University of Pardubice

Faculty of Electrical Engineering and Informatics

Hand analysis in gesture-based interfaces : design of an accurate real-time algorithm

MSc Pavel Jetensky

Dissertation

2016

Study program: Electrical Engineering and Informatics
Branch of study: 2612V070 Information, Communication and Control Technologies

Supervisor:

Karamazov Simeon, prof. Ing. Dr.
Department of Information Technologies
Faculty of Electrical Engineering and Informatics
University of Pardubice

Co-supervisors:

Mgr. Jan Vaněk, Ph.D., Ing. Bruno Ježek, Ph.D.
Deparment of Informatics and Quantitative methods
Faculty of Informatics and Management
University of Hradec Králové

Declaration of author

Pardubice, 8th December 2016

Pavel Jetensky

Acknowledgment

I am using this opportunity to thank professor Simeon Karamazov for a research-friendly environment that enabled me to finish this work. I want also to thank to my co-supervisors, Mr. Jan Vanek and Bruno Jezek for their guidance as they helped me to stay focused on the scientific benefits of my work. I also thank Mr. Vojtech Muller for his help on developing the prototype of a multi-touch surface called TouchTable. My gratitude goes also to Jaroslav Marek and Josef Rak, co-authors and friends that helped with their deep math skills so that research dealing with linear regression of finger knuckles could be finished. User experiments with TouchTable could not be done without my testers whom I would like to thank mainly for their patience and for the courage to undergo stress, related to operating an unknown interface: Jan Hridel, Otakar Kovarik, Vojtech Muller, Zbynek Kopecky, Dana Jablonska, Monika Bendova, Ilona Kucerova and Martin Zabka. My thanks goes as well to Paul Hooper for his kind help with an English review of this thesis. Finally, my gratitude and love flows to my wife and my kids, for their patience and support during my studies and during the time I was working on this dissertation.

Abstract

Depth sensor technology has undergone a large quality increase in recent years. Devices like Microsoft Kinect proved to provide reliable 3D data about the surrounding environment. This led to an emergence of new user interfaces based on a tracking of user's hands. In spite of advantages in natural interfaces, where user is free of any input device, a keyboard and mouse is still a major paradigm for human computer interaction in 2015. The main reason for this stems from the high intricacy of the hand feature detection and gesture classification tasks to properly interpret gestures of human hands.

This work presents a detection algorithm for accurate calculation of fingertip coordinates and other hand properties, based on a hand image captured from a depth sensor. It properly analyzes both straight and bent fingers and is rotation invariant. The algorithm can be used for a development of user interfaces based on a human gesture analysis, in a same way as it was used in our TouchTable interface. TouchTable is a multi-touch, multimodal user interface developed as a part of the dissertation. The user of this interface can relax his hands on a working surface, which prevents fatigue from longer usage. This fatigue is known from traditional touch screens and in-air gesture based user interfaces. Advantages of the proposed algorithm over other methods of fingertip detection are described. The algorithm is evaluated using DepthTip, a new public dataset that can be used to evaluate accuracy of fingertip detection algorithms.

Keywords

3D, auditory memory, curvature, blob extraction, convolution matrix, data retrieval, data storage, depth sensor, depth camera, digital image processing, ergonomics, finger, fingertip, gesture, hand, HCI, Human computer interaction, human hand, information processing, Kinect, MoCap, multi-touch interface, linear regression, TouchTable, template matching, user interface, visual memory, data retrieval, data storage.

Název práce

Analýza ruky v uživatelských rozhraních s použitím gest : návrh přesného algoritmu s výkonem v reálném čase

Anotace

Hloubkové sensory prodělaly v posledních letech velký kvalitativní vzestup. Zařízení jako je Microsoft Kinect prokázaly svou schopnost poskytovat spolehlivá 3D data o okolním prostoru. To vedlo ke vzniku nových uživatelských rozhraní, založených na snímání lidských rukou. Vědci nyní zkoumají rozdílná paradigmata uživatelské interakce, nicméně ideální řešení dosud nebylo nalezeno, neboť na uživatele jsou stále kladena mnohá omezení. I přes velké pokroky v oboru přirozených uživatelských rozhraní, kde je uživatel osvobozen od práce s jakýmkoliv pomocným vstupním zařízením, klávesnice a myš je v roce 2015 stále hlavním paradigmatem interakce s počítačem. Hlavním důvodem je vysoká komplexnost úloh analýzy lidské ruky a interpretace gest. Tato práce prezentuje algoritmus detekce ruky pro přesný výpočet souřadnic špiček prstů a dalších parametrů ruky na základě snímku ruky získaného z hloubkové kamery. Algoritmus správně analyzuje jak rovné, tak ohnuté prsty a je rotačně invariantní. Algoritmus lze použít pro vývoj uživatelského rozhraní založeného na gestech rukou, jako je TouchTable. TouchTable je multi-modální uživatelské rozhraní vyvinuté v rámci této dizertace. Práce prezentuje výhody navrženého algoritmu oproti dosud publikovaným metodám. Přesnost algoritmu je vyhodnocena pomocí DepthTipu, nové veřejné datové sady, která slouží k vyhodnocování algoritmů počítajících souřadnice prstů.

Klíčová slova

3D, sluchová paměť, zakřivení, ektrakce blobu, konvoluční matice, získávání dat, ukládání dat, hloubkový senzor, hloubková kamera, digitální zpracování obrazu, ergonomie, prst, špička prstu, gesto, ruka, zpracování informací, HCI, Interakce člověk-počítač, Kinect, snímání pohybu, multidotykové rozhraní, lineární regrese, TouchTable, shoda se šablonou, uživatelské rozhraní, vizuální paměť, získávání dat, ukládání dat.

Contents

Chapter 1

Introduction

The remarkable growth of human-computer interaction (HCI) has been an ongoing process during the last 30 years. In HCI, researchers are trying to find the best possible user interface where users could interact with a computer in an ideal way. There are multiple criteria that are used to evaluate the quality of such an interaction; productivity, ergonomics, ease of use, and learning curve being some of them. Before starting to work on this thesis, I originally worked for 15 years as a software engineer. I spent a huge amount of time in front of the computer screen. During periods of large work load I even developed health problems with my wrists. So I share, and well understand, the goals of the HCI discipline. For me personally, it is crucial to use the computer in a healthy, productive way. How this interaction should look, is exactly the question that HCI is answering.

Traditionally, sending commands to computers was possible thanks to input hardware devices that were operated by movements of user hands and fingers. A computer keyboard was first used more than 70 years ago, followed by a mouse and later also a touch screen. Though productivity of working with a mouse and keyboard is high, ergonomics is not ideal. Long-term usage of these devices requires thousands of tiny muscle movements, causing repetitive small strains on hand tissues. It can graduate in time to various health problems like Repetitive Strain Injury (RSI) or Carpal Tunnel Syndrome (CTS). This happens because the hand is forced to perform such movements for which it is not well suited. That is why researchers are trying to design such a user interface where users could work with their hands and perform gestures to control the computer. However, the health benefit of such a user interface, where there is no input hardware, is not the only motivation. Parkale suggests [21] that hand gestures have the advantage of being easy to use, natural and intuitive. Researchers sometimes combine a gesture control with speech commands [28] to increase productivity.

In spite of advantages in natural interfaces where a user is free of any input device, the keyboard and mouse is still a major paradigm for human computer interaction in 2015. The main reason for this, stems from a high complexity of hand feature detection and gesture classification steps, which are needed to properly interpret gestures. This is not only caused by very diverse physical conditions during the scene digitization phase, but also by difficulties during the gesture classification phase. The human hand is an awkward shape formed by non-straight curves. It is thus hard to detect in a digital image and it is also challenging to model it in some simple, mathematical way. In different positions, parts of the hand can be occluded by other parts, and thus not visible to a sensor; for example if the user has his thumb hidden under his palm.

This thesis aims to shift human knowledge in the feature detection phase by finding a suitable algorithm to detect hand features from 3D input, with the focus on fingertips.

Chapter 2

Existing user interfaces using human hands

This chapter introduces a reader to theory behind the design of user interfaces that are using human hands as a main modality of input. By studying the existing interfaces, the thesis research goal will be furmulated in further text. The theory related to user interface design is defined by a scientific discipline called Human-computer interaction (abbreviated as HCI). This discipline studies the process of communication between a man (a user) and a computer. HCI researchers are developing new paradigms of user interfaces, experiment with new available hardware and create prototypes of user interfaces. Further, they measure their productivity and ergonomics in user experiments.

2.1 HCI discipline

The goal of HCI is to find an optimal user interface for communication with the computer. This search is performed under given constraints (like an environment, light conditions, and age of users etc.). A learning curve, ergonomics and productivity are factors being optimized. One of the main tools to verify given goals of HCI research is a user experiment, where the selected computer user is working with the user interface. Experiments answer one or more specified research questions, for example whether the new user interface paradigm is more productive than the baseline.

2.2 User interfaces using human hands

User interfaces that are using human hands as a main method of user input can be divided into two groups. When in-air gestures are used, the user is performing operations through movements of his hands in the air. Touch based interactions use a touch of the fingers on the working surface to execute user intended actions.

2.2.1 In-air gestures

Many systems attempted to use in-air gesture user interfaces in scenarios where users were controlling what is happening on a computer screen. The user would, for example, wave their hands to select and choose menu items that were moving around on the surface of the monitor. This approach is not very compatible with our human nature and with the way our brain understands interaction with real world objects. The problem is with the position of objects that users are trying to manipulate. Humans normally use their hands, not to manipulate something that is out there (on the screen), but what their hands are physically touching. The consequence is that in-air gestures do not work well as a substitute for traditional devices, that manipulate with on-screen content, like keyboard and mouse (in [13], p. 104). They are much more useful in scenarios where a user can submerge in an augmented or virtual reality. Advances have been made by joining Oculus rift virual reality glasses, and the Leap motion controller into a single user interface. Precise detection of finger movements together with an in-place display of a virtual object result in an appealing way how to interact with a virtual world.

2.2.1.1 Gorilla arm syndrome

A user of an in-air gesture based system must be working with an extended arm. In a short time, she starts feeling discomfort and tension in the shoulder; called Gorilla arm syndrome. This condition prevents the user from long-term usage of the interface and decreases user accuracy. The low resolution related to Gorilla arm syndrome may be solved by changing the method of detection, by attaching EMG sensors to hands [5]. Even though EMG sensors can handle the detection issue, they have a disadvantage of forcing the user to wear some hardware.

2.2.2 Touch based user interfaces

When working with these user interfaces, users are interacting by physically touching some objects on the working surface. These objects can be projected on the surface or can physically exist there. Physical objects can not only be manipulated by users, but also by the computer to create bi-directional communication, as in [25].

2.2.3 Multimodal user interfaces

This user interface can be operated by single modality (like keyboard, mouse or touch) or by a combination of more modalities. Rather than focusing on single modality interfaces, our collective goal, as a community of researchers, should be to understand how to most effectively design systems that use input modalities in combination, where the strengths of one modality compensates for the limitations of another (in [13], p. 98). As an example, the CAD system can use a physical manipulator (like wooden cube) to rotate a view, a mouse and keyboard to edit the 3D scene, and a voice to input description of edited components. The same enrichment of user experience can be achieved by a combination of HCI output modalities. This trend is supported by development of virtual and augmented reality user interfaces ([13], p. 388). Tab. 2.1 shows possible input and output modalities used in user interfaces.

Table 2.1: Possible input and output modalities of user interfaces

Modality	**Input/Output**	**Notes**
Vision	Output	2D and 3D displays with images and videos
Hearing	Output	Speakers, headphones, 3D sound
Haptic (sense of touch)	Output	Physical manipulators, e.g., digital clay [19].
Kinaesthetic (sense of balance)	Output	Devices where user can walk on movable surface, while remaining in one spot.
Gustation (taste)	Output	Devices producing taste (applications in future VR user interfaces)
Olfaction (smell)	Output	Devices producing smell (applications in future VR user interfaces)
Keyboard	Input	Space multiplexed input device.
Mouse	Input	Time multiplexed input device, attached to virtual UI elements by clicking.
Touchpad	Input	Space multiplexed input device with multi-touch possibility.
Voice	Input	Voice input analyzed by voice recognition algorithms.

Chapter 3

Related work on human hand analysis

In this work, in-air gesture based user interface is not considered as an option and instead, touch based user interface is designed. This is because the aim is to create an interface, which is used on a daily basis. While performing in-air gestures, users suffer from Gorilla arm syndrome which is not desired.

In any case, human hand is a main input device. In detail, this chapter explores the state of the art research in the area of human hand analysis. This relates mainly to the feature detection phase of an algorithm which interprets human gestures. As an input of the human hand analysis, digital data acquired from the scene digitization phase is used. The output is recognized hand and fingertip features with calculated 2D or 3D coordinates. Having the digital image of the scene, the algorithm needs to know whether there are some human hands in the image and if so, where are they and where are their fingertips.

As already mentioned in the introduction, due to the awkward shape of the human hand, a reliable fingertip detection algorithm is still a vivid research area. Research interest into the fingertip detection problem has grown significantly in recent years. The trend is apparent from the graph in Fig. 3.1. It displays a number of scientific publications that were found between 2007 and 2013 by the academic search engine Google Scholar, using the **'Fingertip detection'** search term.

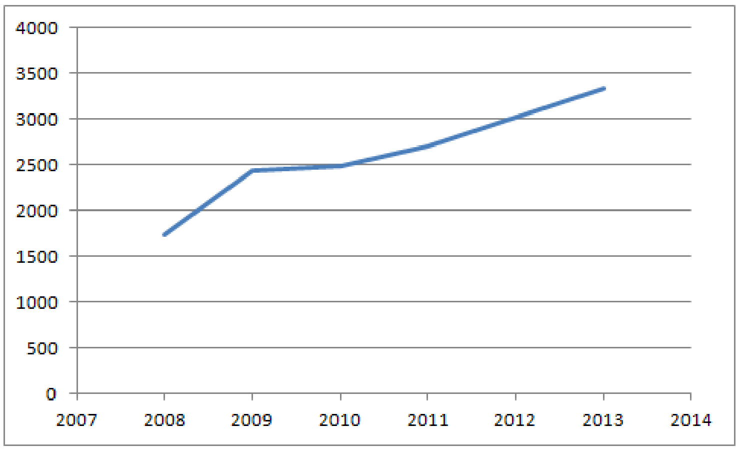


Figure 3.1: Fingertip detection research trend

3.1 Phases of algorithm interpreting human gestures

While building a gesture based user interface, several phases of the computation need to be solved. This chapter describes these phases and studies challenges associated with them. Phases are following:

1. **Scene digitization phase** - the aim is to capture the scene by a suitable sensor, i.e., a visible light camera or a depth sensor.
2. **Feature detection phase** - the aim is to extract information about human hands and fingers from the digitalized scene.
3. **Gesture classification phase** - the aim is to study the position of hands to recognize gestures.
4. **Gesture to user interface mapping** - recognized gestures are mapped to user interface commands.

3.1.1 Scene digitization phase

Obtaining reliable digital data about the scene used to be a very difficult task. Several techniques based on an image captured from a visible light camera were investigated. If two cameras are used for capturing, a 3D model of the scene can be calculated. Optical based systems can be divided into two major groups - marker based and markerless. Marker based systems place high visibility markers on joints and important parts of the human body. In this way, an accurate detection of body movement is achieved. Markerless systems put no physical constraint on the user, however they obviously deliver worse data quality.

In 2009, Microsoft Kinect was introduced to the market. It is able to calculate a 3D depth image from a captured scene and contains software that returns information about full human body motion. The scientific community welcomed Kinect immediately as it enabled them to get robust, reliable data in an affordable way [11]. The depth sensor is usually placed horizontally in front of the user. The user extends his arms and performs various hand gestures, as in [24].

Microsoft Kinect is just one of the hardware options that can be chosen to build up a gesture based user interface. A list of novel, most commonly used devices, that are not based on visible light cameras, is shown in Fig. 3.2.

Wii

Wii is a game console controller developed by Nintendo. It entered the market in November 2006. The device is held in hand in a similar way as if holding a tennis racket. Wii detects a movement in three axes thanks to built-in accelerometers and an infrared light camera.

Microsoft Kinect

Kinect is a depth sensor developed by Microsoft. It is able to reconstruct a depth image (with resolution 640x480 pixels), of the surrounding environment at a rate of 30 Hz. The device projects a grid of small infrared points, to objects in the scene. Depth information is calculated from the distortion of these dots, captured by an IR camera with a resolution of 1280 x 1024 pixels. Based on the depth information, together with depth pixel-level classification of the human body, the device calculates positions of all important body joints. It can provide motion capture data for up to four people in the scene. Its advantage compared to visible light cameras is that it works properly under any ambient light conditions. Kinect also contains a motor, capable of tilting the device up and down, which adapts the view of the sensor to the changing distance from the users.

Asus Xtion PRO

This depth sensor was brought to the market in 2011 by PrimeSense and is based on the same technology as Kinect. Unlike Kinect, it has no motor and is not so widely used.

Leap motion

Leap Motion was developed by the same-named company and was brought to the market in June 2013. It is placed on the table close to the monitor, capturing movements of the hands and 3D trajectory of finger movements. The user does not need to touch the screen when using Leap motion. The device is equipped by two IR cameras and three IR diodes as a source of IR light. Unlike Kinect, the device can capture only hands, but provides much more information about hand features.

The list above intentionally does not contain devices where additional wearable hardware, like gloves or bracelets is required to be worn on a body of the user, as the trend is to build a user interface without any physical constraints.

(a) WII controller

(b) Microsoft Kinect

(c) Asus Xtion Pro

(d) Leap Motion

Source: www.nintendo.com, msdn.microsoft.com, www.asus.com, www.leapmotion.com

Figure 3.2: Modern MoCap devices

3.1.2 Feature detection phase

Feature detection is one of the disciplines of computer vision. Its aim is to compute abstractions of digital image information called features. During the process, local decisions at every image point (called a pixel) are made, whether there is an image feature of a given type at that point, or not. It is necessary to classify hand and finger pixels and separate them from background pixels of the scene. This is done as a part of an image segmentation step, where image pixels are classified by some of the computer vision methods, like HSV thresholding. If working with a depth map from some depth sensor, classification can be done by comparing depth of finger pixels compared to depth of background pixels from calibrated depth frame where hands are not present.

Based on the needs of the natural user interface, the feature detection phase may not only classify pixels and separate a hand from the background, but also extract other information about more detailed features. These can be exact positions of fingertips, the center of a palm, and finger vectors. Simple algorithms work only with a 2D projection of a 3D scene and extract located feature positions only in two dimensions. More complex algorithms are able to reproduce 3D coordinates that can be used to build more advanced gesture detection systems.

Fig. 3.3 shows segmented pixels of a hand, separated from the background, and a detected hand contour with fingertips as a result of the feature detection algorithm.

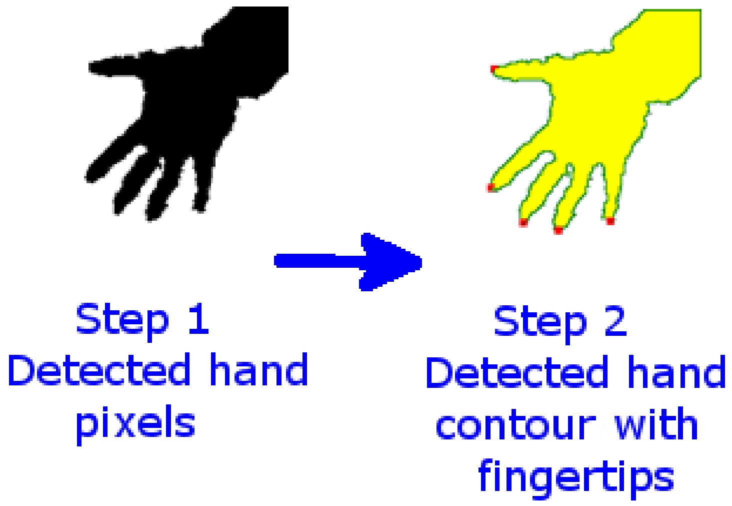


Figure 3.3: Steps of feature detection algorithm

Hand and fingertip feature detection methods can be divided into following groups, based on the chosen approach. State-of-the-art methods used in feature detection phase are described in chapters mentioned in the brackets for each approach:

- Hand contour analysis (described in 3.2)
- Topological image skeleton approach (described in 3.3)
- Detection using circular patterns (described in 3.4)
- Random forests (described in 3.6)

3.1.3 Gesture classification phase

Gestures are used in user interfaces for different purposes. For example a pinch gesture (where the user is pressing together the thumb and the index finger of one hand) can be used for selecting some objects in a 3D scene to interact with them [20]. The gesture where two index fingers are touching the working surface can be used for zooming and rotating the scene [22].

There are two major ways how to detect gestures from the scene:

a) Use a feature detection algorithm, where a feature is defined as a whole gesture.

b) From the detected features of the hand and fingers, calculate a mathematical hand model. Use this model to recognize the gesture, by defining mathematical properties of the hand, which is performing the gesture.

3.1.4 Gesture to a user interface mapping phase

This phase is closely related to an optimal way how already detected gestures translate to commands sent to a computer. It is favourable to choose such gestures for a given task that somehow resemble the action in a non-digital context. This is called a metaphor approach. Until now, a lot of research has focused on finding suitable metaphors for a given task, such as in [30], where a bar metaphor is used. It mimics handling of objects that are skewed with a bimanual handle bar. In [A.1], a new user interface component for storing and retrieving of digital content is introduced. It is based on placing the digital content into a 3D space and associating it with some symbols. This is a metaphor for tidying up, where we store some physical items in the shelves, letting the brain create association between a 3D position of the content and the content itself.

3.2 Hand contour analysis

In this approach, a hand contour is used to calculate fingertips. The contour comprises of a set of connected pixels on a boundary between hand and background. An algorithm analyzes sharp turns of the hand contour and classifies them as fingertips.

