Institute of Formal Methods in Computer Science

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis no. 36

Exploring Maps using Leap Motion

Martin Scholz

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Stefan Funke
Supervisor:	Dipl.-Inf. Filip Krumpe
Commenced:	May 5, 2015
Completed:	November 4, 2015
CR-Classification:	H.5.2

Abstract

In this Master's thesis, a system is proposed which allows users to calculate routes between two locations on a map. The system can be controlled by using only hand and finger gestures. The 3D sensor Leap Motion is used to detect the gestures. To find the locations, two approaches were taken. The first one enables the user to search for a location directly on the map. Five input methods were introduced to move the map and zoom in and out of the map. These are called *manual input methods*. The other approach was to search for the locations by entering addresses using *text input methods*. The system employs five different input methods for this approach. To compare the two approaches and their corresponding input methods, three user studies were conducted. The results show that out of the five different *text input methods*, an onscreen keyboard was best suited. The fastest manual input method was a method which is derived from a joystick control and uses finger gestures for zooming. For scenarios where two close locations are used, a combination of the text input and the manual input should be chosen in order to minimize the input time. Possible use cases of the system are ticket vending machines at train stations or indoor navigation systems. Gesture input has an advantage over touchscreens, which are usually used for these systems, in that it is more hygienic since it works touchless. In addition, gesture input can also be used wearing gloves.

Kurzfassung

In dieser Masterarbeit wurde ein System vorgestellt, mit dem Routen berechnet werden können und das vollständig mit Hand- und Fingergesten bedient werden kann. Für die Gestenerkennung wird der 3D-Sensor Leap Motion benutzt. Um den Start und das Ziel der Route einzugeben, wurden zwei unterschiedliche Vorgehensweisen entwickelt. Der Benutzer kann einen Routenpunkt manuell auf einer Karte wählen, indem er die Karte bewegt und zoomt. Alternativ können Routenpunkte auch über Adressen gesucht werden. Für die beiden Vorgehensweisen wurden jeweils fünf unterschiedliche Eingabemethoden entwickelt und in Benutzerstudien miteinander verglichen. Die Ergebnisse dieser Studien zeigen, dass eine Eingabemethode, die einer Tastatur nachempfunden ist, am besten für die textuelle Eingabe geeignet ist. Unter den manuellen Eingabemethoden war eine Methode am besten, die ähnlich wie ein Joystick funktioniert und Fingergesten zum Zoomen verwendet. Liegen Start und Ziel der Route jedoch nah beieinander, bietet es sich an, den zweiten Routenpunkt manuell auf der Karte auszuwählen, nachdem der erste Routenpunkt über die Adresse gefunden wurde, um die Eingabezeit zu minimieren. Mögliche Einsatzgebiete für dieses System sind Ticketautomaten an Bahnhöfen oder Indoor-Navigationssysteme in großen Gebäuden, wie zum Beispiel Einkaufszentren. Vorteile gegenüber Touchscreens, die üblicherweise für diese Systeme eingesetzt werden, sind einerseits die Möglichkeit das System berührungslos und damit absolut hygienisch zu bedienen, andererseits kann es auch mit Handschuhen bedient werden.

Contents

List of Figures

1 Introduction

Gesture control gained lots of attention since it found its way into gaming with the Microsoft Kinect in 2010 [kinb]. In the year 2012, a SDK allowed it to use the Kinect sensor also on the PC [kina]. This created new and more serious use cases for gesture input. Earlier, we introduced *MapKin* [Sch13], a system for controlling a map with hand gestures using the Kinect sensor. The introduction of Leap Motion, a sensor for recognizing hand gestures (see section 3.1) made it possible to work with much more precise gestures. This allows it to use the gesture input parallel to mouse and keyboard while sitting at a desk. In the year 2013, HP introduced the Notebook *ENVY17 Leap Motion SE* [leab] which has a built-in Leap Motion to use gestures without an external device. Gesture recognition is also used in cars. The new BMW 7 series introduced the usage of hand gestures for different actions like answering or declining an incoming phone call or changing the audio volume [ges]. But gesture control can also be used in the office. The program *AirInput* [lead] enables the user to use the Leap Motion as a replacement for the mouse [lead]. Another program called *PRSNTA* [leag] allows the user to change the slides of a presentation using gestures with the Leap Motion. This shows that the gesture input can be equal to the traditional input methods. Chan et al. [CHM15] showed that the Leap Motion can also be used for authentication purposes. With *Leap-Map* [leac] a program was introduced to navigate in maps by only using gestures. Also *Here Maps* worked on integrating the Leap Motion to navigate through 3D models of cities [leaf]. The both above mentioned systems use the gestures only for changing the view of the map. Whereas the system proposed in this thesis, called *MapMotion*, is also able to set route points or searching for addresses to calculate a route by only using gestures.

One possible use case of MapMotion is to search for an address on a ticket vending machine at a train station. Here, two different scenarios are conceivable. The first one searches for the station by name (e.g. *Main station Stuttgart*) or directly by selecting the desired station on the plan. The second scenario is to search for addresses instead of stations. This can be useful if the users do not know the station but only the address they want to go to. The users can search for the desired location again either by typing in the address of the location or select the location directly on the map. Finally, a ticket for the desired route is sold to the user.

A second use case is indoor navigation in large buildings like shopping malls or office buildings. The users can either search for the name of the shop or of a person using the text input methods or selecting the shop or office manually on a plan of the building. Then, instructions are shown, how to walk to the desired destination.

To show the advantages of MapMotion, the alternatives of gesture input have to be considered. Ticket vending machines and indoor navigation systems usually use touchscreen input. Touchscreens have two disadvantages. First, they are used by lots of people every day with their fingers. This can be considered unhygienic. Since the gesture control proposed in this thesis works without touching any surface, it does not have this disadvantage. Another problem with capacitive touchscreens, which are usually used for the aforementioned systems, is that they do not recognize touches if the user wears gloves. In contrast to these touchscreens, the Leap Motion is able to recognizes the hand if the user wears gloves. This can be an advantage especially for the ticket vending machines which are installed outside. The users do not need to take off their gloves for purchasing a ticket.

In this Master's thesis, different input methods were compared in terms of speed and usability. As already mention on the Here Maps homepage [leaf], there are no established input gestures for controlling a map using gestures. For touchscreens on smartphones, the pinch gesture for zooming and the swipe gesture for scrolling are very common. Users have learned these gestures and expect systems to use this gestures. If users however encounter systems using hand gesture recognition for the first time, they usually do not know how to interact. This could be seen in the user studies of this thesis. Some of the participants wanted to interact with the Leap Motion before being told how the gestures work. They tried different gestures until they had an idea, how to control the map. But none of them knew how to move the map on the first try. Since there are no established gestures, completely different input methods were implemented and tested in order to analyze, which methods work best and which of them are the most intuitive ones. The methods can be categorized by *text input methods*, for which the searched location is found by address and *manual input methods* which allows it to search for the location manually on the map. Data of three user studies are used for the comparison. The first user study which was conducted independently of this Master's thesis compared four different text input methods. This study did not use the Leap Motion but the Microsoft Kinect for gesture input. The second user study compared five manual input methods. Last, the best methods from these both studies plus a third input method were compared for different scenarios in the last user study.

Chapter 2 – Description of the System: This chapter gives an overview of the System. The different functions are described an explanation of the different input methods is given.

Chapter 3 – Hardware and implementation The technical background information for the hardware as well as for the software is given in this chapter.

Chapter 4 – First User Study In this chapter, the first user study comparing the five manual input methods is described and analyzed.

Chapter 5 – Second User Study The fifth chapter contains the description and analysis of the second user study comparing the four text based input methods.

Chapter 6 – Third User Study In this chapter, the third user study comparing the best methods of the previous user studies plus the text based keyboard input method is described and analyzed.

Chapter 7 – Summary This chapter sums up the results of the three user studies.

Chapter 8 – Future Work The last chapter shows some remaining weaknesses of MapMotion and proposes ideas on how to improve the system. Additionally possible topics for further research are suggested.

2 Description of the System

2.1 Overview

In this chapter, an overview over the program *MapMotion* is given. The functions are described in detail and an explanation on how to use the system is given. The main function of MapMotion is to calculate a route between two addresses. The starting point and the destination can be set via two completely different approaches.

- By selecting the address directly on the map (called *manual input*)
- or by specifying the address by entering a city name and a street name (called *text input*).

For both approaches, different input methods were implemented, which solely rely on hand and finger gestures. Neither a mouse nor a keyboard is needed. The user can always choose between using a manual input method or a text input method. The manual input methods work directly on the map whereas the text input methods use buttons which are shown in a *drawer*. The drawer is shown on demand on the right side next to the map (see figure 2.13). In the menu at the top of the program, the user can select which input methods to use for the manual input as well as for the text input. Figure 2.1 shows MapMotion.

2.2 Manual Input Methods

In this section, the manual input methods are described. Five different methods were implemented, which are all capable of performing six distinctive operations on the map. First, they can move the map in all directions. Second, each method can zoom in and out of the map. When the highest or lowest zoom level is reached, MapMotion shows a notification that the minimal or maximal zoom level has been reached. Independently of the used input method, the map always zooms in and out with the center of the map being centered. The third operation is to set route points on the map. This

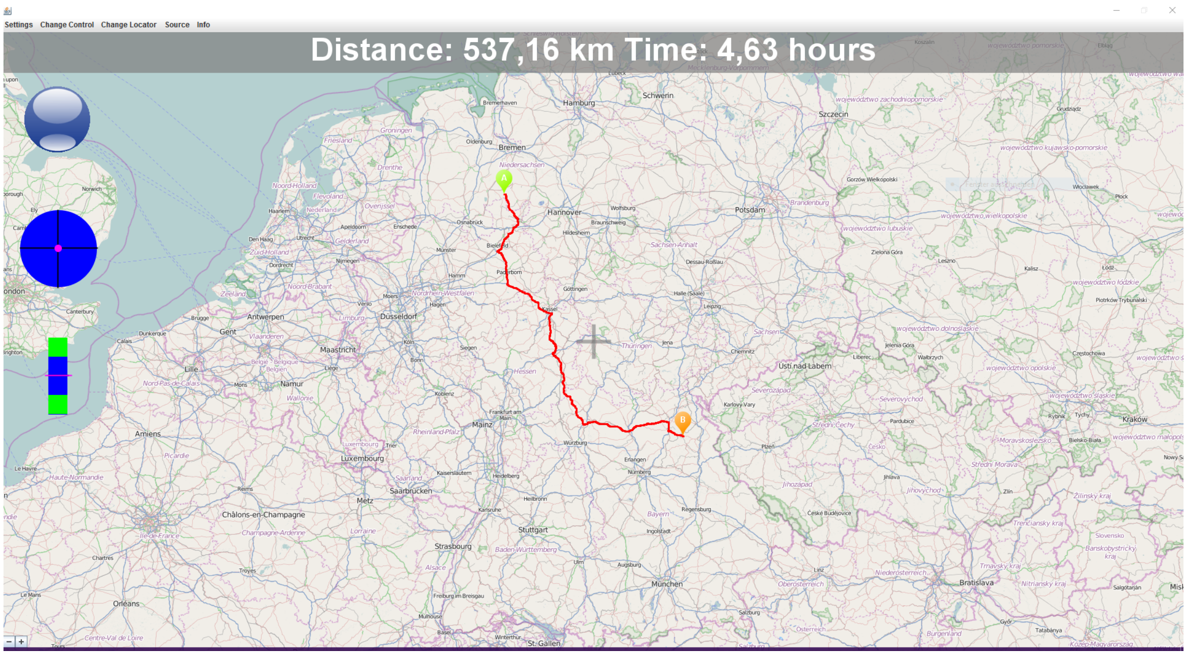

Settings Change Control Change Locator Source Info
Distance: 537,16 km Time: 4,63 hours

Figure 2.1: MapMotion showing two route points and a route.

can be done by a gesture that is identical over all five manual input methods: The hand forms a fist but the index finger is extended. If this gesture is detected, a cursor is shown on the map which represents the fingertip. It follows the finger which is moved in the X-Z plane (see figure 2.6). If the cursor is over the desired location on the map, the user can perform a tap gesture with the index finger similar to clicking the button of a mouse (see figure 2.7). This gesture sets a route point. Since many participants of the user studies extended their thumb as well while performing the tap gesture, the implementation was changed so that the gesture is also detected if the thumb is extended, too. To set another route point, the procedure has to be repeated. If a second route point is set, the route between the two route points is calculated and displayed on the map as a red path (see figure 2.1). If the gesture is performed again, the first route point remains, but the second one is set to the new location. The fourth operation is to clear all route points. This gesture is also identical across all five input methods: The hand forms a fist but the index finger and the middle finger are extended. Now, the hand is moved from the very left of the view of the Leap Motion to the very right. This gesture is called *right swipe* (see figure 2.3). A *left swipe* (which is the same gesture as the *right swipe* but from the right to the left) opens the *drawer* on the right side of the screen. This is the fifth operation. Text based input methods are shown in this drawer. The last operation is to enter or leave the view of the Leap Motion without performing any unwanted actions. The gesture for this operation is again identical across all five manual input methods. The flat hand has to be held over the Leap Motion and all fingers have to be spread as far as possible (see figure 2.2). Now the hand can be drawn out of the view of the Leap Motion without performing any action. The same gesture works for entering the view of the Leap Motion.

All operations can be performed with only one hand. Since it does not matter, which hand is used, the system is also suited for left-handed users.

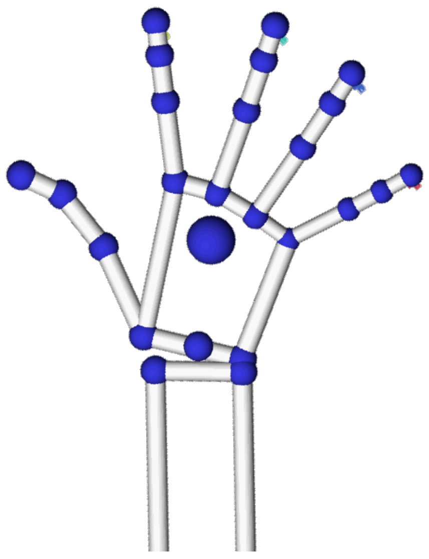


Figure 2.2: The hand is flat and opened and all fingers are spread as far as possible.

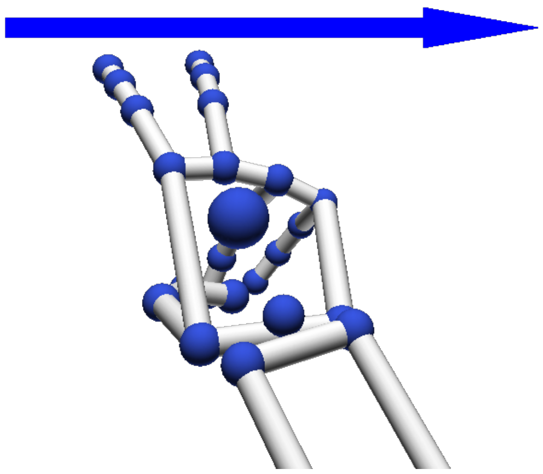


Figure 2.3: Only the index finger and the middle finger are extended. The hand moves from the left to the right.

2.2.1 The Five Manual Methods

The five manual control methods can be divided in three different groups. There are two methods for map movement:

- Position-Control
- and Angle-Control

and two methods for performing zooms:

- Distance-Zoom
- and Finger-Zoom

Each control method is combined with each zoom methods. This results in four different input methods. In addition, there is the *Touchscreen* method which works completely different.

2.2.2 Position-Control with Distance-Zoom

The first input method resembles the usage of a joystick and is called *Position-Control with Distance-Zoom*. For the neutral position, the hand rests opened above the Leap Motion. To drag the map, the hand has to be moved in the X-Z-plane. The map then moves accordingly. If the hand stays off the center of the Leap Motion, the map moves constantly. The speed of the movement can be controlled by the distance of the hand to the center of the Leap Motion. The further the hand is away from the center, the faster the map moves.

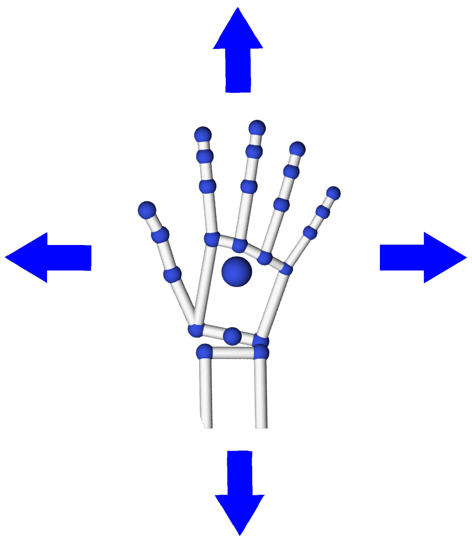


Figure 2.4: The hand moves in the X-Z plane. The map moves accordingly.

