A Gesture Recognizer for Multimodal Geographic Information Systems

Matthias Hinz
University of Münster / Institute for Geoinformatics
Heisenbergstraße 2
D-48149 Münster, Germany

matthias.hinz@uni-muenster.de

ABSTRACT

In this paper I am presenting a browser-based GIS application that supports interaction via Natural User Interface (NUI). A gesture recognizer is described, that is designed for the GIS, but a loosely-coupled independent application by itself. Challenges and questions that occurred during the development process will be discussed. I am proposing that gesture recognizers are value-adding components for map applications and adequate alternatives to traditional mouse and keyboard interfaces.

Categories and Subject Descriptors

H.5.2 [**INFORMATION INTERFACES AND PRESENTATION**]: User Interfaces – *Graphical user interfaces (GUI), Input devices and strategies, Interaction styles.*

General Terms

Algorithms, Design, Experimentation, Reliability

Keywords

Multimodal GIS, Kinect, Gesture Recognition, Natural User Interface

1. INTRODUCTION

Since the invention of the first electronic computers in the mid of the 20th century - large, room-filling machines using punch-cards and keyboard-like input devices - computer hardware has been improved consistently in terms of performance, storage, mobility and variability of input devices. The first touch-screens came up in the 1980s already, starting with the Hewlett-Packard 150, launched in 1983. After the emergence of mobile devices in the 1990s, multi-touch interfaces were not only present in public sectors and consumer electronics; they also became part of todays' PDA's, Notebooks and smartphones. Many of these devices include speech recognizers as well. There is a wide range of multimedia-devices available, permitting new interaction techniques between humans and computers. Some of the most recent gadgets for gesture recognition are related to gaming and interactive virtual reality applications, but they are also useful in domains such as teaching, computer-monitored experiments and GIS: The Nintendo Wii Remote[1], Playstation Eye/Move[2], Microsoft Kinect for Windows[3], Kinect for Xbox 360[4] and Leap Motion[5]. [1]

[1] Nintendo Wii: https://www.nintendo.com/wii

[2] Playstation Eye Camera: http://us.playstation.com/ps3/accessories/playstation-eye-camera-ps3.html

[3] Microsoft Kinect for Windows: http://www.microsoft.com/en-us/kinectforwindows/

Gesture recognizers are integral components of so called Natural User Interfaces (NUI). Within the field of geographic information technology and science, interactive GIS applications enhanced by NUIs can be of great benefit.

The following paper presents a prototype GIS application that supports speech- and gesture-interaction. This application has been developed by geoinformatics master students during a seminar at the University of Münster, Germany. The description focusses on the gesture recognition component, while voice recognition is only considered for completeness. As a use-case, the application is proposed to aid urban planning and to leverage the discourse between urban planners and citizens about the infrastructure of populated places. After introduction and related work, chapter 3 gives an overview over the GIS application first, and then proceeds with details about the gesture recognizer and its architecture. Chapter 4 follows with a quick and informal evaluation done by students from the seminar. Chapter 5 discusses design questions and challenges that occurred during the development process. It is leading up to the conclusion of this paper and further works.

2. RELATED WORK

Gesture recognition and Natural User Interaction are important and ongoing topics within research on Human Computer Interaction (HCI): Villaroman et al [1] evaluated the use of Kinect devices and gesture recognition-assisted learning activities within classroom environments. The hardware- and software used by the authors is similar to the configuration described in the current paper. The authors concluded that Kinect in conjunction with OpenNI [2] provides students with hands-on experiences that are useful to teach Natural User Interaction in HCI-courses. Stannus et al [3] presented a prototype application for gestural navigation in Google Earth. They also used the same hardware as described in this paper. Unfortunately, gesture navigation was outperformed by mouse and keyboard interaction in every criterion they used for their evaluation, i.e. naturalness, speed, strain and accuracy. Daiber et al [4] presented approaches to interact with GIS systems via multi-touch gestures combined with foot-gestures. They used a large-scale display where they performed zoom-, pan- and selection- operations amongst others. Nick Gillian et al [5] [6] developed a number of machine-learning algorithms for gesture recognition, which are part of the Gesture Recognition Toolkit [7], that will be mentioned and used in this research. Wobbrock et al [8] conducted a study about user-defined gestures on a tabletop touchscreen-surface. They provided a taxonomy for surface-gestures and a set of user-defined gestures which fulfill user needs and technical criteria like ease of recognition, consistency,