3.2.1 Local maxima of y-axis contour coordinate approach

In [10], the algorithm calculates fingertips based on a local maximum of the curvature function. It was developed for an interface where tablet users were pressing buttons on a virtual keyboard, placed on a working desk. The algorithm works in the following steps:

- Edges are detected from the red channel of a RGB camera image.
- Short edges are discarded and a simplified contour is extracted.
- Fingertip candidates are found as points with local maxima of the curvature function.
- Additional criteria to reduce false positives are applied, e.g., minimal fingertip width test. Fingertip width is calculated as a distance between two points to the left and right of the fingertip, where curve direction is parallel to the y axis of the image.

Fig. 3.4 shows images created by the last two steps of the algorithm, where white crosses represent detected fingertips. As apparent from the result, the algorithm has two problems:

- Fingertip positions are not very precise, particularly for the thumb.
- Algorithm is not rotation invariant - due to the method how finger width is calculated; it works only if fingers of user are oriented vertically.

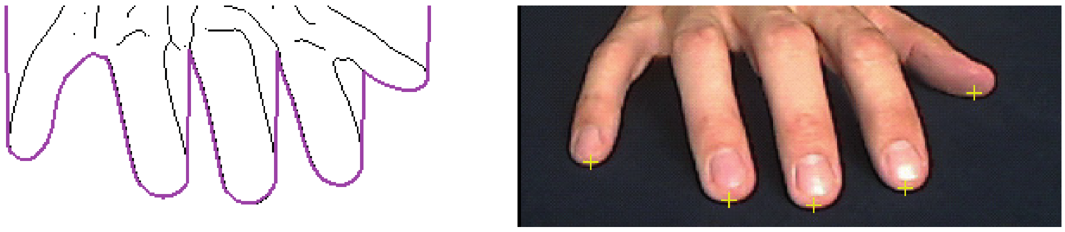


Figure 3.4: Two last steps of the simple curvature based fingertip detection

3.2.2 k-curvature algorithm

In [17], fingertips are detected from the source image using a k-curvature algorithm. Based on a segmented image, the curvature function is calculated from pixels laying on the boundary of hand blob and background. The curvature function describes how fast is changing the direction of the hand contour along the points in it. It is calculated as a first derivative of the angle between subsequent points on the hand contour. Fingertip detection works according to the following steps:

The hand binary image is segmented using a depth image from Microsoft Kinect. Pixels are sorted into two classes - background and hand. An example of such a binary segmented image is in Fig. 3.3, step 1.

The hand contour is calculated from the hand binary image, using a blob search.

Fingertip candidates are calculated using a k-curvature algorithm:

- Three points on the curve L, C and R are selected for an angle test, each having equal distance of k pixels from C.
- The angle between vectors $\vec{LC}$ and $\vec{RC}$ is calculated. If it is lower than a given threshold α, point C is considered as a fingertip candidate.
- Fingertip candidates can also contain valleys between fingers. Valleys are discarded by comparing distances of the candidates from the center of the hand.

The distance k together with the angle threshold α are numerical constants that must be experimentally chosen, based on the proportional size of a hand in an image. The resulting image is in figure 3.5.

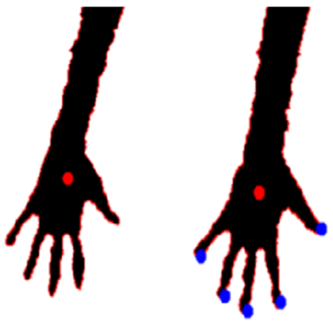


Figure 3.5: Result of k-curvature fingertip detection

In contrary to previous work described in [10], this algorithm has the advantage of being rotation invariant. It means that fingertips are detected no matter what is the rotation of the hand in the image. However, fingertip positions are also not very precise, as a fingertip point is detected only from small surroundings of a fingertip area, comprising of three points on a hand curve.

3.2.2.1 k-curvature algorithm with touch and hover

When two downward-pointing cameras are attached above a planar surface, system can also provide 3D positions of a user's fingertips on and above the plane, allowing us to distinguish between touch and hover [18]. Hover is a finger state, where the finger is lifted slightly above the desk, not physically touching it. Additionally, the hand tracker not only calculates positional information for the fingertips, but also finger orientations. This method thus provides a great amount of extra information on a user's hands, for building a gesture based human interface. Its slight disadvantage is that it can be difficult to setup and calibrate two cameras.

3.3 Topological image skeleton approach

The skeleton of a segmented shape is a thin version of that shape that is equidistant to its boundaries. Because the skeleton emphasizes geometrical and topological properties of a shape, it has been successfully used in several applications in computer vision; such as optical character recognition, fingerprint matching or pattern recognition. If a skeleton is calculated from a segmented image of a hand blob, it can be used to detect fingertips. By reducing hand pixels only to the skeleton, the number of pixels needed to analyze fingertips is reduced. This has a positive effect both on efficiency and also accuracy, as fingertips must be on the axis of the single finger [31]. Example of the skeleton algorithm input and output is shown in Fig. 3.6. As can be seen from this example, a skeleton can contain such endpoints that are not actually fingertips, which can possibly lead to false positives during the fingertip detection phase.

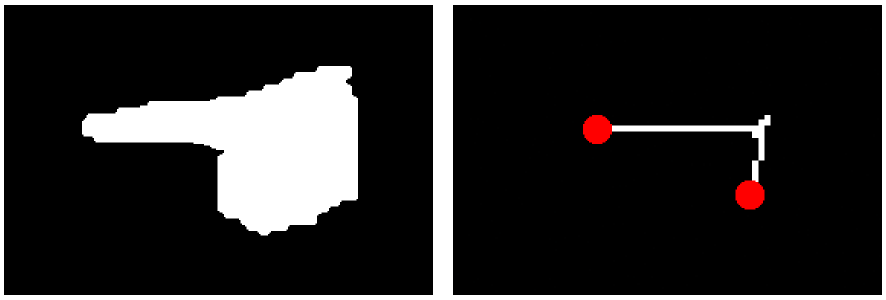

(a) Segmented hand blob (b) Skeleton calculated from hand blob

Figure 3.6: Example of skeleton algorithm input and output

3.4 Detection using circular patterns

A fingertip in the binary image forms a circular shape with one of its ends connected to a larger area of hand pixels, forming a finger segment. Its circular shape can be used to the advantage of the fingertip detection. There are three methods available to do so:

- Template matching
- Applying convolution matrix
- Distance from center of mass method

3.4.1 Template matching

Template matching is a technique in digital image processing, for finding small parts of an image that match a template image. A fingertip can be used as such a template, where initially a sufficient number of fingertip templates is obtained. Though this technique can be effective under defined conditions, the performance may suffer in special cases. This happens if the size of the fingertip in the image can vary, for example if user distance from the camera is unknown. This results in the necessity to have a larger template set, containing different template sizes. Another performance problem can be caused when the input frame needs to maintain a high resolution while being processed, as this increases the number of image region match trials. This can however be overcome by reducing resolution of both the search and template images.

In [8], template matching is applied on a binary image with an extracted hand region (Fig. 3.7b). Authors use L1 norm template matching, with sequential similarity detection algorithm (SSDA) to extract the fingertip. A circular image (Fig. 3.7a) is used as a fingertip template. The circular template may be robust to axial change of the finger. The result of the fingertip extraction is shown in Fig. 3.7c.

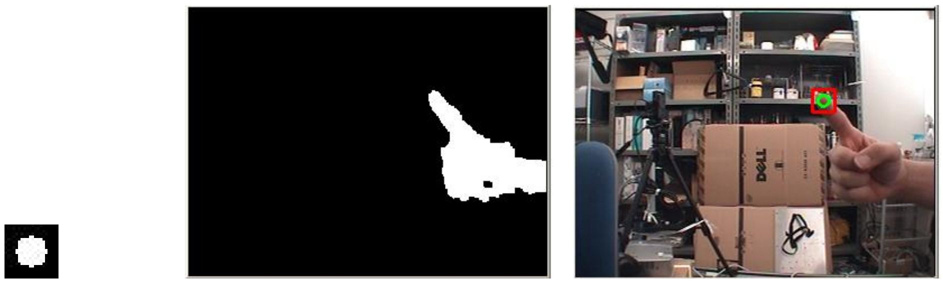

(a) Circular template (b) Segmented hand image (c) Detected fingertip

Figure 3.7: Example of template matching algorithm steps

3.4.2 Convolution matrix

A convolution matrix (sometimes referred as a kernel or mask) is a method commonly used in computer vision for blurring, sharpening, embossing or edge-detection. This is accomplished by means of convolution between a kernel and an image.

The circular convolution matrix can be constructed in such a way that if its center is aligned with the center of the fingertip, a high output response is observed. This can be used for fingertip detection. As with template matching, processing larger hand images of higher resolutions can result in bad computational performance. An example of a convolution matrix used in [6] is in Fig. 3.8. This method not only provides fingertip coordinates, but also finger vectors of last fingertip segments. Even though authors claim that the method calculates angles of bent fingers on both hands, only vectors of the last finger segments are calculated. Vectors of remaining finger segments remain unknown.

The final detected fingertip coordinates are not very accurate, which is apparent mainly for the thumb of the right hand in Fig. 9 in [6].

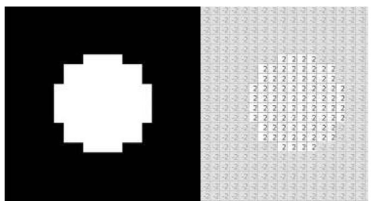


Figure 3.8: Fingertip detection convolution matrix

3.4.2.1 Improved convolution matrix - circular filter

As already mentioned, processing templates or convolution matrices on a larger image can have significant performance impact. This issue is partially addressed in [9] by using a FAST feature detector [26]. Authors are able to recognize gestures at the finger level in real-time at more than 50 fps with commodity computer hardware. The main idea is not to process all pixels of the circular template, but only those that are on the boundary of the circle. The algorithm also uses 3D depth information in the segmentation phase, which yields better results than only using a 2D hand blob. It uses a Bresenham circle of radius 4, to segment the pixel's periphery into arcs of consistently lower, equal or higher depth compared to the center pixel. For comparison, again the maximum deviation threshold is used. The algorithm also detects finger segments (referred to as pipes) and matches detected fingertips with those segments. Although axial finger vectors are provided, its length and orientation is not always properly calculated, as is clearly apparent (despite from low image resolution) from Fig. 3.9 published originally in [9].

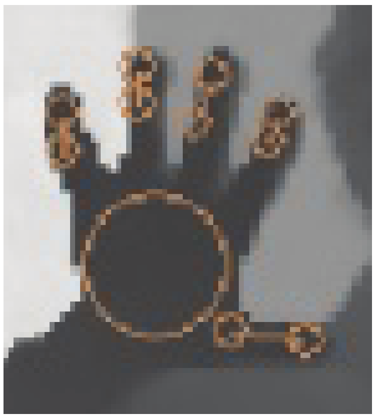


Figure 3.9: Results of fingertip detection using FAST

3.5 Distance from center of mass method

The already mentioned approach for detection of corners in the segmented hands is the curvature function analysis, where an algorithm is searching for local maxima of the function. But if due to some error in the segmentation phase, the boundary of the segmented hand does not become smooth, there can be a very large number of corners detected and extraction of fingertips will become very complicated. This is overcome by the distance from center of mass method [4], which works according to the following steps:

1. User shows the five fingers of the hand in front of the camera for calibration. The hand should be perpendicular to the camera and fully outstretched.

2. Most distant point in the hand contour from the centroid is found and circle is constructed around the centroid.

3. Radius of the circle is decreased (by optimal step size) till five peaks of the graph are eliminated, representing five detected fingers.

Algorithm idea is explained in Fig. 3.10.

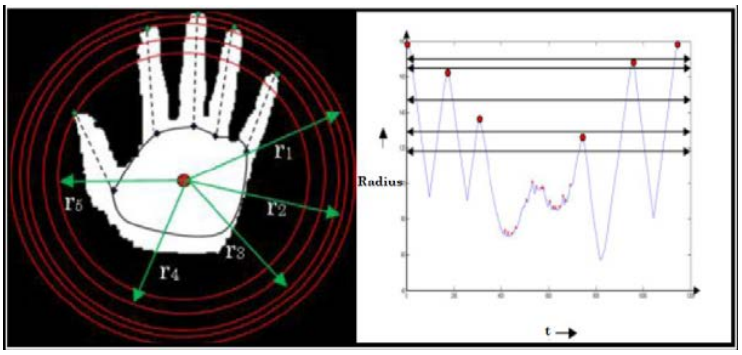


Figure 3.10: Distance from center algorithm for fingertip detection

The algorithm is capable not only to provide reasonably exact fingertip coordinates, but also axial finger vectors of stretched finger. The algorithm will fail though if a finger is bent, as a fingertip may be closer to the hand contour centroid than some other point on the hand curve.

3.6 Random forrests

Shotton et. al [29] published a state of the art real-time human pose recognition algorithm. A single input depth image is segmented into a dense probabilistic body part labeling, with the parts defined to be spatially localized near skeletal joints of interest. Re-projecting the inferred parts into world space, authors localize spatial modes of each part distribution and thus generate confidence-weighted proposals for the 3D locations of each skeletal joint. The final classification of source image pixels is shown in Fig. 3.11.

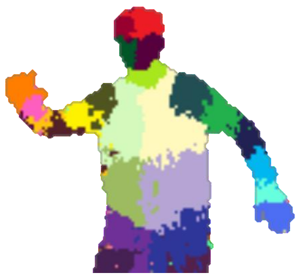


Figure 3.11: Result of human body classification using random decision forest

The algorithm works according to the following steps:

1. Random decision forest classifier is trained using two datasets containing training images: real camera images and synthetised camera images. Realistic synthetic depth images are sampled from a large motion capture database, picturing humans of many

shapes and sizes in highly varied poses. The classifier avoids overfitting by using hundreds of thousands of training images. Decision trees in the forest are trained to classify several very simple depth features, for example a feature that can detect thin structure or a feature that detects pixels high above the floor.

2. Single depth frame pixels are iterated and classified using trained trees by traversing from the root down to leaf nodes:

    - Split nodes consists of a feature f θ and a threshold τ. Traversing down continues left or right according to comparison with threshold τ.

    - When leaf node in tree reaches threshold τ, learned distribution over body part labels is stored.

3. The distributions from all trees in the forest are averaged together to give the final classification.

4. Spatial modes of the inferred per-pixel distributions are computed using a mean shift, resulting in the 3D joint proposals.

Due to the use of very fast tree data structure and simple feature detectors, super real-time performance was achieved (processing of approx. 200 frames per second). Also thanks to the prior training on both real and synthesized datasets earned discriminative approach naturally handles self-occlusions and poses cropped by the image frame.

Though it is excellent for full body pose estimation, it does not provide enough details about hand features. Depth pixels of individual fingers are classified only as right hand or left hand so algorithm can not be used for determining of hand gestures. It also can properly categorize human shape only for poses for whose it was previously trained. For fingertip detection, training or synthesizing the dataset would be tedious task due to an enormous variability of all finger movements.

Chapter 4

Research goals

The ultimate goal of my research is to answer questions about the most suitable design and implementation of a 3D user interface, based on an intuitive and ergonomic human computer interaction. Existing gesture-based interfaces are not meeting the ergonomy goals yet. In-air interfaces suffer from Gorilla-arm syndrome, while touch-based do not allow users to rest their hands or are not achieving needed accuracy.

However, this is a life-time research task. This thesis therefore focuses on a more specific research question, which is a building stone to fulfill the ultimate goal. The main focus of this thesis is to find such a real-time algorithm for human hand analysis, that has a high fingertip detection accuracy compared to already published work. The algorithm in question should also extract additional clues about detected hand features. The final algorithm should therefore meet these objectives, ideally all of them:

- High fingertip detection accuracy: fingertip coordinates should be calculated with precision that allows users to perform accurate computer tasks, e.g., if using fingers instead of a mouse cursor,
- ability to detect fingertips for both straight and bent fingers,
- ability to provide additional hand features: this includes palm center plus forearm and finger vectors,
- ability to provide coordinates of both base and middle joint of each finger, so that information about how the finger is bent can be used for gesture analysis,
- 3D calculated coordinates: if source data from digitization phase contains depth information, a three dimensional position of the fingertip should be calculated,
- real-time performance: at least 30 frames per second processing speed to allow real-time usage in HCI scenario.

As to my best knowledge, an algorithm meeting all these goals has not yet been published.

Secondary objectives of the work are following and are related to 3D user interface design and to the gesture to a user interface mapping phase:

- Explore the relation between the placement and orientation of a depth sensor and advantages that the placement poses for the user interface.
- Study an impact of using other possible modalities in a user interface, that are suitable to accompany human gestures (like human voice or physical manipulators).
- Introduce a new user interface paradigm that is more suitable for particular tasks then if using a traditional mouse and keyboard user interface.

Chapter 5

Design of TouchTable - gesture based user interface prototype

As the main goal of the thesis is to find a hand detection algorithm, a user interface prototype is needed to test this algorithm in real user experiments. This chapter contains the details on designing and implementing the TouchTable - a multi-modal user interface where users are using their hands as an input device.

5.1 Methodology of design of TouchTable

During the design process, the following methodology based on the feedback from users was used. One iteration of the process was organized in the following steps:

- Based on current knowledge, a prototype of touch-based 3D user interface is designed and implemented.
- An experiment is prepared and executed. Users are given a task that they should perform both with traditional UI (keyboard and mouse) and novel prototype.
- The resultant data is collected during the experiment, such as time to complete the task or verbal feedback from users. Results are analyzed and compared.
- Based on the analysis of the results, conclusions are made about the user interface prototype. New knowledge regarding suitability of the design is gathered and a new hypothesis is formed on how to improve the design.
- Prototype is improved.

Apart from this evolution process and inter-related user experiments, other measures how to evaluate the new implemented algorithms can be used. In this work, a hand detection algorithm was also evaluated against a dataset containing a sufficient number of 3D image samples with a hand. During this evaluation, fingertip detection accuracy and algorithm performance were measured. It was critical to achieve best possible fingertip detection accuracy and a processing speed higher than 30 frames per second. This is described in detail in chapter 6.

5.2 TouchTable

The TouchTable is a prototype of a 3D user interface for working with a computer, that was published in [A.7]. User experiments, where users are executing computer operations with their hands, need to be performed. The TouchTable is not only pursuing the search for the ideal 3D user interface, but also providing necessary feedback for further evolution of performance, precision and behaviour of a fingertip detection algorithm.

The TouchTable analyses human hands lying on the desk. It is using a fingertip detection algorithm to determine finger coordinates of both user's hands in real time. It also collects data about the height of the fingers above the desk. The interface is multi-modal as it uses speech recognition to execute voice commands.

5.2.1 Depth sensor placement above the desk

During the scene digitization phase, the TouchTable is using a Microsoft Kinect depth sensor as an input device. The depth sensor is usually placed horizontally in front of a user. The users extend their arms and perform various hand gestures, as in [24]. Whereas the setup with a horizontally facing depth sensor can be quite suitable for capturing a whole body motion, it is not suitable for detecting tiny finger movements. A user of such interface also suffers from Gorilla arm syndrome, as described in chapter 2.2.

5.2.2 TouchTable goal

A TouchTable's goal is to increase productivity of the user, who is working with a computer, with a focus on CAD systems. It tracks both users' hands resting on the desk. Depth information is analyzed and mapped to commands that control the computer. A computer image is projected from underneath, to a semi-transparent working desk. The novel vertical placement of the depth sensor brings the following advantages:

- The user can rest both his arms on the desk and does not suffer from the Gorilla arm syndrome. He can have palms laying on the table. This is an advantage against regular touch displays placed horizontally, where users cannot rest hands on the surface, as palm touch would generate undesirable interactions.

- As user's arms are supported, the precision is good enough to detect fingers tapping on objects, as small as 64x64 pixels. As a user sits at the desk, he can use the new interface as a complementary way to control the computer and, from time to time, switch to a usage of a traditional mouse and keyboard interface easily.

5.2.3 Overall TouchTable architecture

Fig. 5.1 shows a TouchTable setup, consisting of a display and a detection module. The display module is responsible for projecting a computer image onto the projection desk. The detection module captures hands of a user, analyzes the data and sends appropriate control commands to a computer.

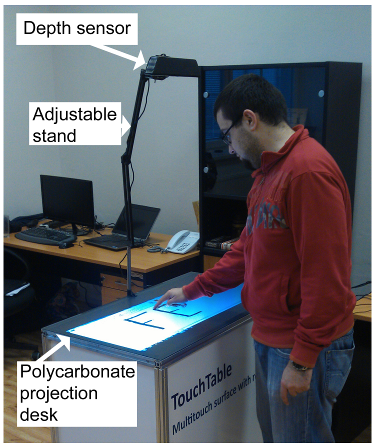


Figure 5.1: Layout of TouchTable interface

The display module consists of a table with a semi-transparent polycarbonate desk attached to the top. The desk is 2 mm thick and has a size of 900x540 mm, placed 87 cm above the floor. A special projector with a short projection range and high luminance (3500 ANSI) is hidden inside the table, projecting the computer image to the desk from below. Finger positions and an image projected onto the polycarbonate desk are synchronized, so that an interaction with the desktop happens in the same place where the finger touches. Sometimes, areas in the returned depth map, contain blobs with depth reported as 0. This is caused by a malfunction of the Kinect device under some unspecified light and reflection conditions. If there are a lot of such invalid depth pixels, the whole user interface can no longer work properly. By trial and error, I have realised that these false spots can be minimized by placing the already mentioned polycarbonate desk on another desk from chemically brushed glass. This has an effect of somehow helping the Kinect to properly determine the depth of nearly all pixels in the depthmap. It even slightly sharpens a projected image from the short-range projector.

The detection module consists of the depth camera (Microsoft Kinect), placed on an adjustable stand 89 cm above the working surface. The camera is attached to a processing computer, IBM Lenovo ThinkPad, 3MB RAM, 2.2 GHz Intel Core 2 Duo. Software, implemented in .NET and using EmguCV image processing library. To communicate with Kinect device, TouchTable was originally implemented with OpenNI and NITE software libraries. Later, the implementation was changed to Microsoft Kinect SDK because of the difficulties to properly install OpenNI drivers.

For speech recognition, the TouchTable uses a Microsoft Kinect microphone array consisting of four microphones. Word commands recognized from an audio stream are accessed through Microsoft Kinect Speech API.