This is similar to moving an invisible joystick with the palm of the hand. To make it easier to keep the map steady, the neutral zone is enlarged. Since the system cannot give haptic feedback on the current position, like a physical joystick would do by snapping back to the neutral position, a gauge is shown on the left side of the screen (the second round gauge from the top on the left in figure 2.1). This gauge shows the position of the palm relative to the neutral position as pink bubble. The enlargement of the neutral position results in a magnetic-like behavior of the bubble of the gauge. The zoom level is controlled by the Y-position of the hand. The height of the hand is shown by another gauge (see the lower square gauge in figure 2.1). If the height of the hand is increased beyond a certain level, the maps zooms out. If it is held below a certain level, the map zooms in. The threshold, which indicates when the zooming is performed is visible on the gauge as green area. If the hand is held below or above the according level, the map zooms in the according direction at a rate of one zoom level per second.

This input method was chosen because it maps the view of the map directly to the hand movement in three-dimensional space. If the users move their hand in any direction, they get to see the part of the map in this direction. This works for the X-

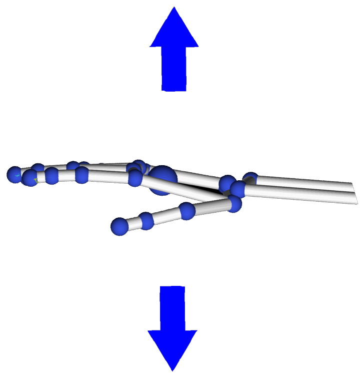


Figure 2.5: The Y-position of the hand controls the zooming of the map. If the hand is raised, the map zooms out, if it is lowered, the map zooms in.

and Z-plane as well as for zooming. If the users want to see more details of a location, they move their hand down towards the Leap Motion. This approach was considered very intuitive.

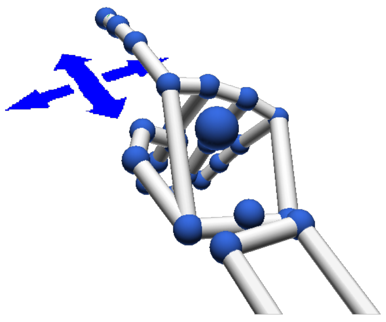


Figure 2.6: The hand forms a fist but the index finger is extended. A cursor on the screen follows the fingertip which can be moved in the X-Z plane.

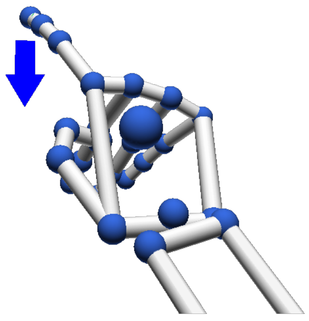


Figure 2.7: The hand forms a fist but the index finger is extended. To set a route point, the user clicks an imaginary mouse button.

2.2.3 Position-Control with Finger-Zoom

The gestures for moving the map are identical to the method *Position-Control with Distance-Zoom*. However the zoom level is controlled by finger gestures instead of the height of the hand. To zoom in, the index finger has to be bent down while the other fingers have to stay extended. To zoom out, the index finger has to be bent up. Since this can be hard for some people, it is also possible to bend down the middle finger instead of bending up the index finger. The Y-position of the hand does not influence the gestures. Therefore, the bar gauge which was used for the *Distance-Zoom* is not shown when using this method.

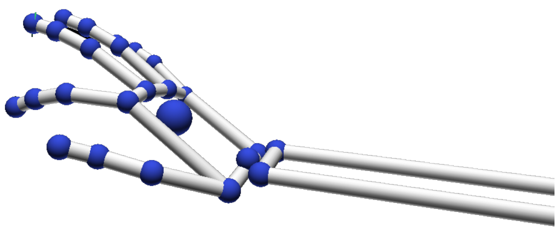


Figure 2.8: To zoom in, the index finger has to be bent down.

The *Finger-Zoom* was chosen instead of the *Distance-Zoom* because early tests with users showed that the *Distance-Zoom* was difficult to learn. Since the Leap Motion

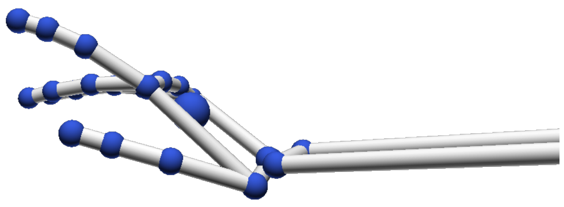


Figure 2.9: To zoom out, the index finger has to be bent up.

is capable of recognizing finger gestures and these were not used yet by this input method, the zooming was mapped to finger gestures.

2.2.4 Angle-Control with Distance-Zoom

The zooming in this method is done exactly as in *Position-Control with Distance-Zoom* by lowering or raising the hand. The gauge, which indicates the height of the hand, is visible. However the gesture for moving the map is different. The flat hand rests over the center of the Leap Motion. Instead of moving the hand in any direction as in *Position-Control with Distance-Zoom*, the users tilt their hand in the direction the map should move. The angle relative to the neutral position sets the speed of the moving. The more the hand is tilted, the faster the map moves. The extent of tilting can be seen in the bubble gauge.

Since users usually support their hand with the elbow on the table, the neutral position varies from user to user. In order to make the input method as comfortable as possible, the neutral position has to be calibrated for each user. This is done automatically when *Angle-Control with Distance-Zoom* is selected and the hand of the user enters the view of the Leap Motion for the first time. Then, a notification is shown telling the user to hold the hand in a comfortable position. MapMotion then collects the information about the tilting and calibrates the neutral position accordingly. See chapter 3 for further details. If the hand is relatively low during the calibration process, this leads to sensitive zooming. Therefore, the hand should be positioned as high as possible while still feeling comfortable.

This method was implemented as an alternative to the methods with *Position-Control*. In contrast to these methods, *Angle-Control* needs less space for the hand and can be performed with smaller movement which could be more comfortable for the users. The gestures used in this method are very similar to the control method used by *Here Maps* [leaf]. But since MapMotion shows the map in a top view, the movement of the

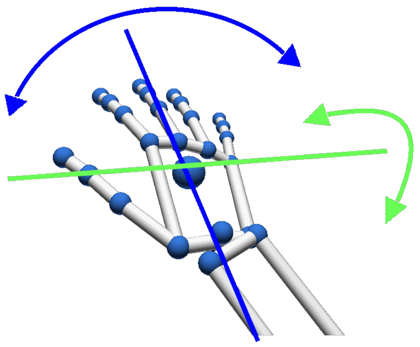


Figure 2.10: To drag the map, the hand has to be tilted.

hand results in a different movement of the map than for the control method of *Here Maps* which shows the map in an inclined view.

2.2.5 Angle-Control with Finger-Zoom

This method combines the map movement gestures from *Angle-Control with Distance-Zoom* with the zoom gestures from *Position-Control with Finger-Zoom*. The gauge indicating the height of the hand is not visible when using this method. The bubble gauge again shows the extent of tilting. This method also uses the calibration process described in section 2.2.4. However the calibration for the average height is not used for this method. Since the finger gestures are used for zooming, the height of the hand is irrelevant.

2.2.6 Touchscreen-Control

This method differs from the previously described ones. It is inspired by the way a device with a touchscreen is controlled. The gestures of controlling maps on a touchscreen are well known. Therefore this method is assumed to be the most intuitive one. To understand this method, one has to imagine an imaginary touchscreen at about 10cm above the Leap Motion. The gesture for dragging the map is the same as for setting route points. The hand has to form a fist while extending index finger (see figure 2.6). Again, a cursor is shown which follows the fingertip. If the finger is above the imaginary touchscreen, the cursor moves over the map without moving the map. If the finger is lowered below the imaginary touchscreen, the map follows the finger. This is analogue to a finger on a real touchscreen. If the finger is above the touchscreen, nothing happens, but as soon as the finger touches the touchscreen, the map follows the finger. To visualize the fingers' height in relation to the imaginary plane, the cursor is color coded. If the finger is below the plane, the cursor is blue. The higher the finger is held, the more the cursor fades to red. So the color changes from blue (finger is low) to red (finger is high). After the second user study, a visual effect was added to indicate when the finger is below the plane (see chapter 5.3.1). The zoom gestures are similar to the usual pinch-gestures on a touchscreen. To zoom in, the thumb and the index finger have to touch each other and the hand is above the imaginary touchscreen. Then, the hand is lowered below the imaginary plane and the thumb and index finger are spread. To simplify the use case of zooming in several levels, the fingers can stay spread. Then, the map zooms in at the rate of one zoom level per second. The gesture for zooming out is similar. The index finger and thumb have to be spread and the hand has to be above the imaginary touchscreen. Then, the hand is lowered below the imaginary plane and the fingers have to be brought together. As described in the zoom-in gesture, the fingers can stay in this position for continuous zooming. Since the gesture for setting route points and moving the map are identical, the height of the fingers is used to distinguish these gestures. If the finger is above the imaginary plane, route points can be set by performing the tap gesture (see figure 2.7), if the finger is below the imaginary plane, the map can be moved.

The *Touchscreen-Control* was implemented as an alternative for the *Position-Control-* and *Angle-Control* method. Since this method uses the well-known gestures and movements also used for touchscreen applications, it is considered to be easy to learn for users.

2.3 Text Input Methods

Instead of manually selecting route points on the map by pointing to the locations on the map, MapMotion also supports setting route points by entering an address. In this case, the city name has to be entered first, followed by the street name. All the input methods used in this approach have in common that they use buttons which are triggered by a tap gesture. Similar to the gesture for setting route points used for the manual input methods (see section 2.2), the index finger serves a representation of the cursor. The tap gesture triggers the button below the cursor (see section 2.2 and figure 2.7). The *T6* input method uses six buttons, all other input methods use

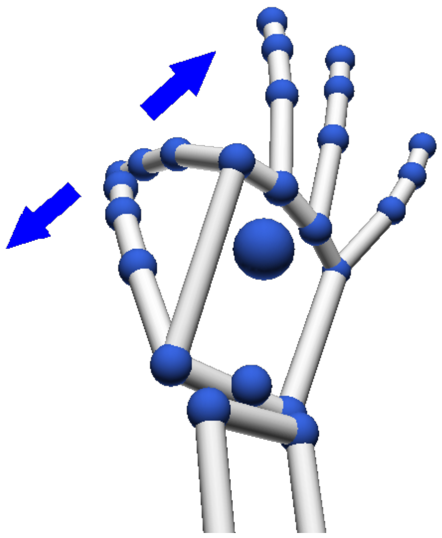


Figure 2.11: To zoom out, the thumb and the index finger have to be lowered below the imaginary plane and then spread.

four buttons. To enter the menu for these text input methods, a left-swipe gesture can be used while MapMotion shows the map. This gesture will open the *drawer* on the left side (see figure 2.13). The drawer consists of the buttons for the selected input method and two additional text labels. The first label is shown above the buttons and displays instructions (either to enter the city name or to enter the street name for the according city). The second label is located below the buttons and shows information about the current input. This information depends on the selected input method and is described in detail in the according section.

For all input methods, the user can perform a left-swipe gesture for undoing the last input. For detailed information see the section of the according input method. However all input methods have in common that the view switches back to city-input if this gesture is performed on an empty street-string. The previously entered city is maintained. While the *drawer* is shown, the user can perform a right-swipe gesture to hide the *drawer*. It is possible to combine the manual input methods with the text input methods. The user can first zoom in to the general area of the address and then use the text input method. Since the text input methods only consider cities which are visible on the map when the *drawer* is opened, this can lead to less button presses and therefore to a faster input time.

In the following chapters, the different text input methods are described.

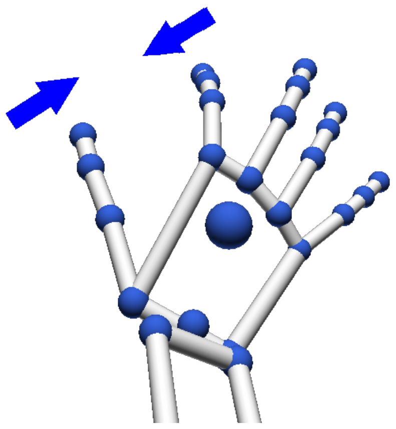


Figure 2.12: To zoom out, the thumb and the index finger have to be lowered below the imaginary plane and then brought together.

2.3.1 Primitive Input

For this input method, the characters of the city or street have to be typed one by one. All possible characters at the current position of the input are split equally distributed into four subsets. These subsets are then mapped to the four buttons. If the user now triggers one button, there are two different cases. For the first case, we assume that only one character is mapped to this button. Then this character is added to the input string and the procedure starts again for the next character. In the second case, there is more than one character mapped to this button. Then, the characters mapped to this button are again split equally into four subsets which are then mapped to the four buttons. If there are less than four characters mapped to one button and this button is triggered, the rear buttons are left empty and cannot be triggered.

If there is only one possible character during the input, this character is skipped and automatically added to the input string.

Example: *Stuttgart* and *Stutengarten* are possible options. The current input string is *Stu*. The only possible character in this situation is *t*. So the *t* is automatically added to *Stu* and the user now has the choice to enter either *t* or *e*.

If the current overall-string is a complete city name or street name but there are still other possibilities, the fourth button is not used for characters, but labeled with this city name or street name instead. If the user triggers this button, the according route point is set on the map if it is a street. If the name of a city was mapped to this button,

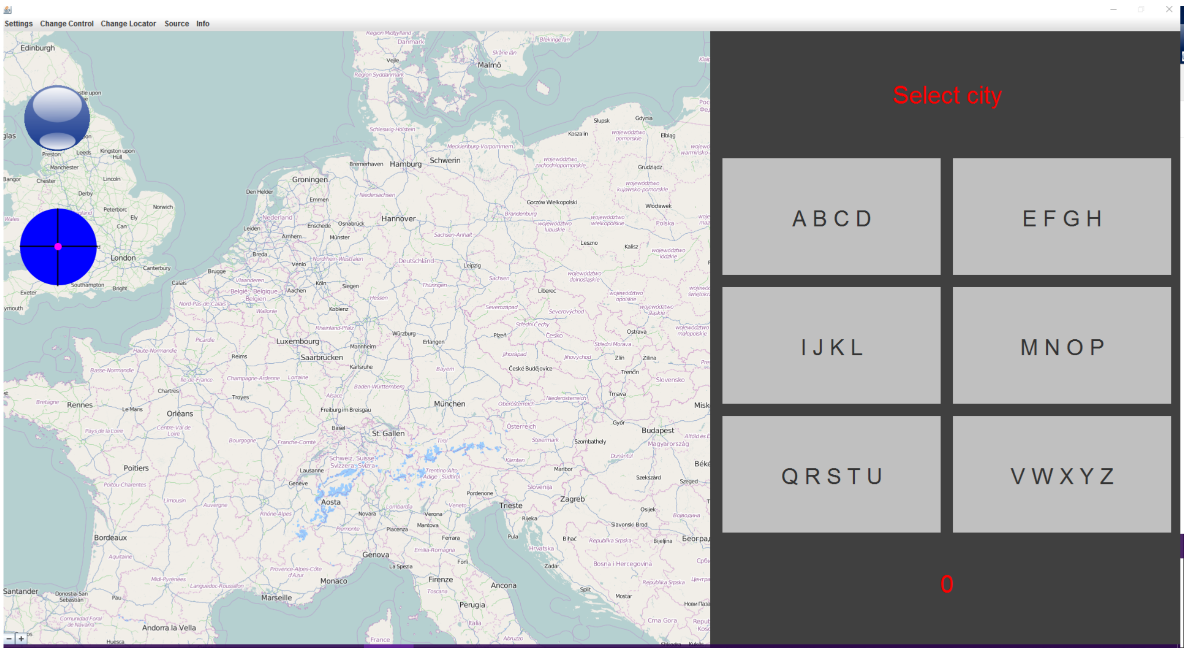


Figure 2.13: This menu allows the user to enter locations by name.

the input for the street name for this city starts. Example: The current overall-string is *Rohr*. This is a valid city name. However there is also a city called *Rohrbach*. In this case, *Rohr* is shown on the last button and the other three buttons are used to type other characters.

The back gesture leads back to the previous view, which does not necessarily entail that a character is deleted from the input-string. The characters entered so far are shown on the information label below the buttons.