[4] Kinect for Xbox 360: http://www.xbox.com/en-US/Kinect

[5] Leap Motion: https://www.leapmotion.com/

reversibility and versatility through aliasing. Hoang et al [9] presented an augmented-reality approach for in-situ modelling of 3D objects, performing hand gestures with Ultrasonic Gloves and wearing a head-mounted display.

3. A MULTIMODAL URBAN PLANNING GIS PROTOTYPE

3.1 Overview

The Urban Planning GIS is a browser-based application designed to aid urban planners and citizens of a populated area in an interactive discourse. It is a casual map application where urban planners and citizens can create points and lines, add descriptions and comments. It also contains a tool for taking measurements on the map and for navigating to the *home location* and other locations that have been saved. The browser-interface also contains a speech recognizer based on the Google speech API, which enables users to execute speech commands like *left*, *right*, *zoom in*, *start measurement* and *go home*. The application can be handled via keyboard and mouse on a normal PC, but in conjunction with the gesture recognizer component described below, with a beamer or large screen and a headset, it can be controlled from a Natural User Interface. A screenshot of the Graphical User Interface is displayed in Figure 1 below.

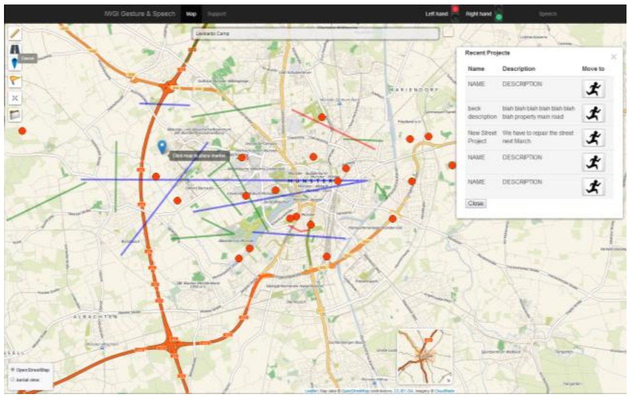

Figure 1: Screenshot of the Urban Planning GIS - GUI

3.2 Gesture Recognizer

The gesture recognizer is a standalone desktop application written in C++. The source code as well as binaries and a technical documentation are publically accessible on GitHub[6]. It was developed with Visual Studio 2012 for 32-bit window platforms. It was successfully tested on windows 7 and 8 operating systems and it is compatible with 64-bit systems. Support for operating systems other than windows was not in scope of the project.

Combining a browser-based GIS with a standalone desktop application may seem odd at first thought, but the proposed solution is simple from the users' perspective and flexible from the perspective of software developers and maintainers: It is simple, because the setup process involves downloading and unzipping a folder only. The included executable file *GRT_Predict.exe* will run immediately, except for the use of Microsoft Kinect for Windows, which requires an additional driver from Microsoft Kinect homepage. In any case, using a Kinect requires some kind of setup on the client PC in order to configure and support the hardware. Even the pure web-based solution JKinect [10] requires installation of the JKinect Web Browser for running Kinect applications. The system architecture of the GIS offers flexibility to developers because the GUI and the gesture recognizer are independent components. They are interchangeable and they can be modified and maintained separately as long as the interface is preserved. In the seminar from the university, there was another team developing a similar GUI to which is described above, and they were able integrate the gesture recognizer with small effort, at least partially. The clue about the interface between recognizer and GUI is a fairly simple idea: Every signal that is send from the gesture recognizer is either a synthesized mouse-event or a keyboard-shortcut. Following this principle, any application that can be controlled via keyboard and mouse can also be controlled via gestures, if it is designed suitably.

3.3 System Architecture of the Gesture Recognizer

The gesture recognizer was successfully tested with both Kinect versions that are currently available on the market: The Microsoft Kinect for Windows[7] and Kinect for Xbox 360[8]. The app is constructed using the Openframeworks toolkit[9]. The gesture recognition is based on the open-source OpenNI2 SDK [2] as the primary library for Kinect. The SDK was combined with the OpenNI-related NiTE2 API [11] and the likewise related PrimeSense Grab Detector. The recognizer also integrates components from Nick Gillians Gesture Recognition Toolkit (GRT): Parts of the source code are based on the Kinect tutorial from the GRT homepage [7]. One component of application enables recognition of static postures based on the Adaptive Naive Bayes Classifier (ANBC) [5]. Therefore, a separate recording app was used to record costume postures for zoom in, zoom out and tab (see below). A similar component was implemented for recording and recognizing dynamic postures using the Dynamic Time Warping (DTW) classifier [6], but it is deactivated for the final version of the prototype, because the algorithm was either predicting to many false positives or the gestures were hardly recognized. However, this result does not necessary mean that DTW is unsuitable for purposes of the application.

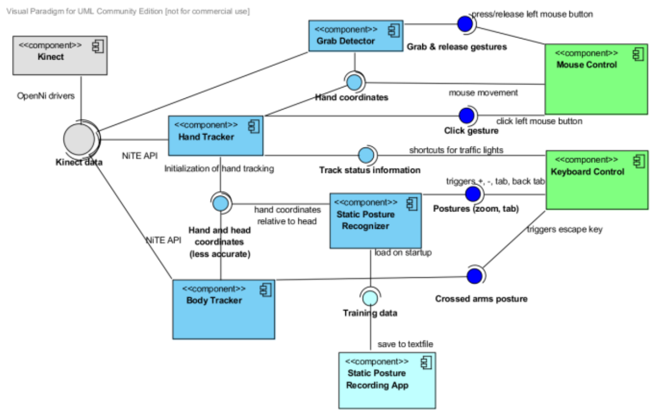
Visual Paradigm for UML Community Edition [not for commercial use]
<<component>> Kinect
OpenNi drivers
Kinect data
NiTE API
NiTE API
<<component>> Grab Detector
Grab & release gestures
press/release left mouse button
<<component>> Mouse Control
Hand coordinates
mouse movement
<<component>> Hand Tracker
Click gesture
click left mouse button
Initialization of hand tracking
Track status information
shortcuts for traffic lights
<<component>> Keyboard Control
triggers +, -, tab, back tab
Postures (zoom, tab)
hand coordinates relative to head
<<component>> Static Posture Recognizer
Hand and head coordinates (less accurate)
triggers escape key
load on startup
<<component>> Body Tracker
Crossed arms posture
Training data
save to textfile
<<component>> Static Posture Recording App

Figure 2 Component diagram depicting the gesture recognizers' software architecture

6 https://github.com/MatthiasHinz/Gesture_Recognizer_of0.8_Nite2_Openni2_x86

7 http://msdn.microsoft.com/en-us/library/jj131033.aspx

8 http://support.xbox.com/en-US/xbox-360/kinect/kinect-sensor-components

9 http://www.openframeworks.cc/

Figure 2 shows the architecture of the gesture recognizer in an abstract schematic way. Most of the components match with one or more C++ classes from the program, except for the *Kinect* (grey) and the *Static Posture Recording App*, which are external components corresponding to the system. It is important to understand the flow of inputs and outputs from one component to another. Both versions of Kinect provide data from a RGB camera, a multi-array microphone and an infrared (IR) emitter and IR depth sensor [12]. Particularly the 3D-data are relevant for the application in order to track parts of the body. The body tracker and hand tracker are both based on the NiTE2 API [11] and used in conjunction: While the body tracker provides data for the *Static Posture Recognizer* (hand coordinates relative to the head position) the hand tracker provides accurate data for the hands. Coordinates from the right hand are used to control the mouse movement. Note that the less accurate hand coordinates from the body tracker are used to initialize the hand tracking, although hand tracking can also be initialized with waving gestures. The NiTE2 based hand- and body- trackers also support some gestures and postures that have been used within the application. Other gestures are supplied by the *Grab Detector* component and the GRT-based *Static Posture Recognizer*. Each gesture either triggers a keyboard shortcut or a mouse click, synthesized by the *Mouse*- and *Keyboard Control* components (green) using the native windows C++ API. These components form up the interface to the Urban Planning GIS GUI. Keyboard shortcuts can also be used to communicate other events than gestures to the GUI. This is done in case of lost or found hands. The GUI communicates with traffic lights to the user whether hand is tracked or not. Table 1 gives an overview about all gestures and postures along with their associated functionalities supported by the gesture recognizer application.

Table 1: Gestures, postures and functionality supported by the gesture recognizer

Gesture / Posture	Description	Associated output / functionality
Grab gesture	Clench a fist with one hand	Left mouse button down (start panning map)
Release gesture	Open a clenched fist	Left mouse button up (stop panning map)
Click gesture	Quickly move one hand towards the sensor	Left mouse button click
Combined click and grab gesture	Perform a grab gesture with one hand Perform a click gesture with the other hand Release the grabbed hand	Left mouse button double click
Waving gesture	Swing one hand from left to right and back for a couple of times	Initializes hands if not automatically detected
Crossed-arms posture	Cross both arms in front of the body	'Escape' key
Zoom-in posture	Hold both hands above your head	'+' key
Tab posture	Put both hand to your right	'tab' key
Backtab posture	Put both hand to your left	'shift' + 'tab' keys

4. EVALUATION

The Urban Planning GIS was informally evaluated in the classroom, by capturing thinking-aloud protocols while users had to perform tasks. Although the gesture recognizer reached a certain level of reliability within the development process, small errors still remain: Some people had problems with doing the grab- and release gestures; others had difficulties to reach parts of the screen with the mouse. Sometimes, the gesture recognizer did not work properly and had to be restarted. The Kinect also seemed much depended on background, lightning and user position. On the other hand, participants started getting confident with the recognizer after a short time of learning and being instructed. Speech and gesture input are intended work conjunction: For some interactions, where gestures are not available or where it would be too complicated to use gestures, speech could be used. However, during the test many words were not recognized properly, or wrong words and sentences were recognized. Combining speech and gestures is an interesting concept, but it would require more reliable speech recognition.

5. DISCUSSION

The development team of the presented prototype application had to do many design- choices and changes during the development process. One of these choices was considering hardware: The team worked with both, Leap Motion and Kinect in parallel, but finally decided for a Kinect-based gesture recognizer because Leap Motion was perceived as less accurate and too sensitive for movements. Also, the Kinect was more suitable for use in conjunction with beamers and large screens, because it allows the user to stand freely. Finally the group chose Kinect, because there were more devices available. Hence the groups' criteria about the hardware were availability, accuracy, sensitivity, ease of use and usefulness according to the proposed NUI configuration.

Another design choice was related to software: Alternative to the OpenNI2 SDK one could also choose for the Microsoft Kinect SDK or the libfreenect software developed by the OpenKinect project[10]. One reason to choose for OpenNI2 was that it fully

[10] http://openkinect.org

supports Microsoft devices, while the support from Microsoft for Kinect for XBox 360 is very limited [12]. The OpenNI2 and NiTE2 APIs were perceived as better understandable. There were many other libraries related to OpenNI that could be combined within the software.

When choosing the right software and hardware, licensing should be considered: Developers should consider whether they need licenses for commercial or non-commercial use and whether licenses of different components are compatible with each other. Both, OpenNI2 and libfreenect are distributed under open source licenses while the Microsoft Kinect SDK is available with a license for non-commercial and for commercial use. In many cases, open-source licenses are not compatible with proprietary software.

Another important point about gesture recognition is reliability and usability: Both are related, because reliable gesture recognition is essential for an application that is convenient to use. Gestures should be intuitive, but unique. If the gesture is not intuitive, users will have difficulties in performing and remembering it. On the other hand, gestures have to be unique in a way that it is not triggered during random movements of the hands and in a way that it can be distinguished from other gestures. The quality of gesture recognition is restricted by hardware and software.

The browser-based GUI solved its purpose as a demo application for urban planning, integrating both speech- and gesture recognition properly. However, the use-cases for urban planners and citizens were quite simple and they have never been discussed or evaluated with neither urban planners nor citizens, so one could argue that the practical usefulness of this application beyond demonstration purposes is highly questionable.

6. CONCLUSION AND FURTHER WORK

Gesture recognition and Natural User Interfaces are a value-adding component for the use of GIS application. This paper presented a prototype application that combines gesture- and speech recognition with a browser-based map interface. According the student evaluations, the program is still error-prone. Especially the speech recognizer makes many false predictions. But it was also shown that students were getting comfortable with the NUI after some training. They actually performed many tasks successfully. Conceptually, any application that can be controlled via mouse and keyboard can also be controlled via gestures if appropriately designed. Hence, the described application offers an adequate alternative to keyboard and mouse interfaces.

The prototype has to be brought to a more reliable and sophisticated state, which was not yet feasible due to restrictions in knowledge, manpower and time. Further work would involve a more in-depth evaluation of the application, in particular considering whether or not the proposed use-case of an Urban Planning GIS somehow matches demands in everyday life. It is also important to explore other domains, where similar GIS applications could be demanded, for instance teaching.

7. REFERENCES

[1] N. Villaroman, D. Rowe and B. Swan, "Teaching Natural User Interaction Using OpenNI and the Microsoft Kinect," *Proceedings of the 2011 conference on Information technology education. ACM,* pp. 227-232, 2011.

[2] PrimeSense Ltd., "OpenNi. The standard framework for 3D sensing," [Online]. Available: http://www.openni.org. [Accessed 21 February 2014].

[3] S. Stannus, D. Rolf, A. Lucieer and W. Chinthammi, "Gestural navigation in google earth," *Proceedings of the 23rd Australian Computer-Human Interaction Conference. ACM,* pp. 269-272, 2011.

[4] F. Daiber, J. Schöning and A. Krüger, "Whole Body Interaction with Geospatial Data," *Smart Graphics. Springer Berlin Heidelberg,* pp. 81-92, 2009.

[5] N. Gillian, R. B. Knapp and S. O'Modhrain, "An adaptive classification algorithm for semiotic musical gestures," *the 8th Sound and Music Computing Conference,* 2011.

[6] N. Gillian, R. B. Knapp and S. O'Modhrain, "Recognition of multivariate temporal musical gestures using n-dimensional dynamic time warping," *Proceedings of the 11th intenational conference on New Interfaces for Musical Expression,* 2011.

[7] N. Gillian, "Gesture Recognition Toolkit," [Online]. Available: http://www.nickgillian.com/software/grt. [Accessed 22 February 2014].

[8] J. O. Wobbrock, M. R. Morris and A. D. Wilson, "User-defined gestures for surface computing," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM,* pp. 1083-1092, 2009.

[9] T. N. Hoang and B. H. Thomas, "Distance-based modeling and manipulation techniques using ultrasonic gloves," *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on. IEEE,* pp. 287-288, 2012.

[10] Octo Technology, "jKinect - Kinectify your web application," [Online]. Available: http://jkinect.com/. [Accessed 21 February 2014].

[11] PrimeSense Ltd., "NiTE 2 API Programmer Tutorial Guide," 25 April 2013. [Online]. Available: http://www.primesense.com/wp-content/uploads/2013/04/PrimeSense_NiTE2API_ProgTutorialGuide_C++Samples_docver0.2.pdf. [Accessed 21 February 2014].

[12] Microsoft Coporation, "Frequently asked questions about Kinect and the Kinect SDK," [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/faq.aspx. [Accessed 21 February 2014].

[13] OpenKinect, "OpenKinect," [Online]. Available: http://openkinect.org. [Accessed 21 February 2014].