5.2.4 Processing algorithm

- When the first frame is processed, the depth map is stored as an **EnvironmentDepthMap**.
- When a frame with a hand on it is processed, hand pixels are recognized by comparing the actual distance of a pixel with the **EnvironmentDepthMap** from the previous step. If the distance is bigger than a threshold of $\tau = 1cm$, the pixel is regarded as a hand pixel (value=1) in a 1-bit image, representing the hand and the table.
- The hand map image from the previous step is first eroded, and then a non-hierarchical list of contours is extracted.
- Fingertips are calculated using some fingertip detection algorithm, described in chapter 6.2. For performance reasons, the number of points in a contour can be reduced by replacing all the points from straight line segments with their end-points.
- A heuristic is applied to match the detected fingertips to finger indexes, as described in the chapter 'Finger adherence algorithm' in [A.5].
- A smoothing of the fingertip coordinates is applied. A position of the finger from the last few frames is averaged to avoid shaky movements of the mouse cursor, controlled by the finger. Best results are achieved when using the last four frames. Should this value be too high, an undesirable effect is experienced, where cursor movement follows finger movement with noticeable delay.

5.2.5 TouchTable gesture to user interface mapping

Once finger positions and movements are properly detected, they must be mapped to specific computer commands. To do it in a meaningful way, some insight was needed to understand how a user works in a CAD system. This is why a user experiment was conducted. Users were asked to use a mouse and a keyboard (referred as regular interface) to draw a simple house (in Google SketchUp) with a pointed roof and exact dimensions (Fig. 5.2). Timing and frequency of commands were than measured using a proprietary keyboard and mouse monitoring application.

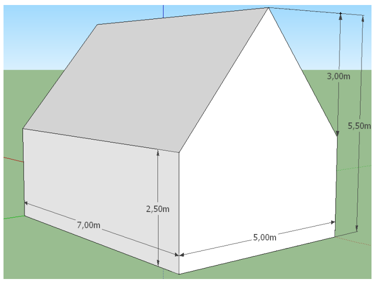


Figure 5.2: The sample house that testers were asked to draw in Google Sketchup

Fig. 5.3 shows a graph with information extracted from the time measurements for a regular interface.

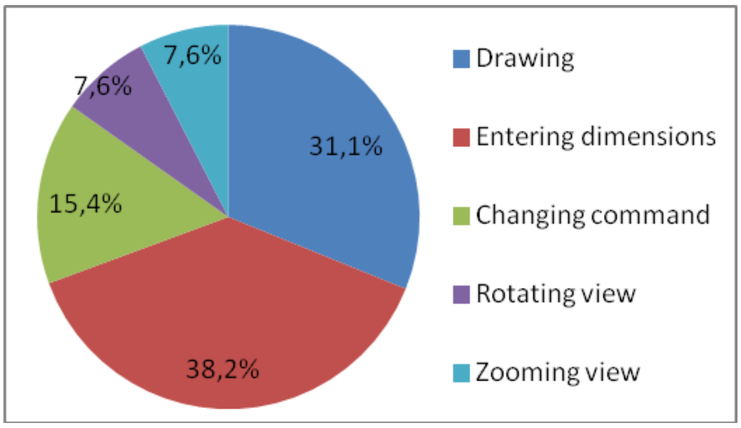


Figure 5.3: Percentage of the time spent in user activities, when drawing using regular interface, grouped by the type

Once better understanding of user behavior was gained, subsequent mapping of user activity to the user interface command, for TouchTable interface, was chosen. The mapping is listed in table 5.1:

User Activity		**User Interface Command**
The index finger touches the desk	+ thumb is hidden in a palm.	Left mouse click
	+ thumb is sticking out of a palm, touching the surface.	Rotating camera around the center of screen
	+ thumb is sticking out of a palm, not touching the surface.	Double mouse click
The index finger is moving on the desk.		Mouse drag & drop
Two index fingers are performing Zoom/Unzoom gesture.		Zoom/Unzoom viewport of 3D scene
A user voice command "Pulling", "Hand", "Draw Line", "Move It", "Cancel".		Activate the tool Pull, Hand, Line, Move,,Undo
The user pronounces figures, e.g., "two point five".		Type exact dimension for last design operation

Table 5.1: Mapping of user activities to commands

5.2.6 User experiment

To evaluate the TouchTable prototype and its fingertip detection algorithm, 5 users (referred to as U1 to U5) were asked to draw houses 10 times. Firstly, they used a regular interface and then they used the TouchTable. U1 was an experienced TouchTable operator, testers U2-U5 had no such experience and before the experiment, were allowed to perform only two trials. If a user made a mistake during the experiment, he pronounced 'Undo' as a command to revert it. The time to fix the error was also recorded, resulting in a longer drawing time. Table 5.2 shows the result of the experiment.

Measured Values	**Time to Finish a Drawing of Sample House (in Seconds)**	
	Mouse & keyboard	**TouchTable**
Average	27.74	56.24
Standard Deviation	11.19	26.72
Min	17.16	22.80
Max	78.34	141.15

Table 5.2: TouchTable interface compared to a regular (mouse and keyboard) interface

The experiment revealed that an average time to finish a task using the TouchTable was slower by 28.5 seconds compared to using the regular interface. The best TouchTable time of the experienced user U1 was slower only by 5.64 seconds, compared to the best regular interface user of 17.16. By analyzing data and observing user behavior, the following conclusions were made:

- Longer times for using the TouchTable were mainly caused by frequent mistakes, as a finger often tapped on an unintended location. This manifested in a high average changing command time of 8.8 seconds (users pronouncing "Undo"). The main reason for mistakes was not caused by users' faults, but by low accuracy of a used fingertip detection algorithm, which was the 'Hand contour polygon simplification algorithm' described in chapter 6.2.2. Further research was then made to improve this accuracy as described in chapter 6.2.

- TouchTable users were interleaving voice commands with the finger movement to work fluently. Users reported this as very comfortable. However, no significant time was saved this way, compared to traditional and very fast keyboard shortcuts.

- A significant learning curve is related to the TouchTable interface for proper pronunciation of commands, for finger movements, and hand coordination.

- TouchTable users did not need to change an active command if they wanted to rotate or zoom the current view, they just used more fingers. A regular interface user needed to press the associated keyboard shortcut. However, no significant time gain was observed.
- Zooming by using two fingers was as fast as using the mouse scroll wheel.
- It was difficult for users to get used to the fact, that they can (and should) rest their palms on the surface. It is against their experience with classical touch displays, e.g., smart phones. It took users several minutes to actually relax the hand on the surface fully. In comparison with the normal touch screens, this allowed them to be free from the strain put on a shoulder and arm muscles, while operating with user interface.

Chapter 6

Design of accurate fingertip detection algorithm

Humans use their fingertips as actuators very naturally, for their interaction with the real world. They use the index fingertip to point to some destination, they can tap on some objects to bring them into focus or they are using fingertips for different gestures. A hand based user interface must be in compliance with this natural schema of interaction. To build a successful user interface, understanding of human hand in the captured image is a crucial task with respect to fingertips. It is necessary to compute where exactly the fingertips are. This chapter explains what is understood by the word 'fingertip' and contains proposed implementation and evaluation of fingertip detection algorithms, that are solving the goals stated earlier in this thesis.

6.1 Definition of a fingertip

Before fingertips in a scene can be detected, proper definition of 'fingertip' and how to understand its coordinates must be clarified. Based on this fingertip definition, fingertip detection algorithms can be then designed and tested for accuracy. In general, two approaches can be chosen for fingertip definition:

- Mathematical hand model approach
- User defined approach

6.1.1 Mathematical hand model

In [33], Deliang Zhu and col. propose a 3D mathematical model of a hand and all fingers with their joints. More coordinate systems are defined:

- Global coordinate system of input device, like a depth sensor

- Local coordinate system of a palm, that is simplified as a rectangle. Origin with x,y,z = (0,0,0) is located in the center of the palm.
- Local coordinate system for all fingers, defined relative to the palm coordinate system.

Authors define several constraints for the hand model, like static distance of finger roots from the palm center, or perpendicularity of finger planes from the palm plane (with exception of thumb plane which may not be perpendicular). During movement of the hand, it should follow the static constraints and dynamic constraints of the hand. The hand model can be seen in Fig. 6.1. Fingertips are defined as final joints in the hand skeleton. The final accuracy, when used in user interface, depends on how well was the hand model fitted over the hand, using the captured data from scene digitization phase (described in chapter 3.1).

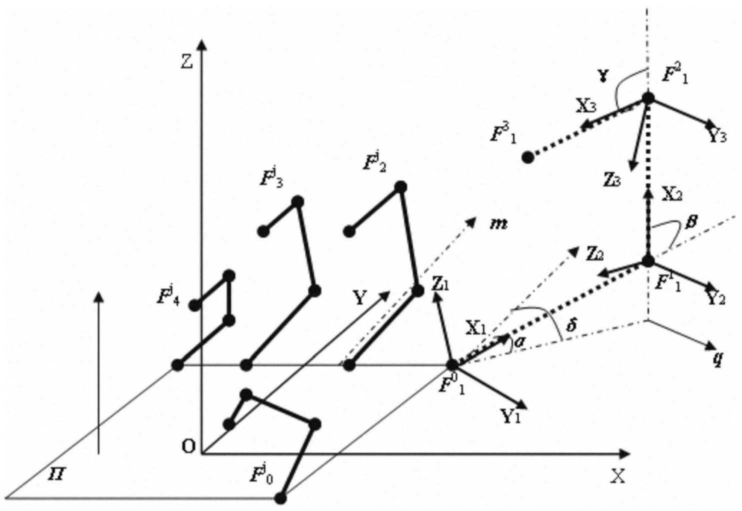


Figure 6.1: 3D skeleton hand model

6.1.2 User-defined fingertip definition

As a human hand is an organic shape that is hard to define mathematically, each user can understand in a different way, where is the tip of his finger. Thus, the system can be taught by first letting users to manually define their fingertip positions in captured sample images. The implemented algorithm's accuracy can be then compared against this annotated dataset. In an experiment, an algorithm can analyze all samples in the dataset and the distance between the users' expected fingertip position and the calculated position can be then measured. The algorithm can then be improved to minimize this difference. For the purpose of the dissertation, the DepthTip dataset was created to serve for user defined fingertip detection.

6.1.2.1 Using a dataset to evaluate fingertip detection algorithm

For the purpose of evaluation, any dataset containing test images of the hand was needed. At first, ColorTip [16] was considered for the purpose. Unfortunately, it contains images of the whole body of a user and does not have sufficient resolution for accurate evaluation of the fingertip detection. This is the reason why DepthTip [2] was created. DepthTip is a set of 73 test images, where images of the right hand are captured using Microsoft Kinect. 10 images are captured for each of the fingers with different bent finger angles. The database also contains 3 images with all five fingers visible and 14 where none of the fingers are extending out of the palm contour. DepthTip images are manually annotated to mark exact fingertip coordinates. This is done by coloring a single pixel with the marker, as can be seen in Fig. 6.2. As stated earlier, this allows us to validate fingertip detection algorithms.

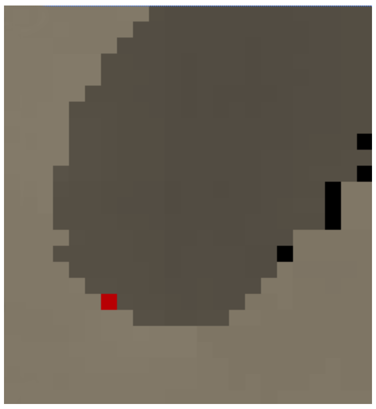


Figure 6.2: Single pixel marked as fingertip in DepthTip dataset

6.2 Fingertip detection algorithms

As a part of the thesis, several algorithms were designed for a fingertip detection, that are able to extract hand and fingertip features from the depth sensor. They differ mainly by processing speed and fingertip detection accuracy:

- Simple algorithm based on hand curvature analysis
- Hand contour polygon simplification algorithm
- Finger cut-off algorithm
- Circular scan algorithm (all pixels)
- Circular scan algorithm (contour only)

In this chapter, these algorithms are described. Based on the feedback from the TouchTable interface, fingertip detection went through an evolution with aim, to increase fingertip detection accuracy, increase number of finger and hand features detected or improve the processing speed.

The source code for all fingertip detection algorithms is available for download, together with sources to run the TouchTable user interface. Necessary information can be found in Appendix B - Dissertation source code.

6.2.0.2 Preconditions for algorithms to work properly

As an input, the algorithms take a binary image of hand pixels, where white pixels represent the hand and black pixels represent the background. An example can be seen in Fig. 6.3). It can be created by any method of 'feature from background' subtraction, like applying HSV threshold on a frame from an RGB camera. Additionally, noise reducing filters may be applied. In the TouchTable, a binary hand image is calculated from a depth image retrieved from a depth sensor. On this binary image, a minimal finger thickness threshold τ is applied. If the pixel has a distance from the working desk higher than τ, it is considered a hand pixel. Table distances are measured during a device calibration, where no hands are physically present in the frame.

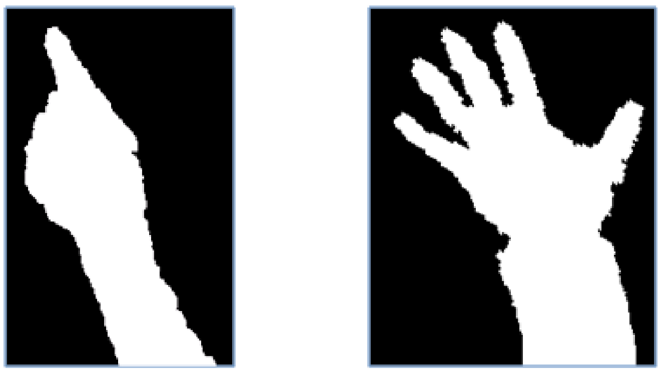

(a) (b)

Figure 6.3: Input images for finger detection algorithm

6.2.0.3 Algorithm phases

Algorithms are implemented in C++, using EmguCV. EmguCV is a wrapper of the frequently used OpenCV software library. However, any similar image processing library can be used. The input image is analysed in two phases:

1. Contours of blobs of hands are retrieved.
2. Based on the detected contours, fingertips are calculated.

6.2.0.4 Phase I - contour detection

The aim of this phase is to extract contours of the hand as a sequence of 8-connected pixels for further hand shape analysis.

1. The input image is eroded using a $3x3$ rectangular structuring element to reduce the amount of noise.
2. Contours of hand blobs are detected using a regular blob detection algorithm. In the TouchTable implementation, the OpenCV Method cvFindContours in `CV_CHAIN_APPROX_SIMPLE` mode is used.
3. Contours that do not meet a minimal bounding box limit are discarded, as they do not belong to blobs of the hand, but rather to a background noise. In our setup, the size of this limit was experimentally set to $30x30$ pixels. This size depends on the image resolution and proportional size of the hand's blob inside the captured image. Also contours that do not meet the minimal hand border length threshold (of 150 pixels) are discarded too.

6.2.0.5 Phase II - fingertip detection

The aim is to calculate finger features based on the preprocessed hand image, and the hand contour. This phase differs for each of the presented fingertip detection algorithms.

6.2.0.6 Performance experiments

For a reasonable HCI use case, it is necessary for a fingertip detection algorithm to work in real-time. Performance was therefore measured using a computer system time, while calculating results 1000 times on a set of all DepthTip images. This was the experiment setup:

- Image resolution: $343x324$ pixels
- Computer spec.: Intel(R) Core(TM) i7-3632QM
- CPU speed: 2 x 2.20GHz

Measured times can be found in chapters related to each individual algorithm. Overall comparison of algorithms can be found in chapter 6.3.

6.2.1 Simple algorithm based on hand curvature analysis

This algorithm was implemented first in April 2011. It is an implementation of a k-curvature algorithm. It analyzes the binary hand image in three phases:

1. All pixels in one hand contour are iterated. These are considered as fingertip candidates.

2. Fingertip candidate is discarded, if it is not part of a convex hull of the hand contour. This allows us to discard such points, that represent locations between adjacent fingers, close to the palm center.

3. The angle between the fingertip candidate pixel and two pixels in the sequence (distant by 40 pixels to the left and right from the current) is measured. If the angle is less than the defined angle threshold α (40 degrees), the pixel is considered to be a fingertip. If more pixels in a sequence are analyzed as fingertips (typically along the curve of the fingertip), they are merged into a single fingertip and the middle pixel of this sequence is used as a fingertip.

All constant values were chosen experimentally based on the characteristics of a hand blob and finger shapes in a depth image. This is possible in this scenario because the distance of TouchTable user's hands from the sensor is approximately constant.

The algorithm has very poor performance, processing approximately one frame per 3 seconds, which makes it unusable in a real-time scenario. Fig. 6.4 shows an input and output of the algorithm. Yellow pixels represent recognized hand pixels, green lines represent recognized hand contours and red spots represent detected finger tips. Moreover (as can be seen in the example output picture), fingertip detection accuracy is biased towards the non curved sides of the fingers.

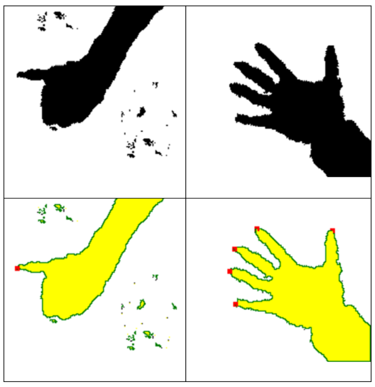


Figure 6.4: Output of the simple algorithm based on a hand curvature analysis

6.2.2 Hand contour polygon simplification algorithm

The main goal of this algorithm [A.7] is to increase the performance of the previous algorithm, so that it can be utilized in user experiments. Its average processing speed for one frame was measured against DepthTip dataset. It works extremely fast, processing 1 frame in less than 1 millisecond. Average processing speed is 0.08 milliseconds which is completely sufficient for HCI interaction.

The idea of this algorithm is to simplify the hand contour by polygon approximation, as can be seen in Fig. 6.5. The approximated polygon is marked in the figure along the hand shape, fingertips are marked as diagonal crosses. The rest of the algorithm works in the same way as Simple algorithm based on hand curvature analysis, where fingertips are found as points with sharp angles between left and right neighbors.

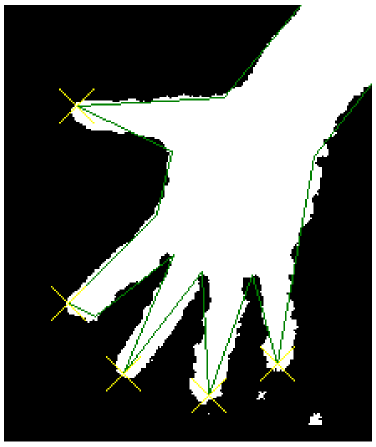


Figure 6.5: Polygon simplification of hand contour

6.2.2.1 Algorithm failures

Algorithm was tested against the DepthTip dataset and failed to detect 11 fingertips out of 95 (11.58%). It does not correctly detect fingertips in situations where the finger is bent. In that case, the fingertip is closer to the palm center and is not part of the convex hull of the hand contour. Therefore, it is discarded by the algorithm. Fig. 6.6 (a) shows such a failure for the DepthTip sample no. 10. Failed detection happened for following DepthTip samples: 10, 39, 40, 48, 49, 53, 54, 55, 58, 70 and 71.

Additionally, the algorithm failed not only for bent but also for straight fingers. This is because polygon simplification sometimes flattens the shape of a finger, forming a rectangular fingertip area rather than a sharp fingertip angle. Thus fingertip angle does not meet the threshold criteria and the fingertip is not detected. Fig. 6.6 (b) shows such a situation as thumb is not detected.

6.2.3 Finger cut-off algorithm

This chapter presents a finger cut-off algorithm for accurate calculation of fingertip coordinates based on hand contours [A.5]. It provides not only information on exact fingertip position, but also orientation and lengths of all fingers in the image. The main goal of design of the 'Finger cut-off algorithm' is higher accuracy of fingertip coordinate calculation. This precision makes this algorithm unique. For straight fingers, it is superior not only to the 'Hand contour polygon simplification algorithm' described in chapter 6.2.2, but also to all other algorithms from literature mentioned in 3.2. The algorithm is real-time; processing 30 frames per second, to allow real-time interaction of the user with the computer.

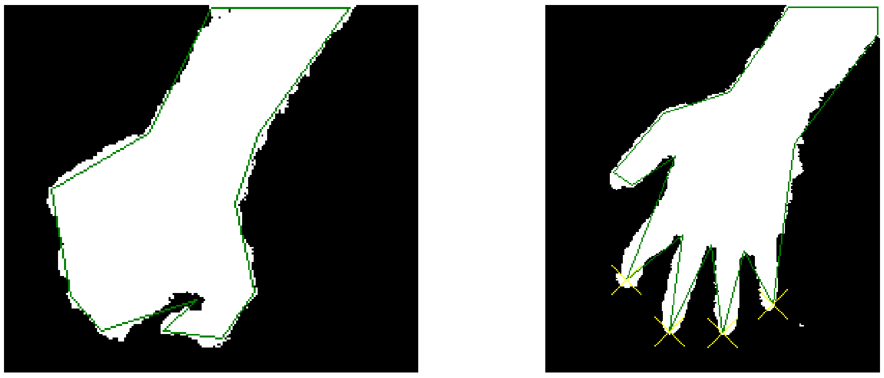

(a) Sample no. 10 (b) Sample no. 71

Figure 6.6: Failed cases of 'Hand contour polygon simplification algorithm'

Implementation was done using the computer vision software library OpenCV. Where appropriate, used library methods are mentioned for reader's reference. Fig. 6.3 shows examples of an input image.

6.2.3.1 Algorithm details

The fingertip detection works in following steps:

1. Polygon approximation of the hand contour is constructed in the same way as for the 'Hand contour polygon simplification algorithm', as seen in Fig. 6.5.

2. Polygon points are iterated and compared to their neighbors. If the distance between two consecutive points is smaller than the minimal threshold of *minDist* pixels, points are merged and replaced by their mid point. This is to avoid a situation where more than one point is part of a simplified polygon, and is still very close to the fingertip. With the *minDist* value setup experimentally to 15 pixels, all close polygon points at the fingertip areas were merged properly.

3. All points in the polygon are analyzed and assigned to one of the following classes, where the number in brackets is a numerical representation of the class: fingertip (class 1), concave finger base (class 2), convex finger base (class 3), concave hand contour (class 4) and convex hand contour (class 5). Fig. 6.7 shows detected points with assigned classes. The classification is done using these rules:

   - Class 1: point in a convex polygon hull, meeting minimal angle to its direct neighbors threshold criteria.

   - Class 2: point not being part of the convex hull, that is direct neighbor of Class 1 point.

- Class 3: point being part of the convex hull, that is direct neighbor of Class 1 point.
- Class 4: point not being part of the convex hull, that is not Class 2.
- Class 5: point being part of the convex hull, that is neither Class 1 nor Class 3.

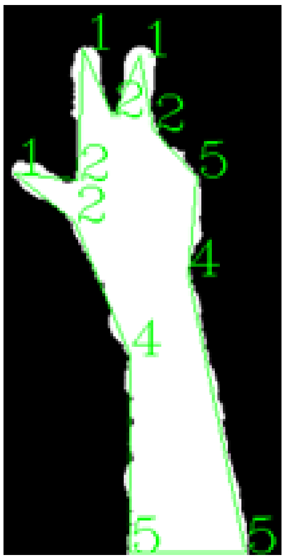


Figure 6.7: Classes assigned to points in the polygon

4. For each of fingertip points (class 1), finger cut-off line is drawn. It separates the blob of associated finger pixels from a blob of palm pixels. This line intersects the neighboring concave finger base point (class 2) and is perpendicular to the finger axial vector, defined in Eq. 6.1.

$$\vec{a} = (\vec{v_1} + \vec{v_2})/2 \tag{6.1}$$

$\vec{v1}$ and $\vec{v1}$ are vectors pointing from the fingertip (class 1) to finger base points (class 2 or 3). Fig. 5 shows a binary image with the cut-off line drawn. If two concave finger base points are detected for the fingertip, only the one closer to the fingertip is used to create the cut-off line. At the end of the cut-off line, a black circle with radius of 3 pixels is drawn. This is to force a clear separation of the cut-off finger blob from the blob of the palm, as the blob detection step works in 8-connectivity mode. For the same reason, width of the cut-off line is two pixels rather than 1 pixel.

5. Blobs are detected in the binary image with cut-off fingers, using 8-connectivity. OpenCV Method cvFindContours in `CV_CHAIN_APPROX_SIMPLE` mode is used. Minimum bounding boxes of blobs are visible in Fig. 6.8.

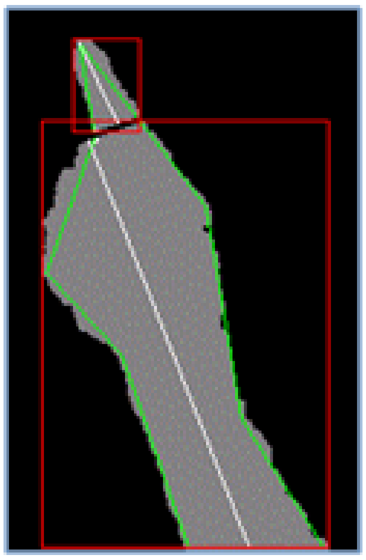
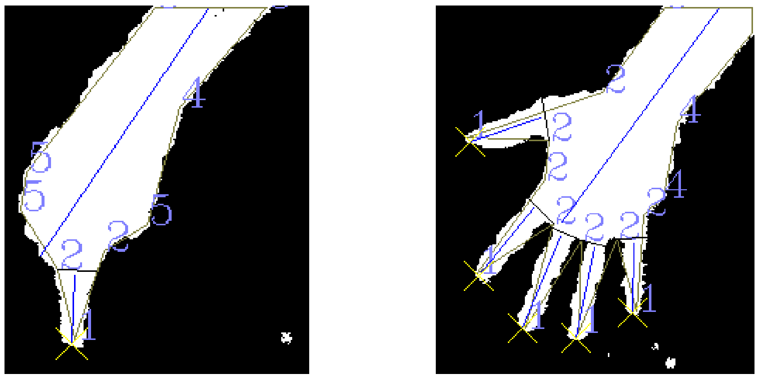


Figure 6.8: Minimum bounding boxes of cut-off blobs

6. For each detected blob, a fitting line is calculated using the least squares method. Coordinates of all pixels in the blob are used as input data for the least squares. The resulting calculated line is an exact representation of the finger axial vector.
7. Length of the finger axial vector is calculated by finding two intersecting points of the finger axis, going through the center of the finger blob, with contour of the hand. Fig. 6.9 shows the result of the algorithm.



(a) One detected fingertip (b) Five detected fingertips

Figure 6.9: Final results and fingertip detection precision

6.2.3.2 Detection results

As already mentioned, the fingertip is calculated as an intersection between the fitting line of all pixels of the finger's blob, and the whole hand convex contour. This ensures that calculated coordinates will be precisely in the medial axis of the finger. Fig. 6.10 show the difference between the old Hand contour polygon simplification algorithm and new algorithm. Accurate finger position, calculated by the new algorithm, is marked with a red circle, while the sharp corner of the simplified polygon used by the old algorithm is drawn in gray (being slightly biased towards the left edge of the finger).

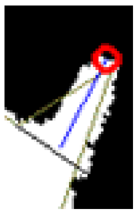


Figure 6.10: Difference in accuracy between 'Hand contour polygon simplification algorithm' and 'Finger cut-off algorithm'

The algorithm also provides additional clues about hand features to help design advanced user interfaces:

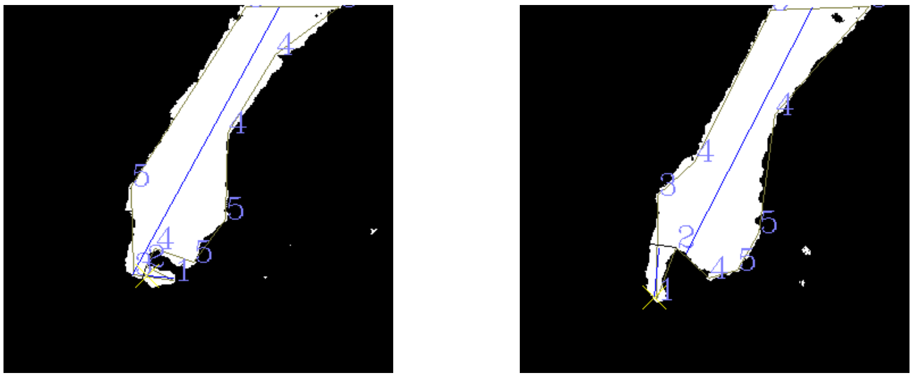
1. Number of detected fingers,
2. length and axial vector for each detected finger,
3. whole hand orientation and visible length; this is possible because the user's palm is detected as the largest blob.

The limitation of this algorithm is that it is not suitable for such hand gestures, where fingers are bent and do not form straight lines. This problem is solved by another algorithm, described in subsection 6.2.4: Circular scan algorithm - all pixels.

6.2.3.3 Algorithm failures

Algorithm was tested against the DepthTip dataset and failed to detect 27 fingertips out of 95 (28.42%). It does not detect a fingertip in situations where the finger is bent. In that case, the fingertip is closer to the palm center and therefore the finger axial vector is reverted. Subsequently, the fingertip is detected in a wrong place. Fig. 6.11 (a) shows such a failure for the DepthTip sample no. 29. Failed detection happened for following DepthTip samples: 8, 10, 16, 17, 18, 19, 20, 25, 28, 29, 30, 34, 36, 37, 38, 39, 40, 48, 49, 52, 53, 54, 55 (2 failed fingertips), 58, 70 and 71. As the algorithm is based on 6.2.2, the problem with flattening the fingertip is happening too (no detected fingertip in sample no. 71).

If a finger is bent only slightly, the fitting line (finger axial vector) crosses a hand contour in a position biased towards the bent side. This can be seen in Fig. 6.11 (b).



(a) Sample no. 29 (b) Sample no. 71

Figure 6.11: Failed cases of 'Finger cut-off algorithm'

Table 6.1: Algorithm calculation times in the experiment

Sample image	**Avg time (in milliseconds)**	**Standard deviation**
1 finger	1.76	0.69
5 fingers	2.63	0.88
Total	1.79	0.81

6.2.3.4 Algorithm performance

Tab. 6.1 shows measured detection times.

Average running speed of the algorithm was 1.79 milliseconds. The number of fingers had an impact on the algorithm performance, but thanks to the overall very fast detection, it does not pose any big issue for usage in HCI scenario.

6.2.4 Circular scan algorithm - all pixels

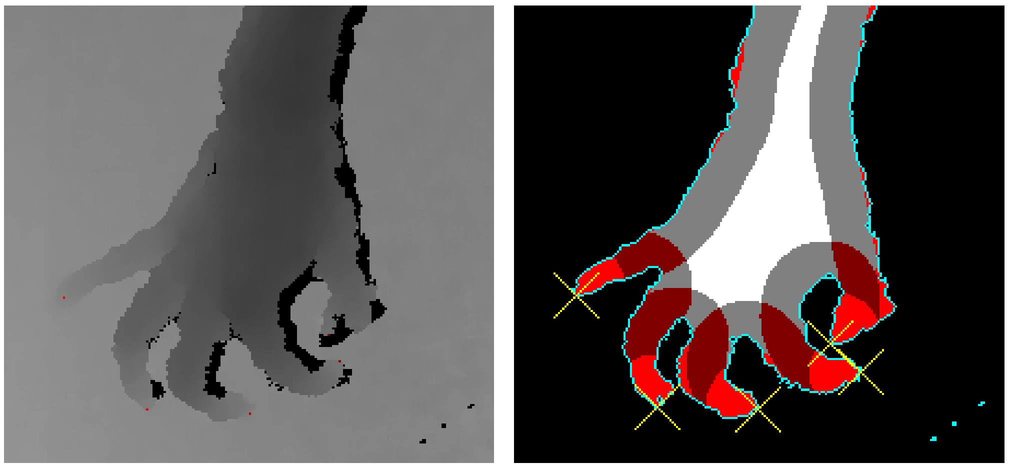
The algorithm described in this chapter analyses finger coordinates using circular scanning [A.4]. It solves the issue of fingertip detection by first classifying hand blob pixels into categories, and later on, to use the classified pixels to analyze the hand blob. In contrast to 6.2.3, it works well both for straight and bent fingers.

The following categories are detected:

- Fingertip (red): These pixels are at the end of each fingers.
- Palm (white): These pixels are inside the palm, far away from an edge of a hand shape.
- Finger segment (brown): These pixels are part of finger between palm and fingertip.

- Hand edge (gray): These pixels are part of the palm, close to the edge of the hand shape.

Color mentioned in brackets matches colors in Fig. 6.12.



(a) Input detph image (b) Detected fingers with categorized pixels

Figure 6.12: Example of algorithm input and output

6.2.4.1 Pixel classification

The pixel classification algorithm works in a way described in the flow chart in Fig. 6.13.

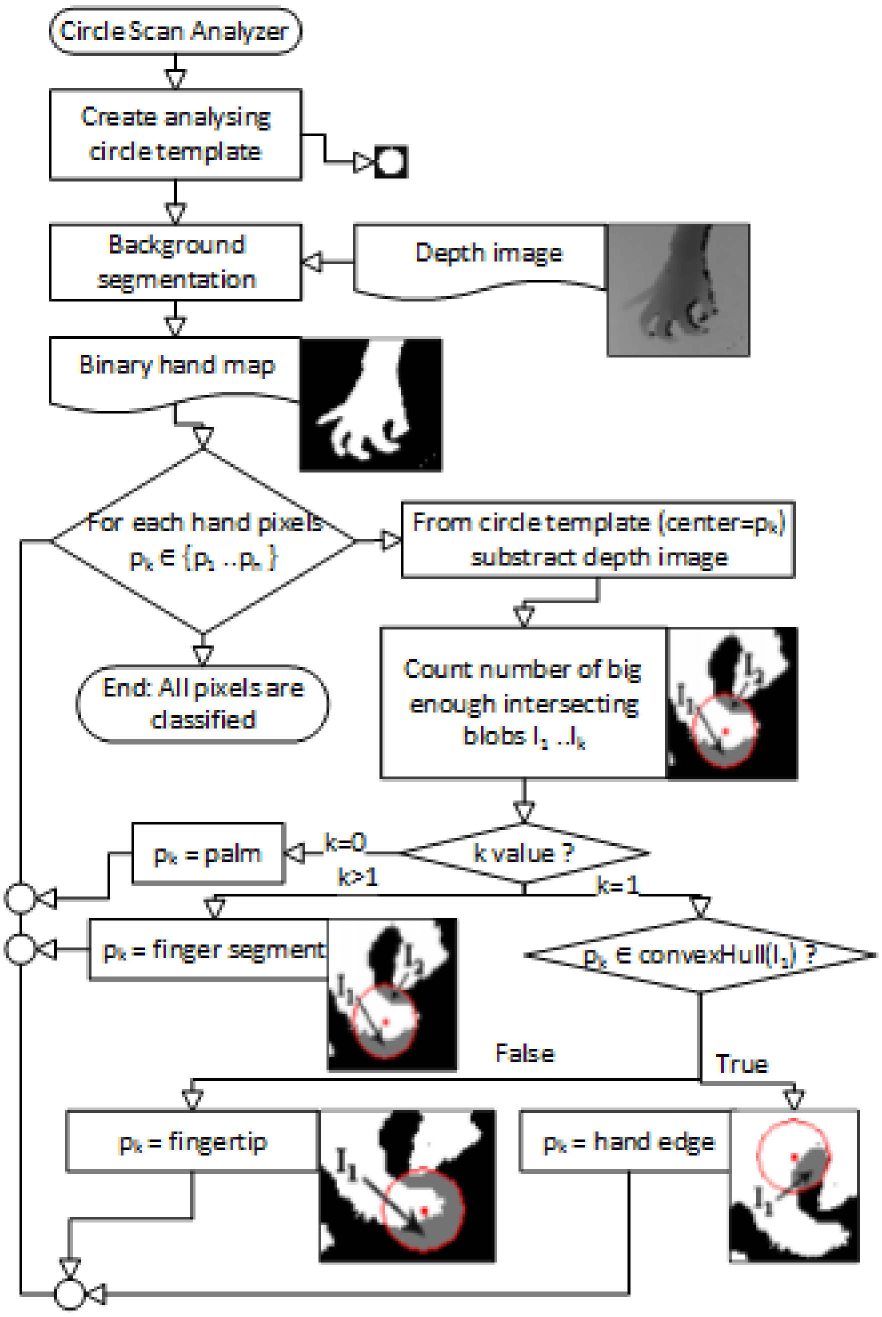

Circle Scan Analyzer
Create analysing circle template
Background segmentation
Depth image
Binary hand map
For each hand pixels $p_k \in \{p_1..p_n\}$
From circle template (center=p_k) substract depth image
End: All pixels are classified
Count number of big enough intersecting blobs $I_1..I_k$
I_1
I_2
k value ?
k=0
k>1
k=1
p_k = palm
p_k = finger segment
$p_k \in$ convexHull(I_1) ?
False
True
p_k = fingertip
p_k = hand edge

Figure 6.13: Pixel classification algorithm

All hand pixels $p_k \in \{p_1..p_n\}$ from a binary hand map are iterated in a sequence, analysing one pixel in each step. A new binary image is constructed, containing k blobs I created by subtracting a binary hand image from a binary image containing a circle of specified diameter d, with a center in the current pixel. Blobs with an area smaller than a given threshold are discarded. Based on the number of intersecting blobs found, the pixel is classified.

All fingertip and finger segment pixels are than sorted into 8-connectivity blobs, where each such blob represents a detected fingertip candidate. Only such fingertip blob candidates that neighbor with at least one finger segment blob, are returned as recognized fingertip blobs. All pixels on the border between these neighboring blobs are stored as *Border*. This step is necessary, as some pixels on small protrusions in the hand shape are incorrectly classified by previous steps as fingertips, even though they belong to a palm.

For each fingertip blob, fingertip coordinates are calculated as a point on a fingertip blob contour that is the most distant from a middle pixel in *Border* sequence of pixels.

The diameter value d depends on an environment and was set to 51 pixels. For the algorithm to work properly, condition 6.2 must be met.

$$d \in (MaxWidth, 2 * MaxWidth) \tag{6.2}$$

It is best to set the value equal to $1.5 * MaxWidth$. $minBlobLength$ was set to $\pi * d$, where $MaxWidth$ is a maximal finger width in the source image.

The algorithm was implemented in .NET, using EmguCV library for computer vision related operations like blob extractions and intersections.

6.2.4.2 Algorithm performance

An average running time of the 'Circular scan algorithm (all pixels)' was 378 ms, compared to 1.79 milliseconds of 'Finger cut-off algorithm'. Though it may be sufficient for static analysis of sample images, for real-time usage in user interface it is not enough. Frequency around 30 Hz (frames per second) is necessary, so that the user is not limited by the slow interface responses.

6.2.4.3 Algorithm failures

In contrast to all previous algorithms, this algorithm detected properly 100% of fingertips in DepthTip sample database.

6.2.5 Circular scan algorithm - contour only

The algorithm described in this chapter combines an idea of two algorithms together.

- Circular scan algorithm - all pixels brings a benefit of high accuracy of detected fingertips both for straight and bent fingers.
- Hand contour polygon simplification algorithm brings a benefit of high processing speed, thanks to an idea to analyze only a hand contour rather than all hand pixels.

The main idea is to process hand pixels in nearly the same way as in Circular scan algorithm - all pixels. But, rather than process all hand pixels, only those that form the hand contour are categorized. This algorithm is one of the main merits of this dissertation. Fig. 6.14 shows the result of this algorithm. DepthTip sample 55 was chosen as an example as it posseses challenging bent finger positions that other algorithms failed to detect properly.

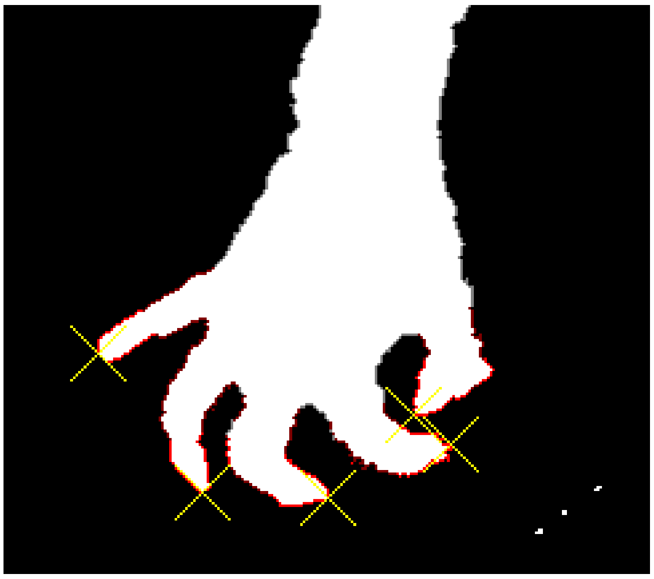
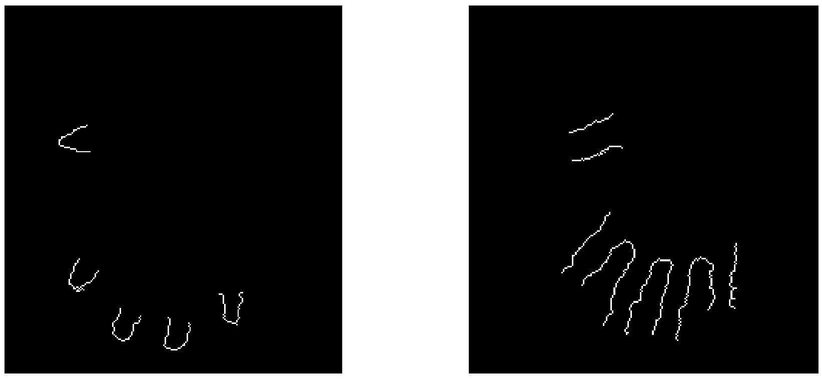


Figure 6.14: Fingertips detected in sample 55

The algorithm works in following steps:

- Categorize all hand contour pixels (into fingertip, palm, finger segment and hand edge categories) in the same way as described in subsection 6.2.4: Circular scan algorithm - all pixels. Fig. 6.15 shows both fingertip and finger segment categorized contour pixels from the DepthTip sample no. 72. The result of the detection can be seen in Fig. 6.16.
- Iterate all detected fingertip sequences in the hand contour. Discard such sequences that:
  - Are not neighbouring to at least one finger segment sequence.
  - Do not meet minimal length of the given threshold τ pixels.
- Calculate fingertip position as a point in the fingertip contour sequence, that is the most distant from the mid point between the start and end point of the sequence. This allows for accurate fingertip detection both for straight and bent fingers.



(a) Fingertip pixels (b) Finger segment pixels

Figure 6.15: Categorized fingertip and finger segment pixels

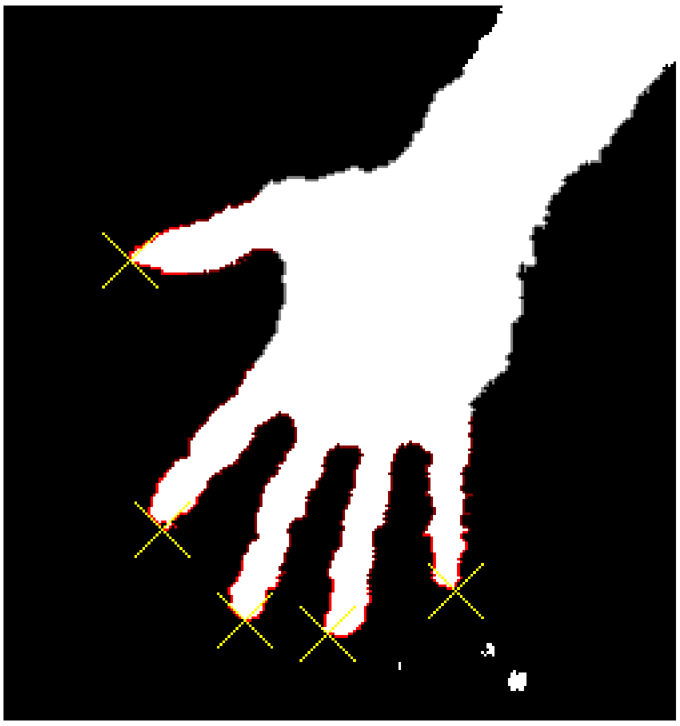


Figure 6.16: Sample 72 detection result

6.2.5.1 Algorithm performance

Average running time of the 'Circular scan algorithm (contour only)' was 21.63 milliseconds, compared to 378 milliseconds of Circular scan algorithm - all pixels. This presents the most desired improvement while the detection accuracy was retained. This algorithm is thus suitable for eal-time usage in a user interface, achieving a processing frequency of 46 Hz (frames per second).

6.2.5.2 Algorithm failures

As a Circular scan algorithm - all pixels, this algorithm detected properly 100% of fingertips in the DepthTip sample database.

6.3 Run-time statistics evaluation of proposed algorithms

This chapter contains comparative results of experiments, where fingertip detection algorithms were tested against the DepthTip dataset [2]. Four proposed algorithms were tested together with our own implementation of 'distance from center of mass' method, as described in section 3.5: Distance from center of mass method. Each algorithm was executed against every DepthTip sample. Distance (in pixels) between the fingertip position annotated in the DepthTip and the actual position calculated by the algorithm was gathered.

6.3.1 Fingertip detection accuracy

Fig. 6.17 shows results of fingertip detection accuracy measurements.

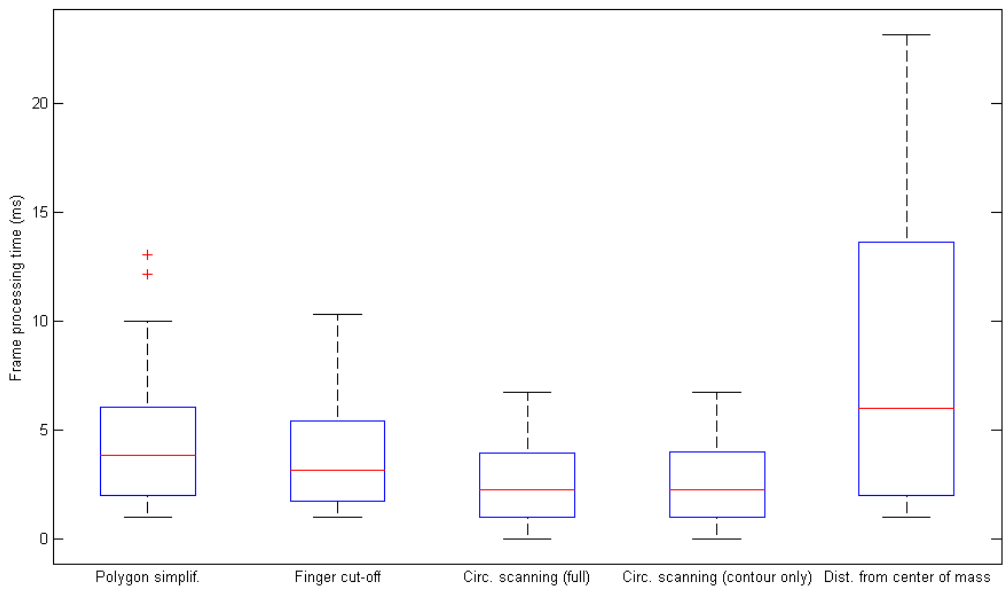


Figure 6.17: Fingertip detection accuracy algorithm comparison

Tab. 6.2 shows numerical values for fingertip detection accuracy statistics. `# Failed detections` column shows the number of samples, where a given algorithm failed to properly detect a fingertip. This happens either, if the annotated fingertip was not detected at

Table 6.2: Fingertip accuracy comparison

Algorithm	Distance			# Failed
	Avg	Min	Max	detections
Polygon simplif.	4.23	1	13.0	11
Finger cut-off	3.78	1	10.3	27
Circ. scanning (full)	2.37	0	6.7	0
Circ. scanning (contour only)	2.44	0	6.7	0
Dist. from center of mass	8.31	0	12	21

all, or if it was detected but its location was miscalculated in a place where no fingertip exists. The reasons for wrong location calculations are mentioned, for example in chapter 6.2.3 when a finger axial vector was detected with wrong orientation. Not to distort accuracy results, failed detections where excluded from the accuracy detection statistics, so only successfully detected fingertips were measured and formed results (in table `Distance` columns `Avg`, `Min` and `Max`).

6.3.2 Performance of the algorithms

Fig. 6.18 shows results of performance, showing time to process a single frame from the DepthTip dataset.

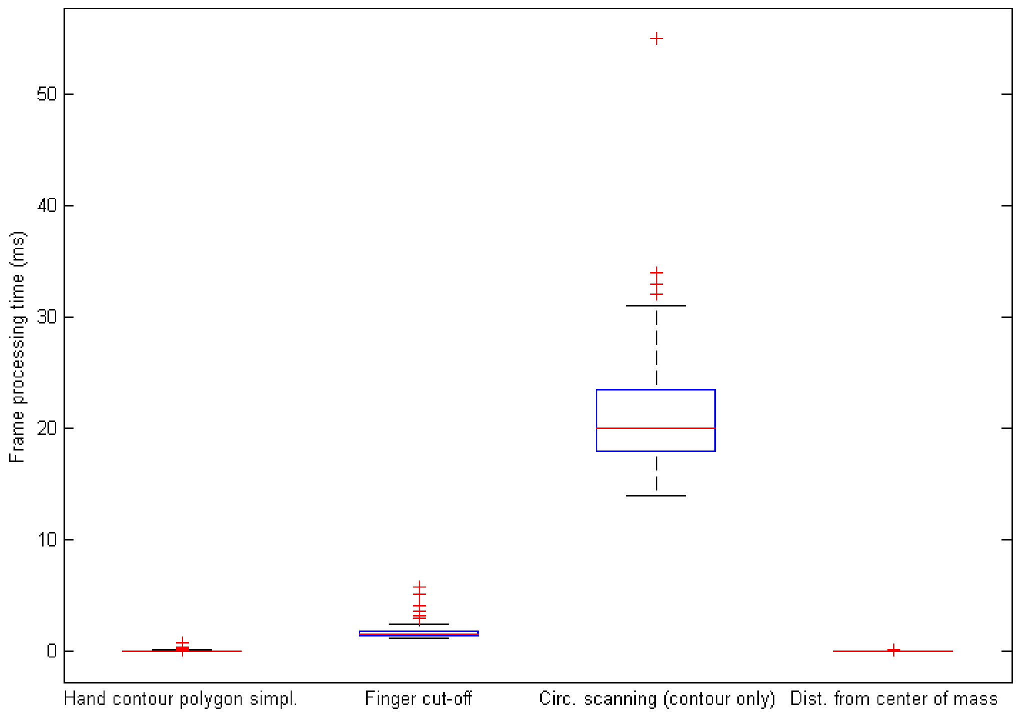


Figure 6.18: Frame analysis time algorithm comparison

Not to lose graph resolution in Fig. 6.18, the 'Circular scan algorithm (all pixels)' was excluded from comparison as it, by far, exceeds the processing times of other algorithms. Its graph can be found separately in Fig. 6.19

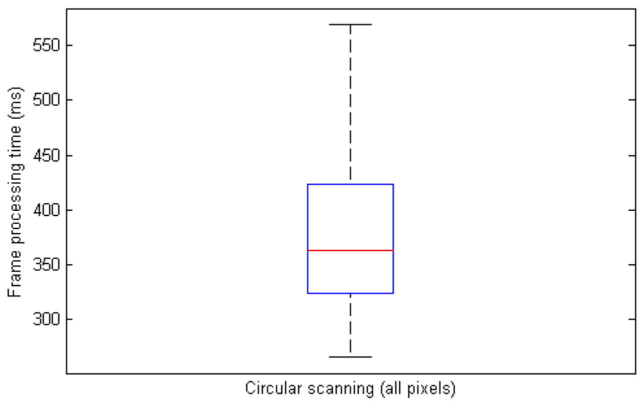


Figure 6.19: Frame analysis time of Circular scan (all pixels) algorithm

All results can be found in folder 'results' in [2], where a nested folder containing related results exists for each of the tested algorithms. In the 'results' folder, fingertip locations are added as yellow crosses into the image of the original depth frame. More information on how to access these results can be found in Appendix A - DepthTip evaluation dataset.

6.4 Summary of fingertip detection algorithms evaluation

All proposed algorithms, and the 'Distance center of the mass' algorithm from literature, were tested in an experiment. Excluding the 'Circular scan algorithm (all pixels)', all have sufficient average processing speed. It is lower than the critical threshold of 30 ms, allowing for real-time user interactions.

For each sample image in the DepthTip, the distance between actual fingertip coordinates (annotated in dataset) and calculated coordinates (output of the fingertip detection analysis) were obtained. Regarding the fingertip detection accuracy, most exact algorithms are circular scan based algorithms, with average fingertip distance between 2.37 and 2.44 pixels. Slightly less accurate is the 'Finger cut-off algorithm', with 3.78 pixels. Accuracy is not good enough for 'Hand contour polygon simplification algorithm' (4.23 pixels) and is worst for the distance from the center of mass algorithm (4.76 pixels). An example of the output for the last mentioned algorithm can be found in Fig. 6.20. Both circular scan algorithms ('all pixels' and 'contour only' variants) successfully detected all 73 DepthTip images; detecting 95 fingertips. The older 'Finger cut-off algorithm' [15] detected properly only 69 fingertips. From the failed 27 detections, 18 were detected with inaccurate coordinates and 8 were not detected at all. Results of both algorithms can be found in respective DepthTip folders called 'circleScanResult' and 'cutOffResult'. The 'Distance from the center of mass' algorithm failed in 21 cases (sample no. 8, 9, 10, 16, 17, 18, 19, 20, 27, 28, 29, 30, 34, 37, 38, 39, 40, 47, 48, 49 and 50). This happened when fingers being detected were bent, which led to locating different points in a hand contour as a fingertip. For this algorithm, only accuracy of straight fingertips was included in statistics (failed samples were not considered).

The 'Circular scan algorithm (contour only)' meets both performance and accuracy requirements and thus can be used as a basis for HCI user interfaces. Unlike contour analysis based algorithms, it works perfectly also for bent fingers. It is apparent that the 'Circular scan algorithm (contour only)' not only properly detects all 73 images in real time, but also does it with higher accuracy.

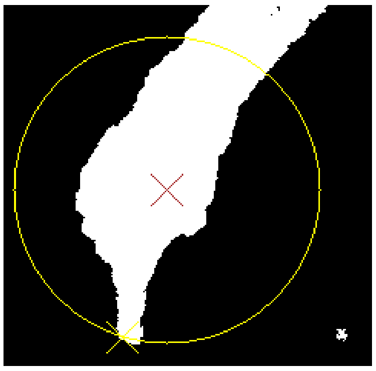


Figure 6.20: Result of distance from the center of mass (sample 12)

6.5 User productivity experiment

Once the winning algorithm for a fingertip detection was known (Circular Scan Algorithm - contour only), the TouchTable user interface was modified to use it. The experiment, originally performed with the former version of an algorithm (as described in chapter 5.2.6), was performed again. This allowed us to evaluate the new algorithm in comparison with the original fingertip detection algorithm. This experiment serves as an additional measure to support the hypothesis that increased fingertip detection accuracy will lead to increased user productivity in an interaction scenario. This chapter contains results from such an experiment. The task given to users remained the same - to draw a simple house, using TouchTable and Google sketchup CAD modelling software.

6.5.1 Experiment setup

As in 5.2.6, 5 users were asked (referred to as U1 to U5) to draw houses 10 times using the TouchTable with 'Circular scan algorithm (contour only)'. U1 was an experienced TouchTable operator, U2 was an experienced CAD software user, and users U3 to U5 were regular computer users with no experience neither with the TouchTable nor with CAD software.

Clicking operation was not mapped to touching the table using the index finger as before, but instead to hiding a thumb of the right hand back to the palm and revealing (extending) it again. Thanks to it, undesirable shifts of the mouse cursor attached to the index fingertip position was avoided.

All other conditions were same as in the original experiment.

6.5.2 Experiment results

Table 6.3 shows the results of the experiment. Figure 6.21 shows achieved times to finish the drawing of the sample house using the TouchTable with the new algorithm. Each of the 5 users performed 10 trials in the experiment.

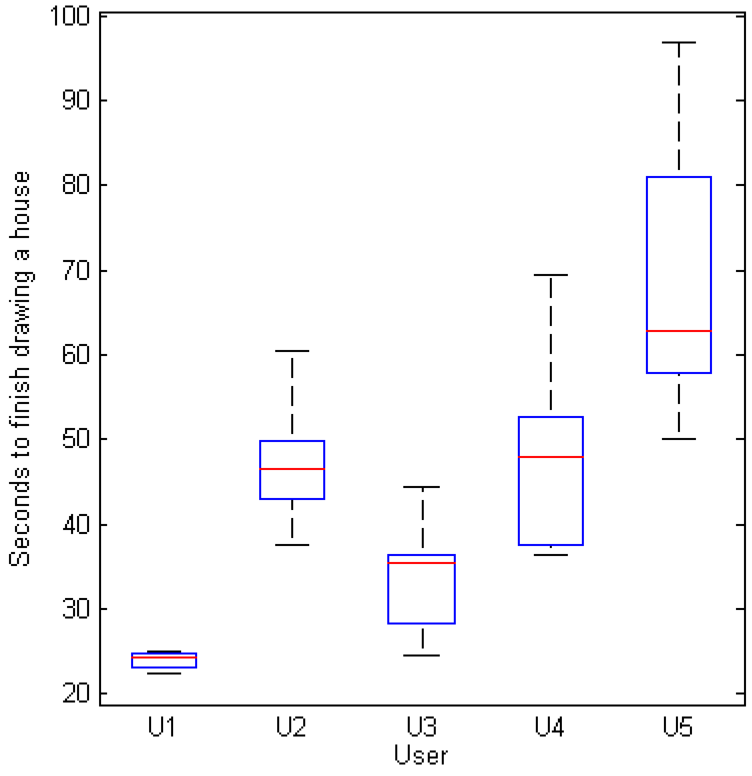


Figure 6.21: Time to finish house - TouchTable with the 'Circular scan algorithm (contour only)'

Measured Values	**Time to Finish a Drawing of Sample House (in seconds)**	
	Hand contour polygon simpl. alg.	**Circular scan alg.**
Average	56.24	43.81
Standard Deviation	26.72	17.69
Min	22.80	22.51
Max	141.15	96.79

Table 6.3: TouchTable interface with low accuracy fingertip algorithm compared to TouchTable with high accuracy fingertip algorithm

The experiment revealed that, with the new more accurate fingertip detection algorithm, together with different mapping of the clicking action, the average time to draw the house using TouchTable was improved by 12.43 seconds (22.1 %). The minimum measured time nearly did not change (from 22.8 to 22.51 seconds). The maximum time improved significantly, from 141.15 to 97.79 seconds.

Lets compare these results to the original experiment results with the regular user interface of mouse and keyboard. An average time to finish the house (43.81) is worse by 16.07 seconds (57.93 %), compared to 27.74 for mouse and keyboard. The minimal time 22.51 is worse only by 5.35 seconds (31.18 %).

By analyzing this data and observing user behavior, the following conclusions were made:

- Improved accuracy of the fingertip detection algorithm indeed resulted in significantly higher productivity, as average time to finish the sample house was lower by 12.43 seconds compared to the original low-accuracy algorithm.
- In spite of the increased productivity, thanks to the new fingertip detection algorithm, using the TouchTable interface was still significantly slower (minimal measured time by 22.51 %) than using a regular (mouse and keyboard) interface.
- A significant learning curve is related to the TouchTable interface for proper pronunciation of commands, for finger movements and hand coordination. It was difficult for users to coordinate movement of their hand with an extended index finger and thumb to achieve mouse clicks.
- During the drawing, users had to curl and hide three fingers in a palm (middle, ring and little finger). Users were thus reporting an unpleasant strain in the forehand. Some of them suggested that it would be easier for them to hold a round object like a small ball in their palm so that these fingers, that are not used for any actions, could rest on the ball instead of being pushed by muscles under the palm. Part of the ball object could also be a fixed extrusion that could be used to detect a cursor position instead of an index finger, and the index finger could rest also on the ball, improving the ergonomics.

6.5.3 Slow speech recognition engine

Slower TouchTable times are caused also by a necessity to input dimensions using a voice, where pronouncing the dimensions leads to unnecessary pauses, compared to direct input of dimensions using a keyboard. Additionally, it takes noticeable time for a voice recognition engine to provide a recognized dimension after words were spoken. To finish the drawing, the following commands needed to be spoken aloud: Five, Three, Two point five, Two point five, Three, Move, Three, Pulling and Five. Recognized numbers were entered by the TouchTable Sketchup plugin as drawing dimensions. Move and Pulling commands changed the active drawing tool to Move or Pull respectively. It was measured that the time to pronounce this sequence alone (without waiting for a speech recognition engine to recognize these words) is quite fast, as it takes only 4.4 seconds in average. However, if speech analysis time is taken also into account, and both pronouncing the word and subsequent recognition of it is measured, the sequence takes 16.55 in average. This duration must be part of the whole time to finish drawing the house in the TouchTable experiment. The regular interface has an advantage that dimensions are entered directly by the keyboard. Hitting the keys is much faster than pronouncing and recognizing the spoken commands.

6.5.4 Failed attempt to decrease user interface complexity

As coordination of thumb and hand movement proved to be difficult, and required a lot of user focus, an attempt has been made to decrease the interface complexity. Instead of using the thumb to perform clicks, voice command *Click* was used in a side experiment. Even though operating in this new user interface mapping was much easier for users, it resulted in increased time needed to draw. This is because *Click* voice command is much slower compared to fast thumb muscle action. For this reason, this user interface variant was abandoned as not being productive enough.

6.6 Mathematical model of bent fingers

Given a mathematical model of a hand, more advanced gesture based user interfaces can be built. For example, the bent angle of the finger can be used to squeeze a virtual object by a given degree. Fingertip detection algorithms usually provide only information about the fingertip coordinates. Some algorithms, e.g., 'Finger cut-off algorithm' described in chapter 6.2.3, provide also an axial vector of the fingers if the fingers are straight and not bent. To build a proper mathematical model of a hand, the position of the knuckles also needs to be known for bent fingers. This chapter describes an approach solving this problem, that uses line approximation methods [A.3]. Pixels that belong to a finger are first collected (using circular scanner as described in chapter 6.2.4) and subsequently two fitting lines are found that ideally fit through the pixels.

6.6.1 Prerequisits for a detection of bent finger's knuckle

Based on the collected finger pixels, lying on the surface of the finger, a suitable model for a finger's characterization needs to be formed. It is based on the transformation of coordinates and a regression model. The regression model gives us a non-smooth function with a change point. The main goal is to find the change point of the approximation function. Coordinates of the change point of this function determine the position of the knuckle.

6.6.2 Change point estimation

For an estimation of the change point value, augmented with its variance, it is necessary to find or design a suitable regression model. An algorithm is based on approximation of two straight lines and is inspired by [12], non-smooth linear function approximation, change point estimation, and transformation of coordinates. The main parts of the algorithm transform measured coordinates to a new coordinate system in such a way that new coordinates can be fitted by a function.

6.6.3 Algorithm details

Algorithm calculates points in several steps:

1. Fingertip and fingersegment classification,
2. two straight lines approximation,
3. finding an intersection of straight lines,
4. transforming coordinates of finger points into a new coordinate system,
5. non-smooth approximation with a change point,
6. transformation of non-smooth function to the old system of coordinates.

6.6.3.1 Fingertip and fingersegment classification

Input of the algorithm is a bitmap with classified pixels, where each pixel has one of "fingertip" or "fingersegment" classes. In the sample images (Figures 2 - 7), the circle pixels

$$T_i, i = 1, \dots, t$$

with coordinates $(T_{x,i}, T_{y,i})_U$ represent "fingertip" and the square pixels

$$S_j, j = 1, \dots, s$$

with coordinates $(S_{x,j}, S_{y,j})_U$ represent "fingersegment". Let us define

$$F_k = \{\{T_i\} \cup \{S_j\}\}$$

and

$$n = s + t$$

for future use. Such classification of pixels help us with the finger approximation.

6.6.3.2 Two straight lines approximation

By the least Squares Method, fingertip pixels are fit to get the following estimator of line

$$f_1 : y = \alpha_0 + \alpha_1 x.$$

Further finger segment pixels are fit to get following estimator of line

$$f_2 : y = \beta_0 + \beta_1 x.$$

6.6.3.3 Finding an intersection of straight lines

Let us define point $P = [P_1, P_2]_U$ as an intersection of lines f_1 and f_2:

$$\begin{aligned} P_1 &= \frac{(\alpha_0 - \beta_0)}{(\beta_1 - \alpha_1)}, \\ P_2 &= \beta_0 + \frac{(\alpha_0 - \beta_0)}{(\beta_1 - \alpha_1)}\beta_1 = \frac{\beta_1\alpha_0 - \beta_0\alpha_1}{(\beta_1 - \alpha_1)}. \end{aligned} \tag{6.3}$$

6.6.3.4 Transforming coordinates of finger points into a new coordinate system

Let us define line g:

$$y = \gamma_0 + \gamma_1 x = (P_2 - \gamma_1 P_1) + \gamma_1 x$$

as a bisector of lines f_1 and f_2. The directions of lines f_1, f_2, g are

$$\begin{aligned} \alpha &= \arctan((P_2 - \alpha_0)/P_1) = \arctan \alpha_1, \\ \beta &= \arctan((P_2 - \beta_0)/P_1) = \arctan \beta_1, \\ \gamma &= \arctan \frac{\sin\alpha + \sin\beta}{\cos\alpha + \cos\beta} = \arctan \gamma_1 \\ &= \arctan\left(\frac{\alpha_1\sqrt{1+\beta_1^2} + \beta_1\sqrt{1+\alpha_1^2}}{\sqrt{1+\beta_1^2} + \sqrt{1+\alpha_1^2}}\right). \end{aligned}$$

So

$$\begin{aligned} g : y &= \frac{\beta_0\sqrt{1+\alpha_1^2} + \alpha_0\sqrt{1+\beta_1^2}}{\sqrt{1+\beta_1^2} + \sqrt{1+\alpha_1^2}} + \\ &+ \frac{\alpha_1\sqrt{1+\beta_1^2} + \beta_1\sqrt{1+\alpha_1^2}}{\sqrt{1+\beta_1^2} + \sqrt{1+\alpha_1^2}} x \end{aligned} \tag{6.4}$$

Let

$$B = P - 2(\overline{F} - P),$$

where coordinates of the point $\overline{F}$ are arithmetic means of F_k coordinates.

Let us define line h as perpendicular to a line g and let $B \in h$. Then

$$h(x) = \frac{h_0}{\gamma_1} - \frac{1}{\gamma_1}x = \frac{B_1 + \gamma_1 B_2}{\gamma_1} - \frac{1}{\gamma_1}x.$$

Projection of the point $F_k = [F_{x,k}, F_{y,k}]$ into the line h is

$$\begin{aligned} \mathrm{proj}_h(F_k) &= [F^*_{x,k}, F^*_{y,k}] \\ F^*_{x,k} &= \frac{\gamma_1^2 x - \gamma_1 y + B_1 + \gamma_1 B_2}{1+\gamma_1^2}, \\ F^*_{y,k} &= \frac{-\gamma_1 x + \gamma_1 B_1 + \gamma_1^2 B_2 + y}{1+\gamma_1^2}. \end{aligned} \tag{6.5}$$

Let us move the line g to the new origin:

$$S = \mathrm{proj}_h(F_m) = (s_1, s_2)_U = (0,0)_V,$$

where

$$m = \mathrm{argmax}|AF_k|$$

with

$$A = \mathrm{proj}_h(P).$$

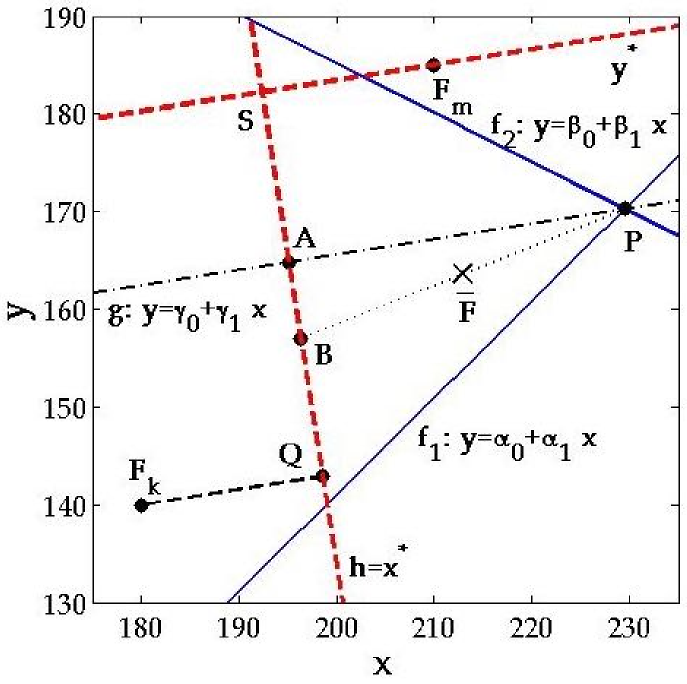
Let the new axis x^* be the line h. Axis y^* is given by a formula

$$y = \omega_0 + \omega_1 x = -\gamma_1 F_{m,1} + F_{m,2} + \gamma_1 x. \tag{6.6}$$

In the new coordinate system V the point $(F_{x,k}, F_{y,k})_U$ has coordinates

$$(F^*_{x,k}, F^*_{y,k})_V = (||\mathrm{proj}_h(F_k) - S||, ||\mathrm{proj}_h(F_k) - F_k||). \tag{6.7}$$

Whole process is described in Fig. 6.22.



190
180
170
160
150
140
130
180
190
200
210
220
230
x
y
y^*
F_m
f_2: $y=\beta_0+\beta_1 x$
S
A
P
$\bar{F}$
g: $y=\gamma_0+\gamma_1 x$
B
Q
F_k
f_1: $y=\alpha_0+\alpha_1 x$
$h=x^*$

Figure 6.22: Process of estimation

6.6.3.5 Non-smooth approximation with a change point

Let us assume that dependency between coordinates is given by two straight lines, and that the knuckle is given by their intersection, respectively by their change point. So the measured coordinates can be approximated by just one non-smooth curve of absolute value function:

$$Y^* \quad = \quad \phi(\boldsymbol{\theta}, x^*) = \theta_1 x^* + \theta_2 + \theta_3 |x^* - \theta_4|. \tag{6.8}$$

Note that parameter θ_4 has a special meaning, it determinates the change point.

Now it is necessary to estimate the values of unknown parameters $\theta_1, \theta_2, \theta_3, \theta_4$ occuring in nonlinear function ϕ.

Let

$$\begin{aligned}
\mathbf{x}^* &= (F^*_{x,1}, \ldots, F^*_{x,n})', \\
\mathbf{Y}^* &= (F^*_{y,1}, \ldots, F^*_{y,n})', \\
\mathbf{Y}^*_0 &= \phi(\boldsymbol{\theta}^0, \mathbf{x}^*).
\end{aligned}$$

The standard procedure of calculation, described in [23], is to linearize the nonlinear function (as seen in Eq. 6.8) at the approximate values $\boldsymbol{\theta}^0$, i.e.:

$$E(\mathbf{Y}^* - \mathbf{Y}^*_0) = \mathbf{X}\boldsymbol{\theta},$$

$$\phi(\boldsymbol{\theta}^0, \mathbf{x}^*) + \mathbf{X}\delta\boldsymbol{\theta} = \mathbf{0}.$$

Now appropriate initial solution can be derived:

$$\boldsymbol{\theta}^0 = (1/2, P_2, -1/2, P_1)'.$$

In [12] there are written all steps of estimation for function (Eq. 6.8).

If $\mathbf{Y}^*$ is normally distributed with mean value equal to $\mathbf{X}\boldsymbol{\theta}$ and with covariance matrix $\sigma^2\mathbf{I}$, then estimator of an unknown vector parameter $\widehat{\boldsymbol{\theta}} = \boldsymbol{\theta}^0 + \widehat{\delta\boldsymbol{\theta}}$ is given by formula:

$$\widehat{\delta\boldsymbol{\theta}} = (\mathbf{X}\mathbf{X}')^{-1}\mathbf{X}'(\mathbf{Y}^* - \mathbf{Y}_0^*), \tag{6.9}$$

where k-th row of matrix $\mathbf{X}$ is

$$\mathbf{X}_{k,.} = \left(\frac{\partial\phi}{\theta_1}(\boldsymbol{\theta}^0, x_k^*), \frac{\partial\phi}{\theta_2}(\boldsymbol{\theta}^0, x_k^*), \frac{\partial\phi}{\theta_3}(\boldsymbol{\theta}^0, x_k^*), \frac{\partial\phi}{\theta_4}(\boldsymbol{\theta}^0, x_k^*) \right).$$

The covariance matrix of $\widehat{\boldsymbol{\theta}}$ is

$$\mathrm{Var}(\widehat{\boldsymbol{\theta}}) = \sigma^2(\mathbf{X}'\mathbf{X})^{-1}. \tag{6.10}$$

The unbiased estimator of σ^2 is

$$\hat{\sigma}^2 = \frac{(\mathbf{Y}^* - \mathbf{X}\widehat{\boldsymbol{\beta}})'(\mathbf{Y}^* - \mathbf{X}\widehat{\boldsymbol{\beta}})}{n-2}. \tag{6.11}$$

An extensive coverage of linearization method and parameter estimation models can be found in books [23] and [27].

6.6.3.6 Transformation of non-smooth function to the old system of coordinates

The point $(x^*, \phi(x^*))_V$ can now be transformed to the point $\xi = (\xi_1, \xi_2)_U$ in the old system of coordinates U by following formula:

$$\begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix}_U = \begin{pmatrix} \cos\psi, & -\sin\psi \\ \sin\psi, & \cos\psi \end{pmatrix} \begin{pmatrix} x^* \\ \phi(x^*) \end{pmatrix}_V + \\ + \begin{pmatrix} s_1 \\ s_2 \end{pmatrix}_U . \tag{6.12}$$

6.6.4 Experiments

A finger in a 3D coordinate system was observed by the Kinect sensor. The measured values used for the experiment come from DepthTip data set and are available online [2].

To illustrate the whole detection and confirmation process, lets study the results of the proposed algorithm for several measurements. One important reason for such an experiment is to check numerical correctness of the results. When testing the quality of the method, the algorithm successfully detected the knuckle in 47 of 50 experiments.

Figures 6.23, 6.24, 6.25, 6.26, 6.27 illustrate examples with successful resulting approximation. Figure 6.28 demonstrates unsuccessful approximation. A poor quality estimator in this case can be caused by numerical nonstability. In the case of a non-linear model, a linearization can be a source of differences in the results. Also, when classification of fingertip and finger segment points is not correct, our procedure cannot be considered to be optimal. An example of such a situation in experiment No. 50 is depicted in Fig. 6.28.

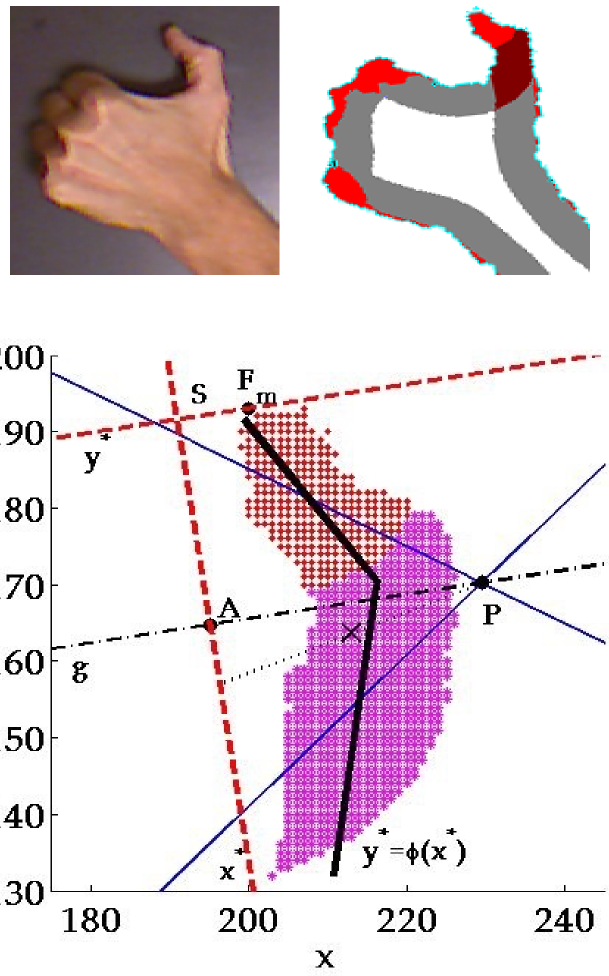


Figure 6.23: Scan, segmentation and approximation of finger: experiment No. 4

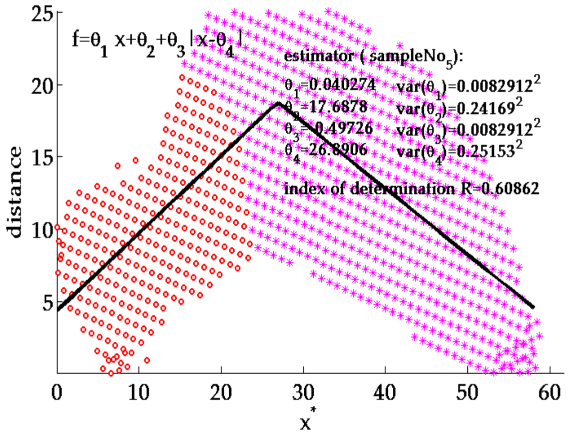


Figure 6.24: Scan, segmentation and approximation of finger : experiment No. 5

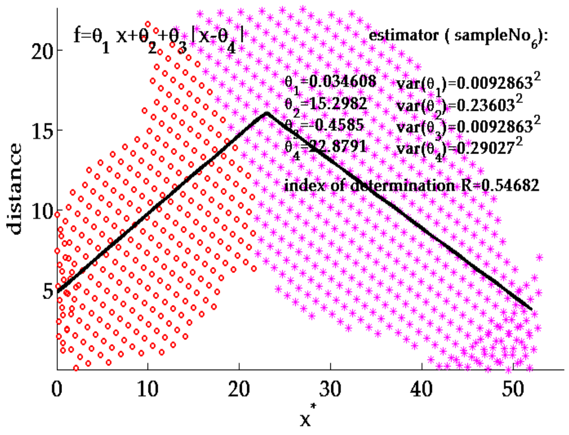


Figure 6.25: Scan, segmentation and approximation of finger : experiment No. 6

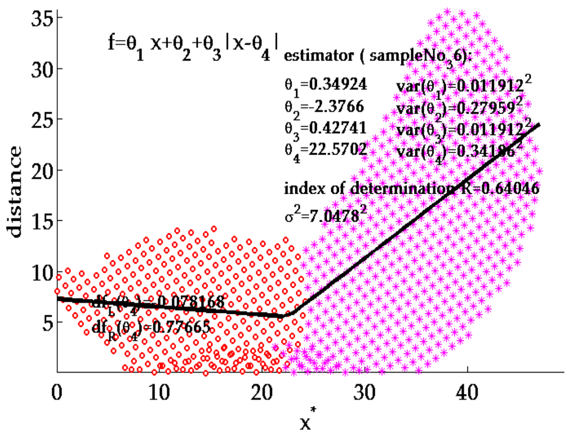


Figure 6.26: Scan, segmentation and approximation of finger : experiment No. 36

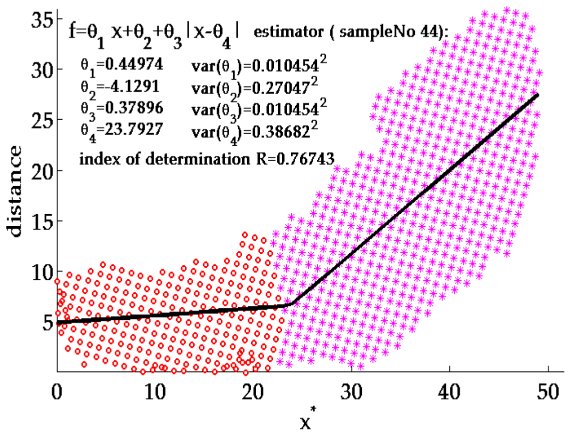


Figure 6.27: Scan, segmentation and approximation of finger : experiment No. 44

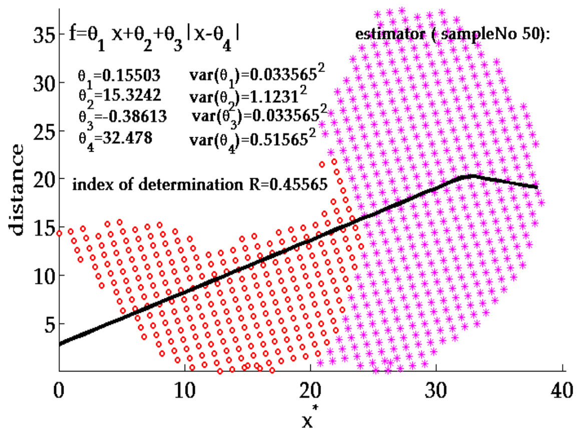


Figure 6.28: Scan, segmentation and approximation of finger : experiment No. 50

6.6.5 Proposed mathematical model summary

The mathematical model able to find knuckle coordinates by finding an intersection of two straight lines in 2D space was created. This is achieved thanks to regression models, grounded in an explicitly defined approximation function of fingers. For our input data with a given classification of points, the presented solution of a finger's approximation is relatively simple and simultaneously approximately optimal.

Chapter 7

Out of the box thinking - new UI paradigms

Though the main focus of this work was touch based user interface, where a user is resting his hands on a working surface, possibilities for novel user interface paradigms are endless. As stated in chapter 2.2.3, multi-modality is a key factor.

This chapter gives an example of such out-of-the-box thinking. It presents an early concept of a user interface component that is designed to help computer users to intuitively organise their digital storage, so that they can easily locate their previously saved documents. Instead of using a traditional hierarchical folder tree, a user is associating his documents with a pair of symbols, organized into three dimensional working space. Positions of user's hands are captured to determine the pair of symbols, being detected either using a depth sensor or RGB camera.

7.1 History insight into how we access digital content

In 1968, Doug Engelbart made his famous demonstration, later retrospectively called "The Mother of All Demos" [7]. The live demonstration featured an introduction of almost all the fundamental elements of modern personal computing: from multiple windows, through hypertext to support for collaborative work. One of the novel features presented was hierarchical tree that user can create and manage to organize pieces of information.

Today, more than forty years later, the way how we are organising our personal data has not changed much. We are still using folder trees to store our documents and our applications of daily use have buttons arranged into hierarchically arranged menu items. Even if we are trying to do our best to keep such hierarchy well organized, we are struggling to find our documents after some time, reverting to fulltext searches.

7.2 3D Associative Symbol Array as Component of User Interface

3D Associative Symbol Array is a new theoretical paradigm for user interface component. It is designed to allow users to access digital information directly, instead of descending into a hierarchical tree, by utilizing the brain ability to associate symbols to each other.

In chapter 7.2.1, concept of 3-dimensional associative array is defined and four basic types of arrays are proposed. Section 7.3 describes how this concept can be used by users to build an association and later access data and also describes prototype of the component.

7.2.1 Concept of associative array

Associative array is a theoretical concept of user interface component that enables users to utilize their associative brain skills to gain fast access to previously stored data or execute previously associated computer commands. Instead of traditional keyboard, where user is manipulating only with letters of English alphabet (keys), in this approach user is using symbol board and "pressing" combinations of two symbols.

To better understand the concept, working with traditional and proposed user interface component can be compared to behaviour of different data structures used in computing. User traditionally using keyboard and mouse to access data through hierarchy of folders is similar to multi-dimensional tree search algorithm. The disadvantage of accessing data through folder tree is that it requires to select multiple targets with an input device like mouse or keyboard to finally reach it, based on depth of the tree. User using proposed associative array component behaves in a similar way as algorithm directly obtaining data through key in a hash map, avoiding tree traversal.

7.2.1.1 Symbol board

Symbol board is a multi-modal representation of the associative array, presented to a user as a part of user interface. It consists of a virtual 3D space, divided to left hand and right hand operational cubes. Each cube represents a 3D 3x3x3 matrix space, containing 27 symbol cells. Each cell contains symbol that can have both visual and audible representation and has x, y and z coordinates, defining particular symbol cell:

$$x \in \{0,1,2\}; y \in \{0,1,2\}; z \ \in \{0,1,2\} \tag{7.1}$$

User selects symbols in symbol cells by placing both hands into appropriate positions in 3D virtual space, as described in detail in chapter 7.3.2. This allows for total number of 729 different associations:

$$N = (3^3)^2 \tag{7.2}$$

While coordinates x and y are determined by a position of the hand on the table in the horizontal plane, z coordinate is changing with the height of the hand above the table. If

$z = 0$, hand is resting on the desk. In such situation, contact with the desk helps user to instantly place his hand to correct height without prior training.

7.2.1.2 Associative array types

Associative arrays can be of different types, based on the category of symbols they contain. Symbols are chosen in a way so that they have easy to remember features like color, shape or other significant properties.

So that user can easily find a position in an array for his hands if he can remember the symbols, placement of symbols in symbol board adheres to predefined logical principles. In this way, brain can estimate the position of symbol cell ahead of opening the symbol board, even if appropriate visual and audible symbols are not yet presented. Such a design is in compliance with a claim, that if we can execute some action in short time mentally, also physical execution of the action is then fast [32]. Each of the chapters dealing with different associative array types contains also a description of these logical principles.

Proposed associative array types are:

- Color array
- Animal array
- Audible sounds array
- Items array

7.2.1.3 Color array

Color array is designed in a way so that colors are as distinct from one another as possible. Appearance of color array symbol board component is in Fig. 7.1. Color placement follows simple logic:

- As x index increases, intensity of red channel increases.
- As y index increases, intensity of green channel increases.
- As z index increases, intensity of blue channel increases.

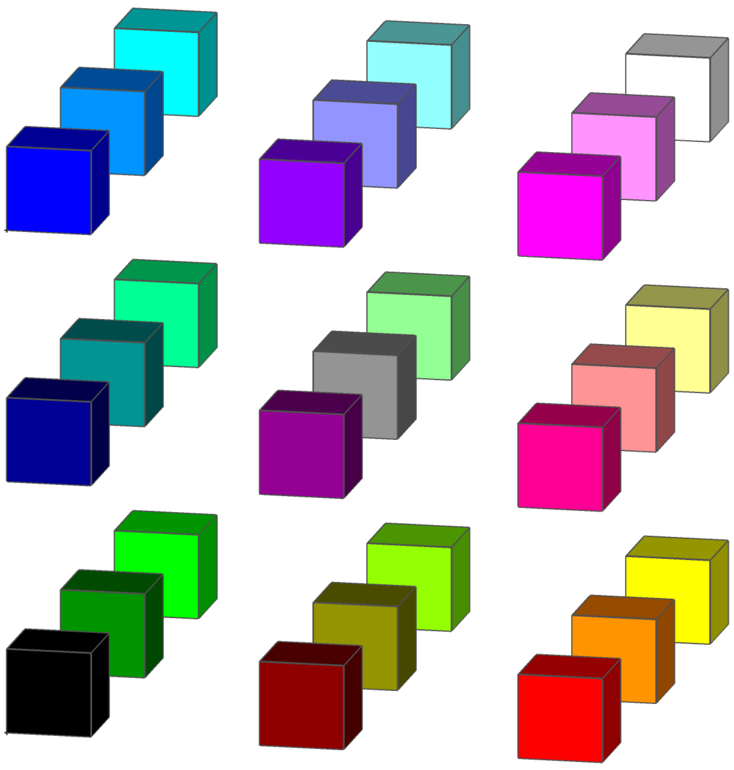


Figure 7.1: Associative color array

Intensity I of red R, green G and blue B channels of array cells is calculated using cell indices x, y and z:

$$I \in \{0, 0.5, 1\} \tag{7.3}$$

$$R = x * 0.5; G = y * 0.5; B = z * 0.5 \tag{7.4}$$

7.2.1.4 Animal array

Animals are easy to associate symbols, as they represent different life styles, live in different environments, vary in size, behaviour and shape. Example of proposed appearance of animal array in user interface is in Fig. 7.2. Placement of animal symbols follows simple logic:

- x array index = size of an animal: 0 means smallest, 2 means biggest.
- y array index = position of an animal in the evolution tree: 0 means high intelligence (typically mammals), 1 means not so intelligent animals but still living, 2 means low intelligent, prehistoric animals.
- z array index = living environment of an animal: 0 means earth, 1 means water and 2 means air.

For example a whale is large intelligent mammal, living in sea, so its coordinates in animal array are $2, 2, 1$.

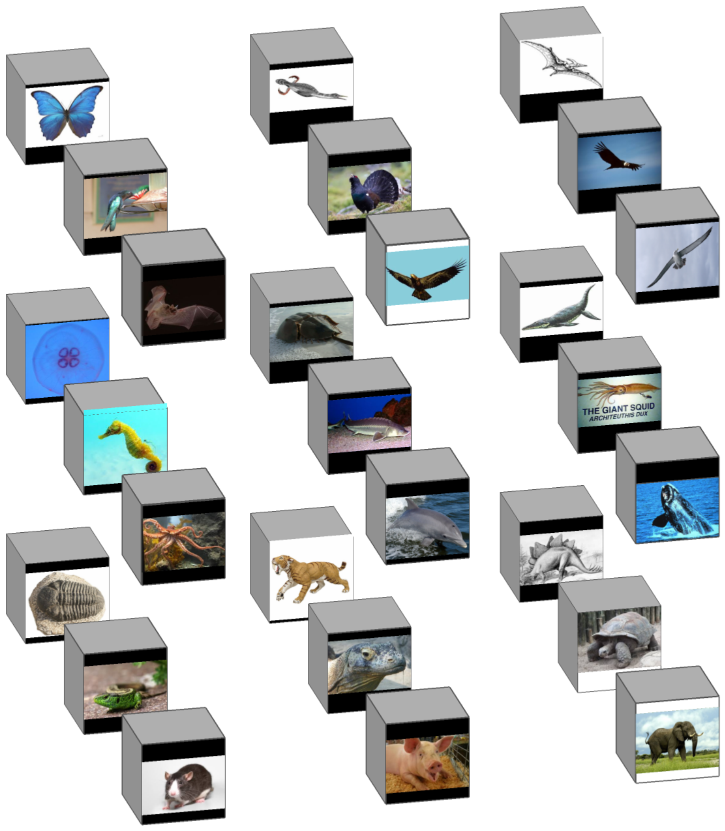


Figure 7.2: Associative animal array

7.2.1.5 Items array

Array consists of symbols of items manufactured by humans. Placement of items inside the array holds the same logic as of Animals array, so that user can remember only one set of placement rules both for animal and items associative arrays. Example of proposed symbols in items array is in Tab. 7.1. Placement logic for items array is following:

- x array index = size of the item: 0 means smallest, 2 means huge.
- y array index = position of item in technology tree: 0 means smart, sophisticated item, 2 means crude, simple item.
- z array index = operating environment of an item: 0 means earth, 1 means water and 2 means air.

For example a symbol of Crane will be associated with position with coordinates $0, 0, 0$. Visual appearance of items array can be same as for animals array, using icons of items placed on cubes in 3D space.

7.2.1.6 Audible sounds array

In contrary to previous associative array types, that can be visually displayed in user interface tool, this array contains audible sounds that are played only when user with his hand activates (enters) symbol cell in 3D array space. This means that user can hear same sound both when storing the document and when recovering it from a data storage. He can use his auditory memory, that can be individually more powerful than visual or motor memory. In this way he can gradually tune to the correct sound while searching for the document and approaching the correct symbol cell. Placement of sounds inside the array follows this logic:

- x array index = the tone of musical instrument that is playing: 0 means C, 1 means E, 2 means G.
- y array index = selection of musical instrument: 0 means violen (most pure sound), 1 means trumpet, 2 means percussion instruments (least pure sound).
- z array index = octave, 0 means C1, 1 means C2 and 2 means C3.

For example a sound of violin playing E2 will be associated with position with coordinates $2, 1, 2$.

7.3 Usage of associative array

7.3.1 Information association step

When user wants to store a new association to an existing document (e.g., some file or an email) so that he can later easily find it, a user interaction follows these steps:

Table 7.1: Proposed symbols in items associative array

	x=0	**x=1**	**x=2**
y=2, z=2	Arrow	Baloon	Airship
y=1, z=2	Bumerang	Ultralight	Concord
y=0, z=2	Qudrocopter	Fighter	Spaceship
y=2, z=1	Paper boat	Catamaran	Rae boat
y=1, z=1	Kayak	Fishing boat	Galley
y=0, z=1	Water scooter	Modern Yacht	Battleship
y=2, z=0	Hammer	Sledge	Stonehange
y=1, z=0	Lantern	Bicycle	Bridge
y=0, z=0	Smartphone	Modern car	Crane

1. The symbol board tool is invoked. Multi-modal representations of L and R associative arrays are displayed on the screen, allowing user to place his right hand to some position in right array R and left hand to left array L, choosing unique combination of two symbols.
2. User decides on a pair of symbols to be associated with this document. Pair consists of one symbol from left hand array L and one from right hand array R. This pair is uniquely defined by two 3D coordinates from arrays L and R.
3. User places both his hands to proper places in 3D space, associated with chosen symbols in arrays L and R.
4. To confirm his decision, user extends his left thumb, as if performing the OK gesture, as seen in Fig. 7.3.
5. Document icon is visual changed in applications that are working with it, so that it is consistent with chosen associative symbol (e.g., a colorful icon with small picture of the symbol). This enforces the association of document with pair of symbols in user's brain.

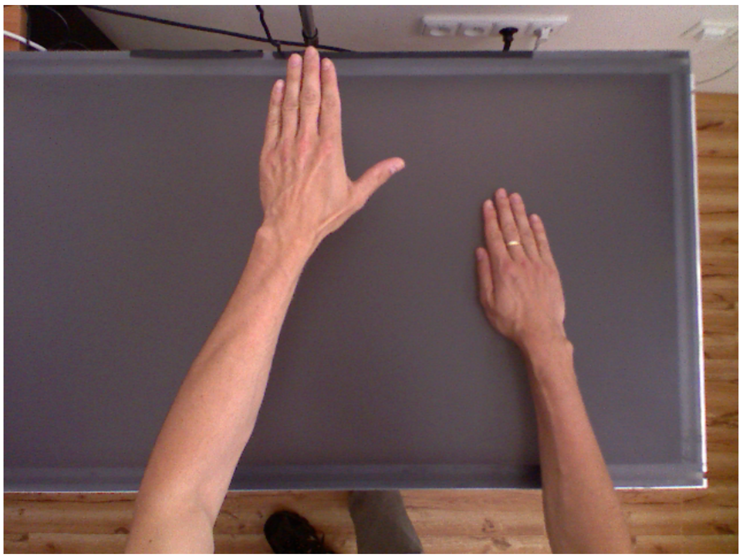


Figure 7.3: User's left hand confirming symbol selection

In step 2) user chooses such symbols that most resemble the document in question. The choice is very individual as every human brain is unique.

7.3.2 Information retrieval step

Information retrieval of previously saved document follows these steps:

1. User decides to access the document. He uses one of two approaches or combination of both to do so:

   (a) User recalls positions of his hands in the 3D space using his motor memory. This is done in a similar way as if pressing frequently used keyboard shortcut without thinking about the keys pressed. Automation of the movement will help to properly place hands even without recalling the symbols. As suggested in [3], this approach can be more suitable for items that are accessed often, as motor memory tends to decay with length of retention interval. Ideal candidates would be shortcuts to frequently used application commands, such as sending an e-mail in mailbox application.

   (b) User recalls from his memory a pair of symbols and looks them up in the arrays in the tool. This can be suitable for document, as user can be choosing symbols based on document content.

2. User accesses the document by placing the hands to the same positions as when he was storing it.

7.3.3 Early prototype of associative array component

For proper operation of component, information about users hands in 3D space is required. RGB camera or depth sensor are suitable input devices for the task. Early prototype of associative array component was built using depth sensor to capture position of user's hands. If using RGB camera, software could calculate z index by comparing area of hand pixels with 2D area of the symbol cell.

7.3.3.1 User hands localization in 3D space of symbol board

A depth camera Microsoft Kinect placed horizontally above user's desk was used to obtain necessary data, in a same way as in TouchTable user interface, described in chapter 5.2. It provides depth map with resolution of 640x480 pixels with frequency of 30 Hz. As clear from [A.8] where similar approach of activating virtual cubes by placing user's fist was used, these parameters are sufficient to create interface with real-time interaction.

The depth image is processed by an auxiliary software, detecting both user's hands in one of 27 segments of associative array 3D spaces. Fig. 7.4 shows sample situation of 3D space, captured by RGB camera. Fig. 7.5 shows result of an analysis of the situation based on a depth frame image, where two segments with detected user's hands are highlighted. Blobs of gray pixels represent detected user's hands. Displayed number shows z index, based on height of the hand above the desk. It is calculated using (7.5), where d is a distance in millimeters of the palm center from the desk surface:

$$z = \lfloor d/10 \rfloor \tag{7.5}$$

7.3.3.2 Hand positioning clues

So that a user can more easily find a place where to put his hands, virtual space can be projected to semi-transparent working desk, as in [A.7]. Some form of physical border (like harsh tape or metal wire) can also be attached to the desk to create tangible feedback helping a user to place hands to desired symbol cell. For help with placement to positions with z=2, glass can be mounted to proper height above the working surface as depth sensor is ignoring it and can provide depth data for all objects under the glass.

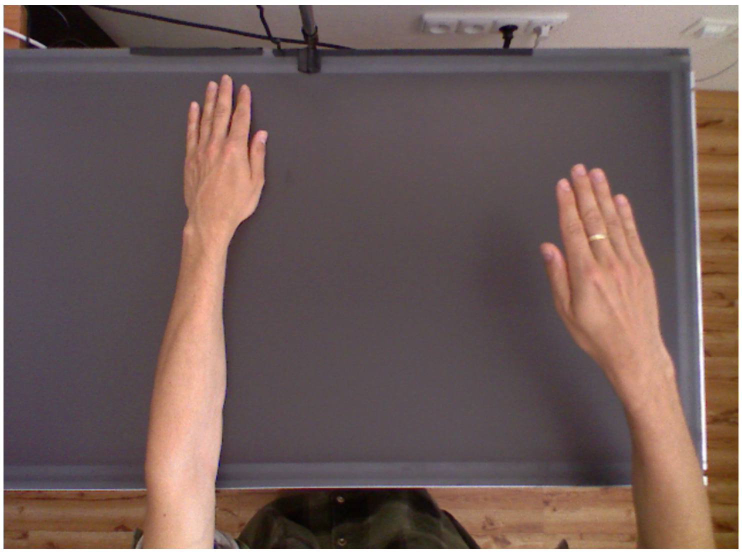


Figure 7.4: User's hands working in a virtual symbol board

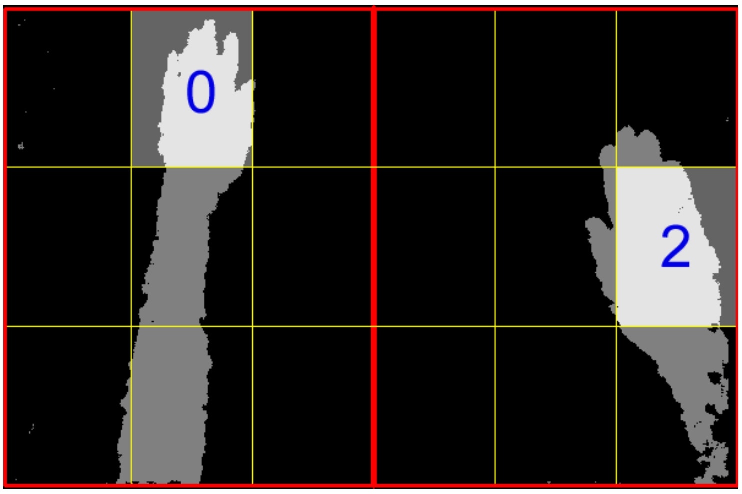


Figure 7.5: Detected user's hands in coordinates L(1,2,0) and R(2,1,1)

7.4 Future work

So that the new proposed user interface component is validated, experiment with real users is needed. In such an experiment, users would be asked to save multiple documents and later find them again, using associative array. Measures like storing and recovering speed, number of failed attempts to store or recover documents and other parameters would be captured. Different types of associative arrays presented in this paper would be examined, comparing their ability to be used for easy association of documents with symbols. However, this experiment has not been done yet and it is a subject of further research. This paper set a theoretical and practical background for such research advance.

Chapter 8

Summary

A high fingertip detection accuracy algorithm called 'Circular scan algorithm' was created. Based on the input image from a depth sensor, the algorithm is able to detect fingertips both for straight and bent fingers in real-time. As to my best knowledge, literature does not mention any other algorithm that would meet these objectives. A new dataset called DepthTip, freely available for the scientific public, was recorded and annotated. Using this dataset, an algorithm processing speed of 22 milliseconds and a detection accuracy of 2.44 pixels was measured and compared to other algorithms. The algorithm detected, without failure, all fingertips in all 73 samples of the DepthTip dataset.

A user interface called TouchTable was created. It works in a similar way as multi-touch surfaces already known (like Microsoft Surface), allowing users to interact with a computer using both their hands. However, compared to regular touch screens, TouchTable has a different method of hand detection, which is based on an observed hand contour and placement of the depth sensor above the table. It brings an advantage that the user can rest his hands on the surface, preventing fatigue during longer work. It also can be built at a fraction of the cost of similar devices available on the market. The hardware for a TouchTable costs approximately 800 euros, compared to a new version of Microsoft surface device, which is valued at 7000 euros.

Additionally, a mathematical model of a bent finger was created, allowing us to find a knuckle position in 3D space, using line approximation methods.

8.1 Lessons learned

Most valuable lessons learned during my research are as follows:

- By increasing fingertip detection accuracy, productivity of a user operating a 3D user interface in a drawing use case, can be increased significantly.
- A 3D user interface that mimics the mouse is not productive enough to replace mouse and keyboard, but has high potential to be used in new paradigms of user interface action mappings, as suggested in chapter 7: Out of the box thinking - new

UI paradigms. Nevertheless, it may be used in scenarios, where traditional devices are not suitable. This can be, for example environments where mouse and keyboard could be stolen or damaged (public information stalls) or where high need for sterility prevents using them (like surgeon operating rooms).

- The most interesting part of my research was to observe how real users were operating in novel ways with a computer, using my proposed user interfaces. It was an exciting experience to watch them and learn through the experiment feedback, in order to later test new ideas in real user interaction. To my surprise, operations that were intuitive and easy to perform for me (as an author of the interface), posed a serious usability issue for others. This understanding allowed me to appreciate the real value of such experiments, where out-of-the-lab ideas, nurtured in the mind of a researcher, are confronted by real users. This brings the most welcomed and desired understanding of the direction where the research should continue.

- Current speech recognition engines are not yet in a stage of hassle-free computer input. For some users, especially those that are not fluent English speakers, pronouncing computer commands so that they can be properly recognized, is a challenging task. Still, if the number of commands is limited, it is already a viable way how to accompany hand gestures in a keyboard free environment.

- User interface must enable users to fully rest all muscles in the time of inactivity. This is necessary if it is supposed to be used in the long term. Most of the time, the user is thinking about the next step and is physically idle. Some interaction with a computer is then performed quickly when the decision is finally made. So, the user interface should require some muscle activity or strain only at the time of action.

- For a hand operated user environment, it is desirable that operations are performed in place. This means that users can see changes in a user interface at the same location where they are working with their hands. In this way, the human vision system and human hand coordination system are synchronized. If the interaction occurs in the air or at the desk, but the user can only watch the computer screen with their virtual hands, this confuses users and forces them to split their cognition models of reality into two - one for their hands and one for the virtual world they are manipulating.

8.2 Future work

In future, I would like to perform more advanced experiments with the TouchTable and with more users. I also want to learn how users interact after they underwent longer training with the TouchTable. I would like to test a physical ball-like manipulator with an extrusion, instead of the index finger, to minimize the strain to hand muscles (as suggested by one of the users). Additionally, I would like to expand the idea of such a 3D user interface that is not just mimicking the mouse in the way that a user operates it, but instead, bring new paradigms for HCI. Experience gained during this work and also creation of hand analysis algorithms are good starting points for such a follow up research.

As a final word I would like to humbly admit, that though my research provided a lot of answers regarding a productive 3D user interface, many more questions were raised to be answered by follow up research.

Appendix A - DepthTip evaluation dataset

DepthTip is a dataset that is used to evaluate precision of fingertip detection algorithm, consisting of 74 images obtained from depth sensor Microsoft Kinect (version 1).

DepthTip is one of the merits of this work and is available for scientific public. This chapter explains the contents of DepthTip and information how to use this dataset for evaluation of algorithms calculating features like fingertips, that analyse binary image of a human hand.

8.3 DepthTip folders

DepthTip can be downloaded as ZIP file [2] and has following folders:

- inputDataSet
- annotated
- results

8.3.1 Folder *inputDataSet*

It contains raw data captured from the sensor, where n in the filename stands for the number of a sample.

- n_depth.png - depth image, where pixel intensity represents a distance in mm from the sensor.
- n_rgb.png - image from an RGB camera for reference. Please note that pixel coordinates are not calibrated with the depth image and are slightly shifted, which is caused by a different placement of RGB and depth cameras on the Kinect device.
- n_hand.png - a binary image where white pixels represent a hand and black pixels a background.

Fig. 8.1 shows an example of three input files for the DepthTip sample no. 1.

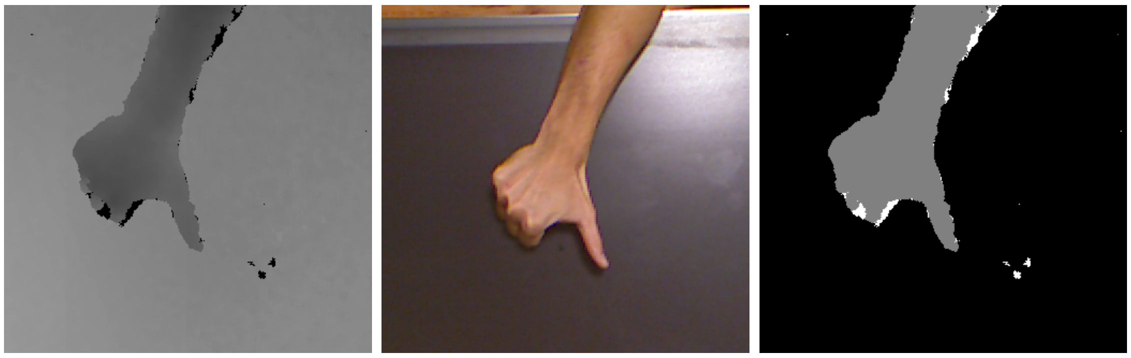


Figure 8.1: Example of input files for DepthTip sample no. 1

8.3.2 Folder *results*

It contains result of fingertip detection when given sample is analysed by one of the proposed fingertip detection algorithms. Each results subfolder contains files with name `(n)_(nameOfAlgorithm).png`, where (n) in the filename stands for the number of a sample. Detected fingertips are marked with yellow diagonal cross.

Additionally, excel file called `algorithmExperiments.xlsx` containing all data created by the run-time experiments with different fingertip detection algorithms can be found in this folder. For each algorithm, spreadsheet contains a two separate lists, one with algorithm running speeds and one for algorithm detection results.

Folder `results` has following subfolders:

- CircleScanAnalyzerContourOnly - results of an execution of the proposed algorithm from 2015 described in chapter 6.2.5.
- CircleScanAnalyzerFull - results of an execution of the proposed algorithm from 2015 described in chapter 6.2.4
- CuttOffFingerPointAnalyzer - results of an execution of the proposed algorithm from 2014 described in chapter 6.2.3. Black cutting lines are visible in the results where a finger blob was cut off by the algorithm from the palm. Numbers displayed at polygon points are pixel classification groups. More details on the pixel classes can be found in the published paper.
- DistanceFromCenterAnalyzer - reference results of an execution of the algorithm from 2012 from literature described in chapter 3.5 for first 50 DepthTip samples. Gray circle display the maximum distance circle found.
- PolygonSimplificationAnalyzer - results of an execution of the proposed algorithm from 2013 described in 6.2.2.

CircleScanAnalyzerContourOnly and *CircleScanAnalyzerFull* contain also red pixels (classified as fingertips) and brown pixels (classified as finger segment pixels).

PolygonSimplificationAnalyzer and *CuttOffFingerPointAnalyzer* display also the simplified hand contour polygon drawn in gray respectively green color.

Fig. 8.2 shows example results for the sample no. 1.

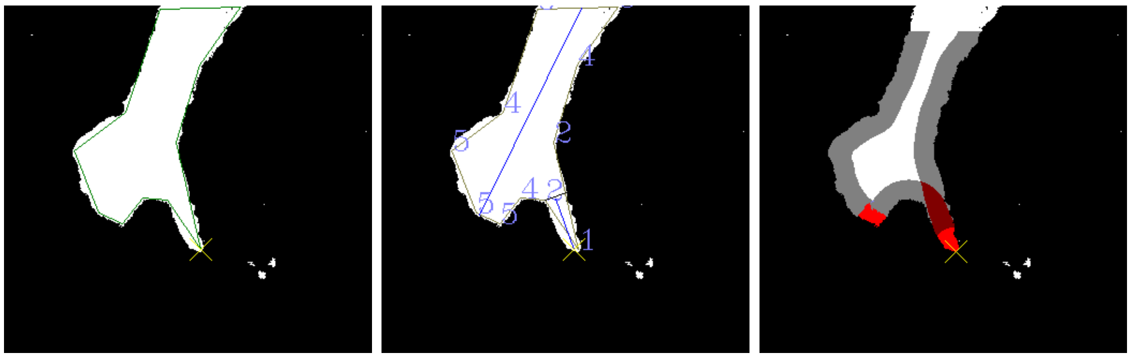


Figure 8.2: Example results of DepthTip sample no. 1 analysis by different algorithms

8.3.3 Folder *annotated*

Each input depth image was manually annotated by setting of fingertip pixels to particular color, depending on the finger index.

- RGB(128,0,0) for thumb
- RGB(160,0,0) for index finger
- RGB(192,0,0) for middle finger
- RGB(224,0,0) for ring finger
- RGB(255,0,0) for little finger

The folder contains also a text file containing coordinates of these annotated pixels. This is an example of the content from file `annotation/1_annotation.txt`.

```
Detected finger index (1=thumb,5=little finger),x coord, y coord)
1,183,229
```

This means that in a depth image the thumb (index = 1) is present at coordinates x=183, y=229.

8.4 How to use DepthTip to evaluate a fingertip detection algorithm

In an experiment, proposed algorithm needs to process all input images and calculate an average distance from annotated text files. This value needs to be compared to the ground truth algorithm implementations from chapter 6.3.1. From that is for example clear that the 'Circular scan algorithm' performs much better compared to the 'Finger cut-off algorithm'.

Appendix B - Dissertation source code

8.5 Source code download

Source code for whole TouchTable and all referred fingertip detection algorithms is freely available for download [14]. This means that whoever is interested in running TouchTable experiments or using it to evaluate his own fingertip detection algorithm in a user experiment is welcome to use it.

The source code contains following modules:

- DepthBasics-WPF: The main executable project that enables to run TouchTable with a newer version of Microsoft Kinect Device v2.
- KinectExplorer-WPF: The main executable project that enables to run TouchTable with an older version of Microsoft Kinect Device v1.
- KinectWpfViewers: A visualisation of depth data from Kinect to a human readable image.
- ReferencesLibrary_x86: All needed libraries to run TouchTable in a 32-bit mode.
- TouchTableCore: The core logic for TouchTable and a fingertip detection. Fingertip detection algorithms reside inside this module in the `analyzer` folder.
- TouchTableUnitTest: Unit tests that are verifying a TouchTable logic and that fingertip detection algorithms work as expected.

The reason why there are two different projects for two different versions of Kinect device stems from the fact that unfortunately, API to both Kinect versions is not backward compatible and different source code is needed to work with both devices.

8.6 Code listings for critical code sections

This chapter contains the source code for the winning CircularScanAlgorithm that can be found under the following path:

`TouchTableCore/analyzer/circleScanAnalyzer/CircleScanAnalyzer.cs`

For better understanding of the algorithm, it is recommended to refer to the diagram in Fig. 6.13.

8.6.1 investigateOneHandPixel method

Input of this method is a single pixel on the hand boundary. Method determines to what category the single pixel belongs and saves the result to the data structure *resultFingerTipSegments* and *resultFingerSegments*.

```
private void investigateOneHandPixel(int col, int row, int scanCircleDiameter, double ↩
    halfCirclePerimeter, int scanDelta, Image<Gray, byte> handContourMap, Image<Bgr, ↩
    byte> displayImg, Image<Gray, byte> maskWithCircle, double minimalIntersectingArea, ↩
    PointF centerOfScanWindow, Image<Gray, byte> resultFingerSegments, Image<Gray, byte>↩
     resultFingerTipSegments)
{
    var scanWindow = new Rectangle(col - scanDelta, row - scanDelta, scanCircleDiameter,
                                   scanCircleDiameter);
    handContourMap.ROI = scanWindow;
    var intersection = maskWithCircle.Sub(handContourMap);

    using (var memStorage = new MemStorage())
    {
        var blobsWithReasonableSize = ImageUtil.FindContours(5, intersection, memStorage↩
            );
        var intersectionBlobCount = blobsWithReasonableSize.Count;
        if (intersectionBlobCount >= 2)
        {
            markResultPixel(displayImg, row, col, 0, 0, 128);
            resultFingerSegments.Data[row, col, 0] = 255;
        }
        else if (intersectionBlobCount == 1)
        {
            // This may be fingertip
            var singleDetectedBlob = blobsWithReasonableSize[0];
            var isPartOfBlobConvexHull =
                singleDetectedBlob.GetConvexHull(ORIENTATION.CV_CLOCKWISE).InContour(↩
                    centerOfScanWindow) > 0;


            var intersectCurveLength = CvInvoke.cvArcLength(singleDetectedBlob, MCvSlice↩
                .WholeSeq, 1);
            var curveIsLongerThenHalfCircle = intersectCurveLength > halfCirclePerimeter↩
                 * 4;

            var isFingerTip = isPartOfBlobConvexHull &&
                              curveIsLongerThenHalfCircle;

            if (isFingerTip)
            {
                // This pixel is fingertip
                markResultPixel(displayImg, row, col, 0, 0, 255);
                resultFingerTipSegments.Data[row, col, 0] = 255;
            }
            else
            {
                // This pixel belongs to palm but is close to edge of hand
                markResultPixel(displayImg, row, col, 128, 128, 128);
            }
        }
        else if (intersectionBlobCount == 0)
        {
            markResultPixel(displayImg, row, col, 255, 255, 255);
        }
    }
}
```

8.6.2 createFingersFromFingertipSequences method

This code travels alongside hand contour pixels and if enough subsequent fingertip pixels are collected, that as a sequence neighbours at least one finger segment pixel, detected finger is created from this sequence.

```
 private static void createFingersFromFingertipSequences(
 Contour<Point> handContour, byte[,,] resultFingerSegmentsData, byte[,,] ↩
     resultFingerTipSegmentsData, List<FingerBlob> resultFingerBlobs)
{
    int pointIndex = 0;
    var handContourPixels = handContour.ToArray();
    decimal pixelCount = handContourPixels.Count();
    int x, y;
    var collectedFingerTipPixels = new List<Point>();
    var lastPoint = handContourPixels.Last();
    bool lastPixelWasFingerSegment = resultFingerSegmentsData[lastPoint.Y, lastPoint.X, ↩
        0] == 255;
    while (pointIndex < pixelCount)
    {
        var handContourPixel = handContourPixels[pointIndex];
        y = handContourPixel.Y;
        x = handContourPixel.X;
        // Check to already analyzed pixel category, stored in ↩
            resultFingerTipSegmentsData structure
        var isFingerTip = resultFingerTipSegmentsData[y, x, 0] == 255;
        if (!isFingerTip)
        {
            // Too short fingertip arches are discarded
            if (collectedFingerTipPixels.Count() > 15)
            {
                // We have fingertip! (sequence of fingertip pixels, neighbouring at ↩
                    least one finger segment pixel)
                if (lastPixelWasFingerSegment)
                {
                    var start = collectedFingerTipPixels.First();
                    var end = collectedFingerTipPixels.Last();
                    var fingerBlob = CreateFingerBlob(
                        start,
                        end, collectedFingerTipPixels.ToArray());
                    resultFingerBlobs.Add(fingerBlob);
                }
            }
            if (collectedFingerTipPixels.Count > 0)
            {
                collectedFingerTipPixels.Clear();
            }
            lastPixelWasFingerSegment = resultFingerSegmentsData[y, x, 0] == 255;
        }
        else
        {
            collectedFingerTipPixels.Add(handContourPixel);
        }

        pointIndex++;
    }
}
```

Appendix C - How to understand box plots

In this work, box plot are used for display of statistical data in a standard way. Fig. 8.3 shows an example of a box plot. Box plots provide a visualization of summary statistics for sample data and contain the following features [1]:

- The tops and bottoms of each box are the 25th and 75th percentiles of the samples, respectively. The distances between the tops and bottoms are the interquartile ranges.
- The line in the middle of each box is the sample median. If the median is not centered in the box, it shows sample skewness.
- The whiskers are lines extending above and below each box. Whiskers are drawn from the ends of the interquartile ranges to the furthest observations within the whisker length (the adjacent values).
- Observations beyond the whisker length are marked as outliers. An outlier is a value that is more than 1.5 times the interquartile range away from the top or bottom of the box. Outliers are displayed with a red + sign.
- Notches display the variability of the median between samples. The width of a notch is computed so that box plots whose notches do not overlap (as above) have different medians at the 5% significance level. The significance level is based on a normal distribution assumption, but comparisons of medians are reasonably robust for other distributions. Comparing box-plot medians is like a visual hypothesis test, analogous to the t test used for means.

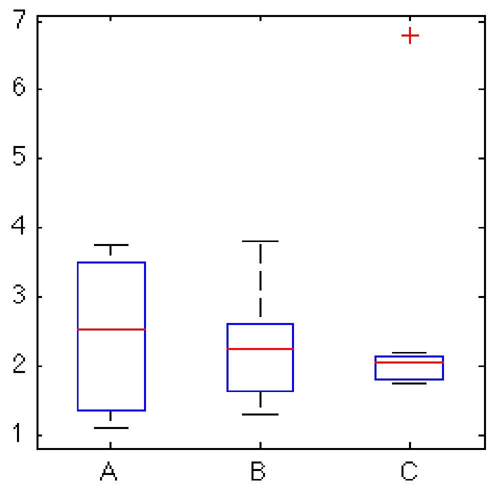


Figure 8.3: Example of a box plot used in the thesis

Abbreviation list

.NET
: A framework for running C# applications, it is an integral component of the Windows operating system.

3D
: 3-Dimensional, data with x, y and z coordinates.

API
: Applicable program interface, a set of routines, protocols and tools for building software applications.

AR
: Augmented reality, a user interface where a user senses real a environment which surrounds him but his senses has an extra digital information from the user interface. An example is meta information like a distance from other planes that is projected onto the cockpit window of jet fighters. It differs from VR in a way that user is aware of the real environment surrounding him.

C#
: C# (pronounced as *Csharp*) is a simple, modern, general-purpose and object-oriented programming language, with a syntax very similar to the Java programming language.

CTS
: Carpal tunnel syndrome, a health problem caused by repetitive movements of tiny hand muscles.

EmguCV
: Emgu CV is a cross platform .Net wrapper to the OpenCV image processing library.

HCI
: Human-computer interaction

HSV
: Hue, Saturation and Value - three channels in the HSV color model, that is alternative to the RGB (Red, Green, Blue) model and is more suitable for a feature detection.

Hi-vis
High visibility material frequently used in a visual motion capture that marks items using highly visible contrast color.

IR
Infrared, light waves with a specific frequency in light spectrum.

MoCap
Motion capture

OpenCV
OpenCV is a library of programming functions mainly aimed at a real-time computer vision, developed by the Intel Russia research center in Nizhny Novgorod.

RSI
Repetitive strain injury, a health problem caused by repetitive movements of tiny hand muscles.

UI
User Interface. System that is used for a bi-directional communication between a man and a machine.

VR

Virtual reality, a user interface where a user is completely submerged with multiple senses in a virtual world. It differs from AR UI in a way that the user is not aware of what is happening in a real environment that is surrounding him.

List of Figures

List of Tables

Bibliography

[1] Box plots - matlab and simulink. (online) [accessed 2015-11-19]. URL `http://www.mathworks.com/help/stats/box-plots.html`

[2] Depth tip: public dataset for fingertip detection. (online) [accessed 2014-12-06]. URL `http://fei-software-public.upce.cz/fingers/DepthTip.zip`

[3] Adams, J.A., Dijkstra, S.: Short-term memory for motor responses. Journal of Experimental Psychology **71**(2), 314–318 (1966), ISSN 0022-1015(Print). DOI 10.1037/h0022846

[4] Bhuyan, M., Neog, D.R., Kar, M.K.: Fingertip detection for hand pose recognition. International Journal on Computer Science and Engineering **4**(3), 501–511 (2012)

[5] Carrino, F., Ridi, A., Mugellini, E., Khaled, O., Ingold, R.: Gesture segmentation and recognition with an emg-based intimate approach - an accuracy and usability study. In: Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on, pp. 544–551 (2012). DOI 10.1109/CISIS.2012.173

[6] Chaudhary, A., Raheja, J.L., Das, K., Raheja, S.: Fingers' angle calculation using level-set method. CoRR **abs/1406.3418** (2014). URL `http://arxiv.org/abs/1406.3418`

[7] Engelbart, D.C., English, W.K.: A research center for augmenting human intellect. In: Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I, AFIPS '68 (Fall, part I), p. 395–410. ACM, New York, NY, USA (1968). DOI 10.1145/1476589.1476645. URL `http://doi.acm.org/10.1145/1476589.1476645`

[8] Goto, H., Kawasaki, Y., Nakamura, A.: Development of an information projection interface using a projector-camera system. In: 2010 IEEE RO-MAN, pp. 50–55 (2010). DOI 10.1109/ROMAN.2010.5598663

[9] Hackenberg, G., McCall, R., Broll, W.: Lightweight palm and finger tracking for real-time 3d gesture control. In: 2011 IEEE Virtual Reality Conference (VR), pp. 19–26 (2011). DOI 10.1109/VR.2011.5759431

[10] Hagara, M., Pucik, J.: Fingertip detection for virtual keyboard based on camera. In: Radioelektronika (RADIOELEKTRONIKA), 2013 23rd International Conference, pp. 356–360 (2013). DOI 10.1109/RadioElek.2013.6530945

[11] He, G.F., Kang, S.K., Song, W.C., Jung, S.T.: Real-time gesture recognition using 3D depth camera. In: 2011 IEEE 2nd International Conference on Software Engineering and Service Science (ICSESS), pp. 187–190 (2011). DOI 10.1109/ICSESS.2011.5982286

[12] Heckenbergova, J., Marek, J., Souckova, J., Tucek, P.: Nonsmooth function approximation in practical change poit problem. In: 11th International conference Aplimat 2012, pp. 895–904 (2012)

[13] Jacko, J.A., Sears, A. (eds.): The Human-computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications. L. Erlbaum Associates Inc., Hillsdale, NJ, USA (2003), ISBN 0-8058-3838-4

[14] Jetensky, P.: TouchTable source code. (online) [accessed 2015-12-03]. URL `http://fei-software-public.upce.cz/fingers/TouchTableSource.zip`

[15] Jetensky, P.: Human hand image analysis extracting finger coordinates and axial vectors: Finger axis detection using blob extraction and line fitting. In: Radioelektronika (RADIOELEKTRONIKA), 2014 24th International Conference, pp. 1–4 (2014). DOI 10.1109/Radioelek.2014.6828474

[16] Josep R. Casas, J.R.H.: Colortip - image processing group. (online) [accessed 2015-01-26]. URL `https://imatge.upc.edu/web/res/colortip`

[17] Lee, K., Min, K.: An interactive image clipping system using hand motion recognition. Information Systems **48**, 296–300 (2015), ISSN 0306-4379. DOI 10.1016/j.is.2014.05.011. URL `http://www.sciencedirect.com/science/article/pii/S0306437914001136`

[18] Malik, S., Laszlo, J.: Visual touchpad: A two-handed gestural input device. In: Proceedings of the 6th International Conference on Multimodal Interfaces, ICMI 2004, pp. 289–296 (2004), ISBN 1-58113-995-0. DOI 10.1145/1027933.1027980. URL `http://doi.acm.org/10.1145/1027933.1027980`

[19] Modarres, A., Cruz-Hernandez, J., Grant, D., Levesque, V.: Simulation of tangible user interface interactions and gestures using array of haptic cells (2014). URL `https://www.google.com/patents/US20140320436`. US Patent App. 14/262,482

[20] Ni, T., McMahan, R.P., Bowman, D.A.: Tech-note: rapmenu: Remote menu selection using freehand gestural input. In: 3D User Interfaces, 2008. 3DUI 2008. IEEE Symposium on, pp. 55–58. IEEE (2008)

[21] Parkale, Y.: Gesture based operating system control. In: 2012 Second International Conference on Advanced Computing Communication Technologies (ACCT), pp. 318–323 (2012). DOI 10.1109/ACCT.2012.58

[22] Ramanahally, P., Gilbert, S., Niedzielski, T., Velázquez, D., Anagnost, C.: Sparsh ui: A multi-touch framework for collaboration and modular gesture recognition. In: ASME-AFM 2009 World Conference on Innovative Virtual Reality, pp. 137–142. American Society of Mechanical Engineers (2009)

[23] Rao, C.: Linear Statistical Inference and Its Application. Wiley series in probability and mathematical statistics. Wiley (1973). URL `https://books.google.cz/books?id=AQyUnQEACAAJ`

[24] Ren, Z., Meng, J., Yuan, J.: Depth camera based hand gesture recognition and its applications in human-computer-interaction. In: Communications and Signal Processing (ICICS) 2011 8th International Conference on Information, pp. 1–5 (2011). DOI 10.1109/ICICS.2011.6173545

[25] Rosenfeld, D., Zawadzki, M., Sudol, J., Perlin, K.: Physical objects as bidirectional user interface elements. Computer Graphics and Applications, IEEE **24**(1), 44–49 (2004), ISSN 0272-1716. DOI 10.1109/MCG.2004.1255808

[26] Rosten, E., Porter, R., Drummond, T.: Faster and Better: A Machine Learning Approach to Corner Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **32**(1), 105–119 (2010), ISSN 0162-8828. DOI 10.1109/TPAMI.2008.275

[27] Seber, G., Wild, C.: Nonlinear Regression. Wiley Series in Probability and Statistics. Wiley (2003), ISBN 9780471471356. URL `https://books.google.cz/books?id=YBYlCpBNo_cC`

[28] Sharma, R., Huang, T., Pavlovic, V., Zhao, Y., Lo, Z., Chu, S., Schul, K.: Speech/gesture interface to a visual computing environment for molecular biologists. In: , Proceedings of the 13th International Conference on Pattern Recognition, 1996, vol. 3, pp. 964–968 vol.3 (1996). DOI 10.1109/ICPR.1996.547311

[29] Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., Cook, M., Moore, R.: Real-time human pose recognition in parts from single depth images. Commun. ACM **56**(1), 116–124 (2013), ISSN 0001-0782. DOI 10.1145/2398356.2398381. URL `http://doi.acm.org/10.1145/2398356.2398381`

[30] Song, P., Goh, W.B., Hutama, W., Fu, C.W., Liu, X.: A handle bar metaphor for virtual object manipulation with mid-air interaction. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '12, pp. 1297–1306. ACM, New York, NY, USA (2012), ISBN 978-1-4503-1015-4.

DOI 10.1145/2207676.2208585. URL
`http://doi.acm.org/10.1145/2207676.2208585`

[31] Xia Yuan, Q.P.: Real-time stereo vision based fingertip detection and tracking. In: ICCEE 2010 (2010)

[32] Young, S.J., Pratt, J., Chau, T.: Misperceiving the speed-accuracy tradeoff: imagined movements and perceptual decisions. Experimental brain research **192**(1), 121–132 (2009), ISSN 1432-1106. DOI 10.1007/s00221-008-1563-x. PMID: 18807021

[33] Zhu, D., Feng, Z., Yang, B., Jiang, Y., Yang, T.: The design and implementation of 3d hand-based human-computer interaction platform. In: Computer Application and System Modeling (ICCASM), 2010 International Conference on, vol. 2, pp. V2–485–V2–489 (2010). DOI 10.1109/ICCASM.2010.5620579

Papers published in association with the dissertation

[A.1] Pavel Jetensky and Simeon Karamazov. 3D associative symbol array as component of user interface. In *2014 ELMAR 56th International Symposium IEEE. Zadar (Chroatia)*. ISBN 978-953-184-199-3.

[A.2] Pavel Jetensky and Simeon Karamazov. ACCESSIBLE CHESSBOARD FOR BLIND THAT IS MOUSE AND KEYBOARD FREE. In *2012 Conference proceedings from International Conference on Applied Electrical Engineering and Informatics 2012. Kiel (Germany)*. Pages 72 76. ISBN 978-80-553-1030-5.

[A.3] Pavel Jetensky, Jaroslav Marek, and Josef Rak. Fingers segmentation and its approximation. In *2015 Radioelektronika (RADIOELEKTRONIKA), 25th International Conference IEEE. Pardubice (Czech Republic)*. Pages 431 434. ISBN 978-1-4799-8117-5.

[A.4] Pavel Jetensky. Human hand image analysis extracting finger coordinates using circular scanning. In *2015 Radioelektronika (RADIOELEKTRONIKA), 25th International Conference IEEE. Pardubice (Czech Republic)*. Pages 427 430. ISBN 978-1-4799-8117-5.

[A.5] Pavel Jetensky. Human Hand Image Analysis Extracting Finger Coordinates and Axial Vectors:Finger axis detection using blob extraction and line fitting. In *2014 Radioelektronika (RADIOELEKTRONIKA), 24th International Conference IEEE. Bratislava (Slovak Republic)*. Pages 73 77. ISBN 978-1-4799-3713-4.

[A.6] Pavel Jetensky. CHESSBOARD SQUARE OCCUPANCY ANALYSIS THROUGH K-MEANS CLUSTERING BASED VISUAL MARKER DETECTION. In *2013 Conference proceedings from XIII. international conference IMEA. Pardubice (Czech Republic)*. Pages 24 31. ISBN 978-80-7395-696-7.

[A.7] Pavel Jetensky. Multimodal Control of Computer Aided Design Software Through Fingers Tracking and Speech Recognition. In *2013 Conference proceedings from II. International Workshop on Human-Machine Systems,Cyborgs and Enhancing Devices. Salford University, Manchester (United Kingdom)*.

[A.8] Pavel Jetensky. VIRTUAL 3D KEYBOARD FOR GESTURE BASED TYPING. In *2011 Conference proceedings from XI. international conference IMEA. Technical University of Liberec (Czech Republic).* Pages 73 77. ISBN 978-80-7372-720-8.