This method was implemented because it gives full feedback while using only four buttons. The users can always see what they have typed. This makes it easy to recognize mistakes in the input (in contrast to *T4* (see section 2.3.3) and *T6* (see section 2.3.4)) and correct them. Since the *primitive* method only uses four buttons, the input does not have to be precise (in contrast to the *keyboard* method (see section 2.3.5)). For example, the whole upper right part of the view of the Leap Motion is mapped to one button. This helps preventing mistakes during the input by typing the neighboring button by mistake. A downside of this method is that it usually needs more than one button press per character.

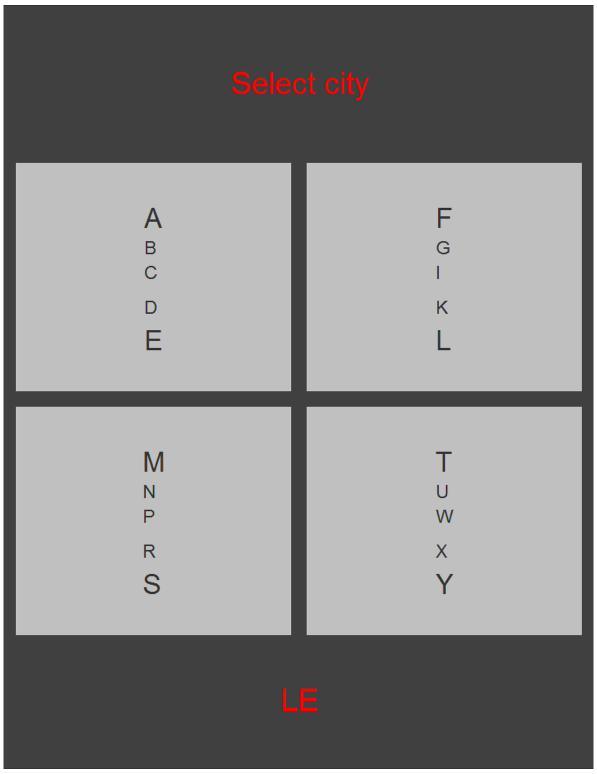


Figure 2.14: The primitive input method. The characters are typed one after the other.

2.3.2 Binary Input

This input method is based on a quaternary tree. The leaves of this tree are results (cities or streets). In the beginning, all results are sorted by name and split into four groups. These groups are then mapped to the four buttons. To visualize the content of on group, the according buttons are labeled in a special way. The first and last result of the group is always shown in big letters. Between these two locations, *hints* are shown. These are locations shown in a smaller font and should serve to let the user know which locations are listed between the first and the last location. To simplify the process of recognizing which group contains the searched location, the locations are not shown in their entirety on the buttons. Only the characters $0 - (n+1)$ of the name of the location are shown where n is the amount of characters which are already determined. If applying this rule leads to the case that the label for the last location of a button b_n equals the first location of a button b_{n+1}, the amount of shown characters is increased until these two labels differ. To make this difference clear, the additionally added characters are shown in red instead of black. Additionally, the characters which are already determined are displayed in grey, so the user can focus on the next characters. This can be seen in figure 2.15. The information label below the buttons shows the characters which are already determined.

If the user triggers a button, the results mapped to this button are again equally split into four subsets and mapped to one button. This is repeated until only one result is mapped to a button. If it is a street, the according result is shown on the map. If it is a city, the input for the street name for this city starts.

If the users performs the back gesture, they get back to the previous view: The parent node in the tree. This does not necessarily entail that a character is deleted from the determined characters.

The *binary* input was implemented because it usually needs less button presses compared to the *primitive method* (see section 2.3.1) while providing full feedback to the user and needing a low input precision as it only uses four buttons (see section 2.3.1).

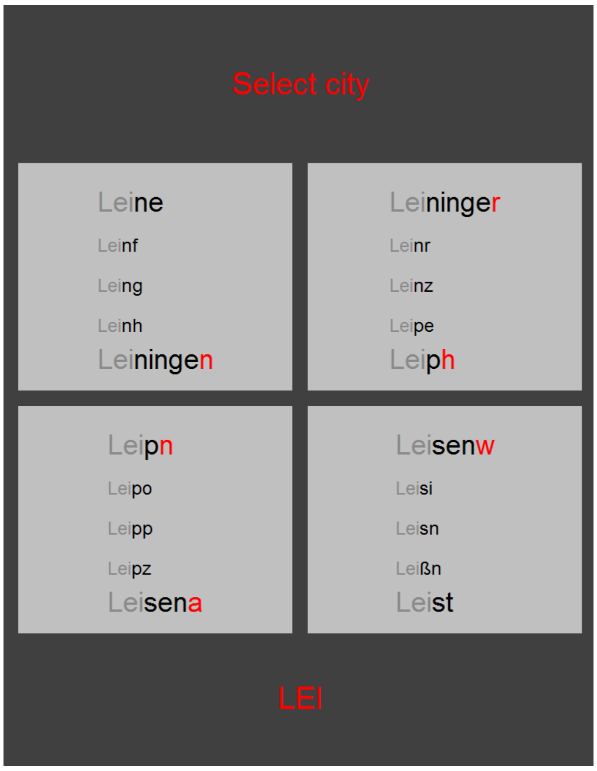


Figure 2.15: The binary input method. The locations are searched for in a quaternary tree.

2.3.3 T4 Input

The *T4* input method is similar to the T9-Input method that used to be common for old mobile phones [t9p]. However there are some differences. The first difference is that there are four instead of nine buttons. Second, the way of choosing the desired word when having finished typing is implemented differently. *T9* has a dedicated key to cycle through possible matches, sorted by decreasing frequency of use. For the *T6* method, the users have to perform the finishing gesture when they has finished typing all characters instead. The finishing gesture consists of holding the flat hand with spread fingers above the Leap Motion for two seconds (see figure 2.2). When the users have performed this gesture, the results are presented on the buttons and the user can choose the result by triggering the according button. The labels of the buttons stay constant during the input process and are labeled as follows:

- Button 1: ABCDEF
- Button 2: GHIJKL
- Button 3: MNOPQRS
- Button 4: TUVWXYZ

Example: If the user wants to type in *Stuttgart*, the sequence would be *3-4-4-4-4-2-1-3-4*.

While typing, two cases can occur. The first case is that the amount of possible results is less or equal than the number of buttons (for *T4* four buttons). In this case the remaining results are mapped to a button each. Now the user can trigger the according button to select the city or the street. This can even occur when the user has not finished typing all characters.

The second case applies when the user has finished typing all letters of the location but there are still several possible results which have names longer than the current search string. In this case, the view does not change as in the previous case. The user has to confirm that the input is finished. This can be done by performing the finishing gesture. Now, only results which have names with the same length as the current search string are considered. This strategy is chosen instead of reserving one button for completed locations like for the *primitive* method (see 2.3.1) because the labeling of the buttons does not remain constant for the *primitive* input. Therefore the users will not be confused if one button is labeled as a location. For *T4* and *T6* on the other hand, the buttons are always labeled the same. It is part of the concept of the method to know which character is mapped to which button. It might be confusing for the user if the mapping of the characters would change in certain situations. Therefore, an alternative gesture is used to tell MapMotion that the input is finished. If the user has performed the finishing gesture, there are two subcases. The first case is that the remaining results are less or equal to the amount of buttons. Then the results are mapped to one button each. But if there are more remaining results, the results are split into sets as in the binary input method (see section 2.3.2). Each set is mapped to one button. Since this is considered a rare case, no special formatting is used for labeling the sets on the buttons. All results of the set are shown on the according button.

The information label shows the number of input characters and then up to three locations (separated by commas) of the list of current possible locations - depending on the size of the list. These should facilitate the recognition of mistakes. A full list of all currently possible locations is not feasible since there can be too many possible locations, especially in the beginning.

One advantage of the *T4* input method is that users usually know the concept of T9 (all of the participants of the first and the third study knew T9) and can adapt their knowledge to the *T4* or *T6* input method. Additionally, this method only requires one button press per character plus the additional selection of the city. Like in the previous methods, only four buttons are used which makes it possible to trigger the right button even with low precision (see 2.3.1). The disadvantage of this method is that the user does not get enough feedback to easily recognize mistakes during the input.

2.3.4 T6 Input

This input method is identical to *T4*, but uses six buttons and therefore less letters per button.

- Button 1: ABCD
- Button 2: EFGH
- Button 3: IJKL
- Button 4: MNOP
- Button 5: QRSTU
- Button 6: VWXYZ

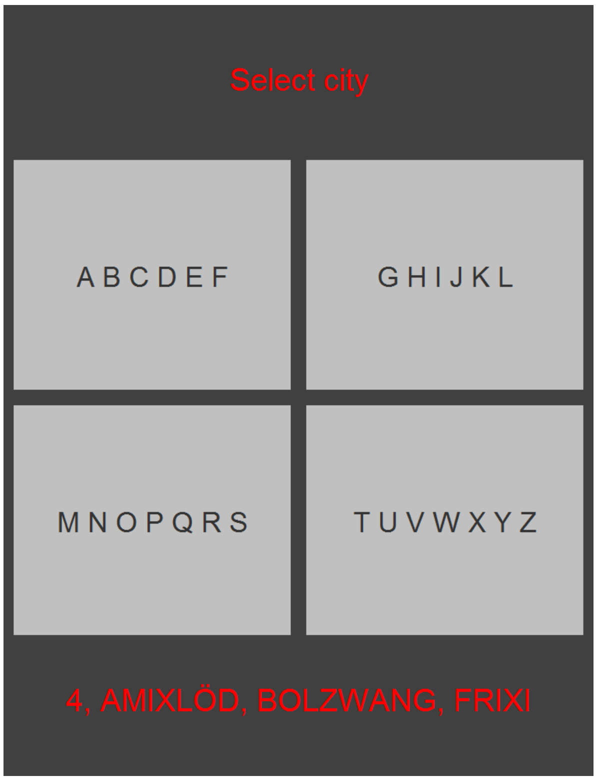


Figure 2.16: The T4 input method. This method enables the user to search for a location using a method similar to T9 on old cellphones.

The advantage of more buttons is that there are less characters mapped to one button, which leads to more distinctive overall-strings and therefore increases the chances of fewer necessary button presses than for *T4* (see section 2.3.3). In theory this should lead to a shorter input time. For a comparison between the *T4* input method and *T6* input method see section 4.3. A possible disadvantage of *T6* is that the buttons are smaller which could lead to more mistakes during the input since the precision of the input has to be higher than for the input methods using only four buttons, like the *primitive*, *binary* and *T4* input method.

2.3.5 Keyboard Input

This input method uses the concept of an onscreen keyboard. The user is shown a keyboard in the QWERTZ layout. The keys are buttons and can be triggered by performing the tap gesture. The street names and city names have to be input letter by letter. To simplify the process for the user, characters which are not possible at the

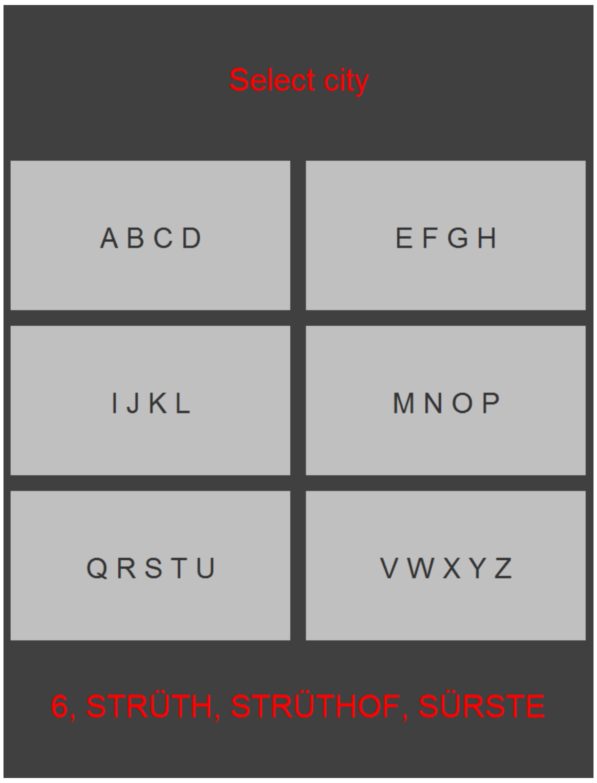


Figure 2.17: The T6 input method. This method enables the user to search for a location using a method similar to T9 on old cellphones.

current position of the input because there are no cities or streets with this substring, are disabled. L_n is the letter at position n. L_n is enabled if $\exists$ location name with letter L on position n. This makes it impossible to type invalid locations. The concept of skipping characters which are unambiguous (see section 2.3.1) is also used for this input method. The currently entered substring is shown on the information label below the keyboard. To delete a character, the user can either use the back gesture or use the backspace button on the onscreen keyboard. When the user has finished typing, the input has to be confirmed by triggering the enter key (button with checkmark icon - see figure 2.18). This button is only enabled if the current substring is a valid location.

The keyboard has 29 buttons (all letters of the alphabet plus a space key, an enter key and the backspace key). Therefore, the buttons are very small and the input has to be very precise. This disadvantage is compensated by the many advantages of the method. First, the users get full feedback on the input and can easily see if they made

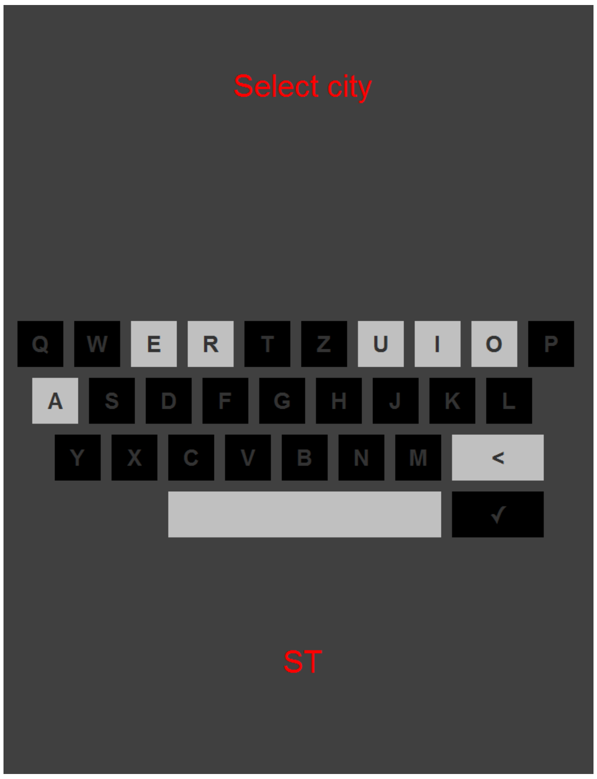


Figure 2.18: The keyboard input method. This method enables the user to search for a location using an onscreen keyboard.

mistakes. Additionally, only one button press per character is needed and the users usually know the concept of an onscreen keyboard well.

2.4 Hardware and Software Requirements

The software of the Leap Motion has some requirements for the hardware as well as for the operating system [leah]. The CPU has to be an Intel®Core™ i3 or equivalent and at least 2GB of RAM are required. The Leap Motion software works with Windows, Mac®OS or Linux but MapMotion was only tested using Windows 8.1 and Windows 10. In order to run MapMotion, an internet connection is required to download the map data and to calculate the routes. Java JRE in version 7 is needed and at least 1GB of space on the hard disk is required. This is mainly because of the size of the database (see chapter 3). If more countries than Germany have to be stored in the database, the required space is accordingly larger.

2.4.1 Suitability for the proposed use cases

The hardware requirements match a low end computer. Therefore, the required computer can be cheap and MapMotion is well suited to be used in ticket vending machines or indoor navigation systems as proposed in chapter 1. The Leap Motion is very small (13cm x 8cm x 1.3cm - see figure 3.1), so it was possible to build it into a notebook [leab]. Therefore it is also possible to build it into the systems mentioned above next to or below the screen.

3 Hardware and implementation

3.1 The Leap Motion

MapMotion uses the Leap Motion [leae] for gesture recognition. This is a device which contains two infrared cameras and three infrared LEDs. The cameras take up to 200 frames/s and can recognize objects in a view angle of 150°. The LEDs illuminate the hands above the Leap Motion and the both cameras capture an image of them. Hands are detected up to a height of about 60cm and also 60cm in each direction. The input space has the form of an upside down pyramid. The green area in figure 3.4 shows the whole interaction space. The red area shows the part of the input space that should be used for the input. While the hand remains in this red area, called `InteractionBox`, it is guaranteed that the hand is in the view of the Leap Motion (see [leaa]). If the hand leaves the `InteractionBox`, the Leap Motion might not recognize the hand reliably. The captured image data is then transferred via USB to the PC where it is analyzed by the Leap Motion Software. The software detects the position of the objects in the view. Since two cameras are used, it is also possible to detect the height of the object. The software then creates a 3D hand model out of the data, which can be accessed via an SDK. MapMotion uses this SDK to recognize the different gestures. Since daylight and some light bulbs also emit infrared light, the light conditions can influence the precision of the recognition to a certain amount. But usually the Leap Motion detects the hand very accurately.

3.2 Implementation

This chapter explains the implementation of MapMotion. MapMotion is derived from MapKin [Sch13]. As MapKit, MapMotion is implemented in Java. The architecture was changed, the text input as well and the four manual input methods were added. Additionally some components were improved. There are several dependencies in order for the system to work. First, it needs the Leap Motion library [leai] in order to use the Leap Motion. To display the map, JXMapKit is used [swi]. This is a java library

Figure 3.1: The Leap Motion.

which shows rendered tiles of OpenStreetMap data [ope]. The tiles are pre-rendered and downloaded as images from a server. Tiles are available in 15 zoom levels. OpenStreetMap provides free map data which are usually up to date since the data can be updated by users [upd]. To provide the route calculation, ToureNPlaner [tou] is used. This is a project of the University of Stuttgart which also uses OpenStreetMap for the calculation of routes. This ensures that the displayed map and the calculated route matches. ToureNPlaner allows further constraints for the calculation as well as multiple route points. Since these features are not relevant for MapMotion, they are not used. A simplified view of the architecture can be seen in figure 3.2. The different components are explained in the following sections.

3.2.1 Database

The information of cities and streets are stored in a SQLite [sql] database. Figure 3.3 shows the schema of the database. The table `Street` contains information of the streets. Each street has an id to identify the street, a name and a location consisting of latitude and longitude. Likewise, each city has an id to identify the city and a name. The attributes `minLat, maxLat, minLon` and `maxLon` specify the boundary of the city. They are calculated by the streets of this city. The maximal latitude of a street belonging to a certain city determines the value for the `maxLat` of the city. The other values are calculated analogously. These boundaries are used to know whether a city is currently visible on the map and therefore needs to be considered as possible location. The values for `lat` and `lon` are located midway between the values of `minLat` and

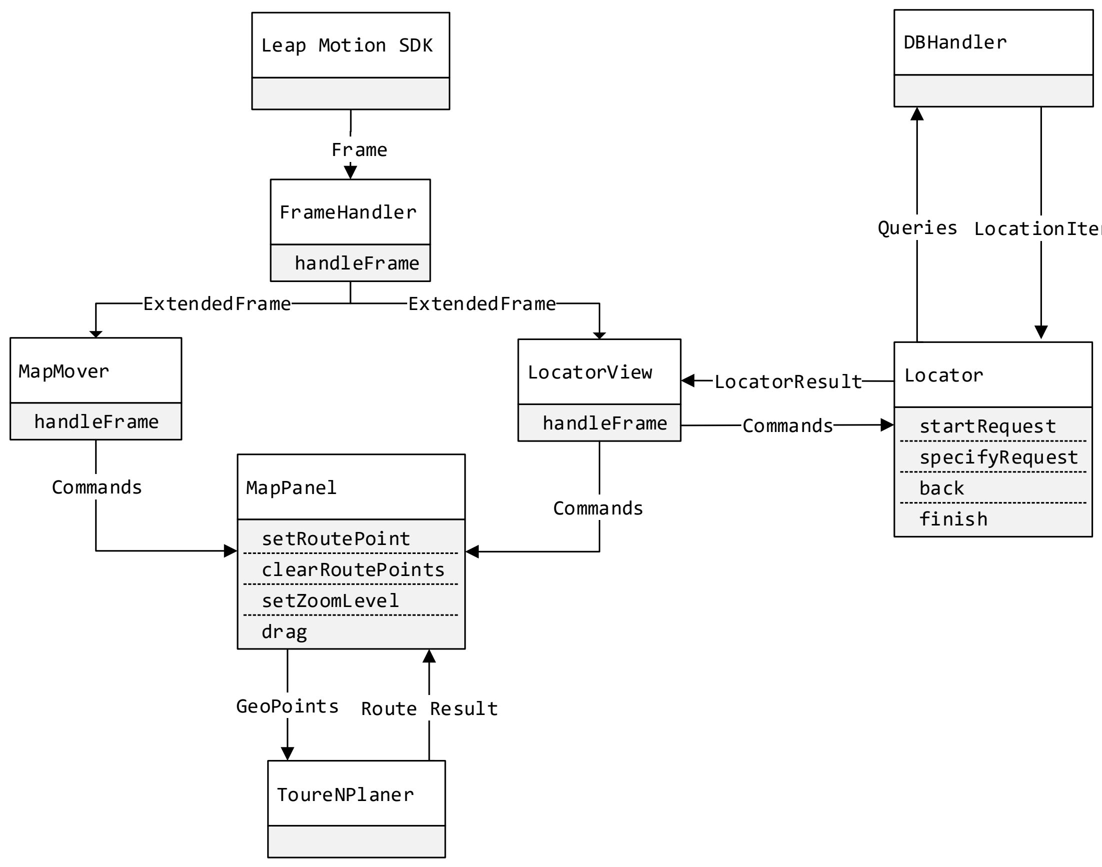


Figure 3.2: Simplified architecture of MapMotion.

maxLat or minLon and maxLon respectively. This location is not guaranteed to be within the city but the city is usually visible on the map when zooming in to this location. The attribute area is currently not used. It is intended to be used to distinguish two cities that have the same name. The area could contain the name of the cities county, for example (see chapter 8). The third table CityStreetRelation is used to store the relations between streets and city. Each entry contains an id idCity of a city and an id idStreet of a street. This means that this street with id idStreet is located in the city with id idCity. One city usually contains multiple streets and one street can be part of several cities. The communication with the database is implemented in the class DBHandler. The representation of the locations in the code are the classes City and Street which are both derived from the super class LocationItem.

The data for the names and locations of cities and streets are extracted from OpenStreetMap data. An external program parses the data and references streets to cities and writes the result to a file. This file can be parsed by MapMotion and then be imported to the database. Streets are represented by n `GeoPoints`. A `GeoPoint` contains the latitude and longitude. Since MapMotion does not support house numbers, the $\lfloor n/2 \rfloor$th `GeoPoint` is used as location of a street.

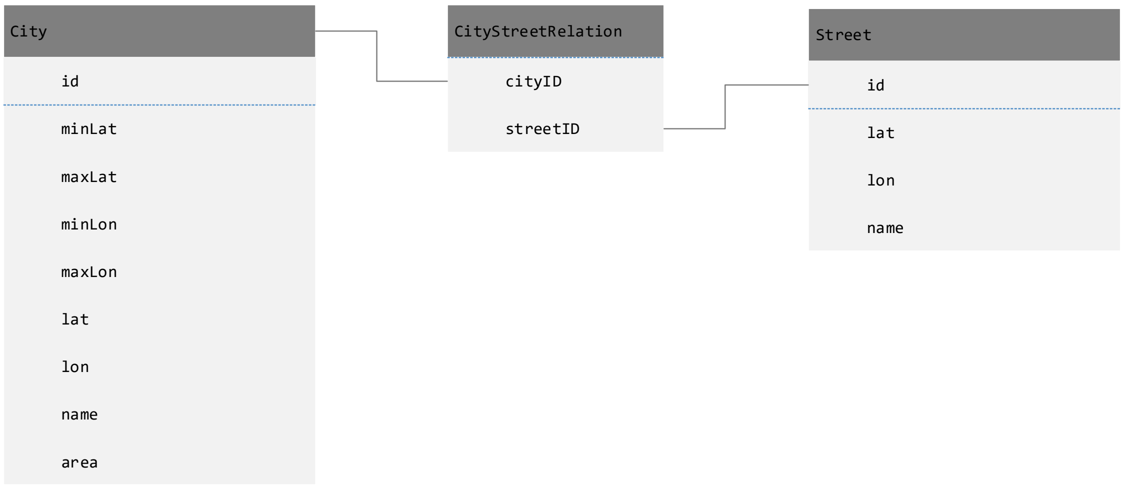


Figure 3.3: Schema of the database.

3.2.2 The Leap Motion SDK

Leap Motion provides a Software Development Kit (SDK) for Java amongst other languages. This SDK is used to implement MapMotion. The SDK provides a high level interpretation of the raw data and provides this by a `Frame`. A `Frame` contains the data at one point of time. Depending on the capabilities of the used computer, the SDK can handle over 100 `Frames` per second. The interpretation of the data results in a model of one or more hands. A visualization of this model can be seen in figure 2.6, for example. The information is built up hierarchically. The root is a list of `Hands`. A `Hand` consists of `Fingers` which can be accessed by type (*thumb, index, middle, ring, little*). This makes it possible to recognize the gestures for right-handed users as well as for left-handed users without differentiation in the code. A `Finger` has two important attributes. First, it can be determined if the finger is extended or not. Second, each finger has a position. The position is a three dimensional point referring to the fingertip and is measured in millimeters from the center of the Leap Motion. Therefore the X-position and Z-position can also be negative. In addition to this position, the SDK also provides a stabilized position of each finger. This position has a little delay but jitters significantly less. MapMotion only makes use of the stabilized position because a steady cursor is needed to aim for buttons or locations on the map. The delay is almost not noticeable and none of the participants of the user studies commented on a possible delay. Apart from the fingers, a hand also has the attribute `Palm`. Like the fingers, the palm also has a normal and a stabilized position. In addition to the position, the palm contains tilt values for all three axes. These values are used for the *Angle-Control*. Again, MapMotion only makes use of the stabilized position. A frame also contains an `InteractionBox`. This is a box which is guaranteed to be inside the view of the Leap Motion (see figure 3.4). The size of the box depends on some settings and is therefore provided in each `Frame`. This `InteractionBox` is used to normalize the positions of the finger to a range between 0 and 1. This makes it easier to map the location of the fingers to the screen. Besides the model of the hand, a frame also contains some predefined gestures. MapMotion uses the `TapGesture` for the triggering of buttons and for setting route points on the map. The recognition of this gesture can be configured in terms of the distance and the speed of the tap. After conducting the second study, the recognition had to be configured to a more sensitive setting, since it was not reliable enough. The second gesture used is the pinch gesture. This gesture provides a value between 0 and 1, which represents the distance between the thumb and another finger of the same hand. This is used for the zooming gestures of the *Touchscreen-Control*. Other possible gestures like the `CircleGesture` (drawing circles with one finger) or the `ScreenTapGesture` (moving one finger fast forward in the Z-axis) are not used in MapMotion. Although the SDK provides a `SwipeGesture`, this was not used since the recognition of the gesture was not reliable enough. Therefore an own recognition of a swiping gesture was implemented (see section 3.2.3).

3.2.3 The Framehandler

In order to avoid redundancies in the code, the data of the hand model is analyzed in one single class, the `FrameHandler`. This class receives all `Frames` provided by the SDK. The `FrameHandler` analyzes the data and detects the different gestures used by MapMotion. The `PointingGesture` (see figure 2.6), which is used for triggering buttons or setting route points on the map is detected, if the index finger is extended or the thumb and the index finger is extended. All other fingers must not be extended. The recognition of the `SwipingGesture` (see figure 2.3), which is used to delete characters for the text input methods, for opening and closing the *drawer* or for removing all set route points is split into two parts. The first part detects the gesture if only the

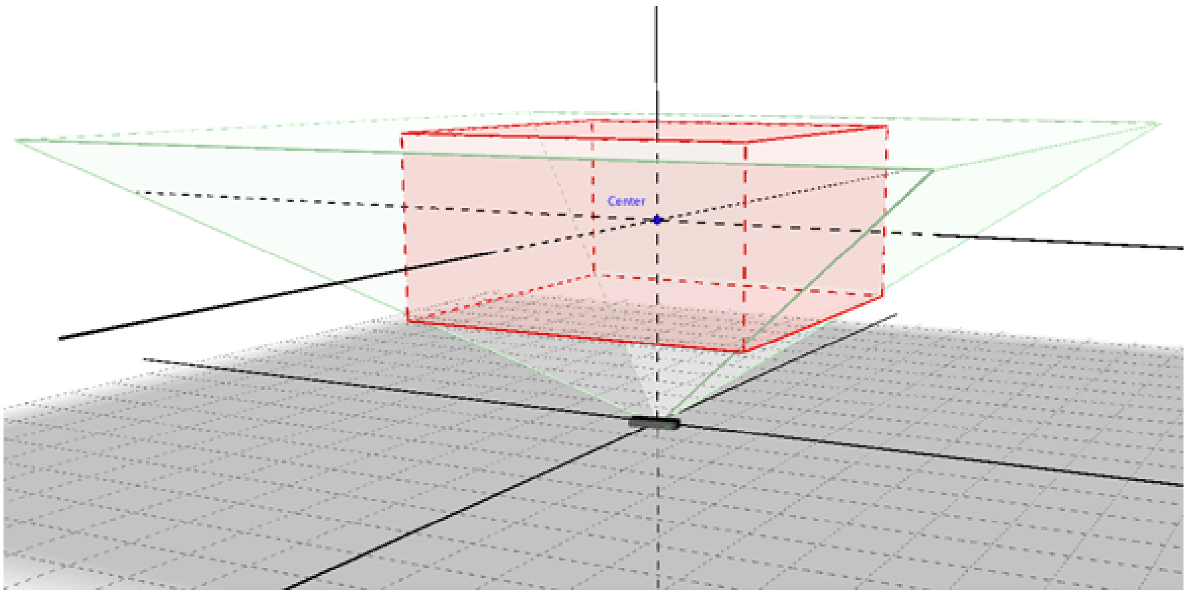


Figure 3.4: The red area is the `InteractionBox` inside the view of the Leap Motion [leaa].

index finger and the middle finger are extended. If the gesture is detected but was not detected in the previous `Frame`, the current position of the palm is stored. If the gesture is also detected in the following `Frames`, the distance between the current position of the palm and the stored position of the palm is calculated. If the distance exceeds a given threshold, a complete *Swipe* is detected. The sign of the distance defines the direction of the swipe. The third gesture to detect is the flat open hand with spread fingers (see figure 2.2). It is used to leave or enter the view of the Leap Motion without performing any actions and is recognized if all five fingers are extended and the angles between all adjacent fingers are above a given threshold. If one or more angles are below the threshold, the flat hand is recognized (see figure 2.4). This is used for moving the map in the *Position-Control* and *Angle-Control* methods. Additionally, the `FrameHandler` normalizes the position of the palm and all fingertips. The results of the gesture recognition and the normalized positions are then stored in an object of the class `ExtendedFrame`. This class is derived from the `Frame` class and can store these additional information.

3.2.4 LocatorViews and Locators

If the *drawer* is opened, the ExtendedFrame is passed on to the currently set LocatorView which is the implementation of the user interface of the according text input method. This can either be the ButtonView or the KeyboardView. The KeyboardView is used for the *keyboard* input method, the ButtonView for all other input methods. Each of these views contains a function which accepts an ExtendedFrame. The views also contain a Locator. This class contains the logic of the text input method. For the ButtonView the Locator is either the BinaryLocator, the PrimitiveLocator or the TnLocator which combines the logic of the *T4* and *T6* input methods. The KeyboardView uses only the KeyboardLocator. To simplify the process of implementing new input methods, an abstract class was introduced. All Locators must extend this class and implement several methods and functions. The first function startRequest is called when the *drawer* is opened and receives boundaries of the currently visible part of the map. In this function, the Locator queries the database for all possible cities according to the given boundaries of the map. Another function specifyRequest is called when the user triggers a button in the according GUI. Additionally the abstract class contains a function back which is called if the user performs the back gesture, and the function finished which is called if the user performs the finishing gesture. The Locator communicates its results to the according LocatorView by objects of the class LocatorResult. This contains information on the labeling of the buttons, strings which are shown above or below the buttons as information, a variable which indicates if the user is allowed to perform the finishing gesture in the current state and an object of the class LocationItem if the search is finished and a specific location was found.

3.2.5 The MapMovers

If the *drawer* is not opened, the ExtendedFrame is passed to the currently set MapMover. A MapMover is an abstract class. The classes which implement the logic of the manual input methods must extend this class. This makes it easy to add new manual input methods in the future. The classes AngleMoverFingerZoom, AngleMoverDistanceZoom, PositionMoverFingerZoom, PositionMoverDistanceZoom and TouchscreenMover extend the super class MapMover. The most important functions they have to implement are a constructor which provides an instance of the MapPanel and a function handleFrame which receives the ExtendedFrame. The data of the frame is analyzed and the according actions are performed on the map. The calibration of the neutral position for *Angle-Control* input methods is also done in this class. When the calibrations has not yet been done and a hand is recognized in a Frame, the calibration process starts.

The pitch and roll values of the palm are stored for each `Frame` until 100 values were collected. For the *Angle-Control with Distance-Zoom* the height of the palm is stored as well. The average out of these values defines the neutral position. The average height of the palm is set as the middle of the height for zooming. The minimal Y-value of the `InteractionBox` is the lowest height for zooming. To detect the gestures for zooming using the *Finger-Zoom* method, the absolute distance of the Y-value between the fingertip of the index finger and the middle finger is calculated. If the distance is above or below a given threshold, the map is zoomed.

3.2.6 The MapPanel

The `MapPanel` contains an object of the class `JXMapKit` which shows the actual map. Additionally, the `MapPanel` contains functions to move the map, zoom in and out of the map and set and reset route points. If two route points are set, the `MapPanel` calls a function of the class `RouteCalculator` to calculate the route between the two route points. The `RouteCalculator` sends a request containing the locations of the two route points to *ToureNPlaner* via the internet. The result of the calculation is received and forwarded to the `MapPanel`. After receiving the result, information about the route such as travel time and distance is shown above the map. In addition, the result contains the actual calculated route as a list of `GeoPoints`. A line of the `GeoPoints` is drawn as an overlay on the map. The result can be seen in figure 2.1.

4 First User Study

4.1 Description

In this chapter, the first user study is described. This study was conducted independently of this Master's thesis. The text input methods *Primitive, T4, T6, Binary* were compared in terms of speed. Instead of the Leap Motion, the Microsoft Kinect was used for input gestures [kinb]. The Kinect is also a 3D sensor but it can track the whole body instead of only the hand. Therefore, the gestures were not performed with a hand or fingers but with gestures using both arms. First, the Kinect was sold as accessory for the game console Xbox 360. However later, Microsoft released a SDK to use it on Windows [kina]. Although the input methods used for the Leap Motion and for the Kinect are different, the results of this study can be used to draw conclusions as to which text input method works best with the Leap Motion. The study was also performed by the author of this Master's thesis and was very similar to the second and third user study of this Master's thesis. The gestures used in this user study are similar to the gestures for the Leap Motion. The views the participants saw while performing the study were identical to the views of MapMotion, since MapMotion is a further development of the system used for this user study. To open the drawer on the right side, the user had to perform a left swipe with both arms. This was done by extending both arms on the right side of the body and then moving them to the left side of the body. For the Leap Motion, the analogue gesture is to perform a left swipe while the index finger and middle finger are extended. The system using the Kinect did not use a cursor to select the buttons like MapMotion does. Instead, the selected button was highlighted. To highlight a button, the user had to extend one arm. The Y-position of the arm controlled the row of buttons which was highlighted, whereby the left arm could be used to highlight buttons on the left column and the right arm to highlight buttons on the right column. To trigger a button, the button had to be highlighted for one second. The user could draw the hand back towards the body to avoid highlighting any button. The finishing gesture (see section 2.3.3) used for the *T4* and *T6* method, could be performed by moving the extended right arm from the left side of the body to the right side of the body. The equivalent for the Leap Motion is to hold the flat hand with spread fingers above the Leap Motion for two seconds. To execute the back action, the extended right arm had to be moved from the right of the body to the left of the body. The equivalent for the Leap Motion is to perform a left swipe with extended index finger and middle finger.

Since the gestures and the controls are related, it is expected that the best method of this study also performs well if it is used with the Leap Motion instead of the Kinect.

4.2 Procedure

First, the participants were told about possible risks of the study (which are none), about the use of the data and that the study could be stopped at any time. The participants had to sign a form to confirm that they understood. Then, a short introduction to all gestures was given and the participants had time to try all of them out. This helped the participants get used to the unknown control method of using gestures. Next, they performed searches for three cities (*Ohlenstedt*, *Schmitten* and *Watzling*) with each of the four methods. The different methods were explained beforehand and the participants had time to practice each method for about three minutes. Then, the time was taken for each search. After completing all four input methods, the participants were asked to comment on each of the input methods. They also had to name their favorite input method for searching for a known city and an unknown city. The participants could choose between:

- each of the four input methods
- zooming in to the general area of the destination, then using one of the four input methods
- searching the location manually on the map

For the unknown city, only the four text input methods could be chosen since searching for an unknown city on the map is pointless. The participants received seven Euro for taking part in the study. The study was conducting using a PC with an Intel®Core™ 2 Duo and 4GB of RAM running Windows 7. A large TV with a resolution of 1920x1080 pixels was used as screen since the participants had to have a distance of about one meter to the Kinect.

4.3 Analysis of the Study

27 participants took part in the study - 3 female, 24 male. All of them were students at the University of Stuttgart. Since one participant was not able to enter *Schmitten* using the *T4* and *Primitive* method, only 26 measurements exist for these input methods for *Schmitten*. The average input time per city can be seen in figure 4.1. It shows that the binary input method was the slowest for each of the three cities. The comments on this input method indicated clearly that the users had difficulties with the method. They described the method as *challenging, difficult and confusing*. The lack of feedback and the need to mentally recite the alphabet were also criticized. Most users said that the cognitive load of this method was the highest compared to the other three methods. This can be seen in figure 4.2. This diagram shows the average time the user needed to trigger one button. The time it takes to press one button can be divided in three parts as described by the Model Human Processor by S. K. Card et al. [CN86]. The time is the sum of the time it takes to perceive the new labels on the buttons, the time it takes to decide which button to trigger next, the time it takes to move the arm to the right position and a second of resting on the according button to trigger it (see section 4.1). The average time for one button press was calculated by dividing the average input time for each city and method by the minimal amount of button presses needed to input the according city name. Then, the average time for the three cities was calculated. An exception was made for the *T4* and *T6 method*. Both methods needed the user to perform the finishing gesture for the city *Schmitten*. The time it takes to perceive the labels and the time it takes to decide which button to press next is considered to coincide with the according times for the tap gesture. The time for performing the gesture of lowering one arm, rising the other one and waiting for one second to trigger the button takes nearly the same amount of time as raising the second arm and moving both from one side to the other to perform the finishing gesture. Therefore, for *T4* and *T6* an additional button press was added to the minimal needed button presses for searching for the city Schmitten. The resulting minimal amount of button presses needed can be seen in figure 4.3. Since two components (movement and resting) of a button press are identical for each method, a long input time per button press can either be a result of a long decision time which indicates a high cognitive load, or a higher error rate. If the users make errors during the input, they have to correct them and trigger the correct button. This results in a longer input time and more button presses. Since the minimal amount of button presses is used for the calculation, making mistakes results in more button presses and therefore in a longer time per button press.
The results suggest that the binary input has either a higher error rate than the three methods or a higher cognitive load. Since both are disadvantages, it is safe to say

that the binary input is the worst method out of these four. Figure 4.3 shows that *T6* needs less button presses than *T4* for all of the three cities. Since *T4* has more letters mapped to one button than *T6*, more results can be excluded from the list of possible locations per button press. Depending on the searched location it is therefore possible that *T6* needs less button presses to enter a location than *T4*. Figure 4.2 also shows that the input time per button for *primitive*, *T4* and *T6* are nearly the same. Despite this fact, the average input time of the *primitive* method is higher than for *T4* and *T6* for all three cities (see figure 4.1). This is because the *primitive* method needs more than one button press per character, since there is usually more than one character mapped to one button, whereas for *T4* and *T6* only one button press per letter is needed. Depending on the city, the finishing gesture is needed for *T4* and *T6* additionally.
According to the participants, the *primitive* method is easy to understand. They liked the feedback and the low cognitive load. However they criticized it for being slow since they needed to trigger more buttons than for the other methods. They also disliked that the labeling of the button changed every time they triggered a button. The comments on *T4* were nearly the same as for *T6*. The users liked both methods for being simple and fast. The only difference was that the users found aiming for the buttons harder with *T6* since the buttons were smaller. The participants remarked that the feedback was not good for both *T4* and *T6* and therefore it was difficult to detect and correct mistakes during the input. This can be seen in figure 4.6. If the users know the city, they prefer to search for it manually. However when the users do not know where the city is located and therefore cannot search for it manually, they prefer the *T4* or *T6* input method. Some of the participants could not decide if they prefer *T4* or *T6*. Their votes were counted as votes for *T6*.
Figure 4.5 shows the average input time for all three cities. This shows that *binary* is the slowest method and *T6* is the fastest for the cities used in this study.

So we can conclude that *T4* and *T6* are better than the *primitive* method since the *primitive* method is slower overall than *T4* and *T6* while the error rate and the average input time per button are similar for these three methods. Additionally, *T6* was the most preferred input method of the participants (see figure 4.6a). Because of these reasons, *T6* was considered the best input method out of the four text based input methods and was therefore used in the third user study.

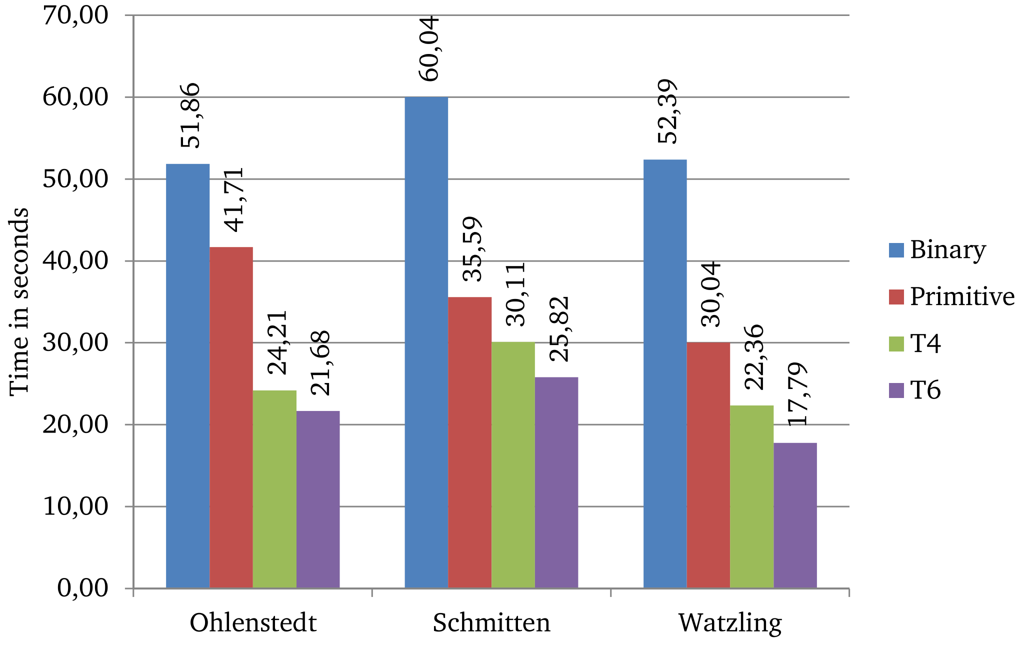


Figure 4.1: Average input time for all three cities.

4.4 Threats to Validity

The most obvious threat to validity is that the study uses a different input system. However since the flaws and advantages of the input methods can be applied to the input methods of the Leap Motion, the data can be used. Another problem is the selection of the participants. Since this study was conducted at the university, only students took part in the study. All participants were younger than 30 years. It cannot be assumed, that the results would have been the same if also older participants would have taken part. Additionally, it would have been better to test the methods with more than three cities to get more reliable results.

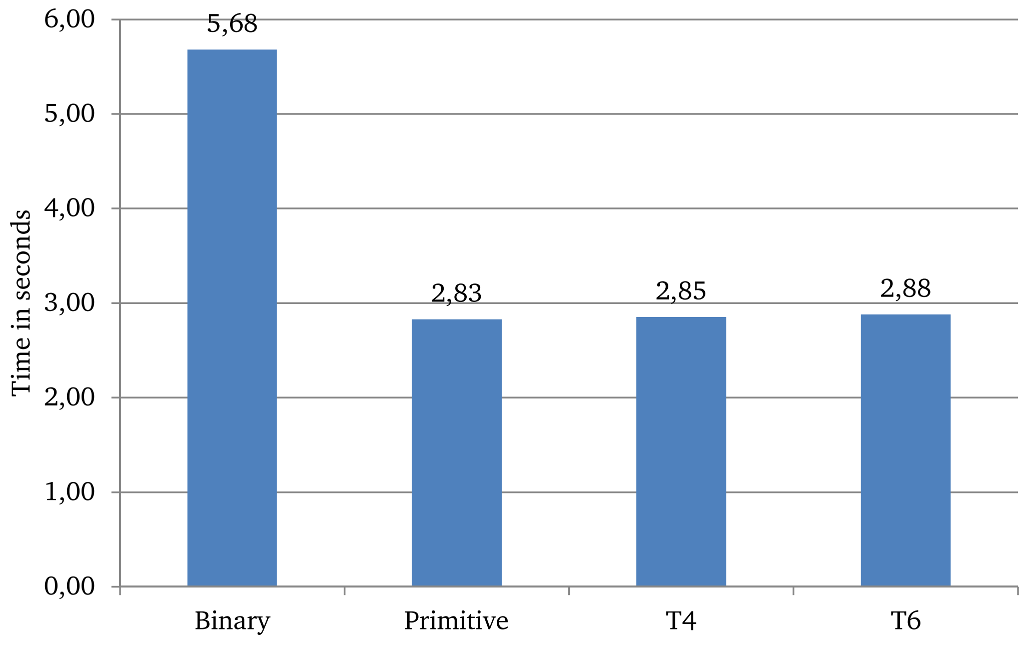


Figure 4.2: Average time per button press.

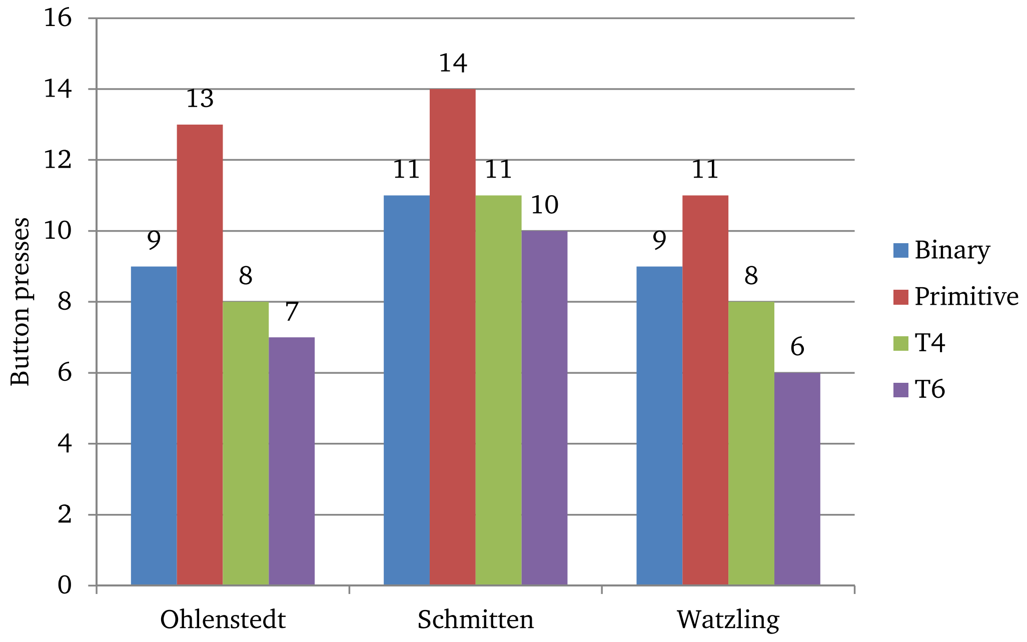


Figure 4.3: Minimal amount of button presses needed.

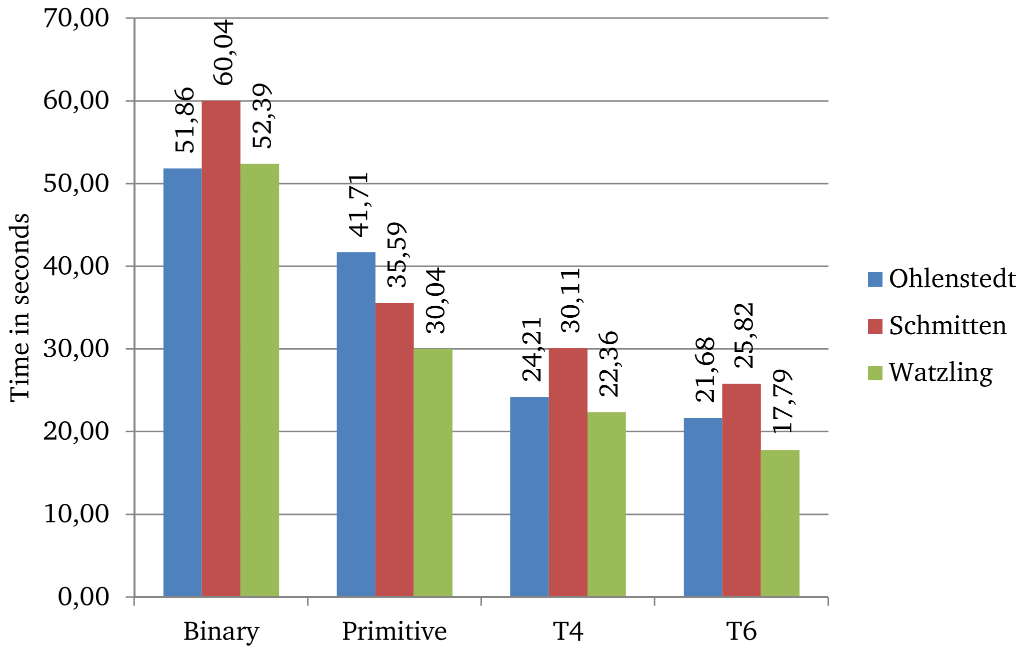


Figure 4.4: Average input time for all four methods.

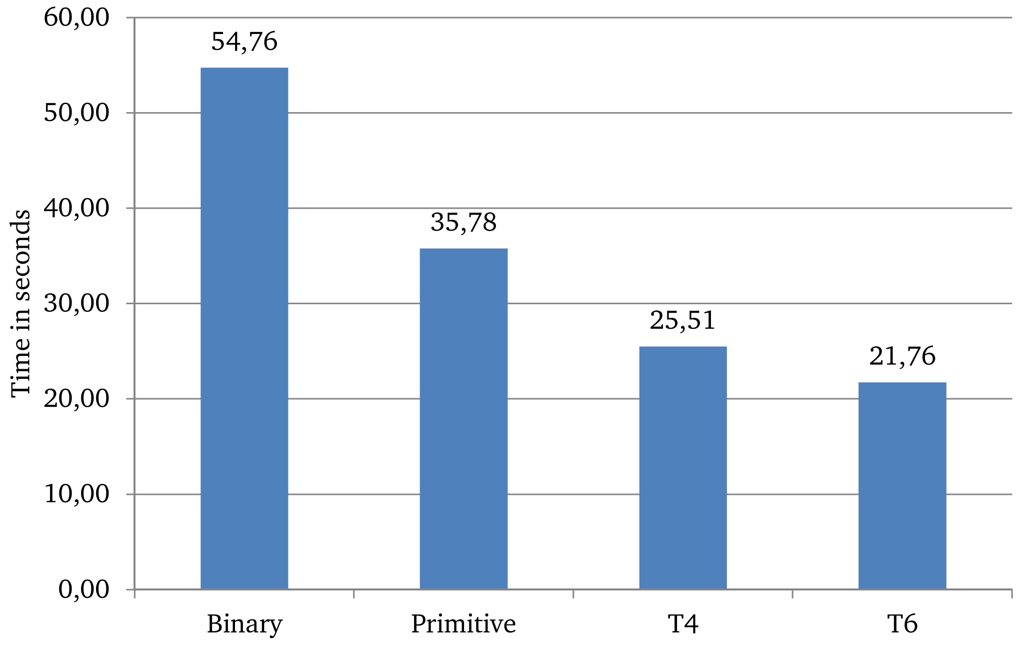
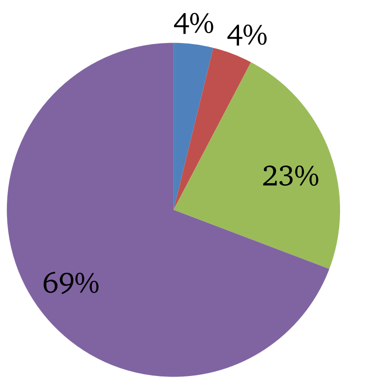
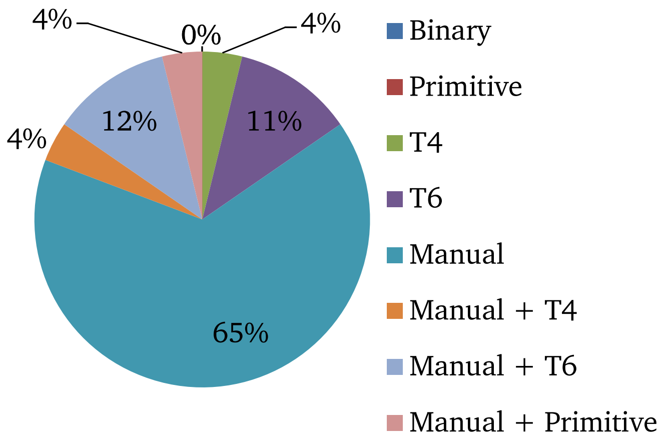


Figure 4.5: Average input time.



(a) Unknown City



(b) Known city

Figure 4.6: Preferred input methods for an unknown and a known city. The votes of participants who could not decide between *T4* and *T6* were counted as votes for *T6*.

5 Second User Study

5.1 Description

In this chapter, the second user study is described. The goal of this study is to evaluate which one of the five different manual input methods is the fastest, most intuitive and most comfortable input method for setting route points on the map. To collect the needed information, a user study was conducted where the participants had to set route points in different cities. To do this, they had to move the map, use the zoom function and set route points. Two city pairs were used for this study. The first city pair is Munich and Berlin. In both cases, a location in the city center should be selected. This pair was chosen because these cities are usually well known and have enough distance to make it necessary to zoom out to find Berlin after having zoomed in to Munich. The second pair uses locations that are in the proximity of each other: The campus Vaihingen of the University of Stuttgart and the city center of Stuttgart. This is a more common scenario for one of the possible usages of the system at ticket vending machines (see chapter 1).

5.2 Procedure

First, the participants were told about risks, usage of the data and what the purpose of the study was as in the first user study (see chapter 4). Next, the locations to search for were shown on the map to make sure the participants knew where the cities are located on the map. Next, a random permutation of the numbers one to five was created which was used as the order of the different input methods. The first input method was explained in detail to the participants. Now, the participants could practice the input method for a few minutes. Practicing with the locations used in this study was not allowed. A summary of the input method was shown on a second screen, so the participants could have a quick look at it to remind them of the gestures if they forgot them. After the practice, the map was set back to the zoom level that shows Germany at a whole. Now, the participant had to set a location point first in the city

center of Munich, then in the city center of Berlin. The time was taken beginning with the first movement of the participant, which does include the time, it takes to place the hand over the Leap Motion. When the first route point was set in the city center of Munich, the first timestamp was taken. The participant then seamlessly moved on to navigate to the city center of Berlin. The second timestamp was taken when the route point in Berlin was set. Then, the zoom level of the map was reset to the overview of Germany and the procedure was repeated for the second location pair - Campus Vaihingen of the University of Stuttgart to the city center of Stuttgart. The next input methods were tested in the same way as explained above. When the last input method was finished, the participants were asked which input method they preferred in terms of usability. Also they were asked to comment on each input method. In the end, the participants received seven Euro.
The study was conducted using a notebook with Windows 10 and an Intel®Core™ i7, 16GB of RAM and a 14 inch screen with a resolution of 1920x1080 pixels.

5.3 Analysis of the Study

The study was conducted with 15 participants. Eight of them were female, seven male. The age of the participants was between 23 and 27 and all of them were students. For each participant, approximately 40 minutes were needed. For some participants, some methods did not work as expected. Especially the pinch gestures for zooming using the *Touchscreen* method did not work for all participants. Also the *Distance-Zoom* was too hard for some participants to complete the input. Therefore some data is missing. The recognition of these gestures was improved after conducting the study. MapMotion could not be altered during the study since this would have also altered the results. The following list shows, which data is missing.

- 1 participant was not able to use the *Finger-Zoom*
- 1 participant was not able to use the *Touchscreen-Control* and the *Angle-Control with Distance-Zoom*
- 1 participant was not able to use the *Angle-Control with Distance-Zoom*.
- 3 participants were not able to enter *Berlin* using *Angle-Control with Distance-Zoom*
- 2 participants were not able to enter *Berlin* using *Position-Control with Distance-Zoom*
- 1 participant was not able to enter *Berlin* using *Touchscreen-Control*

- 2 participants were not able to enter *Campus Vaihingen* using *Angle-Control with Distance-Zoom*
- 1 participant was not able to enter *Campus Vaihingen* using *Touchscreen-Control*
- 2 participants were not able to enter *Stuttgart* using *Angle-Control with Distance-Zoom*
- 1 participant was not able to enter *Stuttgart* using *Position-Control with Distance-Zoom*

When asked about their preferred input method afterwards, none of the 15 participants preferred one of the input methods using *Distance-Zoom* (see figure 5.2). This was also confirmed by the participants' comments after the study. Most of them said they found the *Distance-Zoom* difficult since they were unable to hold their hand at a constant height while moving the map. The gauge which showed the position of the hand at the left of the screen (see figure 2.1) could not counteract this problem since the users said that they had to focus on the map while navigating and the gauge was out of their view. This led to undesired zooming and eventually to a longer input time.

An interesting finding is that the input time for Munich is faster than for Campus Vaihingen - independent of the input method. Both Munich and Campus Vaihingen are the starting points of the two routes and had to be searched for from the same starting view. The amount of zooms and the distance the map has to be moved are approximately the same. Therefore the input time was expected to be similar. A possible explanation is that Munich is a bigger city and therefore already visible in the overview of Germany. Vaihingen on the other hand, is a suburb of Stuttgart. The participants had to zoom in first to search for Vaihingen, which could lead to the higher input time. The difference could also be caused by a learning effect since the order of the locations was not randomized in this study and the Campus Vaihingen was always input before Munich. This explanation would indicate that the time for practicing the method before performing the actual study was too short.

Another finding is that the average input time for Stuttgart is considerably shorter than for Berlin although these two locations are each the end points of the route (see figure 5.1). This result was expected since Stuttgart is very close to Campus Vaihingen (about 6.5 km in a beeline) whereas Berlin is about 500 km from Munich in a beeline. This makes it necessary to zoom out and move the map after having zoomed in to Munich in order to find Berlin. Whereas there is no need to zoom at all to find Stuttgart after having zoomed in to Campus Vaihingen.

Although the average input time for the *Distance-Zoom* is not the longest (see figure 5.1), the *Distance-Zoom* was not considered to be the best approach for zooming since the participants did not like it. The *Touchscreen* method was not considered the best input method either since it is the slowest method for Berlin and the second slowest method for Campus Vaihingen and Munich (see figure 5.1). Only three participants preferred it over the other methods (see figure 5.2). Judging by their comments, this is probably because they found the pinch gesture for zooming difficult. Many participants had to try out which gesture zooms in and which one zooms out. It was found not to be intuitive enough, although the gestures are identical to these used on touchscreens of smartphones.

Position-Control with Finger-Zoom was considered the best method for several reasons. First, out of the remaining two methods, *Angle-Control with Finger-Zoom* is slower than *Position-Control with Finger-Zoom* for all cities. In addition, this method has the disadvantage of needing calibration before working comfortable. Furthermore, *Position-Control with Finger-Zoom* is fastest on average for all cities (see figure 5.3) and most often considered the preferred input method overall. Therefore, *Position-Control with Finger-Zoom* was chosen to be used for the third user study.

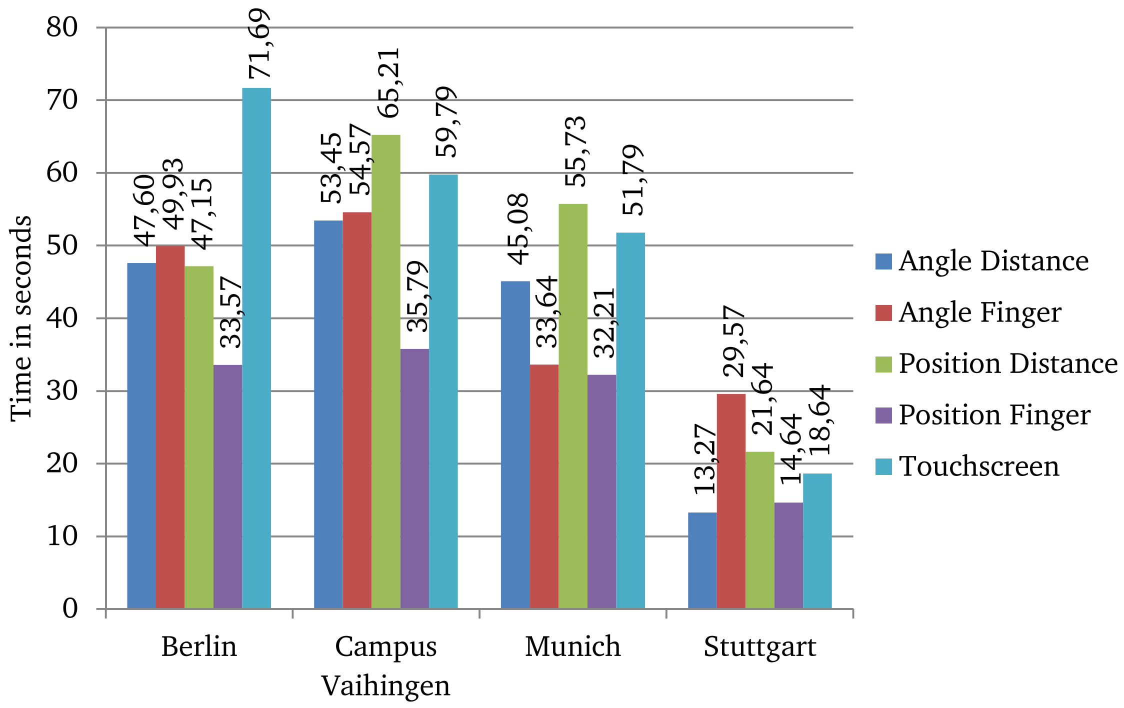


Figure 5.1: Average input time for the different manual input methods by city.

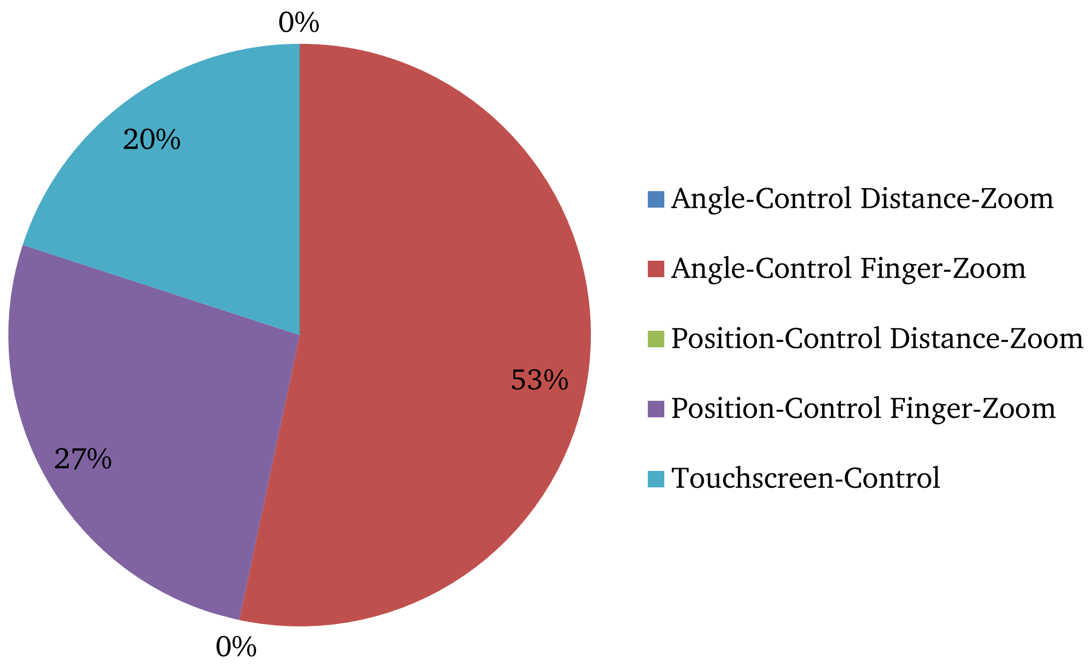


Figure 5.2: Preferred input method for the manual input methods.

5.3.1 Comments and Improvements

In this section, the comments of the participants are summarized and it is explained how MapMotion was improved based on the given comments. Some users stated that it was tiresome to hold the arm in an unnatural position. This is an issue which cannot be improved. However for the possible use case as explained in chapter 1, the user uses the system only for a very short time in contrast to a duration of about 40 minutes during this study. A majority of the users said that zooming using the *Distance-Zoom* was more difficult than zooming using the *Finger-Zoom*. This is one reason for choosing *Position-Input with Finger-Zoom* as best manual input method. It was stated that the color coding for the indication of the height of the finger did not help to know when the finger was above or below the imaginary plane. Therefore, a visual effect was added to MapMotion which shows a circle with increasing radius if the finger is moved from above the imaginary plane below the plane. One participant said the she/he expected the *Position-Control* to work inverted. She/he moved the hand often to the right instead of to the left. But since the other 14 participants did not have this problem, nothing was changed. Some users complained that it was difficult to make use of the height indicator while using *Distance-Zoom* since the gauge was

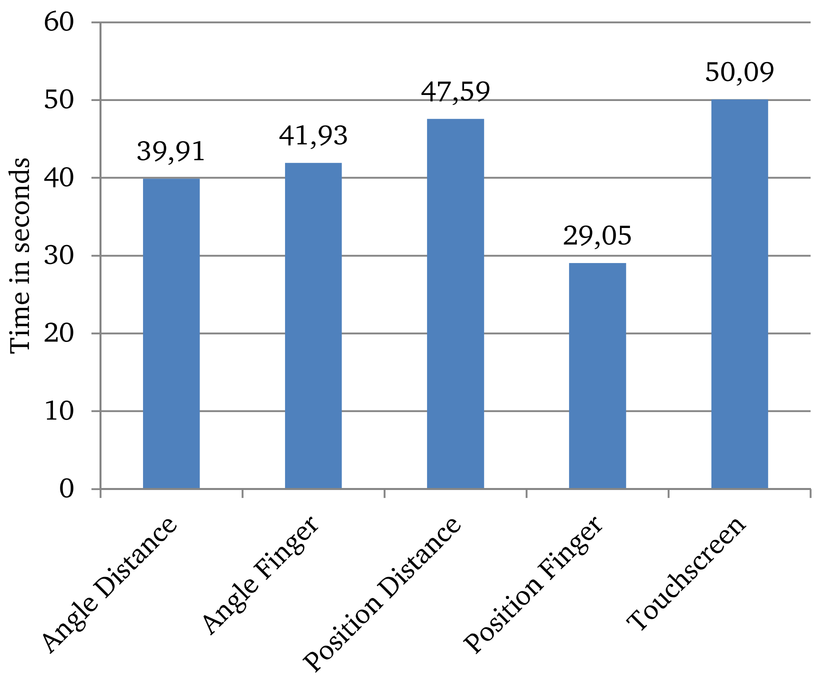


Figure 5.3: Average input time for the manual input methods.

on the left side of the screen but they had to focus on the center of the map. This is another reason for not considering a method with *Distance-Zoom* the best method of this study. During this study, there was no grey cross in the center of the map as shown in figure 2.1. It was difficult for the users to keep the overview while zooming because they did not know exactly where the center of the map was. Therefore their estimated center shifted after performing a zoom gesture and they had to take a little time to regain orientation. As a consequence, the grey cross was added to the map, as can be seen in figure 2.1. Most users had difficulties to set route points using the tap gesture. They often needed several tries before they succeeded. This problem was fixed after the second user study by increasing the sensitivity of the recognition. Especially users with small hands had problems performing the pinch gestures for the *Touchscreen* method. This was fixed by reducing the needed distance between thumb and index finger to recognize the fingers as spread. Eventually, sounds were added for all swipes and taps. This improvement serves as a feedback that a gesture was recognized. According to the participants of the third study, these sounds were helpful.

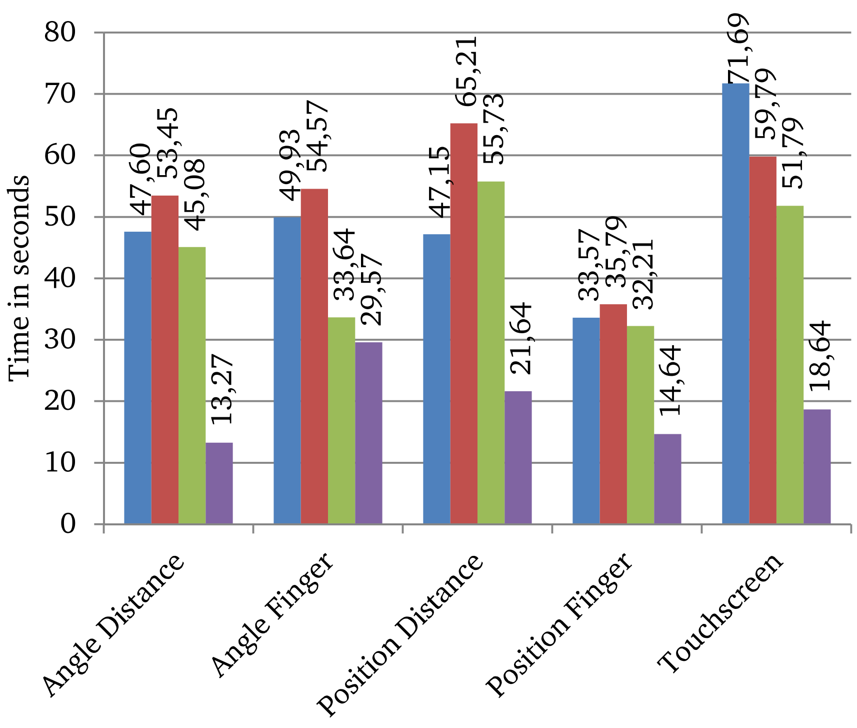


Figure 5.4: Average input time for the different manual input methods by method.

5.4 Threats to Validity

There are some possible influences which could have altered the results of the study. As in the first study, only students took part in the study and again, all of them were younger than 30 years. This leads to the same threat as in the first study: It cannot be assumed that the results would have been the same if also older users would have participated. Another possible influence is that the study was conducted at different locations. Therefore, the light conditions were not the same for all participants, which could have influenced the precision of the recognition of the Leap Motion. Additionally, the speed of the internet was not the same at the different locations. This led to a different loading time for the tiles and therefore to a different waiting time after zooming in or out. This problem could only be solved by caching the map tiles locally.

6 Third User Study

6.1 Description

In this chapter, the procedure of the third user study is described. This user study was conducted to compare the text input methods to the manual input methods. The best method (*Position-Control with Finger-Zoom*) of the first user study is used for the manual input method and the best method (*T6*) from the second study is used for the text input method. As a third input method, the *Keyboard* method is used in this user study. This text input method was not part of the first user study since the Kinect was not considered to be precise enough. However since the precision of the Leap Motion allows the use of an onscreen keyboard, this input method was added to the third user study.

Different scenarios where chosen to find out which input method is suited for which situation. The first scenario is to calculate a route between two close locations, which is typical for the use case of buying a subway ticket at a ticket vending machine. For this scenario, the starting point was Pfaffenwaldring in Vaihingen which is one of the main roads on the campus of University of Stuttgart in Vaihingen. The destination was Königstraße in Stuttgart which is located near the main station of Stuttgart. To calculate the route between these locations, it is not necessary to zoom out once the first location was set when using the manual input method. Both locations can be seen on the screen simultaneously in the second highest zoom level. Therefore it was assumed that this scenario could be performed quickly using the manual input method. For the text input methods, the distance between the locations is irrelevant.

The second scenario is a route between Sonnenstraße in Munich and Friedrichstraße in Berlin. These cities are approximately 500 km apart in a beeline. The streets are located in the center of the cities respectively. It is assumed that the streets can be easily found once the participants have zoomed in to the center of the cities. This also favors the manual input method but since zooming out is required, it is assumed that this scenario takes more time than the first one for the manual input method. Another reason for choosing these cities is that participants have to use the finishing gesture for each, *Munich* and *Berlin* when using the *T6* text input method.

For the third scenario, two small and relatively unknown cities were chosen. The starting point is Marktstraße in Beutelsbach and the destination is Schützenstraße in Visselhövede. Since these locations are small and relatively unknown, it is assumed that they are hard to find on the map using the manual input method. Consequently, in this case the text input methods should lead to better results. Without knowing the cities, it is nearly impossible for the participants to find them on a map of Germany. Therefore the participants were given some hints as to where the cities are. This reflects to use case of knowing only the approximate location while buying a train ticket, e.g. because the user has never been to that location before. Section 6.2 describes how the approximate location was described to the participants.

6.2 Procedure

As in the previous studies, the participants where told about the risks and the procedure of the study. The task of the participants was to calculate three different routes with three different input methods each. The order of the input methods as well as the order of the routes were chosen randomly to take the learning process into account. For each input method, the first step was to explain the method to the user. After the explanation, the participant could practice the method for about three minutes. Practicing with locations which were part of the study was not allowed. Next, a random scenario was chosen and the participants had to calculate the route for this scenario. At the start of each scenario, the map was reset to show the whole of Germany on the screen. The time was taken each time the participants set a route point. For the two text input methods, the gesture for opening the drawer was included in the taken time. This procedure was repeated for all three input methods.
For the manual input method, the participants were shown the locations on the map before the time was actually taken to make sure each participant had the same knowledge about the locations. Since the third scenario is about searching for unknown locations on the map, Beutelsbach and Visselhövede were not shown to the participants. Instead they were given hints to find these locations. The locations were explained as follows:

- *Pfaffenwaldring, Vaihingen*: Zoom in to Stuttgart. Vaihingen is in the south-west of Stuttgart and Pfaffenwaldring is in the very north of Vaihingen.
- *Königstraße, Stuttgart*: Zoom in to Stuttgart. The main station can be easily seen in the center of Stuttgart. The street which leads to the main station is Königstraße.

- *Sonnenstraße, Munich*: Zoom in to Munich. The historic center is labeled in the city center of Munich. There is a ring around that historic center and Sonnenstraße is the left part of that ring.
- *Friedrichstraße, Berlin*: Zoom in to Berlin. The park Tiergarten can be easily found in the city center. Friedrichstraße is the third of the bigger streets to the right of Tiergarten.
- *Markstraße, Beutelsbach*: Beutelsbach is near Weinstadt which is east of Stuttgart. Stuttgart is shown on the map.
- *Schützenstraße, Visselhövede*: Visselhövede is in the north of Hannover near Truppenübungsplätze Bergen. Visselhövede is in the north-west of the Truppenübungsplätze. Hannover is shown on the map.

The study was conducted using the same notebook as described in section 5.2.

6.3 Analysis of the Study

The study was performed with 15 participants. Among the participants, there were 8 women and 7 men between 22 and 27 years. All of them were students. However MapMotion had problems recognizing the finishing gesture for the *T6* input method or the participants were not able to correct mistakes during the input because they missed further feedback. Therefore only 13 participants were able to complete *Stuttgart, Königstraße* using *T6*. Figure 6.3 shows the average input time for all of the locations. The first noticeable finding is that the manual input was clearly slower for the third scenario with unknown cities (route from Beutelsbach, Marktstraße to Visselhövede, Schützenstraße). This is presumably because the participants had to search for both cities on the map. This, the assumption that the text input methods would perform better for this scenario than the manual input was confirmed. The manual input method is the slowest for all locations except for Stuttgart, Königstraße. This can be explained by the fact that Königstraße is the destination of the route and it is very close (about 6.5 km in a beeline) from the starting point of the route. Therefore, the participants did not need to zoom and could set the second route point quickly after setting the first one. For the starting point of the route, *Vaihingen, Pfaffenwaldring*, however, the manual input method was 15 seconds slower than the keyboard input method on average. The conclusion for this scenario is that a text input method should be used to find the first route point and a manual input method should be used afterwards to find a nearby location. For this use case, MapMotion would have to be modified (see chapter 8).

The second scenario (route from *Munich, Sonnenstraße* to *Berlin, Friedrichstraße*) is interesting as well. For the starting point at *Sonnenstraße*, the three input methods needed nearly the same time. For the end point however, the manual input is about 17 seconds slower on average than the keyboard input method. The fact that the user first needs to zoom out of *Munich* if using the manual input method is a possible explanation. For the text input methods however, the input of the destination can start immediately after having searched for the starting point.

To compare the two text input methods, the average time to trigger one button has to be compared as in the first user study (see chapter 4). This can be done by dividing the average input time per location by the minimal button presses needed for this location. For the *keyboard* method, this can be done easily. However for the *T6* method, the finishing gestures have to be handled differently for the calculation. First, the waiting time of two seconds is subtracted from the average time for the locations that need the finishing gesture. Additionally, the time it takes to change to the flat hand gesture and back to the pointing gesture, as well as the time it takes to perceive and decide have to be considered (see Human Processor Model of S. K. Card et al [CN86]). Since no related data was collected during the study, an assumption had to be made. The time for the decision and perception is considered identically for performing the finishing gesture as well as for triggering a button. The time for the movement of the hand to another button can be compared to changing the hand from the pointing gesture to the flat hand. Therefore it can be assumed that the time needed to trigger one button is about the same as the time needed to perform the finishing gesture without the waiting time. Consequently an additional button press was added to the cities that need the finishing gesture. The minimal number of needed button presses can be seen in figure 6.2. The results of this comparison is shown in figure 6.1. It shows that on average, *T6* is faster than the *keyboard* method. This is possibly caused by the fact that the users' gestures need to be more precise for the smaller buttons used in the *keyboard* method. Therefore, the aiming takes more time than with *T6*. However despite the faster input time per button, the *keyboard* method needs less button presses for four of the used locations than *T6*. Additionally, the keyboard does not need the finishing gesture. Therefore, the *keyboard* method is faster than *T6* in the overall average of the input methods (see figure 6.5). Additionally, the *keyboard* method is also preferred by users for unknown cities as well as for known cities (see figure 6.4). According to the comments, this is because the users missed feedback for the *T6* method as already mentioned in the analysis of the first user study (see section 4.3). Still, the *keyboard* method has flaws of usability, too. Even though it was explained to the participants that characters are added to the input if they are non-ambiguous, the users were taken by surprise and made mistakes during the input, especially for *München (Munich)*. After typing the *c*, the *h* was automatically added but most participants did enter the *h* anyway, which resulted in *Münchh*. Most of the participants that made this mistake only noticed the mistake because the *e* was disabled after the second *h* (see chapter 8 on how this problem could be solved). Another huge advantage of the *keyboard* method is that it is well known by the users since onscreen keyboards are broadly used, for example on smartphones and tablets. For the *T6* method however, the users needed an explanation. Most of the participants thought that the input method works like *multi-tap*. Multi-tap was used before the *T9* input method [t9p] on mobile devices. Several characters are mapped to one button and the user needs to press the button several times to enter one character. For example, the button *7* on phones contains the characters *PQRS*. To enter *R*, the button has to be pressed three times. Since the *keyboard* input method is on average faster than *T6*, is preferred by the users and does not need an explanation, it is considered superior to the *T6* input method. It is also considered to be superior to the manual input with the exception for near locations.

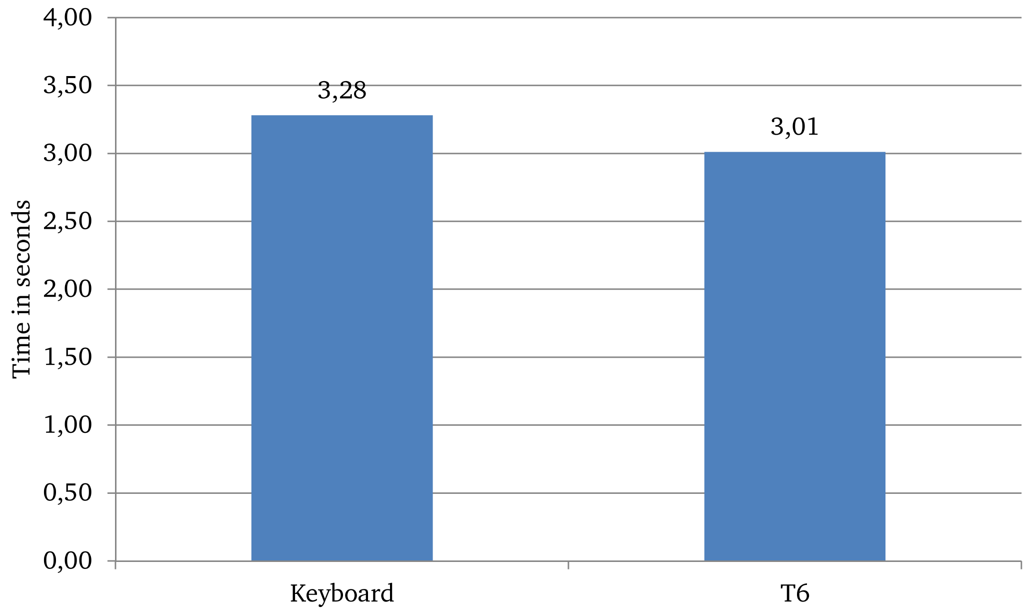


Figure 6.1: Average time per button press.

6.4 Threats to Validity

There are some influences that could have altered the results of this study. There is another threat in addition to the threats of the second user study (see section 5.4). Some of the participants of the third study had already participated in the second

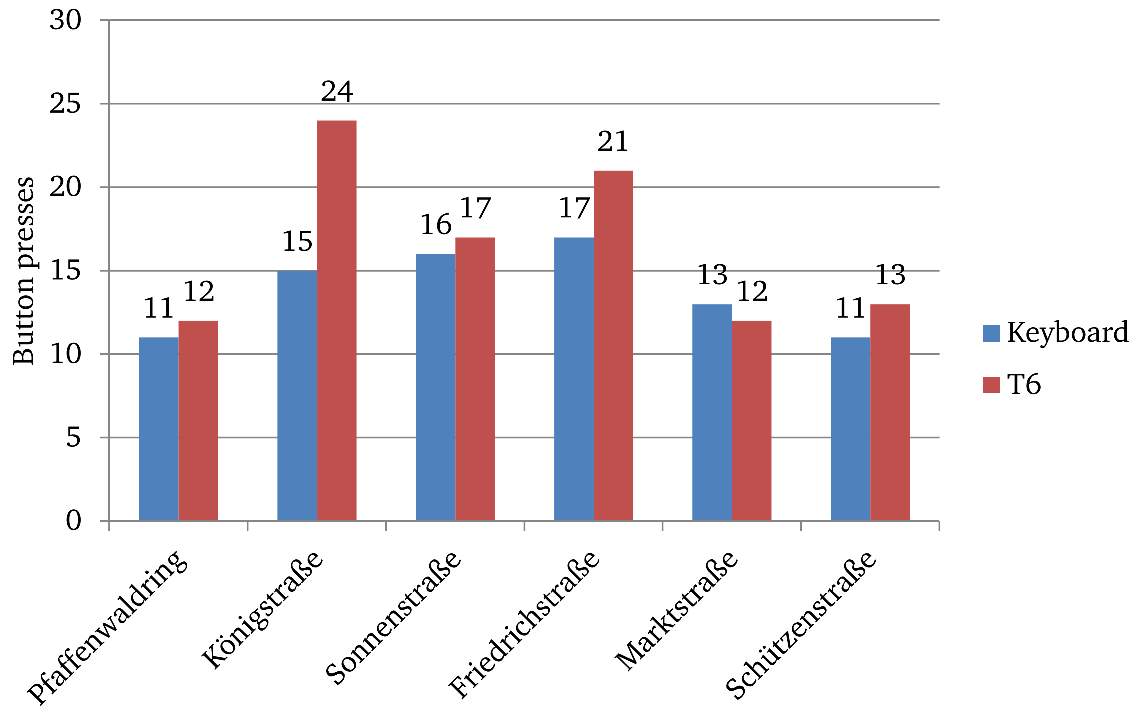


Figure 6.2: Minimal button presses needed.

user study. Therefore those participants might have had more practice than the others. However since the user studies were several weeks apart, the input methods were different and all participants had time to practice the methods for some minutes, this is considered not to be severe. Also it was possible that some participants knew the location of *Beutelsbach* or *Visselhövede* in advance. In this case, they would have had an advantage for searching for them on the map. However when asked, none of the participants knew these cities.

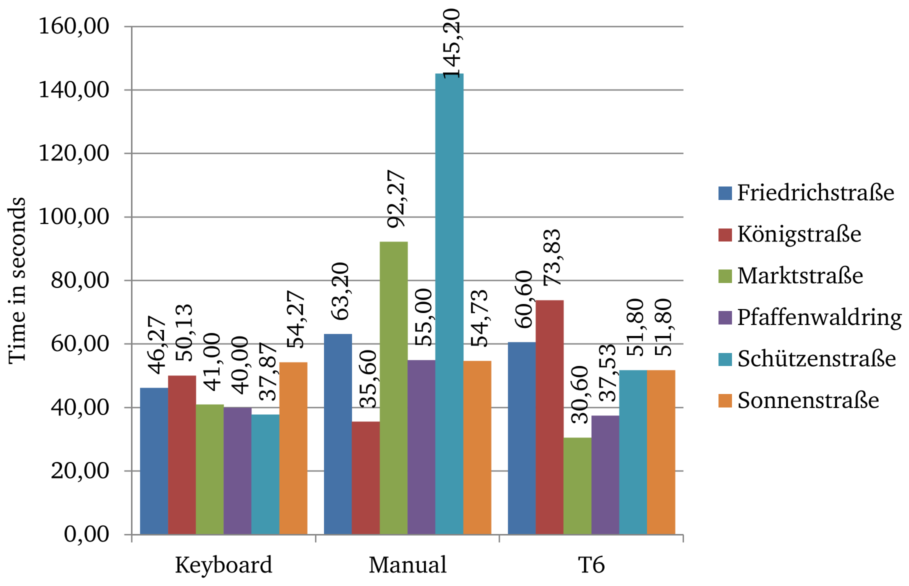
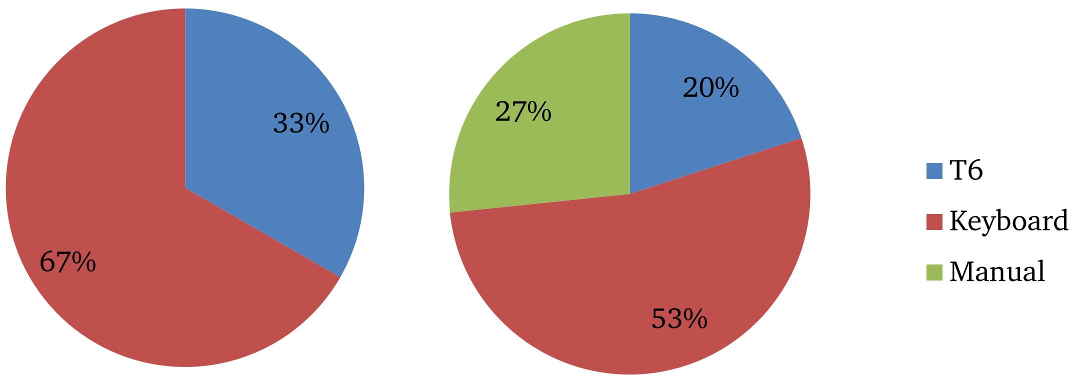


Figure 6.3: Average input time by method.



(a) Unknown location. **(b)** Known location.

Figure 6.4: Preferred input methods for an unknown and a known location.

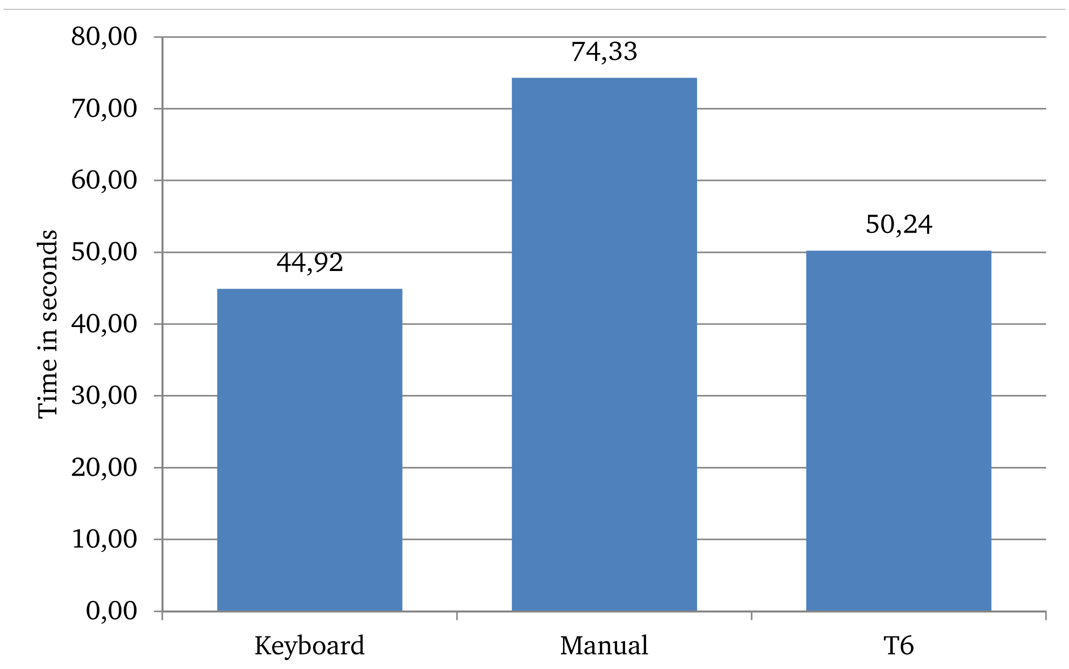


Figure 6.5: Average input time per method for all locations.

7 Summary

This chapter sums up the results of this thesis.

MapMotion, the system that was introduced in this thesis, is intended to calculate routes on a map. It is controlled by solely using hand and finger gestures which are detected by the 3D sensor Leap Motion. The advantage of gesture input is that they work touchless in contrast to touchscreens, which is more hygienic. To calculate a route, users can either set route points directly on the map or search for addresses using text input. For each of the both possibilities, different input methods were implemented and compared by conducting three user studies. The first study compared the different text input methods. This user study was conducted with the Kinect instead of the Leap Motion. The results show that the input method *T6* which is similar to *T9* [t9p] used on cell phones was superior to the others. In the second study, the manual input methods were compared to each other. The results of the study showed clearly that the height of the hand should not be used as input parameter since it is difficult for users to keep it in a constant height. A input method which uses similar movements as actuating a joystick and which uses finger gestures for zooming was the best input method of this study. In the last study, two text input methods were compared to one manual input method in different scenarios to identify which method works best for each scenario. The two best methods of the previous study as well as a new text input method, an onscreen keyboard, were used during this study. The results show that using the onscreen keyboard, which is controlled with hand and finger gestures, is the fastest of the text input methods. It should be preferred of the other methods. Only for scenarios where the second route point is close to the first one, it is faster to select the second route point manually on the map after having found the first one using text input. MapMotion can be used to buy train tickets at ticket vending machines for selecting the desired starting point and the destination either on the map or by searching for the station name. If MapMotion is changed to not display a map showing streets, but the floor plan of a building, it can also be used as an indoor navigation systems at malls or office buildings.

8 Future Work

This chapter proposes possibilities to improve the system further and provides suggestions for future research.

The way identically named cities are handled could be improved. Currently, the user has no way of distinguishing two identically named cities. Therefore, cities of the same name are currently enumerated. While performing the three user studies, only one city with the searched name was stored in the database. This prevented confounding the participants unnecessarily. To solve this problem, information should be added that enables the user to distinguish and correctly identify cities. This could be for example the county. A column in the database layout is already reserved for this purpose (see section 3.2.1). Additionally, house numbers could be introduced to the streets. This would improve the usability for the user and *MapMotion* could serve as a fully-fledged navigation system.

The map itself could also be improved. Currently, only fixed zoom levels are possible. While loading, grey placeholder tiles are displayed. This can lead to a loss of the overview after zooming. Therefore, a map renderer which supports continuous zooming could vastly improve the user experience. To make sure that the displayed route and the underlying map match, this renderer should also use OpenStreetMap data for rendering.

Taking into account the results of the third user study (see section 6.3), some changes have to be made to the text input methods. Currently, the map does not zoom in to the set location when a text input method is used. This is because the text input methods only consider locations which are currently visible on the map. If the map would zoom in to the first location, the users would have to zoom out if they wanted to search for a second location further away. Considering the two use cases, the users should be able to choose between zooming in to a found location or not when they trigger the last button on the text input methods. This could be done by using one or two fingers to trigger the final button. To visualize this, a button could change its color if it is the final button for a city.

Further user studies could compare the keyboard input with a *T9* input method. Since the third user study showed that the precision of the Leap Motion is sufficiently precise to trigger the buttons on the onscreen keyboard reliably, the number of buttons of the *T6* method could be increased. This could prevent the need for the finishing gesture in more cases and eventually lead to a shorter input time while still needing less precision than the *keyboard* input method.

The advantage of autocompletion for the *keyboard* input method could be analyzed. Doubé and Beh conducted a study to compare the advantage of autocompletions for websites between older and younger users [DB12]. A result of this study was that older users with less experience looked on the keyboard until they finished typing. Therefore, they ignored all proposals of the autocompletion. We have to consider that this behavior also applies to gesture control. Since the users are not used to this form of input, they focus on the onscreen keyboard, ignoring that the input was autocompleted. A study could be conducted to compare the speed of the input using the autocompletion to the same input method without using the autocompletion. Additionally, a visualization of the autocompletion could be introduced by adding an animation directly on the keyboard. This could help the user to recognize the autocompletion and eventually reduce the amount of mistakes made during the input.

In the comments of the third user study, the users stated that they liked the feedback of the *keyboard* method. The keyboard gives full feedback about the input in contrast to *T6*. The disadvantage is however, that the keys are smaller and the precision of the input needs to be higher. To counteract this problem, the keyboard could be improved. In the current implementation, keys which cannot be pressed at a certain position of the input are disabled to make invalid inputs impossible. If only a few keys are enabled, the size of these keys could be increased to also use some space of disabled keys. It can be assumed that the increased buttons will not confuse the users because the keys are still located at their normal position but since the size is increased, we assume that the aiming is easier. Another user study could be conducted to confirm this assumption.

Bibliography

[CHM15] A. Chan, T. Halevi, N. Memon. Leap Motion Controller for Authentication via Hand Geometry and Gestures. In T. Tryfonas, I. Askoxylakis, editors, *Human Aspects of Information Security, Privacy, and Trust*, volume 9190 of *Lecture Notes in Computer Science*, pp. 13–22. Springer International Publishing, 2015. doi:10.1007/978-3-319-20376-8_2. URL http://dx.doi.org/10.1007/978-3-319-20376-8_2.

[CN86] T. P. Card, S.K; Moran, A. Newell. The Model Human Processor: An Engineering Model of Human Performance. 1986.

[DB12] W. Doubé, J. Beh. Typing over Autocomplete: Cognitive Load in Website Use by Older Adults. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, OzCHI '12, pp. 97–106. ACM, New York, NY, USA, 2012. doi:10.1145/2414536.2414553. URL http://doi.acm.org/10.1145/2414536.2414553.

[ges] Gesture input BMW. URL http://www.bmw.com/com/en/newvehicles/7series/sedan/2015/showroom/innovative_functionality.html.

[kina] Kinect for Windows. URL http://blogs.msdn.com/b/kinectforwindows/archive/2012/01/09/kinect-for-windows-commercial-program-announced.aspx.

[kinb] Kinect release. URL https://news.microsoft.com/2010/06/13/kinect-for-xbox-360-is-official-name-of-microsofts-controller-free-game-device/.

[leaa] Coordinate system of the Leap Motion. URL https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Coordinate_Mapping.html.

[leab] HP ENVY17 Leap Motion SE. URL https://www.leapmotion.com/news/world-s-first-computer-embedded-with-leap-motion-technology-to-hit-shelves-this-fall.

[leac] Leap Map. URL http://jaxzin.github.io/leap-map/.

[lead] Leap Motion AirInput. URL https://apps.leapmotion.com/apps/airinput-trial/windows.

[leae] Leap Motion Hardware. URL http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/.

[leaf] Leap Motion Here Maps. URL https://web.archive.org/web/20131220160004/http://developer.nokia.com/Blogs/Code/2013/05/16/here-leap-motion/.

[leag] Leap Motion PRSNTA. URL https://apps.leapmotion.com/apps/prsnta--2/osx.

[leah] Leap Motion Requirements. URL https://www.leapmotion.com/setup.

[leai] Leap Motion SDK Download. URL https://developer.leapmotion.com/.

[ope] OpenStreetMap. URL https://www.openstreetmap.org.

[Sch13] M. Scholz. Gestensteuerung von Routenplanern mittels Kinect, 2013.

[sql] SQLite Repository. URL https://bitbucket.org/xerial/sqlite-jdbc/downloads.

[swi] SwingX Download. URL https://java.net/downloads/swingx/releases/1.6.2/.

[t9p] T9 Patent. URL http://worldwide.espacenet.com/publicationDetails/biblio?locale=en_EP&CC=EP&NR=1256871.

[tou] TourenPlaner. URL http://tourenplaner.informatik.uni-stuttgart.de/.

[upd] Update OpenStreetMap. URL http://wiki.openstreetmap.org/wiki/Getting_Involved.

All links were last followed on October 27, 2015.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature