch in der Klasse *CameraGraphics*. Diese erhält die von der der *CameraView* verarbeiteten Daten und zeichnet sie in das Tiefenbild ein. Die *CameraView*-Klasse ist also für die Logik, die *CameraGraphics*-Klasse für die Darstellung zuständig. Ist die Kalbrierung abgeschlossen, leitet die *CameraView*-Klasse die Ergebnisse an das *DisplayPanel* zurück, welches sie wiederum der *MapView*-Klasse mitteilt.
Nun wechselt die Ansicht zur *MapView* und diese erhält fortan die neusten Daten der Kinect. In der *MapView*-Klasse findet die Verarbeitung der Daten und die Gestenerkennung statt. Außerdem stößt diese Klasse die Routenberechnung an, die in die Klasse *RouteCalculator* ausgelagert ist. Diese benutzt wiederum diverse Klassen aus ToureNPlaner und kommuniziert mit dem ToureNPlaner-Server. Die Anfragen werden mittels HTTP an den Server übertragen, welcher das Ergebnis als JSON-Element versendet. Hierbei wurde der Großteil aus der Android-App von ToureNPlaner übernommen. Die Klassen wurden nur soweit angepasst, dass sie einfach für dieses Projekt benutzt werden konnten. Das Ergebnis der Berechnung wird entgegengenommen, verarbeitet und zurück an die *MapView*-Klasse gegeben, welche ihrerseits die Daten zum Zeichnen an die *MapGraphics*-Klasse weitergibt.
Diese übernimmt auch das Zeichnen der Karte, der Overlays, des Gestenindikators und aller weiteren Objekte, die in dieser Ansicht sichtbar sind. JXMapViewer stellt ausschließlich die Karte dar. Desweiteren benutzt es einen Painter, der Overlays über die Karte zeichnen kann. Alle weiteren Elemente werden mithilfe dieses Painters gezeichnet. Dabei ist zwischen statischen Elementen, wie dem Gestenindikator, den Routeninformationen, sowie den Joint-Indikatoren und dynamischen wie der Karte, der Routenpunkte und der Route zu unterscheiden. Die statischen Elemente sind unabhängig vom Verschieben der Karte, die anderen müssen sich entsprechend mit der Karte bewegen.
Um die Karte zu bewegen, wurde ein sehr einfacher Weg gewählt. Normalerweise lässt sich die Karte von JXMapViewer bewegen, indem man die Karte bei gedrückter Maustaste mit der Maus verschiebt. Beim Bewegen des Scrollrades wird ein Zoomen ausgelöst. MapKin steuert JXMapViewer, indem es diese Mausaktionen emuliert. Wird die Verschieben-Geste erkannt, wird das Halten der linken Maustaste emuliert. Bei Bewegung der Hand, wird die Bewegung direkt als Mausbewegung emuliert. Wenn die rechte Hand zurückgezogen wird, wird das Loslassen der linken Maustaste emuliert. Bei der Zoomgeste wird das Drehen des Scrollrades emuliert. Beim Hineinzoomen findet nach dem eigentlichen Zoomen noch eine Verschiebung der Karte statt, sodass sich die Position auf der Karte, in die hineingezoomt wurde, nach dem Zoomen wieder an derselben Stelle befindet. Auch dieses Verschieben wird durch Emulation von Mausaktionen umgesetzt.

3.2 Die Gestenerkennung

Bei der Auswahl der Gesten wurde großer Wert darauf gelegt, dass diese einfach zu erkennen sind. Deshalb erfordert die Erkennung der Gesten wenig Aufwand. Zunächst werden die Daten der Kinect in einzelne 3D-Punkte umgewandelt. Für die Erkennung der Gesten werden folgende Punkte benötigt:

- Kopf
- Torso (entspricht in etwa der Höhe des Bauchnabels)
- linke und rechte Schulter
- linke und rechte Hand

Die X- und Y-Werte der Punkte entsprechen den Koordinaten auf dem Tiefenbild. Da dieses eine Auflösung von 640 x 480 Pixel hat, ist dies auch der Wertebereich, den die Koordinaten annehmen können. In der Kartenansicht werden die Koordinaten auf die komplette Bildschirmgröße der Anwendung umgerechnet, um alle Position auf der Karte erreichen zu können. Die Z-Koordinate hat keine Einheit. Deshalb wird hier von *Tiefeneinheiten* gesprochen. Die Tiefeneinheiten können Punkte im Bereich von 0, einer Position direkt vor Kamera und 2047, einer Position in großem Abstand zur Kamera, annehmen. Die Werte für die Z-Koordinate sind linear, was bedeutet, dass ein Unterschied von einem Zentimeter in der Z-Koordinate immer im selben Unterschied in Tiefeneinheiten resultiert, unabhängig davon in welcher Distanz von der Kamera dieser Unterschied gemessen wird.
Im Folgenden wird auf die Gesten für die Kartensteuerung eingegangen. Die Erkennung der Gesten für die Kalibrierung werden im Kapitel *3.3* auf Seite 30 behandelt.
Zunächst muss der Status, der Hände festgestellt werde – also *vorne*, *hängend* oder *sonstiges*. Für den Status *vorne* wird überprüft, ob der Abstand der Z-Koordinaten zwischen Hand und Torso die Armlänge multipliziert mit dem Faktor 0,6 überschreitet.
Ähnlich einfach ist auch die Überprüfung des Status *hängend*. Hierzu muss festgestellt werden, ob der Y-Wert der Hand unterhalb dem des Torsos liegt. Ist die Hand weder *vorne*, noch *hängend*, hat sie den Status *sonstiges*.
Durch diese Status kann die Geste für das Verschieben der Karte leicht erkannt werden. Hat die linke Hand den Status *hängend* und die rechte *vorne*, kann die Bewegung der rechten Hand direkt auf die Karte übertragen werden. Die Erkennung für die Geste, mit der Routenpunkte gesetzt werden, funktioniert identisch. Hierbei muss die rechte Hand den Status *hängend* und die linke den Status *vorne* haben. Bleibt die linke Hand nun für 0,8 Sekunden innerhalb eines Radius von 30 Pixel, wird ein Routenpunkt gesetzt.
Die Erkennung der Zoomgeste hingegen ist etwas komplizierter. Beim Kalibrieren wird der Abstand zwischen den beiden Händen in der XY-Ebene festgestellt. Dieser Abstand entspricht 100%. Nun werden Bereiche innerhalb dieses Abstands definiert. Beim Übergang von einem in den nächsten Bereich wird ein Zoomevent ausgelöst.

- 1. Bereich: 0-39%
- 2. Bereich: 40-69%
- 3. Bereich: 70-100%

Sobald beide Hände in den Status *vorne* übergehen, wird die Distanz zwischen den beiden Händen gemessen. Daraus wird bestimmt, in welchem der zuvor genannten Bereiche die Hände zu Beginn sind. Solange die Hände den Status *vorne* beibehalten, wird kontinuierlich deren Abstand gemessen. Geht dieser in einen anderen Bereich über, wird ein Zoomevent ausgelöst. Beim Übergang in einen größeren Bereich wird in die Karte hinein gezoomt, beim Übergang in einen kleineren herausgezoomt. Da das Erkennen dieser Geste den Abstand zwischen den Händen in X- und Y-Richtung benutzt wird und sich dieser mit dem Abstand von Benutzer zur Kinect ändern würde, ist es nicht erlaubt, dass sich der Benutzer zu stark vom Abstand, den er während der Kalibrierung zur Kinect hatte, entfernt. Deshalb wird der Abstand während der Kalibrierung gespeichert. Dazu wird die Z-Koordinate des Torsos verwendet. Weicht dieser Wert während des Steuerns um mehr als 120 Tiefeneinheiten vom ursprünglichen ab, wird der Benutzer dazu aufgefordert, wieder den ursprünglichen Abstand einzunehmen.
Die Geste, um die Route zu löschen findet als einzige dicht am Körper statt, womit die Wahrscheinlichkeit für eine versehentliche Auslösung größer ist. Deshalb muss diese Geste besonders genau überprüft werden. Um die Geste auslösen zu können, müssen beide Hände den Status *sonstiges* annehmen. Nun wird überprüft, ob beide Hände auf derselben Höhe sind. Dazu werden der Y-Werte der Hände verglichen. Der Unterschied der beiden Werte muss in einem bestimmten Bereich liegen, der abhängig von der Distanz des Benutzers zur Kinect und dessen Größe ist. Zudem darf die Z-Koordinate der Hand höchstens 70 Tiefeneinheiten vor der entsprechenden Schulter liegen. Es wurde die Schulter als Referenzpunkt für die Z-Koordinate gewählt, da dies die Erkennungsgenauigkeit gegenüber dem Kopf als Referenzpunkt erhöht. Wenn der Benutzer nicht exakt auf die Kinect gerichtet ist, sondern leicht gedreht zur Kamera steht, ergeben sich größere Unterschiede bei den Z-Koordinaten der Hände, obwohl diese, relativ zu Kopf gesehen, dieselbe Z-Koordinaten haben. Würde also der Kopf als Referenzpunkt gewählt werden, würde die Geste nur bei perfekter Ausrichtung des Benutzers zur Kinect zuverlässig erkannt werden. Da die Schultern hingegen bei leichter Fehlausrichtung zur Kinect dieselben Differenzen in der Z-Koordinate aufweisen, sind sie ideal für den Referenzpunkt bei der Erkennung dieser Geste geeignet.

3.3 Die Kalibrierung

Die Kalibrierung ist wichtig für die präzise Steuerung der Karte. Es werden verschiedene Abstände gemessen:

- Abstand zwischen Torso und Hand in Z-Richtung bei ausgestrecktem Arm (Armlänge)
- Abstand zwischen den Händen auf der XY-Ebene bei ausgestreckten und geöffneten Armen
- Wert der Z-Koordinate des Torsos (Abstand des Benutzers zur Kinect)

Um die einzelnen Kalibrierungsschritte abzuschließen, müssen die Positionen der Hände jeweils bestimmte Bedingungen erfüllen – sie müssen sich in gewissen Zonen für eine bestimmte Zeit aufhalten. Diese Zonen wurden „Ziele“ genannt und durch transparente, weiße Flächen auf dem Tiefenbild visualisiert. Dadurch weiß der Benutzer, wie er die Hände halten muss. Die Ziele sind jedoch nicht statisch sondern abhängig von der Position des Benutzers. Bewegt sich der Benutzer, bewegen sich auch die Ziele mit, wodurch das Zielen einfacher wird, da der Benutzer nicht darauf achten muss, sich so zu positionieren, dass er in der Mitte des Tiefenbildes erscheint.

Als erstes wird die Armlänge gemessen. Diese wird dazu verwendet, festzustellen, wann ein Arm den Status *vorne* hat. Für die Kalibrierung muss der Benutzer die Hände in etwa schulterbreit vollständig ausstrecken. Die Ziele liegen entsprechend schräg außen über den Schultern (siehe Abbildung 3.2).

Abbildung 3.2: Kalibrierung der Armlänge. Zu sehen sind die weißen Ziele über den Schultern, die mit den Handindikatoren getroffen werden müssen. Die Kalibrierung der Armlänge wird dazu verwendet, präzise feststellen zu können, wann die Hand als *vorne* erkannt wird.

Die Positionen der Ziele sind von den Positionen der Schultern abhängig. Die Mitte des Ziels wird leicht über den Schultern platziert. Die Größe ist abhängig von der Entfernung des Benutzers zur Kinect. Je weiter der Benutzer von der Kinect entfernt steht, desto kleiner wird das Ziel. Dies wurde deshalb so gewählt, da kleine Bewegungen in größerer Distanz kaum auf dem Tiefenbild wahrgenommen werden können. Dementsprechend kann der Kreis auch klein sein. Ist der Benutzer hingegen sehr dicht vor der Kamera, machen kleinere Bewegungen des Arms größere Distanzen auf dem Tiefenbild aus. Deshalb muss in diesem Fall der Kreis größer sein.
Die Mitte des Handindikators muss für 1,5 Sekunden innerhalb eines Vierecks, das tangential zu den kreisförmigen Zielen liegt, bleiben. Da die Handindkatoren jedoch kreisförmig sind, wurden ebenfalls Kreise für die Ziele gewählt, um die Anschaulichkeit zu verbessern. Verlässt eine Hand das Ziel wieder, bevor 1,5 Sekunden verstrichen sind, startet die Zeit beim Wiedereintritt erneut bei 1,5 Sekunden. Weiterhin muss für das erfolgreiche Kalibrieren der Abstand zwischen Torso und den Händen in Z-Richtung mehr als 300 Tiefeneinheiten betragen. Dies ist ein sehr optimistischer Wert und sollte immer geringer als die maximale Armlänge sein – auch bei Kindern. Sind diese Voraussetzungen erfüllt, speichert MapKin den maximalen Abstand zwischen Torso und dem Durchschnitt der Z-Koordinaten der beiden Hände, der während den 1,5 Sekunden Kalibrierungszeit gemessen wurde. Dieser gilt fortan als Armlänge.
Im nächsten Schritt wird die Distanz der Arme bei geöffneten und ausgestreckten Armen

gemessen. Diese Distanz wird für die Zoomgeste bei der Kartensteuerung verwendet. Auch hier gibt es wieder Ziele, in die der Benutzer die Hände halten muss. In diesem Schritt sind diese Flächen jedoch rechteckig. Die Höhe der Rechtecke ist identisch zum Durchmesser der Kreise im vorherigen Schritt – auch hier passt sich die Höhe der Distanz des Benutzers zur Kinect an. Die Y-Position der Rechtecke ist ebenfalls identisch zu der der Kreise im letzten Schritt. Für die linke Hand beginnt das Rechteck am linken äußeren Rand des Tiefenbildes und erstreckt sich bis zu einem Punkt, der links der linken Schulter liegt (siehe Abbildung 3.3). Der exakte Abstand bis zur Schulter ist wieder abhängig von der Distanz des Nutzers zur Kinect. So wird sichergestellt, dass eine Mindestdistanz eingehalten wird. Gleichzeitig ist das Maximum der Distanz nur durch die Aufnahmegröße der Kamera beschränkt. Für die rechte Hand ist die Position des Rechtecks analog.

Abbildung 3.3: Kalibrierung der Armweite. Zu sehen sind die rechteckigen Zielflächen seitlich der Schultern, die mit den Handindikatoren getroffen werden müssen. Die Kalibrierung des Handabstands wird für die Zoomgeste verwendet. Durch die Kalibrierung kann festgestellt werden, welches der maximale Armabstand ist. Somit können die Abstände der Stufen bei der Zoomgeste berechnet werden.

Eine weitere Bedingung ist, dass die Hände als *vorne* erkannt werden. Dazu wird in diesem Schritt bereits die Armlänge aus dem vorherigen Kalibrierungsschritt benutzt. Liegen die Hände für mindestens 1,5 Sekunden innerhalb der Ziele, wird die maximale Distanz, die während dieser Zeit gemessen wurde als Distanz der Arme gespeichert. Zur gleichen Zeit wird der Abstand des Benutzers zur Kinect festgehalten – dafür wird die Z-Koordinate des

Torsos verwendet. Dies ist deshalb nötig, da die Distanz der beiden Hände zueinander in Pixel gemessen wird. Würde der Benutzer den Armabstand beibehalten, sich aber gleichzeitig nach hinten von der Kinect entfernen, wäre der gemessene Abstand geringer. Deshalb ist der Wert für die Armlänge nur für den aktuellen Abstand zur Kinect gültig. Aus diesem Grund wird im Falle einer zur großen Positionsabweichung des Benutzers zur Position während der Kalibrierung eine Meldung ausgegeben, die den Benutzer dazu auffordert, wieder die ursprüngliche Position einzunehmen.

Nach diesem Schritt ist die eigentliche Kalibrierung beendet. Es folgt jedoch noch ein weiterer, der zwar zum Kalibrierungsvorgang gehört, aber nicht zur Kalibrierung beiträgt. Er besteht darin, die Hände, die aufgrund des letzten Kalibrierungsschritts noch nach vorne ausgestreckt sind, wieder nach unten zu nehmen. Um diesen Schritt erfolgreich abzuschließen, wird überprüft, ob der Y-Wert beider Arme unterhalb des Y-Wertes des Torsos liegt. Die Kalibrierung ist beendet, sobald diese Bedingung zutrifft. In diesem Schritt gibt es kein Zeitlimit, in dem die Hände in dieser Position verweilen müssen. Das Ziel wird als Rechteck, das das gesamte Bild unter dem Torso einnimmt, visualisiert (siehe Abbildung 3.4).

Abbildung 3.4: Letzter Schritt der Kalibrierung. Zu sehen ist die Zielfläche, die den gesamten Raum unterhalb des Torsos einnimmt. Beide Arme müssen nach unten hängen, sodass sich die Hände innerhalb der Zielfläche befinden. Dieser Schritt ist nötig, um die Hände des Benutzers in eine neutrale Position zu bewegen, bevor er die Steuerung der Karte übernimmt.

4 Inbetriebnahme

In diesem Kapitel wird erklärt, wie MapKin und die benötigten Komponenten installiert werden.
Da MapKin in Java implementiert wurde, ist es grundsätzlich sowohl auf Windows, als auch auf Linux ausführbar. Auch OpenNI und NiTE sind für Linux und Windows verfügbar – sowohl in der Version 1.5, als auch in Version 2. Problematischer hingegen sind die Treiber, die die Kinect unter Linux unterstützen sollen (Eine ausführlichere Erklärung dazu findet sich im Kapitel *5.2* auf Seite 47). Deshalb wird an dieser Stelle dazu geraten, MapKin unter Windows auszuführen. Hier läuft die Installation der benötigten Komponenten erfahrungsgemäß problemlos ab.

4.1 Installation der benötigten Software

4.1.1 Installation unter Windows

Da sich die Einrichtung der Komponenten bei OpenNI 1.5 und OpenNI 2 unterscheidet, sind auch diese Anweisungen dementsprechend zweigeteilt. Sowohl für Version 1.5 als auch für Version 2 muss zunächst die Java Runtime Environment in Version 7 installiert sein. Die Version der Java Runtime Environment muss zur Architektur des Betriebssystem (32bit/64bit) passen. Auf das Vorgehen zum Installieren des JREs wird an dieser Stelle jedoch nicht weiter eingegangen, da sich dazu ausführliche Anleitungen im Internet finden lassen. Des Weiteren ist darauf zu achten, dass die Kinect erst angeschlossen werden darf, nachdem die Installation des Treibers abgeschlossen ist.

Version 1.5

Version 1.5 von OpenNI funktioniert in Zusammenhang mit der Kinect ausschließlich unter Windows 7. Dies liegt daran, dass der Treiber, der für Version 1.5 verwendet werden muss, nicht korrekt signiert ist und deshalb unter Windows 8 nicht ohne Umwege installiert werden kann.
Zunächst muss OpenNI 1.5 installiert werden. Dazu wählt man auf der OpenNI-Seite [Ope13c] die entsprechende Version für sein Betriebssystem aus (32bit/64bit) und installiert diese. Weitere Einstellungen sind nicht nötig. Als nächstes wird NiTE 1.5 benötigt. Dieses kann von der gleichen Seite heruntergeladen werden. Auch hier ist auf die Version für

das eigene Windows (32bit/64bit) zu achten. Danach muss NiTE installiert werden. Diese Reihenfolge der Installation ist einzuhalten, da NiTE ohne vorherige Installation von OpenNI nicht installiert werden kann.
Zum Schluss wird der Treiber für die Kinect benötigt. Dieser kann von einem GitHub-Repository [Sen13] heruntergeladen werden. Dazu lädt man das komplette Repository als ZIP-Datei herunter. Im Ordner *Bin* finden sich die Treiber für die unterstützten Betriebssysteme. Hier muss entweder SensorKinect093-Bin-Win32-v5.1.2.1.msi oder SensorKinect093-Bin-Win64-v5.1.2.1.msi ausgeführt und installiert werden. Wird während der Installation gefragt, ob der Treiber installiert werden soll, muss dies bestätigt werden. Nun kann die Kinect angeschlossen werden. Nachdem Windows die Kinect vollständig eingerichtet hat, ist die Installation beendet.

Version 2

Da Version 2 von OpenNI mit dem Kinect-Treiber von Microsoft zusammenarbeitet und dieser korrekt signiert ist, kann Version 2 auch unter Windows 8 ausgeführt werden. Zunächst muss OpenNI heruntergeladen und installiert werden. Die Beta-Version von OpenNI 2 kann von der OpenNI-Seite [Ope13d] heruntergeladen werden. Wird Windows in der 64bit Version benutzt sollte auch grundsätzlich die 64bit Versionen von OpenNI und NiTE benutzt werden. Bei der Installation wird gefragt, ob ein Treiber installiert werden soll. Dies kann verneint werden. Die Installation des Treibers wird später beschrieben. Als nächstes wird NiTE 2 benötigt. Die Beta-Version kann von der OpenNI-Seite in der Kategorie *Middleware Libraries* heruntergeladen werden [Pri13]. Dazu muss man jedoch einen Account bei OpenNI erstellen. Nachdem man sich angemeldet hat, kann NiTE 2 heruntergeladen und installiert werden. Bei Version 2 ist die Reihenfolge der Installation nicht relevant, da NiTE 2 auch ohne OpenNI installiert werden kann. Zuletzt muss der Treiber installiert werden. Dieser ist Bestandteil des Kinect SDKs von Microsoft, das von der offiziellen Seite [Mic13a] heruntergeladen werden kann. Nachdem die Installation abgeschlossen ist, kann die Kinect angeschlossen werden. Bei der Installation von MapKin in Version 2 gibt es im Unterschied zu Version 1.5 noch einen weiteren Schritt. Die Ausführung von MapKin funktioniert nur, wenn die Jar-Datei in das Sample-Verzeichnis des NiTE-Ordners kopiert wird. Deshalb müssen die Datei MapKin2.jar und gegebenenfalls auch die beiden Batch-Dateien in den Ordner *Samples\Bin* im NiTE-Installationsverzeichnis kopiert werden. Dieses befindet sich üblicherweise in *C:\Program Files\PrimeSense\NiTE2*. Nun ist die Installation beendet und MapKin kann gestartet werden.

4.1.2 Installation unter Linux

Grundsätzlich ist es möglich, die Kinect unter Linux zu benutzen. Jedoch war der Treiber, der für die Ausführung unter Linux benötigt wird, auf einem Testsystem nicht lauffähig. Es gibt jedoch einige Beschreibungen, nach denen er in Kombination mit OpenNI und NiTE zuverlässig funktionieren soll. Deshalb wird hier dennoch eine kurze Beschreibung zur Einrichtung der Kinect unter Linux gegeben. Auch hierbei wird wieder zwischen Version

1.5 und Version 2 unterschieden. Jedoch konnte keine der beiden Versionen funktionsfähig eingerichtet werden.
Wie auch bei der Installation unter Windows, muss zunächst OpenNI und NiTE installiert werden. Sowohl OpenNI 1.5, wie auch NiTE 1.5 können von der OpenNI-Seite im Bereich *History* [Ope13c] heruntergeladen werden. OpenNI 2 steht in der Kategorie *Download* [Ope13d] und NiTE 2 unter *Middleware Libraries* [Pri13] zum Download bereit. Für den Download von NiTE 2 muss ein Benutzerkonto auf der Seite erstellt werden. Die Installation für beide Versionen verläuft identisch. Zunächst müssen die heruntergeladenen Archive entpackt werden. Sowohl OpenNI, als auch NiTE enthält ein Skript (*install.sh*), das ausgeführt werden muss.
Nun werden die Treiber benötigt. Für Version 1.5 von OpenNI kann, wie auch für Windows, das Paket aus dem GitHub-Repository [Sen13] verwendet werden. Hierbei muss jedoch der entsprechende Treiber für Linux verwendet werden. Ein Treiber, der mit Version 2 von OpenNI benutzt werden kann, ist im GitHub-Repository des Freenect-Treibers [ope13a] zu finden. Nach dem Entpacken des Archivs muss der Treiber mit dem Befehl *.\waf configure build* kompiliert werden. Nun befindet sich der Treiber libFreenectDriver.so im Verzeichnis *build*. Von dort muss diese Datei in den Pfad *Redist/OpenNI2/Drivers* im OpenNI-Verzeichnis kopiert werden.

4.2 Benutzung der Kinect

Bevor MapKin genutzt werden kann, sollte die Kinect korrekt positioniert werden. Dazu platziert man sie direkt unter oder über dem Bildschirm. Besonders wichtig ist es, dass die Kinect direkt auf den Benutzer gerichtet ist, da sonst unter Umständen die Gestenerkennung nicht korrekt funktioniert. Bei der Platzierung der Kinect ist darauf zu achten, dass diese nicht im Gegenlicht steht, da das einfallende Licht die von der Kinect erzeugten Infrarotpunkte überstrahlt. Aus selbem Grund kann sie bei Tag nicht im Freien verwendet werden. Laut dem Handbuch der Kinect [Mic13b] sollte man einen Abstand von etwa 1,5m zur Kinect einnehmen. Sowohl die Hardware, als auch die Software ist in der Lage, Bewegungen in einem Abstand von 0,7-6,0 Metern zu erkennen. Die Erfahrung zeigt jedoch, dass die Erkennung der Joints in zuvor genanntem Bereich präziser funktioniert. Besonders bei zu geringem Abstand zur Kinect funktioniert die Erkennung der Joints nicht mehr ideal. Es kann dann vorkommen, dass die Hände aus dem erfassten Bereich gestreckt werden, wodurch NiTE nicht mehr in der Lage ist, die Hände zu erfassen. Dies resultiert darin, dass die Position der Hand unvorhersehbar „springt“, was häufig zu einer Fehlbedienung von MapKin führen kann. Im Gegensatz dazu gibt es keine Bedenken bei zu großem Abstand zur Kamera, solange 6 Meter nicht überschritten werden. Jedoch sollte der Abstand so gewählt werden, dass die Karte gut lesbar ist. Dies sollte bereits bei der Kalibrierung berücksichtigt werden, da eine Neupositionierung des Benutzers nur durch eine erneute Kalibrierung möglich ist.

4.3 Benutzung von MapKin

4.3.1 Kalibrierung

Nachdem MapKin sowie sämtliche benötigte Software installiert sind, kann das Programm benutzt werden. Um es zu starten, kann die MapKin1.5.jar bzw. MapKin2.jar direkt ausgeführt werden.
Ist die Kinect angeschlossen und wird erkannt, befindet man sich direkt in der Kalibrierungsansicht (siehe Abbildung 4.1). In dieser sieht man auf der linken Seite das Tiefenbild, das die Kinect aufnimmt. Rechts erscheinen bei der Kalibrierung graphische Illustrationen, um zu sehen, wie man sich positionieren muss. Darunter befindet sich ein Text, der Anweisungen für den Kalibrierungsprozess zeigt.

Abbildung 4.1: Das Bild zeigt die Kalibrierungsansicht im ersten Schritt der Kalibrierung. Links ist das Tiefenbild mit den Zielen (weiße Flächen), sowie den Handindikatoren (grüne Kreise) zu sehen. Außerdem werden die Joints des Kopfes und des Torsos durch rote Kreise visualisiert. Der Benutzer wird blau eingefärbt. Rechts befindet sich eine graphische Illustration, die verdeutlicht, welche Pose der Benutzer einnehmen soll. Darunter befindet sich der Anweisungstext.

Anfangs wartet MapKin darauf, dass eine Person im Bild erkannt wird. Solange dies nicht geschehen ist, erscheint der Text „Warte auf Erkennung...“ . Nun sollte man sich so positionieren, dass der eigene Körper möglichst vollständig im Bild zu sehen ist. Es ist ausreichend,

wenn nach unten hin nur die Oberschenkel zu sehen sind, der Kopf muss jedoch vollständig sichtbar sein. Insbesondere ist auch darauf zu achten, dass beide Hände zu sehen sind. Diese sollten sich also nicht hinter dem Rücken befinden, sondern locker nach unten hängen. Der Abstand zur Kinect ist so zu wählen, dass der Benutzer die Arme vollständig zur Seite ausstrecken kann, ohne dass diese das Bild verlassen. Jedoch sollte der Abstand auch nicht zu groß gewählt werden, da ansonsten die Karte später schlecht zu erkennen ist. Dies hängt jedoch auch stark von der Größe des benutzten Bildschirms ab. Grundsätzlich sollte ein Abstand von 6 Metern nie überschritten werden. Außerdem ist darauf zu achten, dass sich die Kinect etwa auf Tischhöhe befindet und möglichst frontal auf den Körper gerichtet wird. Falls die Erkennung nicht sofort geschieht, reicht es meist aus, wenn sich der Benutzer ein wenig bewegt und dabei die Hände nach vorne vor dem Körper hält. Sobald die Erkennung stattgefunden hat, wird die Silhouette des Benutzers blau gezeichnet und der Kalibrierungsprozess startet. Dieser ist notwendig, um die Karte später präzise steuern zu können. Näheres dazu befindet sich im Kapitel *3.3* auf Seite 30. Während des Kalibrierungsprozesses werden Punkte über das Kamerabild gezeichnet – dies sind Joint-Indikatoren, welche verschiedene Punkte des Körpers repräsentieren. In diesem Fall den Kopf, den Torso, sowie beide Hände. Die Indikatoren sind im Normalfall rot, werden jedoch grün, wenn man die Arme ausstreckt. Weiterhin werden während der Kalibrierung transparente, weiße Felder, sogenannte Ziele eingezeichnet. Dies sind die Felder, auf die mit den Händen gezielt werden muss, um die Kalibrierung erfolgreich abzuschließen.
Im ersten Schritt der Kalibrierung wird die Armlänge ermittelt. Dazu müssen die Arme etwa schulterbreit vollständig nach vorn ausgestreckt werden. Die Handfläche sollte dabei Richtung Bildschirm bzw. Leinwand zeigen. Die Hände müssen so positioniert werden, dass die Joint-Indikatoren beider Hände für mindestens 1,5 Sekunden in den weißen Zielen über den Schultern verweilen. Außerdem müssen die Hände als *vorne* erkannt werden, was sich an grünen Handindikatoren erkennen lässt.
Wurde dieser Schritt erfolgreich abgeschlossen, findet direkt im Anschluss die Kalibrierung des Handabstandes zur komfortablen Umsetzung der Zoomgeste statt. Es erscheinen zwei transparente weiße Felder, die über den Schultern beginnen und sich bis zum Bildaußenrand erstrecken. Die Joint-Indikatoren der Hände müssen sich nun innerhalb der weißen Felder befinden und für 1,5 Sekunden möglichst unbewegt bleiben. Dabei sollte der Handabstand so gewählt werden, dass dieser komfortabel erreicht werden kann. Er sollte also nicht zu groß gewählt werden. Insbesondere ist darauf zu achten, dass die Hände während dieses Schritts der Kalibrierung als *vorne* erkannt werden. Sind die Indikatoren bei einer komfortablen Haltung nicht grün, sollte ein geringerer Handabstand gewählt werden, wodurch die Arme automatisch weiter nach vorne gestreckt werden.
Ist dieser Schritt abgeschlossen, ist die eigentliche Kalibrierung beendet. Um in den Kartenmodus überzugehen, müssen die Hände nun nach unten hängen. Dies wird wieder durch ein transparentes weißes Feld verdeutlicht. Hierbei gibt es jedoch kein Zeitintervall. Die Kartenansicht wird angezeigt, sobald sich beide Hände in diesem Feld befinden.

4.3.2 Steuerung der Karte

Im Kartenmodus wird die Anzeige vollständig von einer Landkarte eingenommen (siehe Abbildung 4.2). Die Hände werden weiterhin durch Joint-Indikatoren dargestellt. Die Indikatoren für Torso und Kopf sind für die Kartensteuerung irrelevant und werden daher ausgeblendet. Die Farben der Kreise sagen aus, in welchem Modus sich die Hand befindet. Wird die Hand ausgestreckt, befindet sie sich im Modus *vorne*. Der Kreis wird grün gezeichnet. Lässt man die Hand locker nach unten neben dem Körper hängen, befindet sie sich im Modus *hängend* – der Indikator wird gelb gezeichnet. In allen anderen Situationen befindet sich die Hand im Modus *sonstiges*. Dies wird durch einen roten Indikator dargestellt. Oben links in der Ecke des Fensters ist ein blaues Symbol zu sehen. Dies ist der Gestenindikator, welcher anzeigt, welche Geste aktuell ausgeführt wird (siehe Abbildung 4.3).
Eine Übersicht über alle Gesten für die Steuerung der Karte gibt Tabelle 4.1.

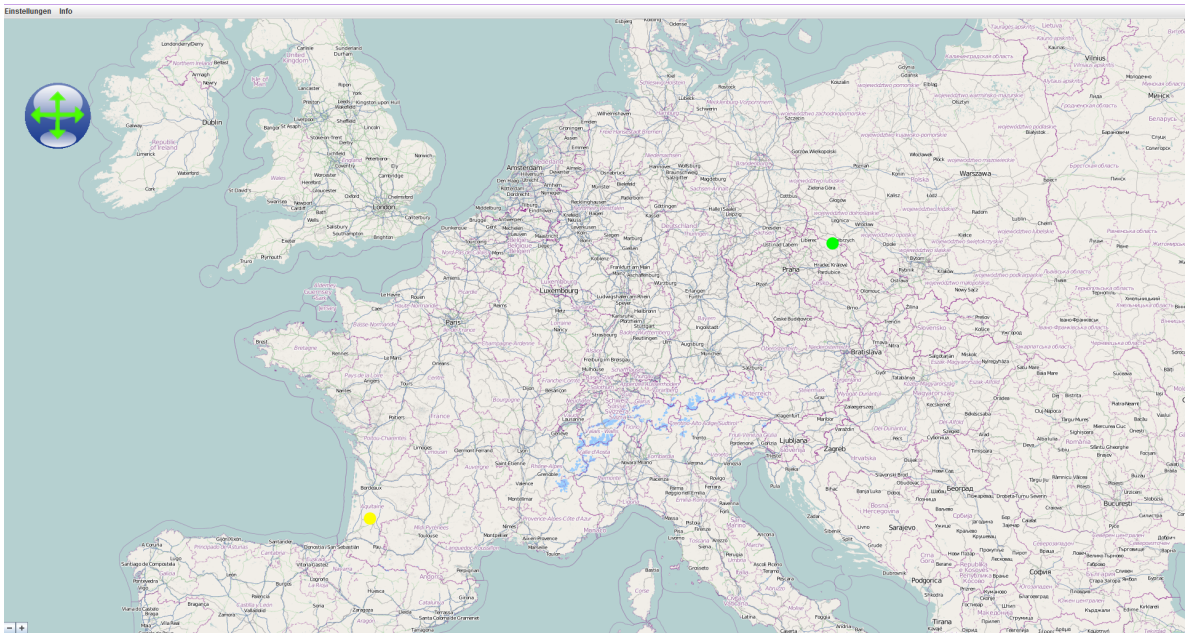
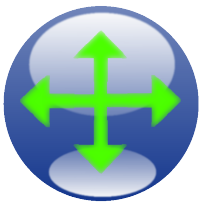
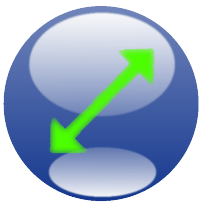


Abbildung 4.2: Ansicht der Karte. Zu erkennen sind die beiden Handindikatoren. Der linke Indikator ist gelb (Hand hängt), der rechte grün (Hand ist vorne). Dies ist die Geste für das Verschieben der Karte, was durch den Gestenindikator oben links im Bild visualisiert wird.



(a) Gestenindikator für die Verschieben-Geste



(b) Gestenindikator für die Zoomen-Geste

(c) Gestenindikator für die Geste zum Setzen von Routenpunkten

(d) Gestenindikator für die Geste zum Löschen der Routenpunkte

Abbildung 4.3: Übersicht über die Symbole des Gestenindikators. Der Gestenindikator zeigt an, welche Geste aktuell ausgeführt wird. Er befindet sich in der Kartenansicht am oberen linken Bildrand.

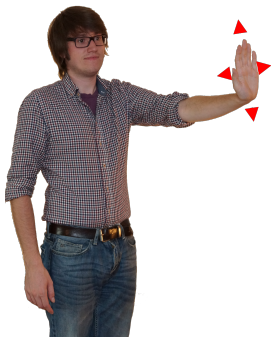
Funktion der Geste	Geste	Kurzbeschreibung der Geste
Karte verschieben		Linke Hand *hängend* Rechte Hand *vorne* Karte folgt Bewegungen der rechten Hand.
Zoomen		Beide Hände müssen Status *vorne* haben Hände zusammenführen: Herauszoomen Hände auseinanderziehen: Hineinzoomen Pro Geste 2x Zoomen möglich. Richtungswechsel jederzeit möglich.
Routenpunkt setzen		Linke Hand *vorne* Rechte Hand *hängend* Handindikator für 0,8 Sekunden auf Position auf Karte verweilen lassen setzt Routenpunkt. Erster Routenpunkt ist Startpunkt. Jeder weitere Punkt ist Zielpunkt. Bestehende Zielpunkte werden überschrieben. Zwischen Setzen zweier Punkte, muss linke Hand abgesetzt werden. Nur Punkte innerhalb Deutschlands möglich.
Alle Routenpunkte löschen		Beide Hände müssen nah am Körper neben den Kopf gehalten werden.

Tabelle 4.1: Übersicht über die Gesten zur Steuerung der Karte. Eine genauere Beschreibung der Gesten findet sich in Kapitel 4.3.2

Verschieben der Karte

Um die Karte zu verschieben, muss sich die linke Hand im Modus *hängend* und die rechte im Modus *vorne* befinden. Bewegt man nun die rechte Hand, folgt die Karte der Hand, solange diese *vorne* ist. Indem man die rechte Hand wieder näher zum Körper bewegt, verlässt sie den Modus *vorne* — die Karte folgt nun nicht mehr der Hand.

Zoomen der Karte

Zum Zoomen der Karte, müssen beide Hände den Status *vorne* haben. Um heraus zu zoomen, bewegt man die Arme mit einem großen Handabstand nach vorne und führt sie dort langsam zusammen. Unterschreiten die Hände eine bestimmte Distanz, wird das Zoomevent ausgelöst. Beim Unterschreiten einer weiteren, kleineren Distanz wird ein zweites Zoomevent ausgelöst. Pro Geste sind maximal zwei Zoomevents in eine Richtung (also hinein, bzw. heraus) möglich.
Um in die Karte hinein zu zoomen, werden die Hände mit einer sehr kleinen Distanz nach vorne geführt und dort langsam geöffnet. Die Funktionsweise entspricht der, beim Herauszoomen. Es ist möglich, die Zoomrichtung zu wechseln, ohne die Hände zurück an den Körper zu nehmen. Beim Hineinzoomen befindet sich der Punkt, der beim Auslösen des Zoomevents in der Mitte zwischen den zwei Kreisen der Hände befindet, auch nach dem Zoomevent in der Mitte der Joint-Indikatoren der Hände. Beim Herauszoomen bleibt der Kartenmittelpunkt jedoch grundsätzlich konstant. Wird beim Zoomen die größte oder kleinste Zoomstufe erreicht, wird dies durch einer Meldung angezeigt. Diese verschwindet nach wenigen Sekunden automatisch.

Routenpunkte setzen

Um einen Routenpunkt zu setzen, muss die rechte Hand den Status *hängend* haben und die linke Hand *vorne* sein. Nun führt man die linke Hand zu der Stelle, an der man den Startpunkt der Route wählen möchte und lässt sie dort mindestens 0,8 Sekunden verweilen. Es erscheint ein grünes Stecknadel-Symbol mit der Bezeichnung „A“ an dieser Stelle. Hat man jedoch eine Position gewählt, an der sich keine Straßen befinden, wird automatisch der nächstgelegene Punkt an der nächstgelegenen Straße gewählt. Um einen Zielpunkt zu setzen, geht man identisch vor. Dabei ist jedoch darauf zu achten, dass die linke Hand zwischen dem Setzen von zwei Punkten kurz den Status *hängend* oder *vorne* einnehmen muss. Ein Zielpunkt wird durch ein orangenes Stecknadelsymbol mit der Bezeichnung „B“ symbolisiert. Beim Versuch weitere Punkte zu setzen, wird jeweils der vorherige Zielpunkt überschrieben. Die Route wird direkt nach dem Setzen des Zielpunktes berechnet und in die Karte eingezeichnet (siehe Abbildung 4.4). Weiterhin erscheinen am oberen Rand der Karte Informationen über die Distanz und die Reisezeit der Route. Dabei bezieht sich die Reisezeit auf eine Autofahrt.

Abbildung 4.4: Kartenansicht mit berechneter Route. Zu erkennen sind die Routenpunkte A und B als farbige Stecknadelsymbole, die Route als rote Linie, sowie die Routeninformationen (Entfernung in Kilometern, sowie benötigte Zeit mit einem Auto in Stunden) am oberen Bildrand. Am unteren Bildrand sind die beiden gelben Handindikatoren zu sehen. Die beiden Hände haben den Status *hängend*. Da dies keine Geste ist, bleibt der Gestenindikator (oben links im Bild) neutral.

Momentan können Routen nur innerhalb von Deutschland berechnet werden. Wird ein Routenpunkt außerhalb von Deutschland gesetzt, wird der nächstgelegenste Punkt auf einer Straße innerhalb Deutschlands gewählt.

Routenpunkte löschen

Die Geste für das Löschen aller Routenpunkte besteht darin, beide Hände über den Schultern auf Kopfhöhe dicht am Körper zu halten. Diese Pose muss für mindestens 0,8 Sekunden gehalten werden. Nun werden beide Routenpunkte gelöscht. Die eingezeichnete Route, sowie die Routeninformationen am oberen Bildschirmrand verschwinden.

Sonstiges

Während die Karte gesteuert wird, darf sich der Abstand zwischen Kinect und dem Benutzer nicht zu stark vom Abstand während der Kalibrierung unterscheiden. Sollte dies doch

geschehen, erscheint eine entsprechende Meldung, die den Benutzer dazu auffordert, sich wieder näher zur Kinect zu bewegen, bzw. einen größeren Abstand zur Kinect einzunehmen. Während diese Meldung erscheint ist eine Steuerung der Karte nicht möglich.
Bewegt sich der Benutzer aus dem Sichtfeld der Kinect, wird nach kurzer Zeit wieder in den Kalibrierungsmodus gewechselt. Das Tiefenbild ist sichtbar. Sobald der Benutzer wieder von der Kinect erkannt wird, wird zurück in den Kartenmodus gewechselt und der Benutzer kann die Karte wieder steuern. Eine erneute Kalibrierung ist nicht nötig. Es ist jedoch anzumerken, dass MapKin nicht erkennen kann, ob die neu erkannte Person der vorherigen Person entspricht. Sollte der Benutzer gewechselt haben, wird die Kartensteuerung mit der ursprünglichen Kalibrierung nicht präzise möglich sein.
Für diesen Fall gibt es unter dem Menü „Einstellungen“ die Funktion „Kalibrierung zurücksetzen“. Dabei werden auch berechnete Routen gelöscht.
Wird diese Funktion gewählt, kehrt die Ansicht in den Kalibrierungsmodus zurück und die Kalibrierung kann erneut vorgenommen werden. Danach befindet sich die Karte wieder in der ursprünglichen Ansicht.

4.3.3 Sprache

Bei der Ausführung von MapKin kann zwischen den Sprachen Deutsch und Englisch gewählt werden. Bei einem normalen Start, richtet sich die Auswahl der verwendeten Sprache nach der aktuellen Systemeinstellung des Betriebssystems. Um die Sprache von MapKin manuell festzulegen, kann beim Starten der Parameter *en* für Englisch, oder *de* für Deutsch angegeben werden. Um dies zu vereinfachen, werden zwei Batchdateien beigelegt, die diese Aufgabe übernehmen.

4.3.4 Fehlermeldungen

Ist die Kinect beim Start des Programms nicht angeschlossen oder fehlt der richtige Treiber für die Kinect, wird eine Meldung ausgegeben, die besagt, dass die Kinect nicht gefunden werden konnte. MapKin beendet sich daraufhin automatisch.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Im Verlauf dieser Bachelorarbeit wurde ein System entwickelt, mit dem es möglich ist, Karten per Gestensteuerung zu bedienen. Dieses soll bei Veranstaltungen wie dem Tag der Wissenschaft zum Einsatz kommen. Für die Erkennung der Gesten wurde die Kinect, ein Gerät, das ein dreidimensionales Bild des Raumes erstellen kann, eingesetzt. Mittels der Software OpenNI und NiTE kann dieses Bild interpretiert werden und auf das sogenannte *Skelett* zugegriffen werden, wodurch eine Erkennung der Gesten möglich ist. Da es zwei Versionen von OpenNI und NiTE gibt, deren API nicht zueinander kompatibel ist, wurden zwei Versionen von MapKin entwickelt, die jeweils mit einer Version von OpenNI beziehungsweise NiTE funktionieren. Die Funktion und das Aussehen der beiden MapKin-Versionen ist jedoch identisch.
Bei der Wahl der Gesten wurde darauf geachtet, dass diese sehr ähnlich zu den Gesten sind, die auf Touchscreens verwendet werden, um die Erlernung zu vereinfachen. Mit der rechten Hand wird die Karte verschoben, mit der linken werden Routenpunkte gesetzt. Das Zoomen wurde mittels einer „Pinch-to-Zoom“-Geste umgesetzt. Um Routenpunkte zu löschen, müssen die Hände auf Kopfhöhe gehalten werden. Da eine präzise Steuerung für verschiedene Benutzer ermöglicht werden sollte, wurde eine Kalibrierung implementiert. Diese misst unter anderem die Armlänge und den Armabstand bei der Zoom-Geste. Dadurch kann sich MapKin auf verschiedene Proportionen der Benuter einstellen.
Für die Routenberechnung wurde ToureNPlaner verwendet. Dieser bietet ein API an, das Koordinaten entgegennimmt und eine Route als Ergebnis versendet. Die Karte wird von JXMapViewer, einer Swing-Komponente dargestellt. Diese zeigt Karten-Tiles mit OpenStreetMap-Karten an, die on-the-fly heruntergeladen werden. JXMapViewer wird durch Emulation von Mausgesten gesteuert. Dadurch ist es grundsätzlich leicht möglich, JXMapViewer durch eine andere Ansicht zu ersetzen und die Gestensteuerung für andere Einsatzzwecke zu verwenden. Routen und andere Elemente werden mittels eines Painters, den JXMapViewer anbietet über die Karte gezeichnet.
MapKin konnte in einer frühen Entwicklungsphase bereits auf einem Tag der Wissenschaft von Benutzern erfolgreich getestet werden.

5.2 Probleme

An dieser Stelle sollen einige Probleme, die während der Arbeit auftraten, dargestellt werden. Zunächst war es nicht vorgesehen, zwei Versionen von MapKin zu entwickeln. Dies hätte

dadurch vermieden werden können, indem direkt OpenNI 2 genutzt worden wäre. Jedoch sollten durch die Wahl der stabilen Version 1.5 anfängliche Probleme vermieden werden. Im Nachhinein erwies sich Version 2 jedoch bereits als äußerst stabil. Die Änderung von MapKin, um die API der neuen OpenNI-Version zu verwenden, war jedoch nicht besonders aufwendig. Die größte Änderung betrifft die Erkennung neuer Personen. Gab es bei Version 1.5 noch Listener, die eine Aktion ausführten, wenn eine neue Person erkannt, oder eine bereits bekannte verloren wurde, so musste in Version 2 nun bei jedem Frame über die Liste der Benutzer iteriert werden, um nach neuen Status für Benutzer zu suchen. Da diese Änderung jedoch auch in den Beispielen, die NiTE in Version 2 beilegt, zu sehen war, konnte die Umstellung schnell erfolgen.
Als großes Problem erwies es sich, MapKin unter Linux zu betreiben. Letztendlich konnte keine funktionierende Konfiguration gefunden werden. Jedoch wurde nur unter Ubuntu 13.04 in der 64bit Version getestet. Hier wurde sowohl OpenNI 1.5 zusammen mit NiTE 1.5, als auch OpenNI 2 und NiTE 2, jeweils in der 64bit-Version installiert und getestet. Für Version 1.5 ist der auf der OpenNI-Seite [Ope13c] erhältliche Treiber *OpenNI-Compliant Sensor Driver v5.1.2.1* verwendet worden, für Version 2, der für diese Version verwendbare Freenect-Treiber [ope13a]. Alle erforderlichen Abhängigkeiten wurden installiert.
Beim Test der Beispielprogramme von OpenNI stellte sich heraus, dass in Version 1.5 die Kinect nicht erkannt werden konnte. In Version 2 hingegen wurde die Kinect erkannt und initialisiert, was am eingeschalteten Infrarotprojektor zu erkennen war. Jedoch konnte weder ein Farb- noch ein Tiefenbild angezeigt werden. Jedes Programm, einschließlich MapKin, das dies versuchte, fror ein. Auch der Befehl *sudo modprobe -r gspca_kinect*, der OpenNI erlaubt, die Kontrolle über die Kinect zu übernehmen, wurde verwendet.
Ein weiteres Problem besteht darin, dass bei der Ausführung von MapKin gelegentlich die Java Virtual Machine abstürzt, jedoch nur bei Verwendung von OpenNI 1.5. Dies ist jedoch nur der Fall, wenn MapKin schnell hintereinander geschlossen und wieder geöffnet wird. Beim normalen Betrieb sollte dies nicht auffallen. Falls es dennoch auftritt, äußert sich dieses Problem darin, dass MapKin nicht startet. In diesem Fall sollten einige Sekunden gewartet werden, bevor MapKin erneut geöffnet wird. Vermutlich tritt dieses Problem dann auf, wenn die Verbindung zur Kinnect noch nicht vollständig geschlossen wurde, aber bereits eine neue Initialisierung erfolgt.

5.3 Aussicht

Zum Schluss wird in diesem Kapitel noch eine kleine Aussicht geben, inwieweit MapKin weiterentwickelt werden kann, beziehungsweise welche Funktionen implementiert werden können.

5.3.1 Die neue Kinect

Zusammen mit der Xbox One hat Microsoft am 21. Mai 2013 eine neue Version der Kinect vorgestellt. Diese soll ein höher aufgelöstes Tiefenbild erstellen können. Im Gegensatz

zur Kinect der ersten Generation wird es zwei verschiedene Modelle für die neue Version geben – eine für die Xbox One und eine für PCs. Die Version für PCs wird sich ohne Adapter über USB 3 anschließen lassen [com13]. Da die neue Kinect zum Zeitpunkt der Erstellung dieser Arbeit noch nicht veröffentlicht wurde, kann nur spekuliert werden, was mit der neuen Kinect möglich sein wird. Sollte die Auflösung ausreichen, selbst die Finger der Benutzer präzise erfassen zu können, wären dadurch völlig neue Gesten möglich. Es müssten keine ausladenden großen Gesten mit den Armen gemacht werden, sondern man könnte die Gesten tatsächlich an die Steuerung an Touchscreengeräte anlehnen, indem alle Gesten mit den Fingern durchgeführt werden. Allerdings könnte dadurch die Fehlbedienungen zunehmen, da sich Finger auch während normaler Gestikulation bewegen. Hier müssten neue Mechanismen entwickelt werden, um zwischen Gestikulation und Steuerung zu unterscheiden.

5.3.2 Suche nach Städtenamen

Während der Präsentation am Tag der Wissenschaft wurde oft gefragt, ob man Städte auch direkt per Name suchen kann, anstatt diese von Hand anzuvisieren. Aktuell ist das noch nicht möglich. Das „Schreiben“ von Städtenamen mittels Gesten würde jedoch mindestens eben so lange dauern. Außerdem wäre das Erlernen der Gesten für das Schreiben der einzelnen Buchstaben für Benutzer nicht in kurzere Zeit möglich.
An der Universität Stuttgart wird zur Zeit in einer weiteren Bachelorarbeit ein anderer Ansatz verfolgt. Stefanie Bahle arbeitet an einer binären Suche für Städtenamen auf einem Touchscreen. Möglicherweise lässt sich dieses Projekt in MapKin integrieren.

5.3.3 Routenberechnung außerhalb Deutschlands

Momentan ist die Routenberechnung, wie bereits zuvor erwähnt, nur innerhalb von Deutschland möglich. Dies liegt daran, dass nur Kartendaten von Deutschland auf den Servern von ToureNPlaner hinterlegt sind. Es ist leicht möglich, Kartendaten von weiteren Ländern in die Datenbank aufzunehmen, wofür jedoch die Entwickler des ToureNPlaners verantwortlich sind.

5.3.4 Weitere Einsatzmöglichkeiten

Es ist denkbar, MapKin dahingehend anzupassen, dass anstatt Karten auch andere Anzeigen gesteuert werden können. Da JXMapViewer mittels Mausgesten gesteuert wird, ist es ohne großen Aufwand möglich, die Karte beispielsweise durch eine Bildanzeige zu ersetzen. Dabei können weiterhin die Zoom- und Verschieben-Geste eingesetzt werden. Dies kann für verschiedene Szenarien hilfreich sein. So könnte beispielsweise in einem Operationssaal ein Röntgenbild betrachtet werden, ohne dabei etwas berühren zu müssen, wodurch die Handschuhe unsteril werden. Auch das Betrachten von Fotos auf dem Fernseher wäre

möglich. Dadurch wäre es jedem Betrachter möglich, die Steuerung zu übernehmen, ohne die sonst dafür verwendete Fernbedienung weitergeben zu müssen.

Literaturverzeichnis

[com13] *Computer And Video Game: New Kinect*, 2013. http://www.computerandvideogames.com/416723/xbox-one-kinect-not-compatible-with-pc-says-microsoft/. (Zitiert auf Seite 49)

[Cup13] Cup of Pixel. *Cup of Pixel: Installation on Linux using Libfreenect*, 2013. http://cupofpixel.blogspot.de/2013/04/how-to-use-openni-2-nite-2-kinect-on.html. (Zitiert auf Seite 17)

[eng13] *Vorstellung des SDKs für Windows: Microsoft SDK*, 2013. http://www.engadget.com/2011/06/16/microsoft-launches-kinect-for-windows-sdk-beta-wants-pc-users-t/. (Zitiert auf Seite 15)

[iFi13] iFixIT. *iFixIt: Kinect Teardown*, 2013. http://www.ifixit.com/Teardown/Microsoft+Kinect+Teardown/4066/3. (Zitiert auf Seite 14)

[kin13a] *Kinect: Fitnect*, 2013. http://www.fitnect.hu/features/. (Zitiert auf Seite 11)

[kin13b] *Kinect: Rollstuhl*, 2013. http://www.heise.de/tr/artikel/Rollstuhl-mit-Kinect-Anschluss-1262882.html. (Zitiert auf Seite 11)

[kin13c] *Kinect: Schaubild*, 2013. http://upload.wikimedia.org/wikipedia/commons/6/67/Xbox-360-Kinect-Standalone.png. (Zitiert auf Seite 13)

[Mic13a] Microsoft. *Kinect SDK: Download*, 2013. http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx. (Zitiert auf den Seiten 17 und 36)

[Mic13b] Microsoft. *Micrsosoft: Kinect Guide*, 2013. http://download.microsoft.com/download/B/E/5/BE5C6674-32E0-488B-8CB6-8B859744E02E/X169902801bro_cropped.pdf. (Zitiert auf Seite 37)

[ope13a] *Linux-Treiber für OpenNI 2: Download*, 2013. https://github.com/piedar/OpenNI2-FreenectDriver/. (Zitiert auf den Seiten 37 und 48)

[ope13b] *Migrationserklärung für OpenNI 2*, 2013. http://www.openni.org/openni-migration-guide/. (Zitiert auf Seite 18)

[Ope13c] OpenNI. *OpenNI 1.5: Download*, 2013. http://www.openni.org/openni-sdk/openni-sdk-history-2/#.Ue-eN23TJqU. (Zitiert auf den Seiten 35, 37 und 48)

[Ope13d] OpenNI. *OpenNI 2: Download*, 2013. http://www.openni.org/openni-sdk/#.Ue-nHG3TJqU. (Zitiert auf den Seiten 36 und 37)

[osm13] *OpenStreetMap Homepage*, 2013. http://www.openstreetmap.org. (Zitiert auf Seite 19)

[Pri13] PrimeSense. *NiTE 2: Download*, 2013. http://www.openni.org/files/nite/#.Ue-nkW3TJqU. (Zitiert auf den Seiten 36 und 37)

[Sen13] SensorKinect. *SensorKinect: Download*, 2013. https://github.com/avin2/SensorKinect. (Zitiert auf den Seiten 16, 36 und 37)

[Tou13] ToureNPlaner. *ToureNPlaner: Karte*, 2013. http://algohol01.informatik.uni-stuttgart.de/#. (Zitiert auf Seite 19)

Alle URLs wurden zuletzt am 27. 08. 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift