Patrick Ruprecht

Hand- and Fingertracking in Augmented Reflection Technology for Post-Stroke Rehabilitation

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

submitted to

Graz University of Technology

Supervisor

Dieter Schmalstieg

Institut for Computer Graphics and Vision

Advisors

Michael Donoser | Samuel Schulter

Institut for Computer Graphics and Vision

Holger Regenbrecht | Simon Hörmann

Department of Information Science

University of Otago (New Zealand)

Graz, Austria, January 2015

TO FABIAN – MY GODCHILD

IN HOPES THAT HE WILL FIND
THE WAY TO COMPUTER VISION

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. The text document uploaded to TUGRAZonline is identical to the presented master's thesis dissertation.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

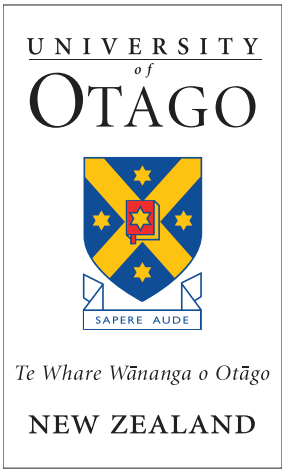
(place/date)	(signature)

Abstract

Augmented reality applications have the potential to enhance lots of established rehabilitation techniques by controlling what people perceive and believe. The therapy with Augmented Reflection Technology (ART) is used for rehabilitating patients who have lost the control of hands due to a stroke. Our system captures hands with cameras and displays them on a screen. The basic idea is to fool the patient's brain by showing that the hands are intact, although movements of the impaired hand are amplified or reflected by the healthy hand. By tracking the position of the hand, we are able to apply manipulations in different ways. Furthermore, the detected fingertips allow interactions in a virtual scene as well as the measurement of movement progresses. The present system provides novel rehabilitation methods that can innovate conventional therapy sessions. Our work focuses on finding the position to which patients are pointing and the estimation of several joint locations on the hand. For the latter, we propose a supervised learning algorithm using random forests, given their capability to learn the appearance of hands from annotated depth image datasets. The algorithm handles various hand articulations and finds the desired joints but there is no limitation to any pose. Our approach is evaluated on a publicly available dataset where we outperform the state-of-the-art prediction accuracy, achieving a localization error of only a few millimeters.

Keywords. Human Hand Pose Estimation, Fingertip Detection, Random Forests, Regression, Classification, Augmented Reality, Stroke Therapy.

Acknowledgments



The presented system is the result of great efforts from lots of researchers and scientists. Honest thanks go to the staff at the Department of Information Science at Otago University in New Zealand. All the people are great and they offer an excellent working environment so that I will never forget my stay there. In particular, I want to thank Holger Regenbrecht who manages the Multi-Media Systems Research Laboratory where our augmented reality application is developed. He works with full enthusiasm on several projects and leads them like none other. His field of knowledge covers lots of topics so that any discussion yields useful information, especially for technical projects. Additionally to the project supervision, he and his family also take care of everyday matters in various fields so that the stay abroad is as pleasant as it can be. Many thanks also to Simon Hörmann who acts as medical contact person for post-stroke rehabilitation. He cooperates with hospitals and supports our group to improve the ART system for clinical use. Last but not least, I give thanks to Jonny Collins for working together in the same office over many months, thank you for the great time.

From the Institute for Computer Graphics and Vision at Graz University of Technology, I want to thank Tobias Langlotz for initiating this Master's Thesis and Michael Donoser for supervising the project as well as the cooperation between the universities. Special thanks go to Samuel Schulter for supporting the main part of my work. It would not be possible to implement everything from scratch, so I am very grateful for the provided random forest framework. I also appreciate his comprehensible explanations of algorithms and his ideas to solve problems. Finally, many thanks to Dieter Schmalstieg and Horst Bischof for providing the opportunity to work at the institute with all the other staff members, yielding lots of helpful and interesting discussions.

Contents

1 Introduction

Worldwide, a stroke involves millions of people a year. It is a traumatic event for victims and often results in hemiparesis, the weakness of the upper and lower limbs where either the entire left or right half of the body is affected. The neural circuits in the brain that are responsible for mediating a movement action are no longer intact [46] but muscles still work. Patients undertake extensive rehabilitation in order to regain some control of the limb [42] where the mirror reflection therapy [16, 37] is a common technique. In therapy sessions, a mirror is placed between the patient's limbs such that the impaired limb is hidden but a reflection by the healthy limb is visible. When performing exercises, the patient looks into the mirror and gets the visual impression that the limb is unaffected due to reflected movements of the other limb. The intention is that patients perceive their own abilities and believe what they see, enabling the healthy part of the brain to learn the controlling of the impaired limb.

1.1 Augmented Reflection Technology

The ART system [41] is used to treat patients with unilateral motor deficits of the arm, in particular for the early stage of stroke recovery. The concept is based on the therapy with mirrors but instead of exploiting the optical reflection, the system captures hands with cameras and displays them on a screen (see Figure 1). An important aspect is the level of illusion when applying such therapies. The study [38] has shown that the system is more believable than the proven rehabilitation with mirrors. The main benefits are that no asymmetric reflection of the arm occurs and that the system provides a variety of manipulations which are unfeasible in the traditional therapy.

(a) **(b)**

Figure 1: Setup of the ART system for post-stroke rehabilitation. We have adapted the current system where (a) the prototype consists of boxes to provide a controlled environment. The proposed system is (b) a free table solution with a colored fabric as background so that the hand in the foreground can be easily segmented.

The system allows to manipulate the experience of reality because the hands and virtually generated objects are rendered in an augmented environment which is controlled by the therapist. In this way, the movement of the impaired hand can be simulated by the healthy hand with the mirrored video stream. Furthermore, an amplification of weak movements is feasible so that patients are able to move the hand as usual. A variety of augmented reality applications can be built for therapy sessions, e.g. applications for playing games or performing different exercises where the patient interacts with objects in the virtual scene. In any case, the original hand from the left or right video stream needs to be displayed on the screen so that the patient is able to identify the hand as its own. This allows to fool the brain because hands visually appear in the same way as in reality.

1.2 Hand- and Fingertracking

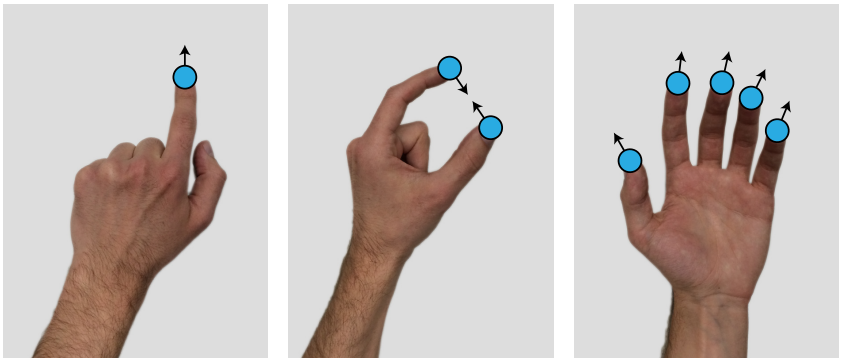
To enable movement manipulations and interactions for rehabilitation purposes, the hands or fingers need to be tracked. Patients are very sensitive to haptics, hence no marker-based tracking algorithms with gloves [12, 28, 36, 54] or similar approaches can be used, even if they would provide a precise result in real-time. The ART system already uses the video stream to keep track of the movement and interaction of the hand. However, the current solution is limited to a few applications for post-stroke rehabilitation due to the naive tracking algorithm and the requirement of controlled light conditions.

In this work, we address the problems in the current system and extend it with reliable tracking functionalities, yielding a robust base system that can be used for future applications in the rehabilitation system with no restriction to an optimal environment.

For a more flexible setup in terms of usability and the opportunity to use different cameras, we replace the box approach with a free table solution. This modification is shown in Figure 1a and Figure 1b, respectively. The illustration in Figure 2 specifies the requirements of future therapy applications. These are the determination of the position to which the hand is pointing, the tracking of some points in order to grab objects in the augmented environment and the measurement of movement progresses for each individual finger so that the therapist is able to analyze the rehabilitation success.

The chosen fabric on the table allows a reliable segmentation on color images, hence standard webcams can be used. To find the pointing direction of the hand, we propose to fit an ellipse on the binary mask of the segmented hand. The algorithm evaluates the complete hand area and yields the desired image coordinate to which the patient points. If only such basic interactions are required, this approach works sufficiently accurate and stable for lots scenarios. In some cases, the hand of the patient can be clenched so that no defined

pointing gesture can be performed. However, by exploiting the elliptical shape, each patient is able to interact with the rehabilitation system because the algorithm detects the pointing direction regardless of the hand articulation.



(a) **(b)** **(c)**

Figure 2: The goals of our work for the rehabilitation system. Future applications need (a) the position to which the hand is pointing, (b) some points for grabbing objects and (c) all fingertips to measure the movement progress of the patient's hand over several therapy sessions.

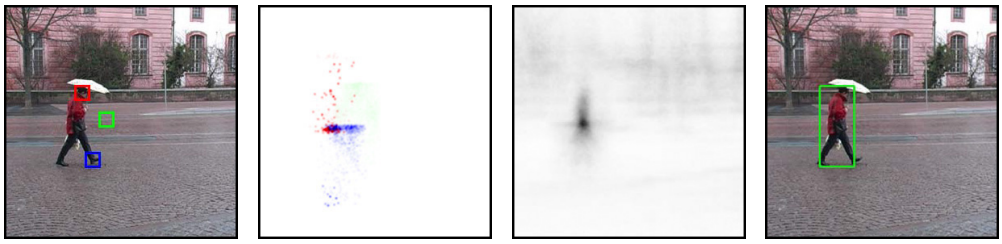
The other requirements, i.e. grabbing of objects for interaction purposes and finding all fingertips for measuring the therapy progress, are solved using a machine learning approach. In particular, we use a random forest that learns the appearance of hands on depth images. For each video frame, the approach predicts the coordinates of desired joint locations on the hand.

1.3 Forest Approaches

The main contribution of our work consists of the development of an efficient method for detecting joint locations on hands (e.g. fingertips) using random forests due to their recent success in various learning tasks on images. At the beginning, we want to give a brief overview of random forests, followed by several related approaches which have initiated the popularity.

Random forests are ensembles of slightly different decision trees where each tree consists of hierarchically ordered split nodes and leaf nodes. For image processing tasks, the forest analyzes patches as features which are extracted from different image locations and forwarded to all the trees. Each split node evaluates the appearance of arrived patches by performing a binary test and passes them to the left or right child. This node can be either a leaf node or another split node where the splitting of patches is repeated. During training, each leaf node stores statistics about the arrived patches, yielding the essential information when similar patches from new unseen images terminate in the same node. For classification forests, the statistics consist of the probability

of each available class that can be computed from the labeled patches. In a regression forest, an offset vector to the annotated image part is assigned to each patch, thus the statistic is formulated as distribution over several vectors, allowing to vote for the desired location.



(a) **(b)** **(c)** **(d)**

Figure 3: Hough forests for object detection. The images are taken from [17] and show (a) the original image with different patches emphasized, (b) the votes for the location of the pedestrian from the selected patches, (c) the aggregated votes from all extracted patches in a Hough space and (d) the detection hypothesis corresponding to the peak in this image. Note, the red and blue patches yield the strongest votes while the green patch from the street shows weak responses.

Typically, a classification task results in image segmentation [6, 43, 44] where the class label is estimated for each patch, consequently each pixel in the image. Furthermore, also image classification [3, 32] or keypoint recognition [31] can be involved in such tasks. An example for object localization is presented in Figure 3 where Gall and Lempitsky [17] formulate a regression problem to predict the location of an object in the image. In this case, a set of patches is subsampled or densely extracted from the color image, followed by passing all patches through the split nodes of each tree in the forest. When a leaf node is reached, the patch casts a voting for the location of the pedestrian. All these votes are aggregated into a Hough space where the peak yields the pedestrian's location. More specifically, the groundtruth data assigns a class label for each pixel. Hence, the authors create a bounding box around the labeled object in the corresponding training exemplar, followed by computing the offset vectors between the patch location and the center of this box. When testing on new unseen images, the leaves also predict the center of the pedestrian. To learn the difference between background and foreground (the predicted classes), they randomly chose between a classification and regression objective at split nodes, allowing to eliminate votes which would worsen the result. Lots of approaches [14, 35, 47] perform both tasks at the same time.

Another research topic is the estimation of head poses as visualized in Figure 4 with depth or color images. Fanelli et al. [13] predict the nose position and its orientation in real-time by using a dataset which contains synthetic depth images from virtual head models. The approach is based on the appearance of depth patches, i.e. the binary tests evaluate the difference in depth to find similarities in patches. In the training phase, each patch of the image gets

assigned a multidimensional vector that contains the world space coordinate of the nose as well as the angles of the head rotation. The blue spheres in the last-mentioned figure visualize all predictions while the green spheres are selected by finding the cluster, yielding the final estimate of the nose. The cylinder indicates the estimated face direction. Their experiments show that the approach is able to handle large pose changes, partial occlusions and facial expressions due to the high amount of annotated images which are fed into the learning process. Dantone et al. [10] propose an approach that estimates facial feature points on color images. They also formulate a regression problem but the measurement for splittings is based on the class uncertainty. In this way, each patch is assigned to a class label according to the distance to facial feature points. When patches terminate in leaves, the voting is conducted for each of these points. Furthermore, the authors present conditional forests where the global head pose is predicted such that trees can be selected which have learned predictions on the estimated pose, allowing to precisely vote for the location of all desired facial feature points.

(a) **(b)**

Figure 4: Head pose estimation. The example images are taken from [13] and [10], respectively. Both approaches estimate the direction to which a head points but (a) estimates the nose position as well as its orientation on depth image patches and (b) estimates the label of the head orientation and then several facial feature points, done on color images.

As shown in Figure 5, another prominent application of random forests is the estimation of body poses. Shotton et al. [45] use a representation in terms of several body parts, allowing to perform a per-pixel classification. After the body is classified, the approach finds the local modes which yield the position of skeletal body joints. Dantone et al. [11] operate on color images and use a classification as well as regression layer. The first layer acts as body part classifier where the forest estimates the probability that an image region belongs to a specific body part. The second layer takes the resulting class distribution and regresses body joints in dependence of the earlier classified body parts. In this way, the ambiguity of similar body parts is reduced because patches get more discriminative when predicting joints for legs or arms.

(a) (b)

Figure 5: Body pose estimation. The example images are taken from [45] and [11], respectively. Again, we show depth and color images where (a) performs a per-pixel classification of different body parts and (b) formulates a regression problem to predict the coordinate of several joint locations of the body.

Due to the success of the previously described predictions on head and body, the estimation of hand poses has also become popular, in particular for the research in human-computer interaction. The approach from Keskin et al. [25] uses synthetic depth images as shown in Figure 6 where the hands are separated into different parts. In the single-layer approach, the algorithm classifies each pixel, followed by clustering the estimations for each class label (hand part) which yields the predicted location of all hand joints. In addition, the authors propose a multi-layer approach that takes hand poses into account. Hence, the first forest assigns pixels to hand poses and directs them to the second forest which has specifically learned the prediction on that pose. Another recent approach is presented by Tang et al. [50] who formulate a regression problem for the hand pose estimation. Some results are shown in Figure 7 on realistic depth images. Compared to other approaches that learn a mapping from the patch appearance to the joint locations, their approach takes the whole hand as feature. The procedure can be considered as structured course-to-fine search. They learn the hierarchical topology of the hand where split nodes iteratively divide the hand into subregions until each joint is isolated from the others. In this way, each leaf node votes for the corresponding joint location. This allows to implicitly learn the kinematic constraints of hands.

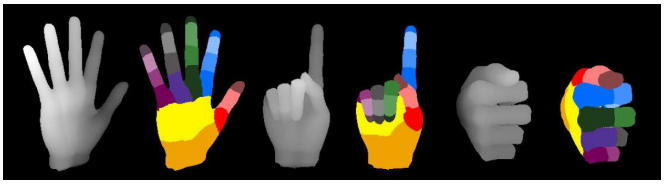


Figure 6: Hand pose estimation with synthetic images. The examples are taken from [25] and show some synthetically generated depth images and the corresponding groundtruth labels of several hand parts.

In our work, we investigate different solutions to build a random forest that learns the appearance of hands on depth images. The approach considers many important factors which need to be taken into account for estimations on sophisticated hand articulations. Our experiments show that we outperform the localization accuracy of state-of-the-art methods by several millimeters.

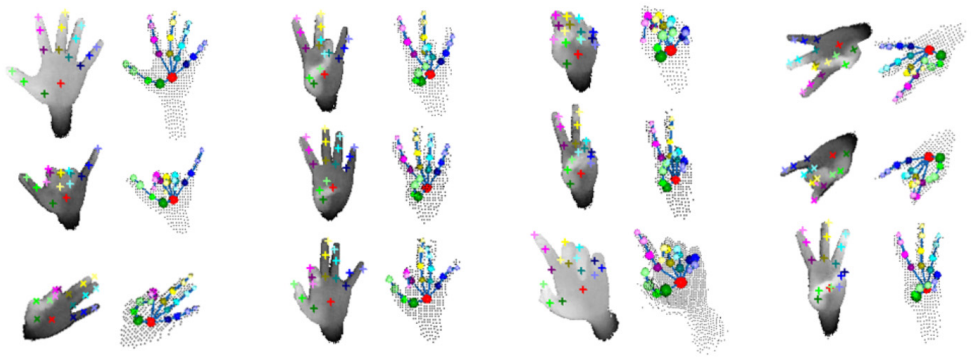


Figure 7: Hand pose estimation with realistic images. The examples are taken from [50] and illustrate different gestures. Each example shows the captured depth image with joint locations as colored crosses and another visualization with connected spheres where the pixels are projected from image space to world space.

1.4 Synopsis

In this section, we give a brief overview of our work by summarizing the following sections of this thesis. In Section 2, we focus on the ART system for post-stroke rehabilitation. The current system is described in Section 2.1 in terms of the hardware concept and the software components. We show several problematic cases and present our solution in Section 2.2 in detail. The adapted setup contains the segmentation process, the ellipse fitting approach, the camera calibration for the augmented environment and the workflow for fingertip tracking in the system. However, for a precise detection of several joint locations on the hand, we need a more powerful approach. Hence, in Section 3, we describe the datasets that are required in Section 4 for learning predictions with random forests. The preliminaries are described with comprehensible examples in Section 4.1, followed by Section 4.2 which discusses features and labels. Subsequently, we present our forest approach in Section 4.3 in terms of split/leaf nodes as well as the final joint detector. All experiments are described in Section 5 where Section 5.1 evaluates our solution for finding the pointing direction of the hand and Section 5.2 as well as Section 5.3 progressively evaluate our random forest approach. The final results are presented in Section 5.4 by comparing to a state-of-the-art approach and visualizing several predictions on depth images. In Section 6, we conclude our work with a summary and the most important remarks, followed by an outlook for future work.

2 Working Environment

In this section, we explain the fundamentals and the interface of our approach for post-stroke rehabilitation which consists of an experimental setup and a tracking solution for hands as well as fingertips. The adapted base system according to Section 2.1 describes the initial hardware concept and software components. This implementation is used as a starting point and extended with reliable functionalities in Section 2.2 such that an improvement can be reached in both, the general setup for therapy sessions and the accuracy for human-computer interactions.

2.1 Isolated Box System

A common method to rehabilitate stroke patients is the mirror reflection therapy with optical mirror boxes (see Figure 8 which shows a mirrored hand). The patient is sitting at a table and the mirror is placed approximately in the middle so that e.g. the healthy right hand is visible and the affected left hand is hidden. By mirroring the hand and giving the visual appearance that both hands move, the patient's brain is fooled into believing it sees both hands working [15], even if the other hand stays clenched behind the mirror. During therapy sessions, the patient performs various exercises to relearn hand movements.

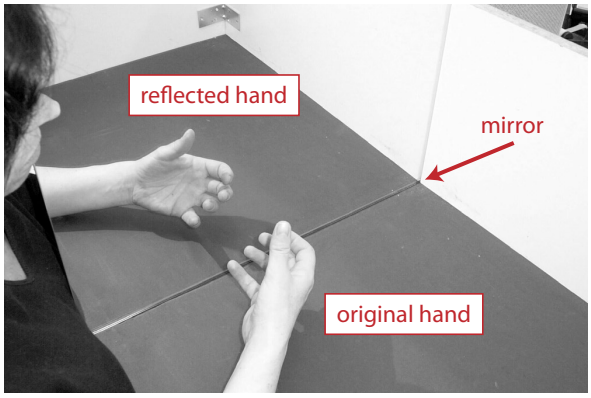


Figure 8: Demonstration of mirror reflection therapy. The picture is an adapted version from [46] and shows a person during a session. The affected limb (left arm) is hidden from the mirror while the healthy limb (right arm) is moving on the table. This creates the perception of bilateral movements because the movement of the left arm is simulated by the right arm.

The proposed ART system by Regenbrecht et al. [41] is an augmented mirror box (a hardware and software setup) which shows similarities to the optical mirror box and acts as base system for our approach. As described in Figure 9 and the next sections, the idea is to capture the hands of the patient as video streams and to display them on a computer monitor in real-time.

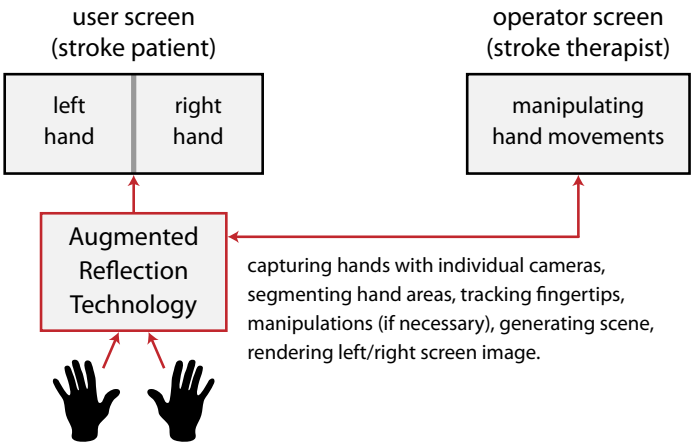


Figure 9: Sketch of the ART system. The real hands of the user are captured with cameras and then displayed on the screen. How the movements appear on the screen, is controlled by the operator who is able perform manipulations on another screen. The system also creates a virtual scene where the hands as well as other objects are rendered.

From now on, the therapy of stroke patients is not limited to mirroring hands but allows to manipulate the perception of reality. The captured hands are segmented and the fingers are tracked so that the therapist is able to control movements with a separate interface, allowing e.g. an amplification of weak hand movements or a complete simulation of movements or anything else. Lots of manipulations are feasible and also described in [38] but another important part of the presented system is the generation of a virtual scene. This allows to place further contents on the screen in addition to hands, resulting in an augmented environment with the possibility to implement various applications for therapy sessions.

2.1.1 Hardware Concept

The system consists of a standard desktop PC which is connected to both augmented mirror boxes, a wide-screen monitor for the patient and another screen with a keyboard as well as mouse for the therapist. The pictures in Figure 10 show the current setup. Each box contains a wide-angle webcam and several LEDs which emit a consistent lighting. The opening of the box is covered with a curtain so that both, the patients are not able to see their own hands and a controlled environment for capturing images is given. The patient's screen is placed above the boxes for viewing the hands while the other screen is placed beside and only visible to the therapist. Both cameras are mounted to the ceiling of the box (close to the upper middle of the curtain) and face downwards to the hand, yielding a realistic viewing angle as if patients would look at their hands on the table.

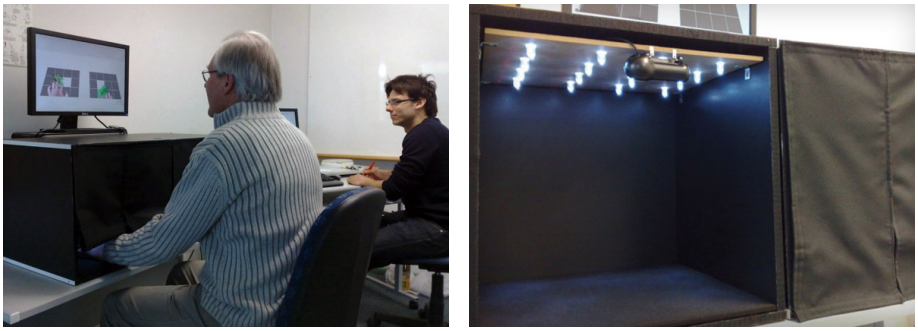


Figure 10: Therapy setup. The pictures are taken from [39] and show the current rehabilitation system with a patient and a therapist. The setup consists of several main parts such as the wooden boxes and the monitor which visualizes the hand within an application. The closer view shows the opened/closed box system with the light sources and the webcam which captures hand movements.

The whole ART system considers ethical aspects for post-stroke therapies because patients are sensitive to haptics or influences from the environment, hence the material of the fabric and the light source (heat source) within the box are chosen such that the patient feels comfortable. A clinical feasibility study for the system is presented in [22] where a specifically implemented application is evaluated for rehabilitation purposes.

2.1.2 Software Components

Apart from the hardware concept, the essential part is the interaction of several software components within the system. As described in [38], the main functions are (*i*) the controlling of video streams, (*ii*) the segmentation process, (*iii*) the tracking algorithm for the patient's hands or fingers, (*iv*) the provision and rendering of the augmented environment by use of a game engine in which (*v*) the box content is merged with the virtual scene and (*vi*) a manipulation of the reality is provided by (*vii*) a control mechanism as well as a GUI for the therapist. In the following, we want to focus on those components which are later replaced with our approaches, yielding a more reliable system without the need of boxes.

The contents, i.e. the hands and other parts in the box, are captured with individual webcams and the usage of the OpenCV library. In order to visualize hands on the screen (in particular in the augmented environment), the images are segmented using a background subtraction by brightness, yielding the hands as foreground area. The extracted hand is processed as RGBA color image where all other parts are set to full transparency. If the segmentation thresholds are adjusted to the skin color (bright or dark hands) and if the box curtain is fully closed, then this implementation delivers suitable results because of the controlled light conditions within the box.

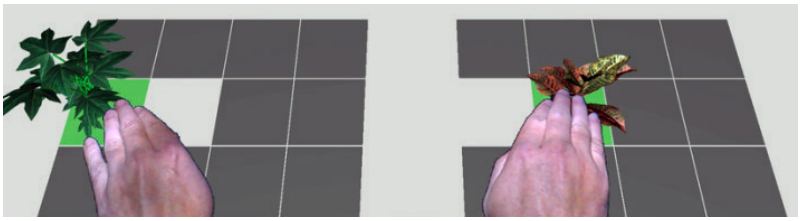


Figure 11: Playing the memory game. The picture is taken from [39] and shows a screenshot of the monitor. The hands are captured with cameras in the left and right box. After segmenting the hand areas, the system finds the pointing direction and opens a tile. If both tiles show the same plant model, the tiles disappear and the user can continue the search for other plants. The operator of the system is able to manipulate the displayed hand movements.

An exemplary augmented reality application for the ART system is the so-called TheraMem [39] which is similar to standard memory games with cards (see monitor screenshot in Figure 11). The patient controls the virtual game using only hands, hence no devices are needed to interact with the game. When the patient moves the hand over tiles (colored in grey) and holds the position for a predefined time, the corresponding tile flips (color changes to red) and shows a virtual plant model which stays activated (tile gets green). If both sides show the same plant, the matching tiles disappear from the screen. The game ends after finding all plant pairs. At each restart of the game, the plants are randomly assigned to tiles.

(a) **(b)**

Figure 12: Example manipulations. The pictures show the current system and exhibit hands on the monitor where (a) demonstrates the effect of a typical mirror reflection and (b) a reversed controlling of the hands. In both cases, the left hand is controlled by the right hand. Of course, during therapy sessions both hands would remain in the box.

The system has been in use for therapies since several years and the presented game allows to perform simple movement tasks. Patients are motivated during therapy sessions, not only due to the game play but rather because of the applied manipulations by the therapist. According to the shown example pictures in Figure 12, only the right hand stays within the box while the left

hand is visible on the screen. Manipulations are individually configurable for all implemented application, e.g. mirroring hands as well as a reverse controlling of hands. Furthermore, each hand can be manually controlled to spatially shift the vertical/horizontal position on the screen. The main feature is the automatic amplification of hand movements, i.e. the therapist has the opportunity to fool the patient's brain by amplifying the visual appearance, leading to larger movements even if the patient is only able to move the hand for a small distance.

For movement amplifications or interactions in applications, like the memory game, the system needs a tracking algorithm for the hand. The camera position is fixed and it can be assumed that the hand always enters from the same direction because of the defined box opening, hence the current approach tries to find the highest point (according to x/y coordinates) in the image plane. Each video frame is analyzed in terms of pixelwise comparisons, i.e. the algorithm searches top-down and left-right in the image until a segmented hand pixel is reached. If the pixel has more than a predefined number of neighboring pixels (concerning possible pixel artifacts due to the segmentation process), then this pixel determines the current pointing direction. The scenario is restricted but as shown in Figure 13, the approach works for most use cases and the tracking accuracy is sufficient for lots of simple applications. Obviously, some problems occur when rotating hands by a few degrees, then the result becomes less accurate or the algorithm delivers completely wrong points.

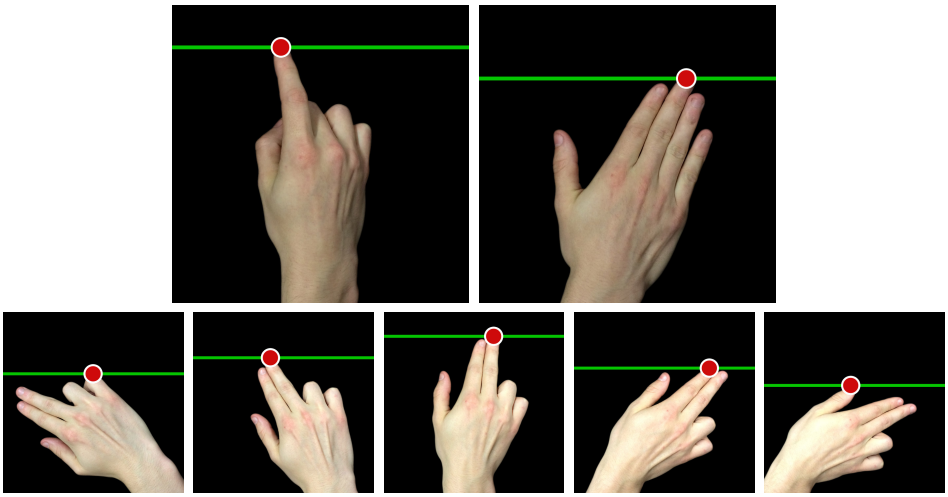


Figure 13: Illustration of the tracking approach in the ART system. The sample images in the upper row visualize correctly found points and the lower row shows the problem when rotating the hand.

To build the augmented environment, the segmented images and the pointing directions (pixel coordinates) are forwarded to the Unity Game Engine which provides a rendering engine for graphics as well as a physics engine. All other engines are currently not used. The scene configuration will be described later when calibrating cameras for the environment.

2.2 Extending the Base System

Our key task is to achieve an improvement of the ART system in terms of the therapy setup and tracking algorithms. The current system is already used in hospitals for post-stroke rehabilitation and several studies [22, 38] prove its feasibility and effectiveness of fooling the brain. Rather than using boxes, we show a new setup in Section 2.2.1 and a stable tracking solution in Section 2.2.2 which is based on color images and can be used for the presented memory game or any application which needs a pointing direction of the hand. In Section 2.2.3 we describe the camera calibration which is required to initialize the game engine. Determining precise fingertip positions is the essential functionality for future augmented reality applications, thus Section 2.2.4 addresses an out-of-the-box method that is used to find multiple fingertips with depth images.

2.2.1 Free Table Solution

The current solution with individual boxes has the drawback that a controlled environment must be created, i.e. such boxes need a lot of space and are difficult to transport as well as heavy. Due to the defined space inside the box, the system is limited to wide-angle cameras with a suitable field of view. Our initial experiments have also shown that the curtain causes troubles with the isolated room. If the vent of the curtain is not completely closed (due to movements of the arm), rays from different light sources outside the box can impair the segmentation process. We propose the setup as shown in Figure 14 where no isolation is required and a stable hand segmentation can be performed, resulting in several benefits for the further processing.

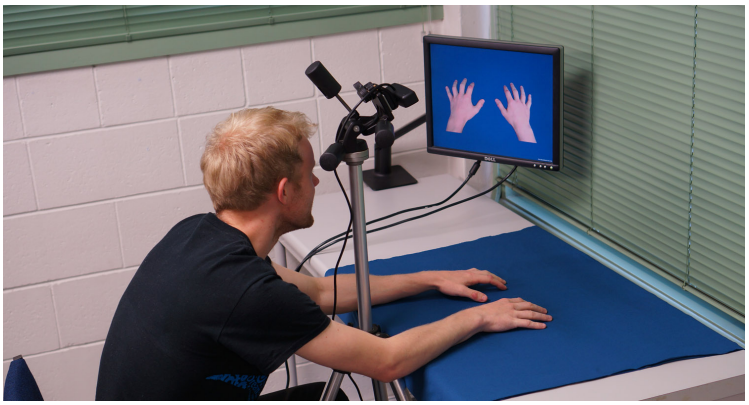
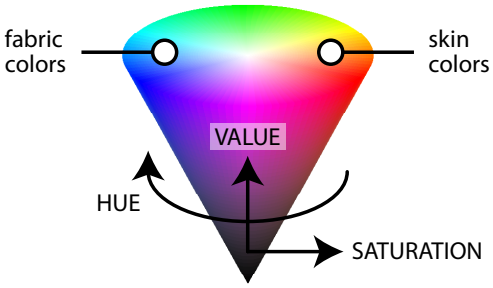


Figure 14: Experimental setup. Instead of the previously described configuration with boxes, the camera is mounted on a tripod and faces towards the table where a colored fabric serves as work space. The screen is mounted on a monitor arm and individually adjustable. Note, for a productive system, we would manufacture a suitable frame so that a curtain hides the arms.



pixel assigned to background if

$$H_{thres_1} < pixel(x,y)_H < H_{thres_2}$$
$$S_{thres} > pixel(x,y)_S$$
$$V_{thres} > pixel(x,y)_V$$

else foreground (hand area)

Figure 15: Color model for segmentation process. We convert the color image from RGB to HSV space and threshold each channel. Typical skin colors lie on the opposite side of our fabric color so that thresholds can be set roughly.

The isolated box system is based on RGB images where the hand is thresholded by the brightness because the inner surfaces are completely black. We use color images in HSV space and apply thresholds for each channel separately. Using OpenCV color notation, image pixels are defined as background if the hue lies between our thresholds $H_{thres_1} = 75$ and $H_{thres_2} = 135$ as well as at least a saturation of $S_{thres} = 25$ and brightness value of $V_{thres} = 10$ is given, otherwise pixels are marked as foreground. The color model in Figure 15 shows that typical skin colors are far away from the blue fabric and therefore simple assumption can be made. The thresholds are manually defined in our configurations and produce good results, even if the captured color varies in dependence of camera settings and light conditions. Also, the shadows from the arm do not cause any problem. However, an automatic background calibration by use of some initial video frames (without foreground hand) would be possible, then the median image of these frames serves as reference image for the hand segmentation.

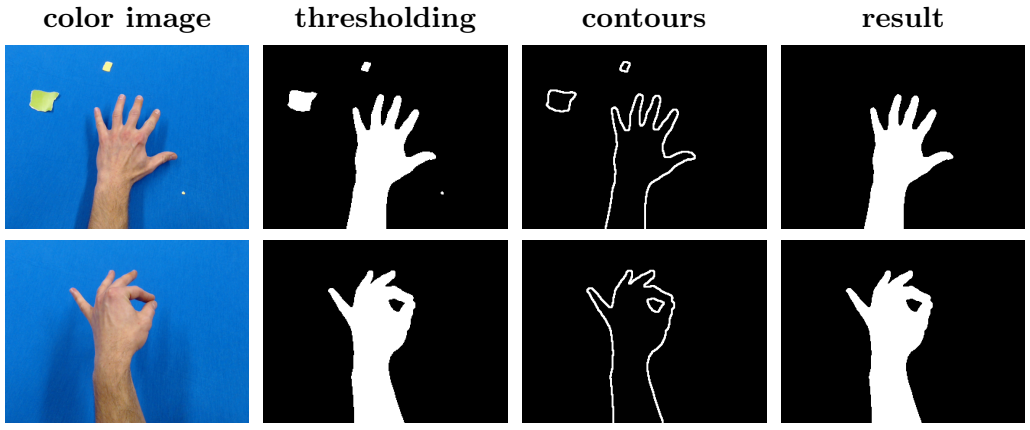


Figure 16: Examples for segmenting hands. We perform several operations to find the hand in the image. As shown in the upper row, small particles are removed and the largest area defines the hand. The lower row visualizes that also holes are handled during the segmentation.

The segmentation process delivers a mask on which additional operations are performed (see also Figure 16). At the beginning, we smooth hand contours and remove artifacts which can appear due to particles on the table. This is solved by a morphological operation where a 3×3 cross-shaped kernel is used for 2 iterations of erosion and dilation. We define the hand as the biggest object in the image. The usage of image contours allows us to compute object areas, even if holes are present. Small objects are removed for areas below a predefined threshold. Contours are handled in a hierarchical way, i.e. if a contour contains other areas, each inner contour is assigned to the outer contour. Finally, the hand can be found by iterating over a few remaining objects in the image mask.

Our implementation contains a split-screen mode where images are captured with a resolution of 1280×720 pixels from a single camera. Then the image is divided into a left and right half so that the approach works for both hands independently. When using the dual solution, each hand is captured with 640×480 pixels from individual cameras. The suitable configuration type will be evaluated in future studies.

2.2.2 Pointing with the Hand

The segmentation mask is used to visualize the hand on the screen and to find the pointing direction (the position to which the fingers are pointing). For stroke patients, it is hard to define a pointing gesture because some people may point with a fingertip and others may point with the whole hand. How well a patient is able to move the hand as well as fingers, depends on the rehabilitation progress. But, the algorithm should deliver an accurate estimation for any scenario, allowing to implement lots of different applications.

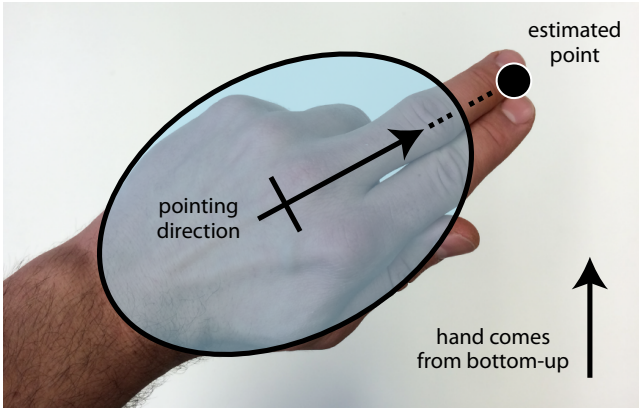


Figure 17: Ellipse fitting. The basic idea is to find the best fitting curve over the segmented mask because hand areas are elongated typically. Using our constraint that the point of view always remains the same, the pointing direction can be easily determined.

According to the memory game and similar applications for the ART system, we need a stable solution that finds the position to which patients are pointing. When looking at hands, the hand area with outstretched fingers resembles an ellipse shape (see Figure 17). Based on this assumption, we fit an ellipse around the hand and observe the orientation of the shape to estimate the pointing direction. To determine which side of the ellipse is the front and the back, we use the constraint that the arm can only enter from the bottom and moves upwards. Hands entering from the left/right side of the image cause no problems but the main orientation is a predefined parameter. This is no restriction to the system because the camera is already mounted close to the patient so that the point of view remains the same.

Our algorithm uses the binary mask of the segmented hand. Additionally, we remove the arm area so that mainly the hand remains in the image. Using the complete mask would also work because the image resembles an elliptical shape anyway. But, if the patient tilts the wrist, the precision is reduced because the significant information is in the hand area. To find this separation, we start at the image top and count the number of pixels in direction to the bottom until the empirical determined threshold $M_{thres} = 35000$ is reached. This threshold varies depending on the resolution of the image, the distance between camera and table, the hand size in general, the pointing gesture and whether the hand is captured in the upper or lower part of the image (due to the camera angle). The latter is solved by weighting each horizontal pixel line as a function of the image height. Since the clinical rehabilitation setup is done once and will not be changed, the factors for resolution and distance are not considered. Finally, the human hand size and the gesture are empirically found to be negligible. The use of a camera calibration or depth camera would provide additional methods to estimate the required hand area but the current approach already produces sufficient results.

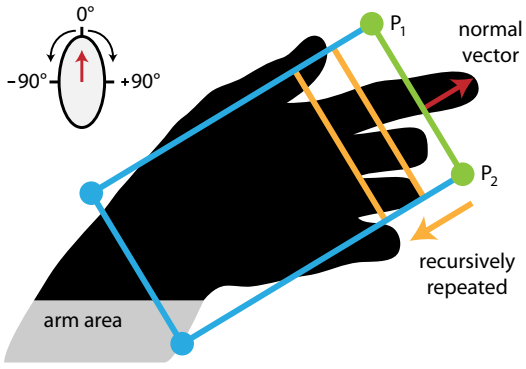


Figure 18: Steps for finding the pointing direction. The mask from the segmentation process is divided into the hand and the arm area. Our visualized rectangle is aligned to the computed ellipse. The orientation can be found due to the predefined side from which the hand enters and the vector perpendicular to the cutting line delivers the final point. If no intersection is found, the process is recursively repeated.

After finding the remaining hand segment (without arm, see mask in Figure 18), we fit an ellipse in a least-squares sense using the OpenCV library. The result is a rotated rectangle in which the ellipse is inscribed. Applying the constraint that the principal direction is predefined, we compute the normal vector

$$\boldsymbol{n} = \begin{pmatrix} +d_y \\ -d_x \end{pmatrix} \cdot k \quad \text{perpendicular to} \quad \boldsymbol{d} = \frac{P_1 - P_2}{||P_1 - P_2||} \tag{1}$$

of the rectangle, yielding the desired direction to which the hand is pointing. The required points in Equation 1 are defined as the highest corner point P_1 and its closest corner point P_2 from the rectangle. For each hand pixel that intersects with the cutting line $\boldsymbol{d}$ between these points, we use the normal vector $\boldsymbol{n}$ and compute the distance to the nearest non-masked image pixel. The intersection between the longest vector and the hand mask results in the final position for the finger. To determine the correct direction, we apply

$$k = \begin{cases} +1 & \text{if } \; 0 \leqslant \alpha < \dfrac{\pi}{2} \\ -1 & \text{otherwise.} \end{cases} \tag{2}$$

The rotation α in Equation 2 is known due to the description of the ellipse in the image. If no intersection with the cutting line can be found, the algorithm is recursively repeated by generating parallel lines which are closer to the hand center. Several illustrations of example images and a user study are shown in our experiments.

2.2.3 System Calibration

Applications for the ART system are built in an augmented environment where the virtual scene is rendered together with real hands. To generate this environment, the Unity Game Engine is used. The illustration in Figure 19 shows a side perspective of the main elements of our setup. A camera is pointing towards virtual objects (e.g. plants in the memory game) and video planes are placed between them. These planes are used to render the left and right color image which contains only the segmented hand because all other pixels are set to full transparency. In this way, the placement allows the hand to occlude any scene content. Concerning our configuration which splits the original camera image into halves, each plane shows a half part with the corresponding hand segments.

The system runs in real-time and the tracked hand or finger allows to trigger interactions with virtual objects in the scene. Similar to [40] we computationally project the tracked pixel position into our world coordinate system which uses metric units. Assuming that the captured space (our table with hands) is flat, the homography matrix [56] gives the relation between the image and world coordinate system as well as the augmented environment, after the system is correctly aligned.

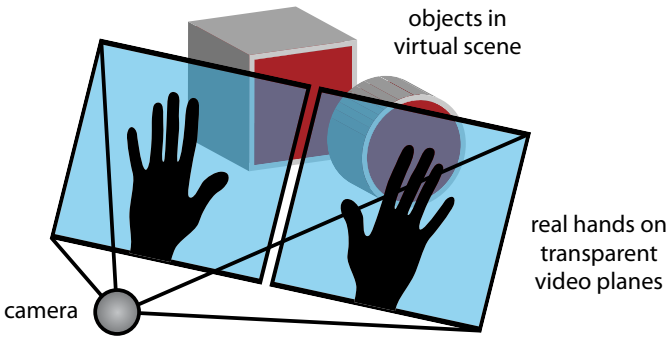


Figure 19: Illustration of our environment setup. The main elements are the video planes for rendering the captured images of real hands, the virtual objects placed in the scene and the camera. The images are rendered on the top of the scene so that the objects in the back are overlaid by the hands.

Based on our setup, we need a calibrated camera, i.e. the intrinsic and extrinsic camera parameters. The calibration is done once during a setup phase and need not be repeated, unless the configuration changes (position and orientation of camera). The functionality is added separately to the system and allows to configure the workspace in a few steps. We want to focus on our practical use case and refer the interested reader to Hartley and Zisserman [20] who describe the preliminaries in detail. As shown in Figure 20, a checkerboard with 9×6 corners yields the relation between image and world coordinates, i.e. the automatically found corners in the pattern (see colored points) correspond with manually defined world points where the z coordinate is initially set to zero (measured in millimeters). For intrinsic parameters, we capture 8 images of the rotated checkerboard from varying perspectives and process the point correspondences using homogeneous coordinates and the OpenCV library. The result is the camera matrix

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

with the focal lengths f_x/f_y and the principal point (c_x,c_y) as well as several distortion coefficients. Each camera has its own K but the matrix stays the same. The rotation matrix R and also the translation vector T belong to the extrinsic parameters which change if the camera is moved, hence

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad \text{and} \quad T = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{4}$$

are computed by placing the checkerboard on the middle of the table and finding the point correspondences as before. The camera faces towards the table middle and its image center defines the origin of our coordinate system.

Due to the lens distortion of every camera, we use the intrinsic parameters in Equation 3 to undistort images during runtime. The extrinsic parameters in Equation 4 allow transformations between world points and camera points because the camera has its own origin and coordinate system.

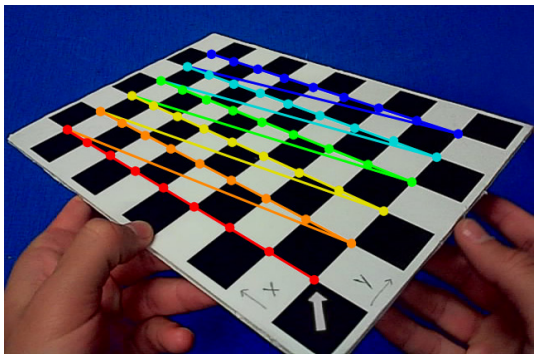


Figure 20: Calibrating the camera. To compute the intrinsic camera parameters, several frames from a rotated checkerboard are captured. Finally, the checkerboard is placed on the table so that the extrinsic camera parameters can be computed.

For the scene configuration, we also need to align the virtual camera with the real camera so that the objects appear at the same place in the augmented environment. Given a point P_C in the camera coordinate system, we can get the corresponding point P_W in our previously defined world coordinate system by applying the rotation and translation as

$$\begin{aligned} P_W &= RP_C + T \\ P_C &= R^T(P_W - T) \end{aligned} \tag{5}$$

where $R^{-1} = R^T$ for orthogonal matrices is performed to rewrite the equation. Thus, we can use the result of Equation 5 to reversely project each world point to the corresponding camera point. This transformation is applied to the virtual camera which is initially placed and oriented in the origin of the world coordinate system. To finish the system calibration, we perform

$$\begin{aligned} C_{position} &= R^T\left(\begin{pmatrix}0\\0\\0\end{pmatrix} - T\right) = -R^T T \\ C_{lookat} &= R^T\left(\begin{pmatrix}0\\0\\1\end{pmatrix} - T\right) \\ C_{upvec} &= R^T\begin{pmatrix}0\\1\\0\end{pmatrix} \end{aligned} \tag{6}$$

to get all required parameters for our augmented environment. As shown in Equation 6, we compute the real camera position $C_{position}$ by transforming the origin. The same computation is applied to the initial look-at point, yielding again the absolute coordinates for C_{lookat} to which the camera is pointing. The final up-vector C_{upvec} is obtained by applying a rotation only because no translation is needed for the camera orientation.

After these transformations, the camera is correctly placed and oriented. Finally, the rendering planes for the video streams are configured by triangulation because the camera's FOV and its position in the coordinate system are known. The camera is pointing perpendicular to the center of the planes and the distance between them is adjustable, allowing to configure the visual size of hands in the system. As a consequence, the size of the video plane can be computed, yielding our fully configured system.

2.2.4 Fingertip Tracking

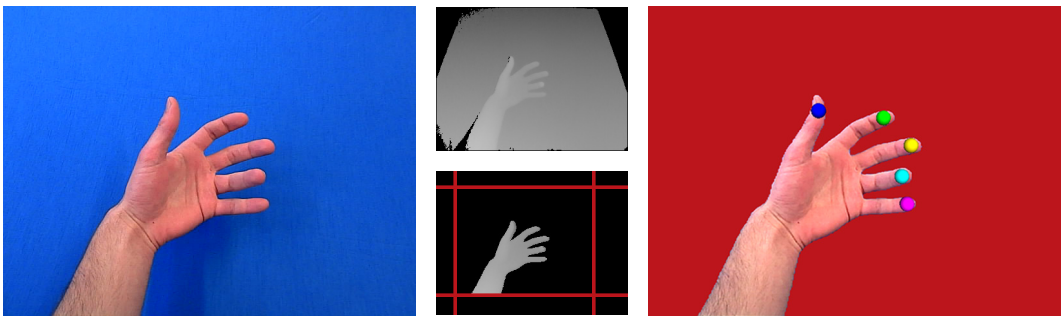
Our implementation is an advanced ART system and contains several methods that work together. For the segmentation of the hand area, we can switch between the established box approach and our free table solution. The ellipse fitting approach improves the system by finding the correct position to which the hand is pointing but it is restricted to simple rehabilitation applications. The implemented calibration procedure allows to build a virtual scene which is aligned to the real setup. The camera streams are forwarded to our scene configuration and rendered in real-time with virtual objects. In sum, the implementation is a complete system for post-stroke rehabilitation where various applications can be integrated. However, lots of such applications require more precise information about the hand to enable different interactions. When the position of fingertips in the image is known, we can implement applications which e.g. allow to grab virtual objects in the scene. Furthermore, therapists get the possibility to measure finger movements of patients. From now on, we want to focus on the recognition of fingertips, yielding additional functionalities for the presented system.

We use a time-of-flight camera which provides depth images in addition to color images. Depth values are more suitable for finding fingertips because hands do not provide a lot of texture and colors strongly vary in different light conditions. More detailed information about such images can be found in the next sections. Currently, we capture both camera streams in our implementation by use of the Intel PCSDK [2] which provides lots of out-of-the-box methods. Beside the streaming capability, the SDK is able to track various joints of the hand. For coming applications in the ART system, we only access the fingertips and forward the tracked positions to our augmented environment. Thus, now we handle several points that allow to interact with the generated scene content.

Using a depth camera also causes problems with our setup. Such consumer cameras capture the distance to objects with an accuracy of plus/minus several millimeters, i.e. if the hand is placed almost flat on the table, the camera is not able to recognize any difference in depth. Thus, no tracking algorithm would work for our system, unless the distance between hand and table gets sufficiently large.

We solve the problem by using the color image and our segmentation method, i.e. we again convert the RGB stream to HSV channels and create a mask. Since the cameras are calibrated, each background pixel of this binary hand mask is mapped to the depth image and sets the corresponding depth pixel to non-valid, yielding the segmented hand as foreground which is used for the subsequent tracking algorithm. The example in Figure 21 shows the smoothed raw data in the upper depth image and our segmentation result in the lower depth image. In addition to the mapping, we cut the borders of the depth image due to the offset between both camera centers and more important, the wider FOV compared to the color camera. Note, depth images are captured with half the resolution of color images, hence no artifacts occur due to the projection. In this way, we get an accurate segmentation which would not have been possible by applying a depth threshold.

The tracked fingertips are visualized by differently colored spheres where the colors (blue, green, yellow, cyan and magenta) correspond the fingertips from the thumb to the pinky. The algorithm works well when performing typical gestures where the fingers are extended and do not touch each other. However, the algorithm sometimes does not issue any result for fingertips and often wrong labels are assigned to them (see Figure 22). In particular, this happens when rotating the hand and showing different viewpoints. This limits the range of applications for our system because a stable fingertip tracking is the fundamental requirement.



(a) **(b)** **(c)**

Figure 21: Tracking with Intel PCSDK and our applied color segmentation. After capturing (a) the original image and performing the segmentation, we also apply the binary mask on (b) the depth image by mapping each masked pixel. Due to the different FOV of the color camera, the depth image border is also set to non-valid. Finally, we (c) visualize the tracked fingertips in our augmented environment by placing colored spheres where in this case, each color is correctly assigned to the fingertips.

To the best of our knowledge, no system on the market is able to precisely find fingertips in images, certainty not with consumer cameras and in real-time. Furthermore, our ART system addresses a special use case with a defined workspace so that algorithms may benefit from our constraints, i.e. we know the side from which the hand will enter or we can define in advance where the left and the right hand will appear.

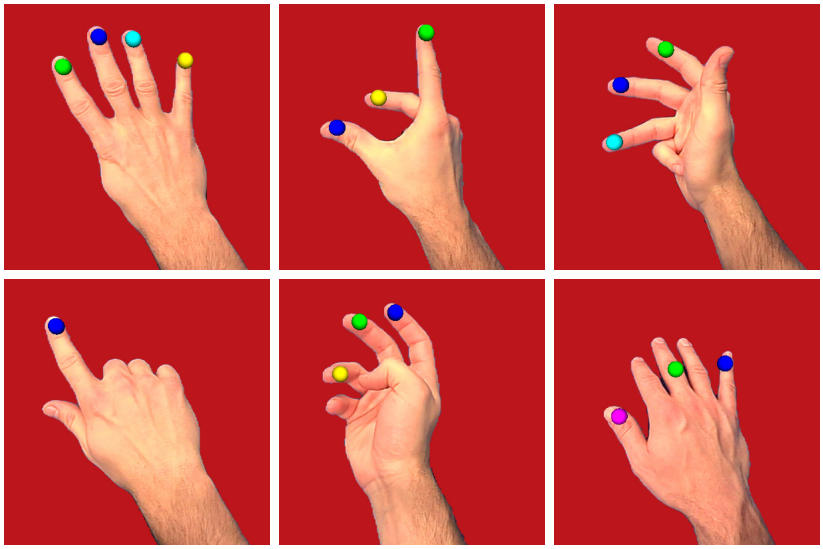


Figure 22: Further example results. The provided algorithm by the Intel PCSDK only works well if all fingers are extended and not too close together. When rotating the hand and showing different viewpoints, the algorithm fails as either lots of fingers are not recognized or wrong labels are assigned to fingertips.

In the following sections, we describe our approach for the estimation of joint locations on hands. The proposed algorithm is based on random forests which learn the appearance of human hands, in particular where joints are present in the image (in our case the fingertips). We need a lot of example data which can be fed into the learning process. Such datasets are described in Section 3 and contain depth images as well as annotated points. At this time, no dataset exists that covers the special use case of the ART system, i.e. the images deviate from our perspective to hands. However, there is no relation to the algorithm so that the dataset can be easily replaced later. In Section 4, we provide the preliminaries for random forests and a detailed description of the implemented methods which can be adapted to the presented rehabilitation system or any other system that depends on an accurate detection of joint locations.

3 Databases

This section introduces datasets that are in use for experiments with random forests. The datasets are described in their characteristic, i.e. the quality and quantity of the captured depth images, the groundtruth data as well as the split into training and test data. Depth values are mapped to the colormap in Figure 23, allowing a detailed visualization of images where annotation are marked with colored circles. The depth images are segmented using a depth threshold according to distance of the captured hand.

The hand illustrations in Figure 24 show the available annotations of the used datasets. These are the ICVL dataset and the ICG dataset which are described in Section 3.1 and Section 3.2, respectively. Finally, in Section 3.3 the differences in the datasets are discussed.

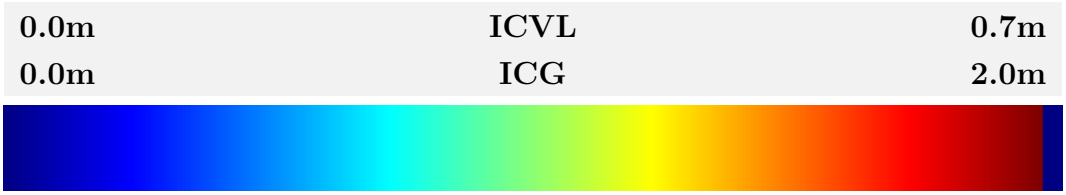


Figure 23: Colormap for depth images. For visualization purposes, each depth value is mapped to a color. Both datasets use the same colormap but the mapping is fitted to the respective metric range. Note, we visualize the segmented background with the same color that is in use for depth values equal to zero so that the foreground hand and its annotations are easier to distinguish.

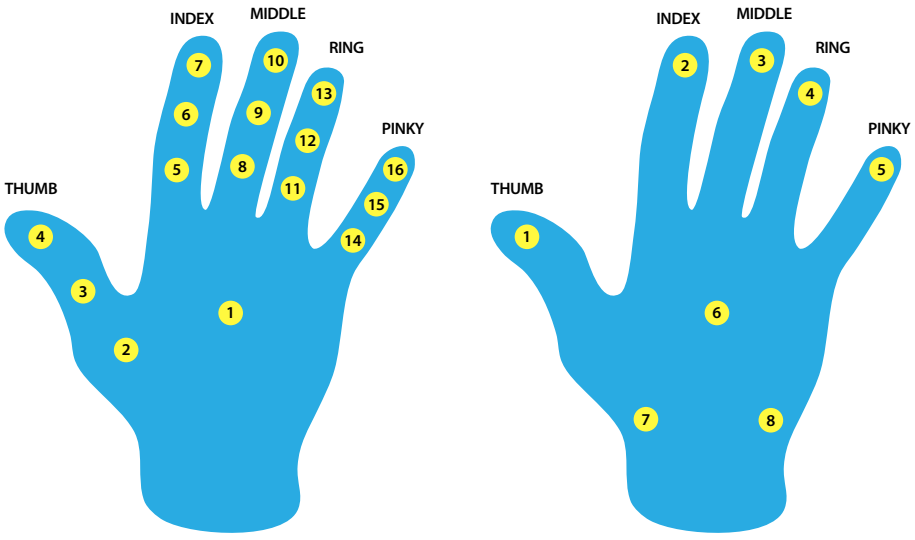


Figure 24: Dataset annotations. The datasets are labeled with several joint locations where the ICVL dataset (left side) and the ICG dataset (right side) show a different amount and numbering of annotations.

3.1 Articulated Hand Posture Dataset ICVL

The ICVL dataset [1, 50] is a set of depth images from several video sequences that show different movements of human hands. Each image, captured with an Intel Creative Interactive Gesture Camera, is of size 320×240 pixels and contains distance values in metric space. The sequences are collected from 10 different subjects that perform 26 various hand poses. A selection of samples in Figure 25 visualizes some depth images. The captured dataset consists of 22K images and by using the additionally provided image rotations, the number of training images can be enlarged many times over. For our experiments, we use the original depth images and 4 rotations ($\pm 22.5°$ and $\pm 45.0°$) as shown in Figure 26, resulting in 110K depth images. The test images are divided into 2 sequences that contain about 700 and 900 video frames. To cover a wide range of possible hand and finger movements, these sequences contain various poses at different scales and viewpoints.

The groundtruth data contain 3 joint locations per finger and an additional annotation in the hand center, yielding 16 annotations in total. The joint locations are not only placed on the hand surface but inside the hand at the half depth. This allows to learn the real joint locations in metric units.

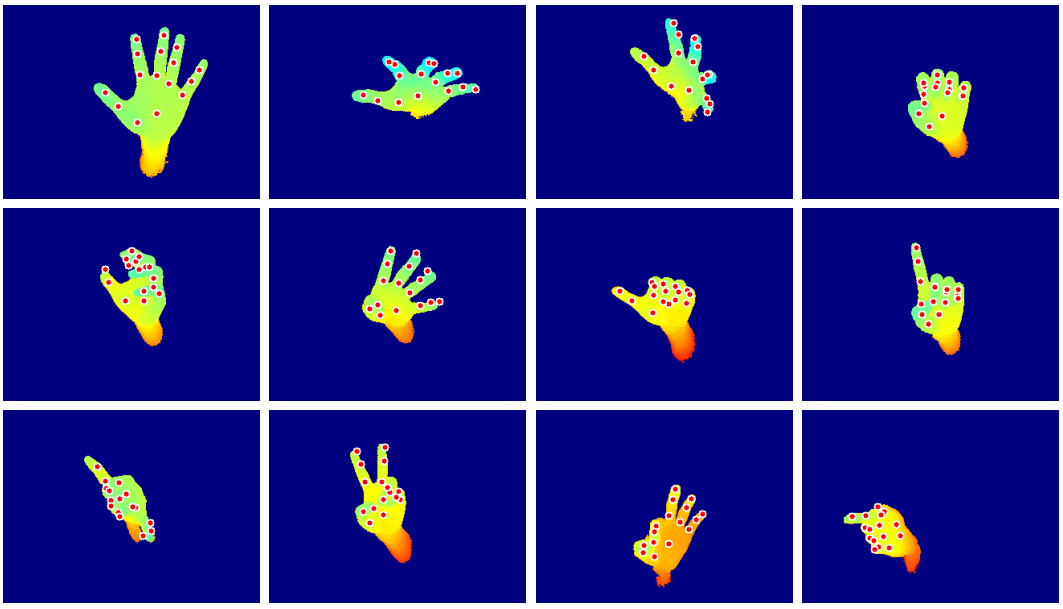


Figure 25: Example images from the ICVL dataset. The images are randomly chosen from the complete set of depth images and the circles indicate all joint locations from groundtruth.

−45.0° −22.5° original +22.5° +45.0°

Figure 26: Rotated depth images. Each captured depth image (figure in the mid) can be additionally in-plane rotated so that several rotations are covered as shown on the left/right side. The complete dataset also contains further rotated images (±67.5°, ±90.0°, ±112.5°, ±135.0°, ±157.5° and 180.0°).

3.2 Articulated Hand Posture Dataset ICG

The ICG dataset (not published yet) presents depth images from a scene where people move their hand in front of the camera. As depth camera, an Asus Xtion Pro Live Motion Sensor is in use, thus the resolution of depth images is 160×120 pixels. Some examples can be found in Figure 27 with additionally provided color images. Note, depth images are taken from a different viewpoint and no warping to color images is applied.

Figure 27: Example images from the ICG dataset. Each image is randomly chosen from the dataset and contains all visible (not occluded) joint locations. In addition to depth images, we provide color images for a better interpretation of the scene.

For training, the dataset is divided into 17 gesture classes, each of them with about 120 video frames. These sequences are collected from 8 people, with the result of almost 1K depth images per gesture and 15K depth images in total. At the testing stage, another person is evaluated with the same amount of different gestures and depth images.

The next difference to the previously presented dataset is the additional label for occluded annotations. Especially for fingers, a self-occlusion can occur many times while showing different poses or rotating the hand, hence this can be a useful information. The groundtruth data consists of 8 joint locations, containing 5 fingertips as well as the palm mid and 2 other annotations on the heel of the hand. But, when considering occlusions, less might be used because the finger is hidden behind another part of the hand. The annotations have been created in image space, hence the corresponding world coordinates can be computed with the depth value and the camera calibration matrix.

3.3 Dataset Comparison

Both datasets show different hands from different people and all fingertips as well as further joint locations are marked with annotations. However, it is not possible to combine the datasets because the frames are captured with different depth cameras and the captured scene is varying. In addition, the joint annotations do not correspond to each other and more important, the different image noise level would cause troubles. Our implemented random forest is able to handle any dataset, each of them with its own parameters and label information, hence we do evaluate each dataset separately.

The main difference in the used datasets is the provided information about gestures and occlusions. While the ICVL dataset just consists of depth images with the corresponding joint annotations, the ICG dataset is separated into gesture classes and all joints contain an additional flag about self-occlusions. Depending on the approach, these labels can provide benefits because more information can be fed into the learning process.

A comparison of Figure 25 and Figure 27 shows another important difference. The ICVL dataset has a well segmented foreground which contains the hand and a negligible part from the arm. The ICG dataset shows a room and almost the whole person as background so that the hand area consists of only a few pixels. The latter dataset is also much more noisy and in combination with the tiny hand, the depth values at the annotated joints vary much more. This yields less accuracy, in particular when the annotations are not exactly placed on fingers.

4 Random Forests

Detecting fingertips or even several joints on hands is a relevant task for many applications. Our related use cases are grabbing of objects for human computer interactions as well as the measurement of movement progresses. This requires a precise detection but lots of divers finger articulations lead to a high variety of different hand appearances and an image which is difficult to interpret. The next sections present random forests that split the complex problem into a set of smaller ones. The preliminaries for the algorithm are explained in Section 4.1, followed by Section 4.2 which describes the features and labels that are used for the learning process. The implementation details in Section 4.3 explain our approach progressively, beginning with splitting criteria and leaf node statistics for patches and ending with the joint detector. We extend our approach with feature maps which are generated by another forest and later additionally used for feature channels in combination with depth images.

4.1 Decision Trees are getting Randomized

Random forests are ensembles of differently learned decision trees. Similar to graphs, a tree consists of nodes and edges but no loops are possible, thus there is no way back and each path is unique because no other way terminates at the same place. Decision trees start at the top with the root node and grow to the bottom where several nodes store a statement. For the set of input data, each split node performs a binary decision so that the output can be passed to the following left or right node. This process is iteratively repeated until a predefined stopping criteria is reached, then the node becomes a final leaf node. Randomness is introduced in terms of multiple trees with randomly optimized nodes and randomly chosen sample data so that generalization [4, 21] on new unseen data can be achieved. Trying lots of different split parameters may demand time but allows to optimize trees. To compute the final result, the individual leaves are collected and combined.

4.1.1 Tree Structure

Although we want to automatically learn how to make the best decisions, the illustration in Figure 28 shows an exemplary decision tree where each node is manually defined. A tree might be able to separate a set of different animal images. For simplicity, we only evaluate the color of pixel areas, even if such algorithms would not be stable enough to correctly handle all kinds of situations. If penguins are typically present on beaches, then split nodes may check known image contents. When looking at the upper part of the image, the area has to be blue because of water or sky. Additionally, the lower part has to be brown

because of the sand. And, if the the middle part is mainly black and white, then we store in the corresponding leaf node that a penguin is visible on the image. Every similar image will end in the same leaf and images with other animals would take a different way, terminating in another leaf that classifies another scene. If penguins are also present on green grass, then other split nodes may check such appearances so that another leaf node does the correct classification. We could extend the tree with heaps of questions because each split helps to analyse the image content. When less questions are sufficient, leaf nodes can also be created earlier as shown in the illustration. However, generating such questions by hand would expend a lot of effort, thus trees should learn the best decisions by use of labeled datasets.

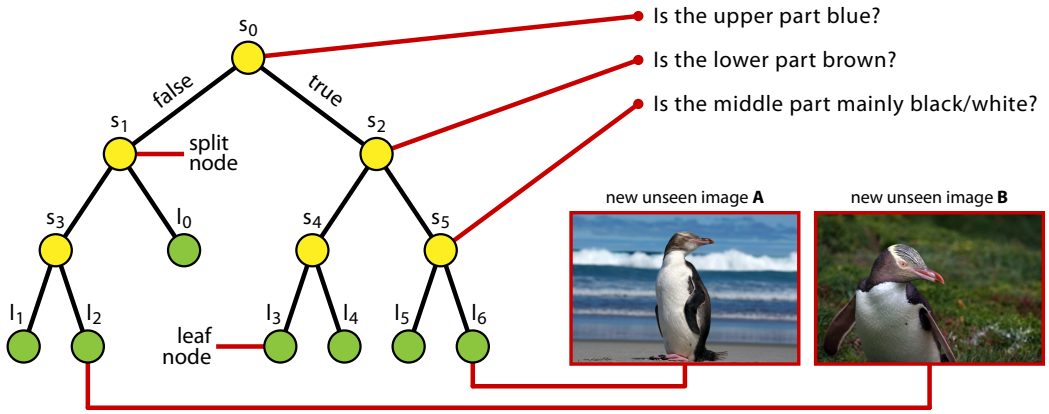


Figure 28: Hierarchical tree structure. Every split node asks a question, allowing to pass images to the left or right (depending on the answer which can be either false or true). Similar images end up in the same leaf node so that properties for new unseen images can be concluded. Even if both illustrated images show a penguin, the way down the tree might be different.

The discussed example is based on classification trees [23, 26, 43] which allow to classify discrete image contents. Another field of application is the prediction of e.g. coordinates of a particular point in the image. This is done by regression trees [9, 48, 55] that produce a continuous result, i.e. vectors to predicted points. Decision trees can be used for multiple machine learning problems but they are always structured with hierarchically ordered nodes. Learned trees store split functions which can be applied to data samples, and leaf nodes store the information to estimate the final result.

4.1.2 Training and Testing

The functional principle can be separated in the training phase, i.e. learning with image datasets as well as the corresponding groundtruth data and the testing phase that uses new unseen images. We want to focus on random forests

for classification and regression problems where data points are sent through the trees. When trees are tested, each input data point $\boldsymbol{v} \in \mathbb{R}^D$ passes several split nodes until a leaf node is reached which predicts a class label $c \in C$ or an output vector $\boldsymbol{w} \in \mathbb{R}^D$ for the regression task. The goal during training is to find split parameters that best separate the dataset $\mathcal{P}$ at the parent node into the sets $\mathcal{P}_L$ and $\mathcal{P}_R$ for children as shown in Figure 29 by the dashed line. For this purpose, a certain number of randomly chosen thresholds is generated so that a measurement between the parent node and child nodes can be applied. As described by Criminisi and Shotton [8], the information gain

$$\mathcal{IG} = \mathcal{H}(\mathcal{P}) - \sum_{i \in \{L,R\}} \frac{|\mathcal{P}_i|}{|\mathcal{P}|} \mathcal{H}(\mathcal{P}_i) \tag{7}$$

measures the uncertainty which is defined by the entropy $\mathcal{H}$ for each participating node. The term $|.|$ in Equation 7 denotes the cardinality of nodes. Finally, the threshold that delivers the best score for the information gain is assigned to the split node.

Data points for classification trees are labeled with classes, thus we are able to compute the probability distribution $p(c)$ as normalized histogram. As mentioned in Figure 29, the parent node contains the same amount of data points from all classes, hence the Shannon entropy

$$\mathcal{H}(\mathcal{P}) = -\sum_{c \in C} p(c) \log\Big(p(c)\Big) \tag{8}$$

gives bad scores because of low probabilities. After splitting the parent node, its children contain less classes, resulting in high probabilities and good scores. Thus, using the entropy according to Equation 8 for the information gain indicates that the children are pure according to their class labels.

In case of regression trees, the data points are related to annotations because the groundtruth data just contain points from which trees have to learn predictions, thus no histograms can be generated. Solving regression problems refers to finding clusters so that the approximated density for the differential entropy

$$\mathcal{H}(\mathcal{P}) = \sum_{\boldsymbol{v} \in \mathcal{P}} \log\Big(|\Lambda_{\boldsymbol{w}}(\boldsymbol{v})|\Big) \tag{9}$$

allows the computation of the information gain in the same way. Instead of counting class occurrences, now a multivariate Gaussian distribution with the covariance matrix $\Lambda_{\boldsymbol{w}}$ [33] is fitted to the data points. For Equation 9, the illustration in Figure 29 visualizes the density in the parent node (high entropy) compared to densities in child nodes (low entropies). The objective for such split nodes is to produce peaky Gaussians for the children. In a simplified manner, the score can also be achieved by using an objective that reduces child variances, resulting in a faster computation time compared to covariances.

The general procedure is to recursively perform splittings until a defined stopping criteria is reached, then a leaf node is created. This means that after finishing the node optimization for splits, all the data points are separated into different leaves. The forest uses the information in leaves and creates statistics which can be used during testing because similar data points will always terminate in the same leaf node. For classification and regression, the average of class labels or vote vectors, can be used to predict the final result on new unseen data points. If several trees are used, the individual leaf statistics are aggregated to the final result.

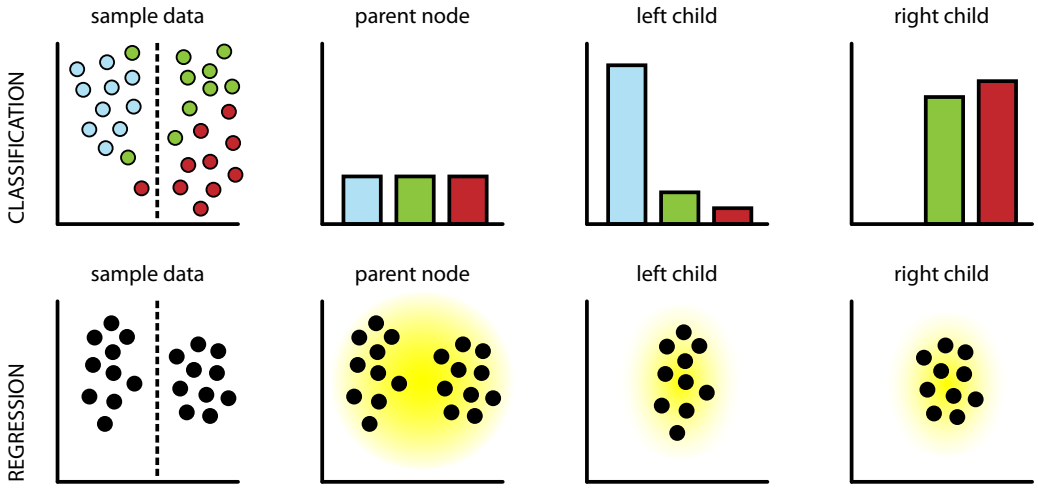


Figure 29: Splitting data points. For classification, a threshold (see dashed line) has to be found so that the child node become as pure as possible. When using regression, no class labels exist but Gaussian-based models can be used to compare the density of data points in children.

Forest are often used for real-time applications because trees can be trained as well as tested in parallel on multiple cores/threads and sending data through split nodes in trees is fast. To decrease the amount of gathered data in leaves, the estimation of labels or vectors is generated at training time so that the computation time for testing phases is reduced. The correct choice of forest parameters and the randomness factor play a central role to make reasonable predictions.

4.2 Features and Labels

A feature is just a collection of data which describes a specific situation. Pretty much each application similar to ours uses a camera to capture the scene at a specific moment. This might be a color image, a depth image, a response of a chosen filter bank or anything else. Due to provided label information in the learning process, the forest is able to find previously seen image parts again.

4.2.1 Using Depth Images

Pixels in depth images store the distance to objects in the captured scene. Compared to color images, a depth camera produces similar images in different light conditions but also the information about depth differences might be more useful than a colored hand with homogeneous texture regions. However, there exist several approaches [24, 29, 30] that rely on color images for hand/finger estimation. When using a depth image, the hand in the front can be easily segmented by a background subtraction with depth thresholds. The illustrations in Figure 30 show some examples where the hand and just a short part from the arm are the only segments in the depth image. But, the main advantage of features from such images is the possibility to distinguish between highly diverse depth regions on articulated hands. As shown on the last-mentioned figure, a hand looks extremely distinct when the viewpoint changes. Furthermore, occluded parts are prevalent when performing hand rotations or all kinds of gestures. Even if several self-occlusions occur, there is a lot of depth information between each individual finger, resulting in distinct features. Depth images are in use for e.g. body part [11, 45] and head pose estimations [10, 13] but compared to hand poses, the variety of depth images is less individual. Hence, the hand posture datasets have to cover all these articulations.

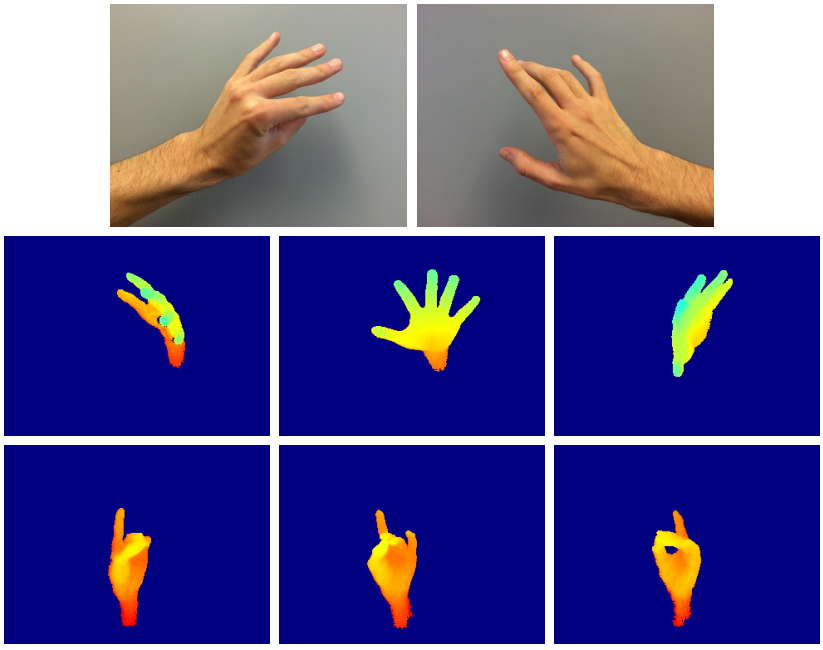


Figure 30: Exemplary color images and depth images. A captured hand produces similar color values over the whole image, resulting in textureless regions. Further problems are illumination changes and shadows, especially at finger overlaps. The depth image sequences show a hand at different viewpoints with several self-occlusions. Depending on the hand articulation, each finger can occlude other parts of the hand but the camera captures the difference in depth.

4.2.2 Learning from Datasets

During the learning process, the labels are used for splitting the data samples and then stored in leaves so that a detection of desired locations in new unseen images can be performed. According to the datasets described in Section 3, each depth image contains marked feature points that refer to these locations. They are called joint locations concerning the human hand structure or just annotations in general. Depending on the used dataset type, the fingertips and further joints on the hand are annotated with 8 to 16 points. Labeling processes are manually done by clicking trough all video frames, or automatically by using other tracking based systems. But, such algorithms are not suitable here because fitting a hand model to any human hand articulation is a hard task.

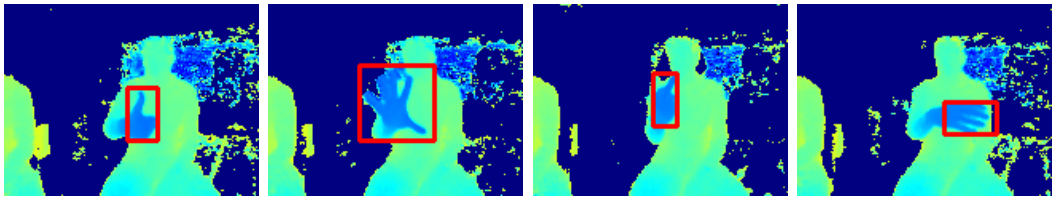


Figure 31: Classification of patches. The bounding box is computed by use of the annotations where the interior region defines the foreground and the exterior region the background, respectively.

Depth images from the ICVL dataset are well segmented and only a regression task is performed where the labels work as targets. The ICG dataset shows much more from the surrounding scene and also contains other parts in the images. As a consequence, we additionally perform a classification between these background parts and the foreground hand (see Figure 31). To divide the depth image into classes, a bounding box around the groundtruth annotations is used. This can be done in image space by finding the minimum and maximum coordinates over all annotations. Additionally, the bounding box is enlarged by a small percentage so that the whole hand is within the box. In this case, each extracted image patch is labeled with the corresponding class, allowing to learn the difference between foreground and background.

4.3 Regression Forest for Joint Localization

The goal of our work is to precisely predict coordinates of joint locations on hands. We train a random regression forest [18, 19] on a huge dataset that contains lots of different depth images and groundtruth annotations. At the beginning, various patches are extracted from the set of depth images and forwarded to the root nodes in the forest (see Figure 32). The distance as well as direction between patch and joint locations are known so that trees can find the best parameters for split nodes. Patches are passed towards the left

or right child until a leaf node is created where the patch terminates. The leaves evaluate all collected patches and store a real-valued vector for each joint location $n \in \{1, 2, \ldots, N\}$ computed from the annotated depth image. When testing the forest, similar patches will end in the same leaves so that these vectors are able to vote for joint locations in new unseen depth images.

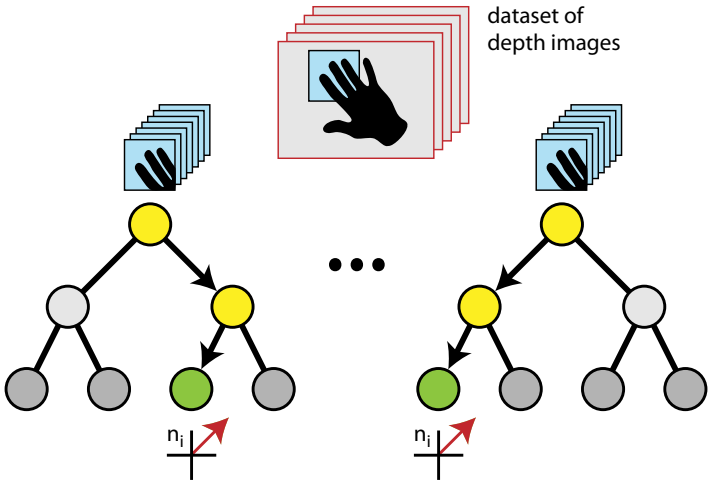


Figure 32: Example of random regression forest. Several slightly different trees are trained with a dataset of depth images. Split nodes direct all extracted patches through the tree until a leaf node is reached. In the testing phase, the corresponding leaves predict joint locations by their stored vote vectors. Aggregating these vectors yields the final result, separately for each joint.

To learn the estimation of desired coordinates, we use regression objectives in split nodes as well as aggregation methods in leaf nodes. Depending on the captured content in depth images, we also perform a classification of foreground/background to distinguish between the hand and the scene. Hence, split nodes randomly choose the regression or classification objective but the forest only votes for joint locations when leaves mainly consist of hand patches. In the next sections, we describe our implementation for split optimization and aggregating leaf votes, allowing to precisely detect all joint locations.

4.3.1 Patch Comparison Features

The learning process is supervised, thus we load a dataset of depth images that are annotated with several joint locations. Each tree in the forest is built from a set of square patches $\{\mathcal{P}_i = (\mathcal{I}_i^f, \boldsymbol{\theta}_i, c_i)\}$ that are randomly sampled from the complete dataset. The visual features $\mathcal{I}_i^f$ are extracted from different feature channels but currently the depth images are used only. The set of real-valued vectors $\boldsymbol{\theta}_i = \{\theta_{i_x}^1, \theta_{i_y}^1, \theta_{i_z}^1, \theta_{i_x}^2, \theta_{i_y}^2, \theta_{i_z}^2, \ldots, \theta_{i_x}^N, \theta_{i_y}^N, \theta_{i_z}^N\}$ describes the offsets between patch center and joint locations, separately for each dimension.

The class label $c_i \in \{1, 2\}$ indicates the class affiliation of patches, either the foreground or the background. Hence, the input data for forests consists of classified depth image pixels and the offset vectors as annotations. This information is used to learn forest parameters so that new unseen images follow the same tree path and produce the desired output.

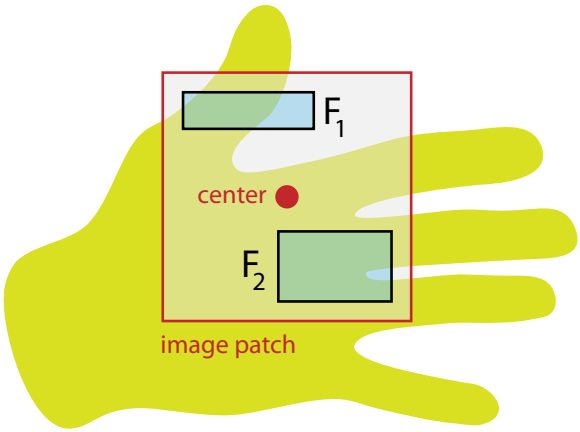


Figure 33: Example of extracted image patch. Binary tests are applied on randomly chosen rectangular areas within image patches. The response of a patch is defined as the difference between the mean values of these areas.

Split nodes separate patches so that the entropy $\mathcal{H}(\mathcal{P})$ for child nodes gets as low as possible. At the beginning, we need a patch comparison feature which evaluates similarities of patches so that a split node is able to direct all patches to its children. The simplest comparison would be a single pixel test on patches where a pixel coordinate $\boldsymbol{v} \in \mathcal{I}^f$ is randomly selected within the patch boundaries, and each $\mathcal{P}_i$ is passed to the left child if $\mathcal{I}_i^f(\boldsymbol{v}) < \tau$ is satisfied, otherwise patches are passed to the right child. This requires a threshold τ which is also randomly chosen. The approach can be extended with pixel pair tests as used in [31] where the difference between pixel $\boldsymbol{v}_1 \in \mathcal{I}^f$ and pixel $\boldsymbol{v}_2 \in \mathcal{I}^f$ is compared with the threshold. We define the patch comparison feature similar to [9, 14, 18] as

$$\left(\frac{1}{|F_1|} \sum_{\boldsymbol{v} \in F_1} \mathcal{I}^f(\boldsymbol{v}) - \frac{1}{|F_2|} \sum_{\boldsymbol{v} \in F_2} \mathcal{I}^f(\boldsymbol{v}) \right) < \tau \tag{10}$$

where F_1 and F_2 describe the randomly chosen rectangles within image patches. This makes binary tests less sensitive to noisy depth images (see also Figure 33). The subtraction in Equation 10 is computed from the mean values of both rectangular areas. Such patch comparison features are similar to Haar features as used in [53] where the sum or difference of multiple predefined rectangles is computed. We use generalized Haar features for our approach where the size and position of rectangles is arbitrary. One may think that the computation time for summing up rectangles is high but we store depth images as integral

images, hence the comparison of feature responses can be computed in constant time. The problem with fixed rectangles for binary tests in split function is the scale. On the one side, the size of human hands varies in general, and on the other side, the camera produces differently sized hands in dependence to its distance. The proposed method in [45] is scale invariant for pixel tests because a normalization term (the depth value at the patch center) is used to weight the relative offset of pixels within patch boundaries. Currently, we do not consider depth invariances but our dataset covers lots of different scales anyway. This allows to learn different hand sizes but normalized binary tests may reduce the number of required samples.

4.3.2 Split Evaluators

We have already described that the response of image patches is computed by the difference of randomly chosen rectangles, thus patches with low responses are passed to the left and patches with high responses are passed to right. The chosen parameters F_1 and F_2 as well as threshold τ and the selected feature channel are evaluated with the entropies $\mathcal{H}(\mathcal{P}_L)$ and $\mathcal{H}(\mathcal{P}_R)$ where $\mathcal{P}_L$ and $\mathcal{P}_R$ define the sets of image patches in the left and the right child node, respectively.

The forest is built following the random forest framework by Breiman [4] where at each split node, a pool of splitting candidates $\phi = (\varphi, \tau)$ with $\varphi = (f, F_1, F_2)$ is generated. For each φ at the node, the parameters are randomly chosen regarding image patches, i.e. rectangles are selected within patch boundaries and the feature is selected from the set of channels. After computing the responses for all patches, the set of randomly chosen thresholds τ with values between the minimum and maximum response of all participating patches can be generated. The next step is to direct patches to child nodes for each threshold, followed by the computation of entropies and selecting the threshold that delivers the best entropy. This procedure is repeated until all candidates ϕ are evaluated. The objective is to select ϕ^* that has achieved the best score. Note, the computation of the information gain $\mathcal{IG}$ is redundant because the entropy $\mathcal{H}(\mathcal{P})$ of the parent node (the current split node) is a constant, i.e. we only compute the weighted sum of the child node entropies to evaluate splitting candidates.

To achieve a performance enhancement, we sort the generated thresholds and patches according to their responses in advance. The parent node directs all the patches to the left child, followed by successively moving patches to the right child until a threshold is reached. The benefit of sorting the data is that the objective function can be evaluated on-the-fly. Thus, after computing the entropy for a certain threshold, the parent node continues with moving patches and incrementally computes the partial result for the entropy. This allows a more efficient computation when evaluating lots of different split parameters.

The variance is a useful measure to determine the compactness of labels in a node. Each patch knows the offsets to all joint locations and all these vectors should point to the same coordinates. If the variance of offset vectors is high, the coordinates are scattered which indicates a bad splitting. A low variance implies that points are close together. The corresponding score

$$\mathcal{H}(\mathcal{P}) = \sum_{n=1}^{N} \sum_i \|\boldsymbol{\theta}_i^n - \overline{\boldsymbol{\theta}^n}\|^2 \quad \text{with} \quad \overline{\boldsymbol{\theta}^n} = \frac{1}{|\mathcal{P}|} \sum_i \boldsymbol{\theta}_i^n \tag{11}$$

can be computed as the sum of all joint variances where $\|.\|$ defines the Euclidean distance between the mean value $\overline{\boldsymbol{\theta}^n}$ and the offset vector of a patch in the node, separately for each individual joint location.

Another method is to model the offset vectors with a multivariate Gaussian distribution as described in [9] where covariances are used to compute the entropy. However, the reduction of variances in Equation 11 is proportional to the differential entropy [17] and more efficient to compute. Hence we evaluate this regression objective in our experiments. The authors in [10] propose to use the class uncertainty measurement for the regression objective, therefore they define the entropy as

$$\mathcal{H}(\mathcal{P}) = -\sum_{n=1}^{N} \frac{\sum_i p\left(q_n|\mathcal{P}_i\right)}{|\mathcal{P}|} \log\left(\frac{\sum_i p\left(q_n|\mathcal{P}_i\right)}{|\mathcal{P}|}\right) \tag{12}$$

$$p\left(q_n|\mathcal{P}_i\right) \propto \exp\left(-\lambda \, \|\boldsymbol{\theta}_i^n\|\right) \tag{13}$$

where $p\left(q_n|\mathcal{P}_i\right)$ indicate the probabilities which are assigned to extracted image patches. In this way, each patch belongs to each joint location according to a certain probability. The illustrated hand in Figure 34 visualizes the class affiliations which are set depending on the distance to joint locations, i.e. the exponential function in Equation 13 results in $p\left(q_n|\mathcal{P}_i\right) = 1$ for a patch $\mathcal{P}_i$ that is directly located at the n-th annotation, or goes towards zero for patches that are extracted far away. The steepness parameter λ controls how fast the function falls, while $\lambda = 0$ disables the class assignments in Equation 12 because each class would get the same probability. This allows to use a discrete set of classes but the class affiliations to each joint is set continuously.

We stop the growing process, when the maximum tree depth is reached or the node contains less then a minimum number of patches. Then the node becomes a leaf and all the patches within the node are evaluated to predict joint locations.

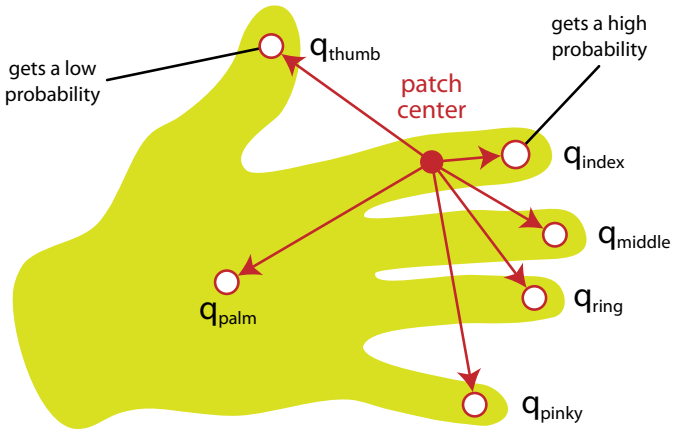


Figure 34: Example of regression objective with class affiliations. Several class labels are assigned to each patch. The probability for classes is set depending on the distance between patch center and joint location. The illustration only visualizes fingertips and the palm mid but the number of classes varies according to which annotations are present in the dataset.

4.3.3 Leaf Statistics

During the training phase, split nodes learn their parameters with a certain amount of patches from randomly generated locations in depth images. All these patches terminate in different leaves of several trees. In the testing phase, patches of equal size are densely extracted from the depth image. If the splitting criteria are well learned, then test patches from new unseen images are directed to leaf nodes where similar patches have already been accumulated. This allows to use the collected information from training to detect the desired joints.

The general workflow is to send a test patch through the forest and to collect the corresponding leaves where offsets from several training patches are stored. In a non-parametric manner, each leaf contains the information of all training patches. The processing of a huge amount of vectors would demand a lot of time, hence we make this information parametric so that testing works faster, e.g. by storing the average value of all offset vectors in a leaf. After sending all patches through the forest, the procedure is repeated by averaging the vote vectors from the collection of leaves, yielding the final vote. Note, the different joint locations are handled independently, hence each patch votes several times.

We aggregate the votes within a leaf node during training so that leaves only deliver a single vector for each joint. For testing, there still remain multiple vectors due to several extracted patches and trees. Each leaf node gets the same weight, regardless of the number of patches that are aggregated. The illustration in Figure 35 shows both, the aggregation of votes at training and test time. Our experiments have shown that no differences in precision occurs but a performance enhancement can be reached with the parametrization.

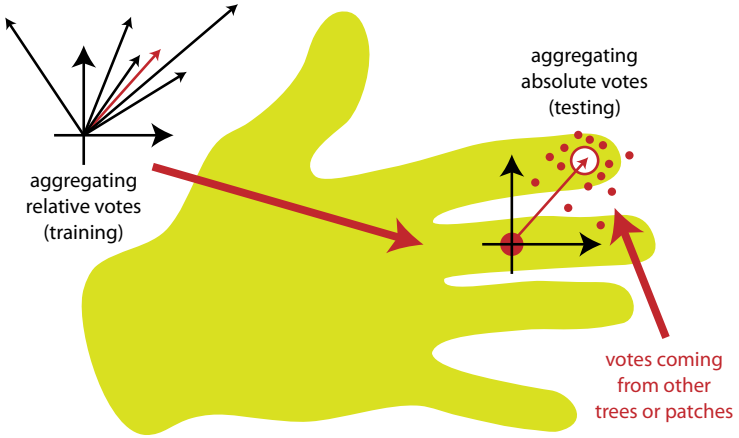


Figure 35: Aggregating votes to predict the final joint location. When creating leaves at training time, the node stores several patches and the associated relative offsets to joints which are aggregated to a final leaf voting. During testing, the patches use these vectors and vote for an absolute joint location in the image, followed by the next aggregation step to generate the final joint voting.

Currently, we use a weighted hill climb [27] as aggregation method, however a mean-shift [7] or any other mode seeking algorithm might also be used. At the beginning, we compute the mean value of all votes in the leaf node, resulting in the initial starting point. Then we again compute the mean value at this point but now in a fixed search radius, followed by shifting the current point towards the new mean value. This process is recursively repeated several times so that the point can climb the hill where the highest data density is present. The mean values for this process are computed with

$$\hat{\boldsymbol{t}}^n = \frac{\sum_i w_i^n \; \boldsymbol{\theta}_i^n}{\sum_i w_i^n} \quad \text{and} \quad w_i^n = \exp\left(-\lambda \; \|\boldsymbol{\theta}_i^n\|\right) \tag{14}$$

where the weights w_i^n are the results of the exponential function so that each offset vector $\boldsymbol{\theta}_i^n$ is weighted by its length. The weights for the mean value $\hat{\boldsymbol{t}}^n$ in Equation 14 are computed similar to the probabilities for the class uncertainty measurement from the previous section.

The idea is to decrease the influence of long distance votes because the variety of different hand articulations is extremely high. A short vote vector only gives predictions for the intermediate neighborhood, yielding lower error rates.

The factor λ again controls the steepness of the exponential function. If the steepness is set too small, then we only allow a local view of the depth image where no difference between fingers can be found. A large value for the steepness leads to a global view of the whole hand, resulting in worse accuracy because the finger positions change significantly when performing various hand poses.

In addition to the vote vectors that result from weighted hill climbs, we store the variances σ^n in each leaf node, computed over all patch offsets and for each joint location separately. This allows to evaluate the original scattering that occurred during the learning process.

4.3.4 Detecting Phase

Rather than using each pixel location as patch center on new unseen depth images, a pixel grid with a fixed stride length can be used, leading to a desired tradeoff between efficiency and accuracy of the final prediction. For noisy depth images, we apply a $(k \times k \times k)$ median filter in the same way as done in the learning procedure, i.e. filtering over image coordinates and the time. Again, integral images are in use so that a speedup on the stored binary tests can be reached.

When a patch is directed through the tree and arrives at a leaf node, we take the stored vote vectors for all joint locations. Concerning forests, each patch votes several times from its location because of several trees. This is visualized in Figure 36 for a selected fingertip. The aggregation works equally as described in the previous section, i.e. we aggregate votes with a weighted hill climb where the weight is a function of the vote vector length.

Before aggregating votes to the final result, we verify the empirically determined threshold σ_{thres} and discard all vectors to joint locations corresponding to the stored joint variances σ^n of the leaf. The precision of final results is much lower when bad leaf votes are involved because they mainly add noise to the detection phase. However, this threshold need to be treated with caution because a low variance is no clue for an accurate voting. Additionally, if too many votes are discarded due to the threshold, the forest is not able to give a prediction for a certain joint location.

After getting the final result for each joint on the hand, we project the world coordinates $(x_w/y_w/z_w)$ back to image space (x_v/y_v with depth as pixel value) and verify the relevant pixels in the depth image. If the depth value of a joint is valid, i.e. not segmented due to the applied depth thresholds, the estimated depth value is replaced with the depth value from the image pixel, otherwise we use the original estimation. This additional step is the reversal process of the learning phase where depth values are read from the annotated dataset. If the x_v/y_v coordinates are already well estimated, taking the original depth improves the result, otherwise the coordinate z_w worsens because the replaced value might come from a completely wrong area in the image.

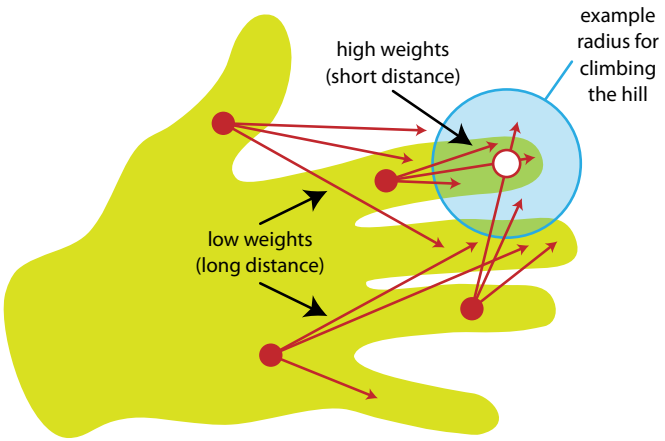


Figure 36: Example votes for a few patch locations. Each patch terminates in several leaves (from several trees) and uses the stored offset vectors to vote for each joint locations. These votes are aggregated with a weighted hill climb to get the final prediction. Note, only a selection of vectors for the index fingertip is visualized, hence this need to be done for all joints.

As the last step, we perform a simple tracking algorithm by use of a FIFO for all joint locations separately, i.e. each new estimation is fed into a queue of fixed size but we only store the latest estimations, thus the oldest entry is removed permanently. If the current estimation is further away than the median value of all the estimations in the queue, we discard the estimation. This ensures that completely wrong estimations are not able to produce a final prediction but both parameters (queue size and allowed distance) have to be set carefully, otherwise no forest result will occur for some joint locations.

4.3.5 Jointly use Classification

Depending on the content which is captured in the depth images of the dataset, we perform a classification of the foreground (hand segment) and the background (other parts), additionally to the regression problem. This allows to distinguish between different image contents so that only the informative patches control the prediction of joint locations. Regarding the available datasets, it is not necessary to classify pixels from the ICVL dataset because the hand is the sole object in all depth images. But, for use cases where additional parts are present in the image, we need trees which are able to classify image patches and predict joint locations. The example images in Figure 37 show the comparison with the ICG dataset. Such images contain the head, the torso, the arm as well as other parts from the scene in addition to the hand. For this reason, we compute a bounding box around all known groundtruth annotations as previously described in Figure 31 and follow the approach in [14] such that trees can handle regression and classification tasks in the same forest.

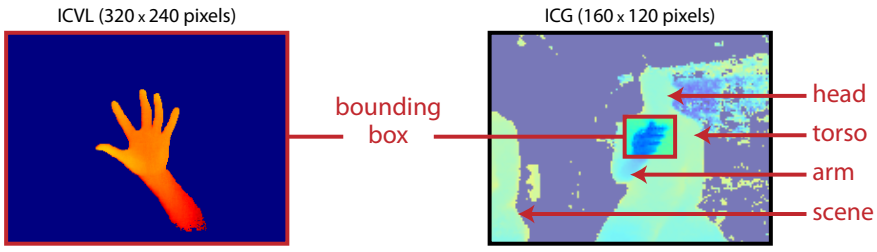


Figure 37: Bounding box and other parts in depth images. Depending on the use case, we additionally compute a bounding box in image space coordinates of annotations so that foreground and background become distinguishable.

When extracting patches from within the bounding box area, we assign the class label $c_i = 1$ for the hand, otherwise the patch $\mathcal{P}_i$ gets the class label $c_i = 2$ that indicates any other content in the image. At split nodes, we randomly choose between regression (see Equation 11 or Equation 12) and classification objectives. For the latter, we use the Gini index [5] which finds splitting candidates with the highest impurity reduction for the next nodes. If the patches in a split node are pure according their class labels, we fix the objective to regression because no classification is needed. If the node contains no more hand patches, a stopping criteria is reached because only patches from the hand are allowed to vote for joint locations. Leaves additionally store the ratio of hand patches $p(c = 1|\mathcal{P}) = 1 - p(c = 2|\mathcal{P})$ that reached them during training. When testing, we only consider leaf nodes with a ratio greater than p_{thres} so that only those leaves are able to give a prediction which mainly consist of patches capturing the hand.

4.3.6 Generating Feature Maps

The previous sections have described the basic functionality of our random forest implementation. Now, we want to extend the approach with feature maps that are generated by an additional forest. The idea is to resolve ambiguities between image patches due to self-similarities in the appearance of fingers. If we are able to obtain more features per image, the forest can use them to discriminate patches, yielding a more precise localization of joints. The concept is based on [52] where similar feature maps are used as context information. We separate our learning algorithm into different stages which are sequentially processed. In the first stage, we learn a forest on depth image patches with the given class labels and joint annotations from the dataset. This forest is used to generate feature maps, i.e. each patch votes for its pixel in several Hough maps. The resulting feature maps are used for the second stage where another forest learns on all these maps, in addition to the original depth image.

The depth images in our hand posture datasets from Section 3.1 and Section 3.2 are labeled with different classes. Each extracted image patch is assigned to

the foreground or background class which is defined by the computed bounding box around the groundtruth annotations. Counting both classes is redundant, hence we are only interested in patches from the hand. Each patch is also assigned to a gesture class $g \in \{1, 2, \ldots, G\}$ depending on which hand pose is captured, i.e. all patches of an image get the same label. For the self-occlusion flag, we assign the label $h \in \{1, 2, \ldots, H\}$ where $H = N$ because each joint annotation can be visible or not. As shown in Figure 38, all these class labels allow to create classification maps. The votes for the location of joints are mapped from 3D to 2D using the camera calibration matrix and accumulated in a regression map for each joint annotation separately. Although we lose the depth information, this procedure allows to achieve much faster computations.

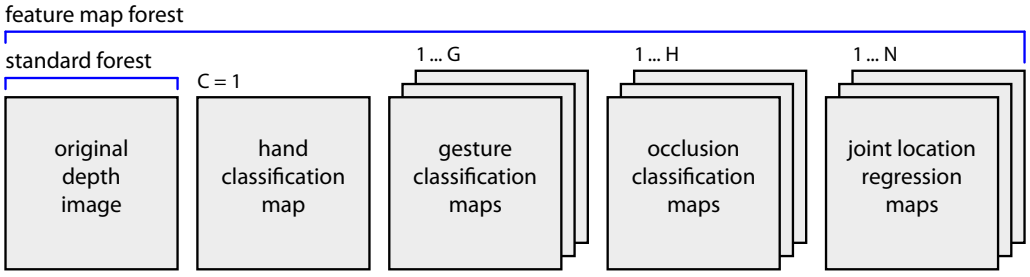


Figure 38: Feature maps. We use the original depth image and several maps that store the probability for each available label. These features are generated by regression labels and different types of classification labels.

Split nodes in the first forest jointly solve the regression and classification problems. For the latter, we randomly perform the node optimization on either the hand class or the gesture classes or the occlusion classes. Leaf nodes store the probability for all $(C + G + H)$ classes and the (N) vote vectors to joint locations as usual. When using the trained forest, the extracted image patches terminate in leaf nodes which accumulate the probabilities in the corresponding classification maps at the patch center coordinate, and store the vote for each joint location in the corresponding regression map if p_{thres} as well as σ_{thres} are fulfilled. We train the second forest nearly the same as described in previous sections, except that $f = \{1, 2, \ldots, 1 + C + G + H + N\}$ for the depth image plus the feature map channels, i.e. the options for randomly chosen splitting candidates are extended with the generated maps.

For test sequences, we perform the same procedure, i.e. for each test image, the first forest generates feature maps which are used by split nodes in the second forest. Finally, patches terminate in leaf nodes and vote with the stored offset vectors for coordinates that are aggregated to the joint locations.

5 Experiments

In this section, we show the results of experiments that were conducted for our proposed approaches. We begin in Section 5.1 with an evaluation of our fitted ellipse and its pointing direction in terms of a user study as well as several sample illustrations. In Section 5.2, we evaluate the most important parameters for our random forests. We continue with Section 5.3 and evaluate feature maps which are generated by an additional forest. The overall results can be found in Section 5.4 where our forest estimations are compared to a state-of-the-art approach. Finally, we visualize several joint predictions on depth images.

5.1 Ellipse Tracking

Our algorithm is optimized for the ART system but the approach can be adapted to any other application with similar requirements. The constraint stays the same, i.e. the hand needs to enter the image from mainly either the upper edge or the lower edge, only then the correct direction of the ellipse can be computed.

5.1.1 User Study

To evaluate our approach for finding the pointing direction of the whole hand, we performed a user study at different locations with 14 participants in total (with 6 people from Dunedin/NZ and 8 people from Graz/AT). All participants completed 4 tasks where the first half was conducted with the original tracking algorithm of the ART system and the second half with the proposed ellipse fitting approach. In order to evaluate how well the algorithms perform when people try to point at a desired position on the table, we used our Unity environment and placed a colored sphere at the estimated position on the hand. The participants were asked to point with typical pointing gestures and to visually follow the sphere on the screen. The other task involved the presented memory game. Here, both hands were used to complete the game but no sphere was displayed, i.e. the participants played the game without knowing where the system tracks the hand or fingertip.

All participants successfully finished the experiment and were asked to answer a questionnaire with 2 questions for each algorithm. The original questions and the result can be found in Figure 39 where a voting from 1 to 10 was given to rate the performance of the individual tasks.

The diagram shows that we improve the accuracy for both tasks but the difference for the memory game is lower compared to sphere experiment. This is because the game consists of large tiles which need to be flipped by the user,

hence the accurate position of the hand is less important. The difference in the result for the other task is almost twice as high because the users see the estimated point in terms of a visualized sphere. This demonstrates that our approach is able to follow the user's hand regardless of the gesture. However, the algorithm will only work if the user performs pointing gestures as humans would point with hands.

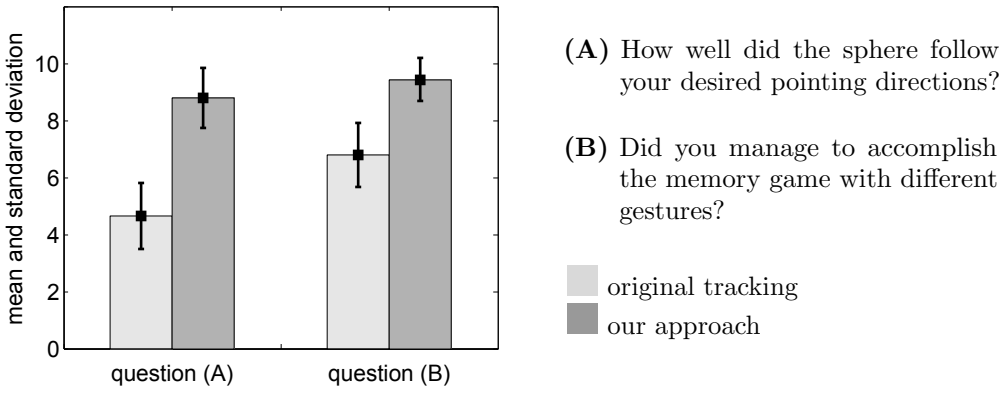


Figure 39: Result of our user study. The designated field of application is the presented ART system for playing the current memory game and several future applications. Our evaluations show that finding the pointing direction works now much more accurate and that users are able to control an application by various hand movements.

5.1.2 Sample Illustrations

Our approach greatly improves the current ART system but we have no groundtruth data, nor an algorithm with which the results can be compared fairly. Furthermore, it is hard to define the position to which the whole hand points. However, several experiments have shown that our approach yields the desired result in almost all cases.

Some examples can be found in Figure 40 where the original color images and the corresponding visualizations of the algorithm are provided. The illustrations show a rotated rectangle in which the ellipse is inscribed, i.e. the rectangle is exactly aligned to the ellipse shape. The major axes are marked with blue lines, the minor axes in green, respectively. The algorithm only uses the upper green corner points, the lower blue corner points remain unaffected. The intersections with the hand mask are marked with smaller yellow points. After finding the longest vector perpendicular to the cutting line (within the hand mask), the final estimation is marked with a red point in both images. As already described, the parameter for the user's hand size (number of pixels) remains the same for all our participants but might be adapted to other use cases, especially when the camera is placed differently and the hand varies in its captured size.

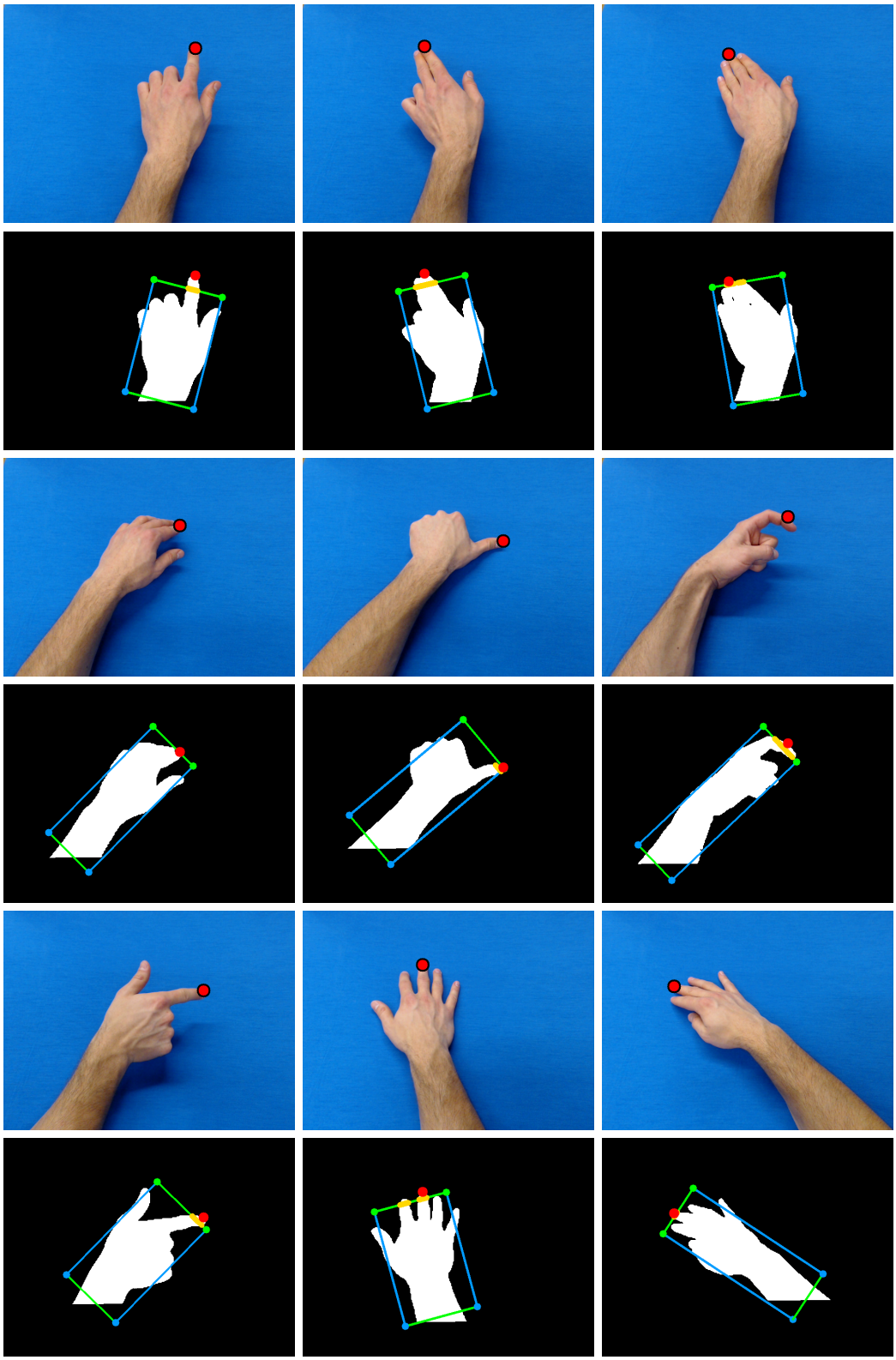


Figure 40: Pointing with hands. The final result is shown in the original color images and the rectangles visualize our ellipse fitting algorithm for the ART system.

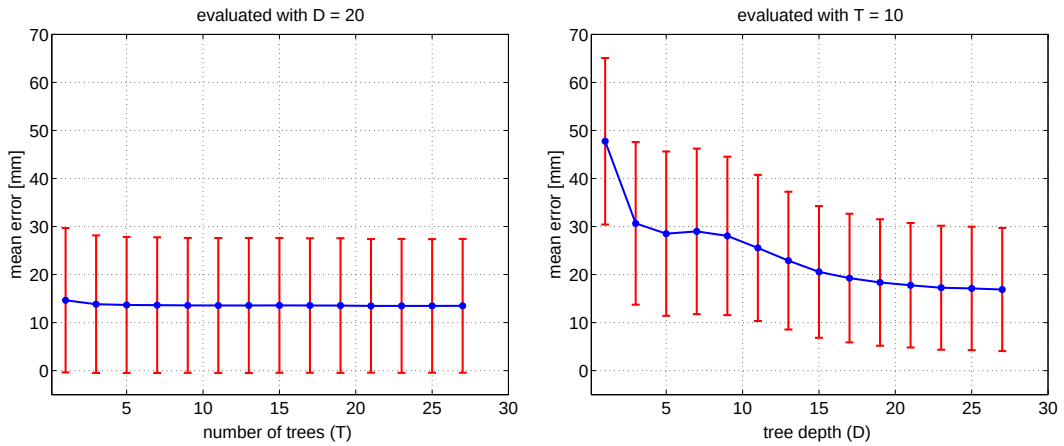
5.2 Forest Parameter Evaluation

The following evaluations show the influence of several forest parameters. Each evaluation is generated with the ICVL dataset because of the higher accuracy in groundtruth labels and less noisy depth images compared to the ICG dataset.

All original depth images in the dataset are in use but we skip the rotated versions to speedup our evaluations, yielding 22.057 images for training and 1.596 images for testing. The results are averaged over 5 independent forest builds where each run is evaluated with the same patch locations, i.e. at the beginning, we randomly generate patch locations and store them for batched runs. This ensures that averaging of equal evaluations only counteracts the random factor of forest parameters. Actually, we should also average over different sets of patch locations. Our evaluations have shown that the result remains the same, due to the fact that the dataset contains a large amount of highly diverse depth images.

5.2.1 General Configurations

The trees are trained with a fraction of the dataset, even though our tests are applied on both available test sequences. This allows a shorter learning phase and a faster evaluation of different forest settings. But, these sequences may contain hand poses which are not learned, resulting in a higher error rate. Nevertheless, we can find the parameters that yield the best result. Our evaluations are also completely independent, i.e. in case of need we disable some functionalities so that only the evaluated parameter influences the result.



(a) **(b)**

Figure 41: Evaluation of number of trees and tree depth. The mean error and the corresponding standard deviation are computed over all joint locations. The diagrams show that (a) using a few trees improves the accuracy and that (b) the correct choice of the tree depth is important to achieve suitable predictions of joints.

We evaluate the forest parameters by comparing the distance between the groundtruth annotation and our estimated joint location. Unless otherwise stated, we compute the mean error and the standard deviation over all available joint locations in the dataset, hence we want to find the best result in average. At the beginning, the impact of the forest size $\mathcal{T}$ and the tree depth $\mathcal{D}$ is evaluated (see Figure 41a and Figure 41b, respectively). The more trees we use, the more estimations are available to predict final joint locations. Our evaluation shows that just a few trees are sufficient to achieve a more precise prediction, i.e. each tree generalizes well because of very different image patches. The tree depth depends on the size of the dataset and the number of extracted patches, i.e. the forest requires deeper trees to split a larger amount of samples. As shown in the diagram, no further performance enhancement is achieved when using deeper trees but also no overfitting occurs. However, the training time increases quadratically with the depth while the runtime for tests stays linear, hence the growing process can be stopped earlier.

Forests contain several parameters and lots of them are evaluated in our experiments. However, the following configurations are empirically determined and fixed for all evaluations in the next sections:

- **number of node tests and thresholds**
  During training at split nodes, we randomly generate 2000 node tests, each of them with 100 thresholds so that the best splitting candidates can be found.
- **number of extracted patches**
  We extract 50 patches from each depth image. If the bounding box for classification is in use, we extract the first half from the interior region and the second half from the exterior region.
- **stopping criteria**
  The growing process of trees ends, if the maximum tree depth is reached or less than the minimum amount of 20 samples are left. Then, a leaf node is created.
- **voting grid distance**
  For test images, we use a grid of 2 pixels to extract image patches. Hence, with regards to image boundaries, less than the half of all image pixels act as patch center.
- **parameters of detecting phase**
  The mean value for hill climbs is always computed over 7 iterations with a search radius of 30 millimeters. The final tracking algorithm stores the latest 5 elements in a queue for each joint location where the allowed distance is restricted to 100 millimeters from frame to frame.

We now start to use diagrams which show the percentage of correctly predicted joint locations within a certain radius, i.e. we count how many of all joints are less then a certain distance apart from the groundtruth annotation. The diagram in Figure 42 evaluates different square patch sizes $\mathcal{S}$ which are chosen in order to illustrate the significance of correct configurations. If patches are too small, then many of them look completely the same because there is almost no difference in the appearance of fingers or other parts of the hand. On the contrary, large patches cover almost the whole hand, causing problems with binary tests because it is hard to find similar patches when performing different hand articulations or capturing the hand from different viewpoints. Our evaluations have shown that the best results are achieved with patches that contain neighbouring fingers, hence the size should be chosen so that the relation to the closest fingers is covered by the patch.

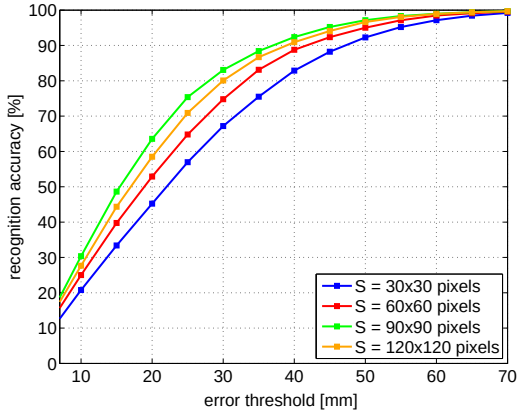


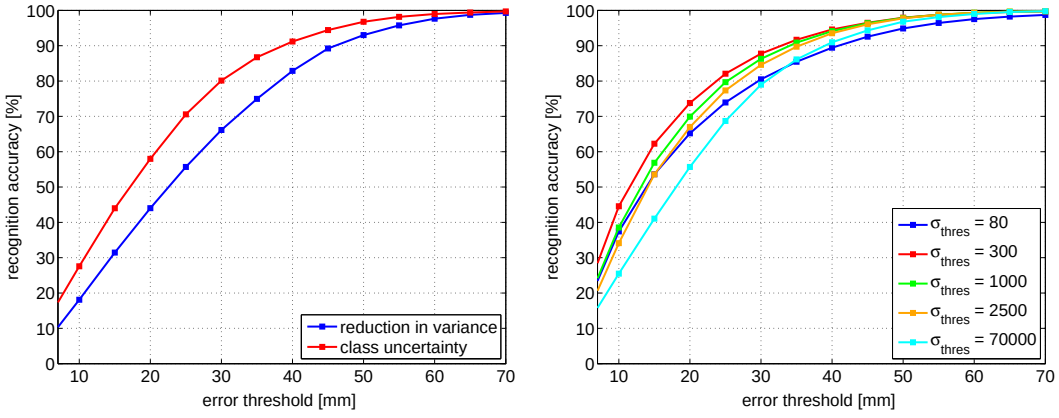
Figure 42: Evaluation of patch sizes with error thresholds in metric space. A small patch size yields very similar looking patches so that it is difficult to distinguish between fingers. Large patches allow to view the relation to other fingers but lead to a vast number of different appearances.

5.2.2 Node Optimization

After selecting a split function and the corresponding threshold at the parent node, each patch is passed either to left or right child node according to the feature response. To select the splitting candidate which minimizes the entropy of child nodes (maximum information gain), we compare score calculations based on reduction in variance and the class uncertainty measurement.

The best information gain is the result of well separated parent nodes, i.e. their children vote to the same joint location in the image. If the variance of the votes is high, then it indicates a poor splitting criterion because similar patches should not vote to different joint locations. The reduction of node variances helps to find reasonable splits but this regression objective is based on single joints, hence the overall variance for multiple joints is the sum of all the individual

joints. The obvious problem is that all patches at the current node have to produce low variances for every joint, hence the overall variance can be high although a few joints produce suitable values, with the consequence that the forest refuses the split. The comparison with the class uncertainty measurement in Figure 43a has shown that the accuracy improves when assigning continuous probability distributions. As described and evaluated in the next section, an extracted patch is associated with a probability for each discrete class label (the label of the joint) in dependence to its distance. In this way, the regression objective works with multi-modal distributions as used for classification tasks.



(a) **(b)**

Figure 43: Optimizing splits and leaves. To generate the best forest, it is necessary to (a) use reasonable regression objectives for split nodes and to (b) exploit additional statistics in leaf nodes which are collected during the learning process.

To estimate the final result, we send new unseen patches through the trees and collect the resulting leaf nodes from the whole forest. But, only those leaves are considered for regression which have variances below the predefined threshold σ_{thres} (all other votes are discarded). According to Figure 43b, the variance is a useful measure to differ between informative and noisy leaf nodes. The threshold is applied for each joint location independently, hence some joints get more votes, others may get less. If the threshold is set too hard, it increases the probability that no tree votes for a specific joint location. Furthermore, the accuracy decreases because low variances give no guarantee for precise votes, in this case the curve flattens faster too. However, a softer threshold discards lots of bad votes what improves the accuracy. Another benefit for real-time applications is the reduction of computation time because less votes have to be aggregated to the final vote.

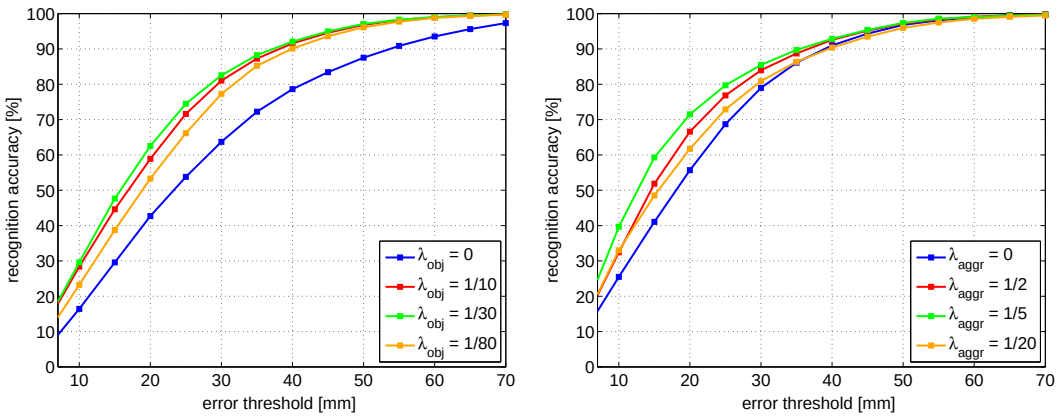
For leaf node statistics, we store the variances and the distribution of hand patches. Hence, we apply the threshold for the maximum allowed variance σ_{thres} as evaluated, and the empirically determined threshold $p_{thres} = 0.9$ for the minimum required hand probability. This ensures that only those leaves are allowed to vote which mainly consist of patches from the hand area. Of course,

we use no bounding box for the ICVL dataset (all patches get the same class) but extracted image patches from the ICG dataset are assigned to the foreground or background class, allowing to exclude the corresponding leaves for the regression task.

5.2.3 Distance Functions

We use an exponential distance function in split nodes and leaf nodes for our random regression forests. The class uncertainty measurement in split nodes defines a class affiliation for each extracted image patch depending on its distance to joints. The closer a patch is located to a joint location, the higher is the probability for this class label. In leaf nodes, we aggregate the patch votes with a weighted hill climb which uses weights depending on the vote vector length. The same is done over trees, when aggregating leaves to a final vote for each joint location which need to be predicted.

The evaluated parameters in this section control the steepness, i.e. a high value corresponds to a short range of influence and $\lambda = 0$ results in the same probabilities or weights for all distances. Although the values are generated by the same function, we want to separately use the parameters λ_{obj} and λ_{aggr} for the class uncertainty measurement and the weighted hill climb, respectively.



(a) **(b)**

Figure 44: Evaluation of distance functions. The parameter λ_{obj} and λ_{aggr} control the steepness of the exponential function that is in use for (a) the class uncertainty measurement in split nodes and (b) the weighted hill climb in leaf nodes. If the parameters are set too hard, the result worsens again.

A correctly chosen steepness improves the accuracy by several percent over the given error thresholds in Figure 44a where regression objectives with low values for λ_{obj} are more like fully randomized forests. The parameter can be roughly estimated depending on the use case because distances between patch center and annotation are specified in metric space. If large values are used for the

steepness, then the probability for the class affiliation of patches is almost not given, resulting in worse accuracies. Several evaluations have shown that the parameter λ_{obj} becomes much more important at deeper split nodes. Beginning at the root node, the split function has to deal with every annotation in the dataset, hence it is hard to find a threshold that effects low entropies for child nodes. Further down the tree, the number of classes gets lower because the patches with other labels are already splitted to other nodes. Therefore, the class uncertainty measurement is able to impact the split function stronger.

Leaf nodes aggregate their patch votes at training time while the testing procedure just aggregates the resulting vote vectors to quickly compute the final result over all trees. Thus, the aggregation has to be done twice but aggregating all votes at once could improve the result. This preserves real-time capabilities because the number of votes is heavily reduced. The diagram in Figure 44b shows the aggregation with a weighted hill climb, done within leaves and over the forest results. On closer examination, it can be seen that the parameter enhances the prediction in particular for short range error thresholds. Smaller values for λ_{aggr} prohibit long distance votes, or in other words, short votes get a higher weight which influences the mode seeking much stronger. This is important because hand poses extremely vary so that patches only know about joint locations in the immediate neighborhood. The well chosen λ_{aggr} permits votes in the range of a few millimeters so that the workflow is similar to classification tasks. However, performing regression with short voting vectors is necessary and yields the best accuracy.

5.2.4 Fingertip Accuracy

The previously evaluated forest parameters contain the errors of multiple joints merged to create a single curve. Fingertips usually are the most important locations for many applications, hence the following diagrams separately illustrate each fingertip of the hand. The forest is built with $\mathcal{T} = 10$ trees and depth $\mathcal{D} = 17$ as well as a patch size of $\mathcal{S} = 90{\times}90$ pixels. Class assignments in split nodes are done with $\lambda_{obj} = 1/50$ and the parameter $\lambda_{aggr} = 1/3$ controls the aggregation of votes. Again, we use the same training dataset (no rotated depth images) and both provided test sequences. The interesting part of the diagram is how far the errors of fingertips drift apart. When considering Figure 45a and the 20 millimeters threshold, the index finger is correctly estimated to 91%, the ring finger and thumb to 82% respectively. Pinky achieves 73% of correct estimations and the extremely bad accuracy of 28% is performed by the middle finger. A closer look to Figure 45b shows an improvement in average, with the difference that the variance threshold $\sigma_{thres} = 200$ instead of $\sigma_{thres} = 30000$ is in use. The index finger loses 4% but the middle finger improves the accuracy by 27%, the ring finger by 4% and pinky by 6% (the thumb remains constant). Again, this shows the importance of exploiting leaf node statistics.

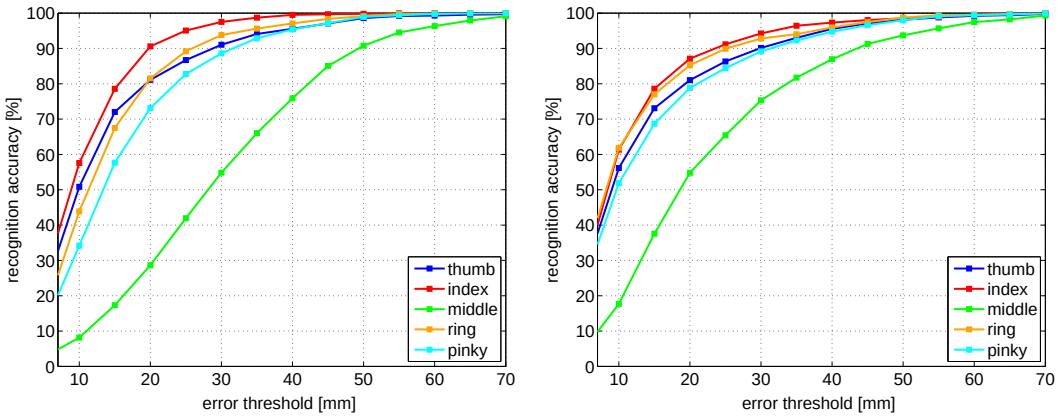


Figure 45: Evaluation of fingertips. Both runs are performed with the same parameters, except that we use (a) a higher variance threshold and (b) a lower variance threshold. The index finger shows the best accuracy, followed by the ring finger and thumb as well as pinky but the middle finger has the worst recognition accuracy in all cases.

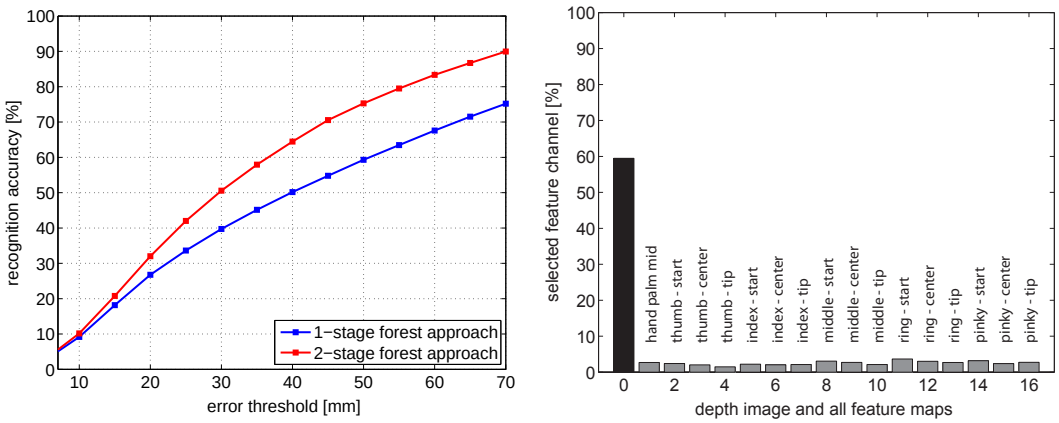
The variance of vote vectors is a useful measurement but precise votes can be eliminated too. The middle finger achieves the greatest benefit because of poor learned estimations with high variances. That is because the middle finger is surrounded by the index finger and the ring finger which look very similar (length, direction etc.), resulting in ambiguous leaf nodes. The index finger is on the exterior and close to the completely different looking thumb. The ring finger has similar advantages because of the much smaller pinky. The patch size $\mathcal{S}$ must be chosen so that several fingers are covered, otherwise it is hard to find indicators in image patches. Generally, hands are not easy to define, especially when performing different gestures, so that finding the best split function is no simple task for random forests. Although we have not implemented it yet, different variance thresholds for every fingertip would additionally improve the result because each joint location is handled individually.

5.3 Feature Map Evaluation

This evaluation shows the benefits when extending feature channels with feature maps that are generated by another forest. The workflow is very similar, i.e. the first forest accumulates the information from leaves in a Hough map and the second forest uses them in split nodes. Due to the erroneous ICG dataset which is evaluated later, the classification maps are not considered. Hence, the evaluation is only performed on the ICVL dataset with regression maps. Therefore, we compare the results with our standard approach and visualize the generated maps for a specific configuration.

5.3.1 Context Stacking

To evaluate the result, we perform a self-comparison with our standard forest which is called 1-stage approach in contrast to the 2-stage approach when using feature maps. We separate our training data into two halves to prevent overfitting, yielding 11K samples for the first forest and 11K samples for the second forest. The 1-stage approach only uses the latter half of training data. The 2-stage approach uses the first half to generate feature maps, i.e. the splitting criteria are learned with the initial depth images and leaf nodes store their vote vectors as usual. Subsequently, the first forest is tested with the second half of training data where the leaf nodes vote into a Hough map for each joint location. The accumulated votes are weighted by the vote vector length, yielding probability distributions (the feature maps) for all joint locations. The depth images and these generated features are fed into the learning process of the second forest, i.e. split nodes try to find the best parameters over several feature channels. The evaluation is completed by testing both approaches with the available test sequences.



(a) **(b)**

Figure 46: Evaluation of generated feature maps. The first stage stores leaf votes in Hough maps and the second stage uses these feature for splitting in addition to depth images. The diagrams show that (a) the results are improved by a few percent and (b) the forest has mainly preferred the original depth image as feature channel. The black bar refers to the depth image channel and the numbering of all gray bars corresponds to the annotation number in the dataset.

The forests are built with $\mathcal{T} = 10$ trees and a allowed depth of $\mathcal{D} = 15$ due to the half amount of sample data. There is no difference in the configuration of the first and second forest, except that we apply a dense voting for the generation of feature maps (patches are extracted at each pixel location when testing), and that we increase the number of node tests from 1000 to 5000 due to the larger amount of feature channels when splitting with feature maps. All the other forest parameters are chosen similar to previous evaluations.

For an absolute comparison to our best result, we are limited to the amount of training data which need to be separated for the 2-stage approach. Furthermore, we need lots of RAM to keep all feature maps for each depth image in the memory. However, we can compare it relatively with the 1-stage approach because the forests are configured completely the same. The curve in Figure 46a shows a slight accuracy improvement when observing the error thresholds of a few millimeters. The mean error is reduced from $48mm$ to $35mm$ and the standard deviation from $35mm$ to $24mm$ over all joint locations and frames. The histogram in Figure 46b shows the distribution of the selected feature channels. In about 60% of the cases, a split node chooses the original depth image, yielding a probability between 2% and 3% for choosing the feature map of a joint location. The shown error rates are extremely high in comparison to our best evaluation results on this dataset. This is due to the fact that the test sequences contain lots of different hand poses which are not learned when using only half of the training images. Nevertheless, we can assume that feature maps will also improve the estimation of joints when more data is available.

5.3.2 Visualization of Hough Space

From each depth image in the test dataset, we extract patches at every pixel location and forward them to all trees in the first forest. The split nodes are trained on depth images and pass the patches towards the next node until a leaf is reached. Each patch terminates in several leaves and votes into a separate Hough map for each joint location, i.e. the collected votes are projected from world space into image space using the camera calibration matrix and accumulated in a feature map where the probability is a function of the vote vector length.

Such maps are visualized in Figure 47 for a selected depth image. The peaks in the images indicate the estimated joint locations. Several visualizations have shown that joints like fingertips are more scattered than joints which are located closer to the hand center. A reason could be that learning from half of the dataset limits the accuracy. However, the forest would only chose a feature map if the split function yields the best score. And, the statistic in Figure 46 shows that feature maps from fingertips are less frequently selected compared to the other joints.

The original depth image and all of its feature maps are forwarded to the second forest which extracts patches according to the defined pixel grid. The split nodes have additionally stored the feature channel from training and perform the corresponding binary tests, either on the depth image or on a feature map. The procedure for leaf nodes works as usual, hence all votes are aggregated using our weighted hill climb that yields the final coordinates of joint locations.

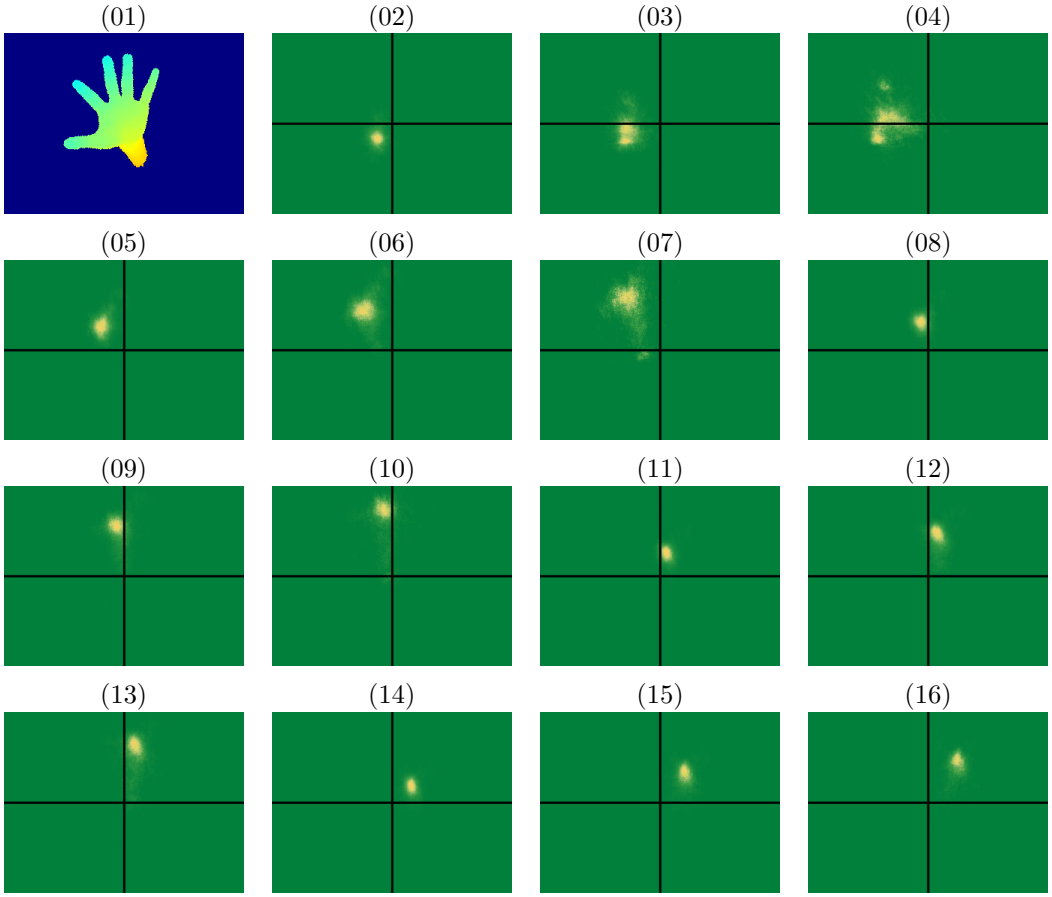


Figure 47: Illustration of generated feature maps. At the beginning, the processed depth image is shown, followed by a Hough map for each joint, except the hand palm mid (the cross indicates the image center). For visualization purposes, each feature map is normalized according to its stored minimum and maximum probability.

5.4 Overall Results

Several forest parameters are evaluated in the previous sections, now we show our result with the best configuration. Consequently, the predictions on the ICVL dataset are compared with another state-of-the-art approach. Since the ICG dataset is not published yet, we only show the accuracy of our estimated joint locations. Finally, we visualize several predictions on depth images.

5.4.1 Comparison to other Methods

We compare our random forest approach with Tang et al. [50] who use the Latent Regression Forest (LRF) to estimate joint locations in real-time without a hand model. Similar to ours, their approach is based on regression but instead of only learning the appearance of depth image patches, they also learn the topology of the hand.

For the comparison, we use our standard approach, hence no feature maps are generated by a separated forest. We train the forest with 110.285 depth images in total, composed of 22.057 original images which are rotated 4 times by a certain angle (yielding 0° as well as ±22.5° and ±45.0°). Using further rotated versions may improve the result. However, this would require more memory to load the images which also increases the computation time for the learning phase. The approaches are compared with all provided test images in the dataset, i.e. sequence $\mathcal{A}$ with 702 images and sequence $\mathcal{B}$ with 894 images, resulting in 1.596 depth images in total. The first test sequence mainly contains a hand which shows different poses at a constant distance to the camera. The second test sequence shows another person who rotates the hand and varies the viewpoints at different distances. Our forest size is set to $\mathcal{T} = 5$ and the growing process stops when the tree depth $\mathcal{D} = 27$ is reached. From each depth image, we extract 50 patches of size $\mathcal{S} = 100 \times 100$ pixels. The class uncertainty measurement for the regression objective in split nodes uses $\lambda_{obj} = 1/30$ to assign probabilities for each joint label. The aggregation of leaf votes is weighted with $\lambda_{aggr} = 1/5$ to restrict vote vector lengths to the immediate neighborhood. The allowed variance of $\sigma_{thres} = 300$ has to be satisfied, otherwise the vote vectors are discarded. All other forest parameters remain the same as configured in the previous evaluations.

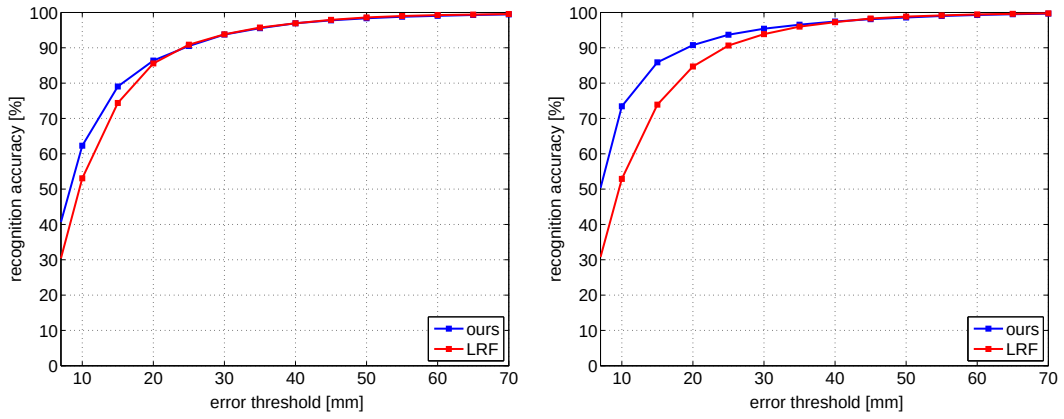


Figure 48: Comparison of test sequences from the provided dataset. The evaluations are performed on (a) test sequence $\mathcal{A}$ and (b) test sequence $\mathcal{B}$ over all joint locations. The LRF approach shows similar error rates for the first sequence but we improve the recognition at close range. Our method achieves a great accuracy enhancement for the second sequence, even if the depth images contain much more varying hand poses.

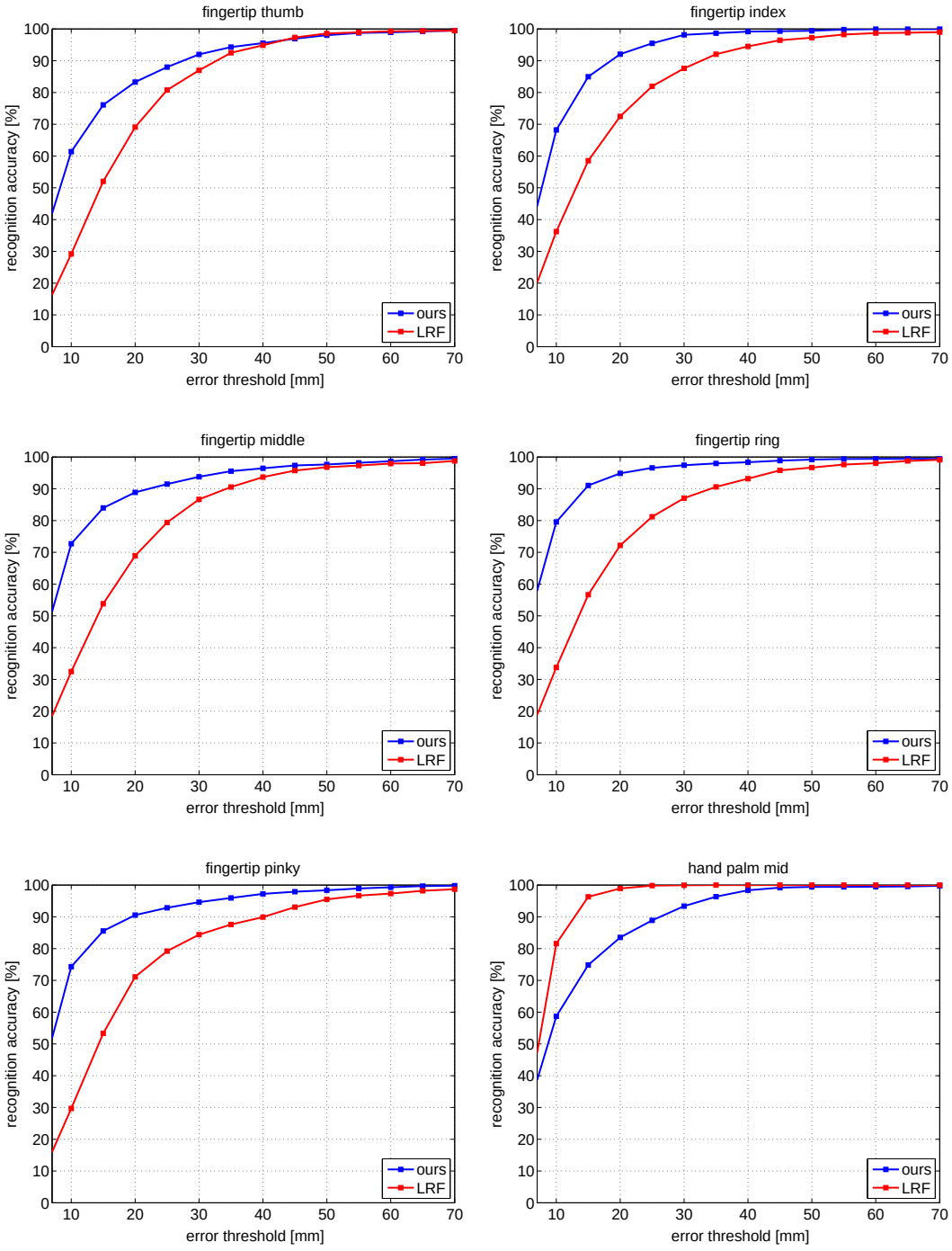


Figure 49: Comparison of individual joint locations. The diagrams show our results compared to the LRF where all fingertips and the hand palm mid are confronted. We greatly improve the accuracy for each fingertip (and several other joints) but the precision for the palm is worse. Both provided test sequences are merged for these evaluations.

In Figure 48a and Figure 48b, we compare our prediction results with the LRF approach for both test sequences separately. The diagrams show the percentage of joint locations which are within a certain distance to groundtruth (error threshold), done for all annotations over all images. The mean error for sequence $\mathcal{A}$ is reduced from $13mm$ to $11mm$ and the standard deviation stays at $11mm$ for both approaches. Much more performance enhancement yields sequence $\mathcal{B}$ where our mean error is $9mm$ compared to $13mm$ for the LRF approach, with $10mm$ standard deviation. For our use case, we are mainly interested in the fingertips, hence Figure 49 evaluates each fingertip as well as the palm mid. We improve the result by 17% when observing the error threshold of 10 millimeters and using all 16 joint locations. But, if only the 5 fingertips are considered, an improvement of 39% is reached. In particular, this shows that we greatly outperform the LRF approach for joint locations on the finger, even if the hand palm mid is inaccurately estimated.

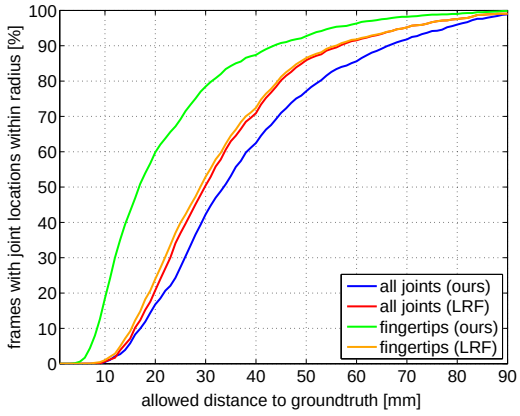


Figure 50: Proportion of frames with joint locations within a radius. This metric shows how many frames have all 16 predicted joints or the 5 fingertips within a certain allowed distance from the groundtruth. Our approach is clearly superior when comparing the fingertip curves but the LRF shows better results when using all available joints.

For the diagram in Figure 50, we use the proposed metric in [51] which counts the number of frames that have all predicted joint locations within a certain distance from groundtruth. This is a more challenging metric because many applications rely on an accurate estimation of several joint locations at the same time. When observing the allowed distance deviation of 20 millimeters, our approach shows a slightly worse result for all 16 joints. On the other side, we achieve an improvement of about 35% for the 5 fingertips. Due to the learned hand structure, the LRF approach is able to precisely vote for all joint locations. Our approach only relies on the appearance of extracted image patches, hence the random forest has no information about the topology of a human hand. Nevertheless, we are superior for fingertips, in particular for allowed distances of a few millimeters which is the most important part of the diagram.

In order to perform a fair comparison, we also have to mention that our evaluation excludes joint predictions in 402 cases out of 25.536 votes (number of annotations times number of depth images). This is due to the fact that in 399 cases the groundtruth annotations are not placed on the hand area in the depth image, i.e. we are not able to obtain the depth value. In the remaining 3 cases, our random forest refuses the voting because of either the variance threshold or the tracking queue. However, this is a negligible amount of votes which has almost no influence on the comparison in all shown diagrams.

5.4.2 Remark to Datasets

The comparison to a state-of-the-art method on the ICVL dataset has shown that our approach achieves great results for the localization of joints in depth images. Now, we want to show our best result for the ICG dataset which can be handled similarly. The dataset is not published yet, hence no comparison to other algorithms is performed. For the evaluation, we train a forest with about 15K depth images from different people and use a test sequence that contains almost 2K depth images from another person. The forest consists of $\mathcal{T} = 8$ trees where a depth of $\mathcal{D} = 20$ is allowed. We extract 25 patches of size $\mathcal{S} =$ 40×40 pixels from inside as well as outside the computed bounding box. The allowed variance in leaf node is set to $\sigma_{thres} = 700$ but all other parameters are retained from the other dataset. Additionally, each split node randomly selects the classification or regression objective.

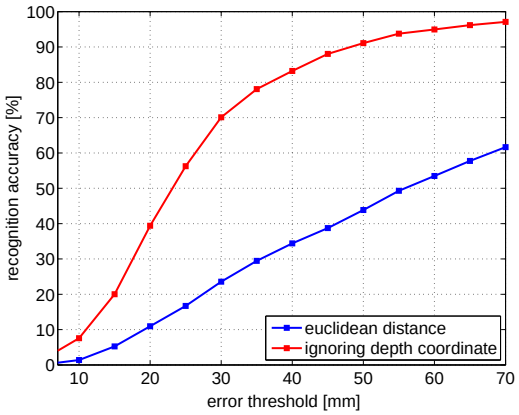


Figure 51: Our best evaluation result. Both curves are constructed from the same detection result over all joint locations in the annotated dataset. In the blue curve we consider the groundtruth distance as usual and in the red curve we ignore the estimated depth coordinate.

Although the forest configuration is adapted to the properties of the dataset, i.e. number of images and their resolution due to the hand size, the evaluation in Figure 51 shows no accurate prediction of joint locations. As usual, we count the number of joints which are within a certain distance from the 8 groundtruth annotations but the error rate is not suitable for detecting joints on the hand.

Furthermore, the forest refuses the voting in about 30% of all votes due to the thresholds for σ_{thres} and p_{thres} as well as the tracking queue which observes several estimations from the past, hence no voting occurs for certain joints that would further deteriorate the result. The mean error yields $59mm$ with a standard deviation of $31mm$ in average over all detected joints and processed frames. Our evaluations have shown that the estimated depth value deviates significantly from groundtruth, hence Figure 51 also shows a curve where the depth coordinate is ignored. In this way, only the estimated x/y coordinates are evaluated because the estimated z coordinate is unconsidered, yielding $27mm$ and $16mm$ for the mean error and standard deviation, respectively. The comparison demonstrates that the algorithm works but the learning of the depth has failed. This is due the dataset which contains noisy depth images and imprecise annotations. Besides the fact that the depth image resolution is very low anyway, the captured hand only exploits a small area so that a finger consists of a few pixels. Due to the noise, we apply a median filter but the annotated depth values still vary by several centimeters from frame to frame. Nevertheless, the major problem are the annotations because lots of them are roughly placed, resulting in an annotated part of the scene rather than the hand area or a fingertip. Forest are able to handle some deviations but for detecting fingers, the dataset is unsuitable due to the low resolution and misplaced annotations which yield wrongly learned regression targets.

5.4.3 Representative Images

Finally, we visualize several estimations of joint locations on depth images. The examples in Figure 52 show the results for fingertips and the hand palm mid where all coordinates are projected from world space to image space with the camera calibration matrix. Usually, the location of fingertips is accurately predicted but the palm shows stronger varying predictions due to the flat hand area which has less significant features in depth images. The accuracy of all other joint locations is similar to fingertips. Some typical failure cases show the images in Figure 53 where the joint locations are visualized as before. Depending on which images the forest has seen during the learning phase, the accuracy for some joints might be weaker than for others. On several frames, the thumb shows a slightly higher drift, i.e. the forest is not able to precisely vote for the location. This is reasonable due to the large variety of articulations which can be performed with the thumb. Many times, the thumb directly faces towards the camera which is hard to predict too. Even if it occurs only rarely

for some frames, a major problem is the similarity between fingers, e.g. the middle finger might be recognized as index/ring finger and vice versa. Our approach considers joint locations as independent detections, hence there is no information about the relation between the individual joints. Learning the structure and kinematic of hands [49, 50] or using a model-based approach [34] that also considers physics, allows to overcome such problems. However, our approach is only based on the appearance of depth patches and also achieves reasonable results.

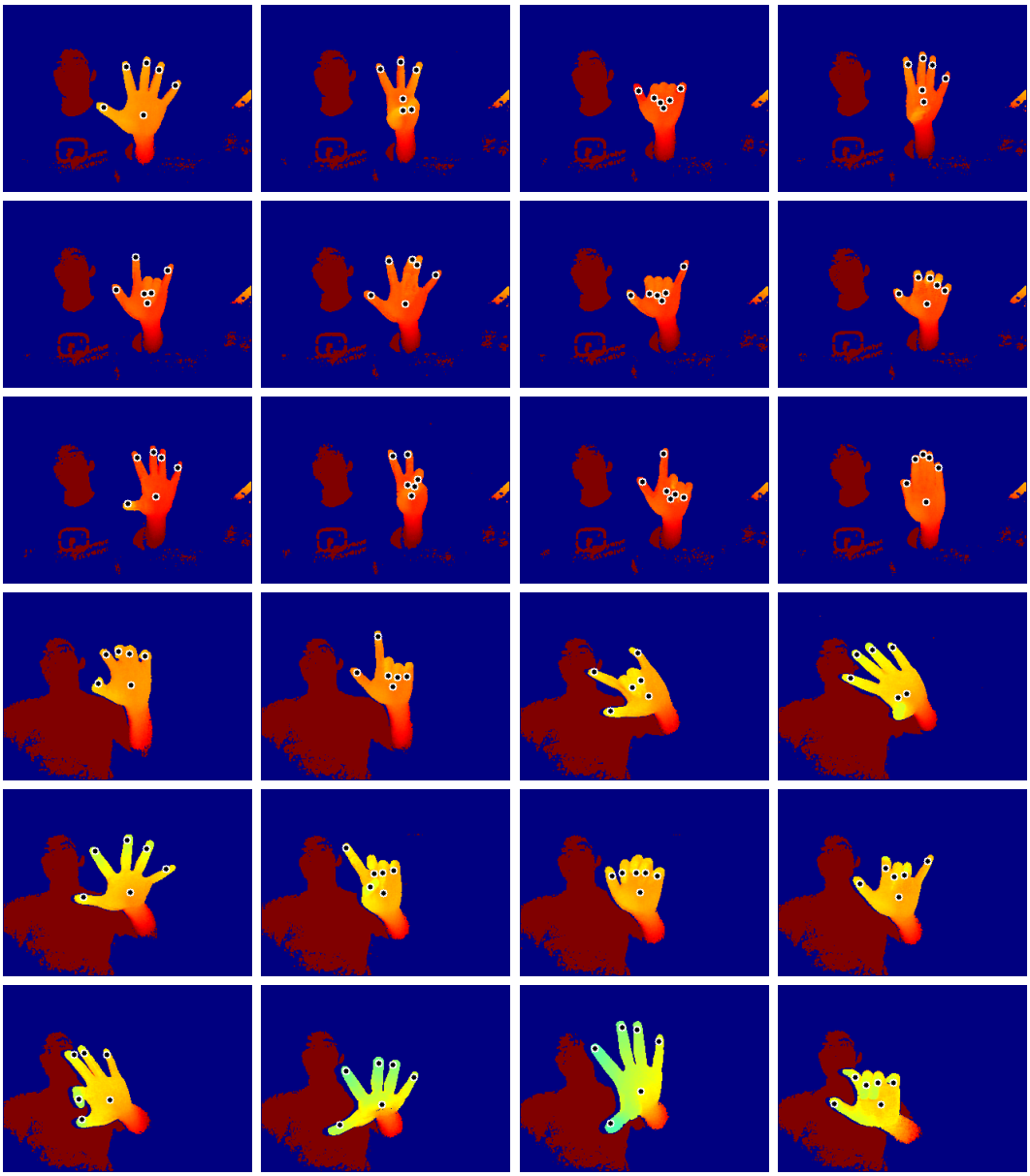


Figure 52: Correctly estimated joint locations. The examples are taken from both test sequences. Due to visualization purposes, we only show the fingertips and the hand palm mid. Each joint location is projected from world space into image space.

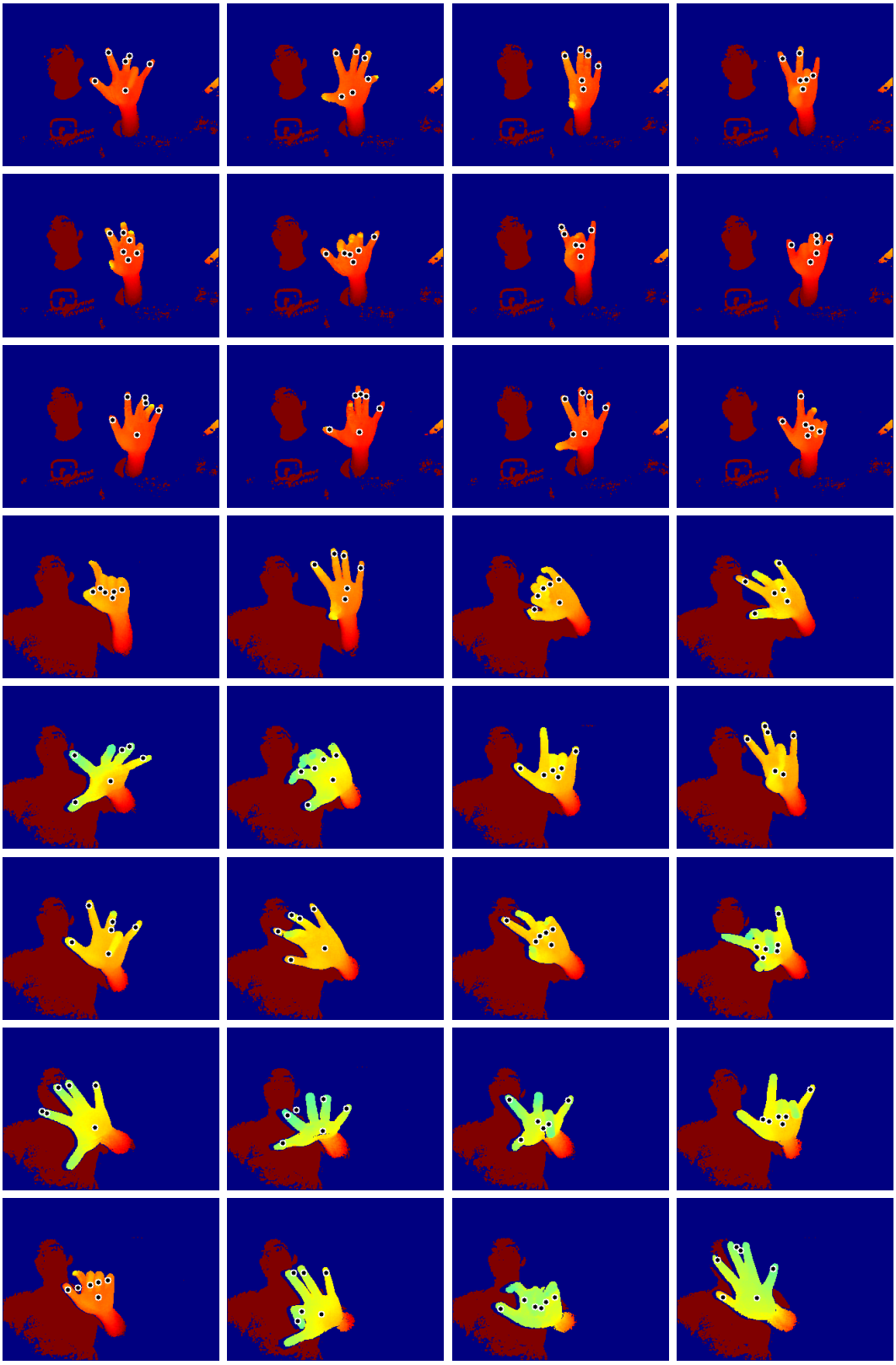


Figure 53: Slightly worse examples from both test sequences. As in the last figure, we only visualize the prediction of fingertips and the hand palm mid but all other joint locations show similar results.

6 Conclusion

This work has presented a post-stroke rehabilitation system which is used for therapies of motor impairments of the arm. For measurements and interactions in an augmented environment, the essential task is to determine the accurate position of the hand and every finger in the image. The following sections discuss the highlights of our contribution and give an outlook for future work.

6.1 Summary

The developed system involves several parts that work together. The camera captures the stroke patient's hands and displays them in an augmented environment by generating a virtual scene that is calibrated with the real rehabilitation setup. By viewing and moving the segmented hand on the screen, the patient is able to control an application which can be a game or any other implementation for therapy sessions.

The presented ellipse fitting algorithm works on color images and allows to find the position to which the hand is pointing. Our evaluations have shown that we achieve an accurate and stable result, allowing to adapt the algorithm into the productive rehabilitation system. Considering the constraints for our use case, the approach can also be adapted to any other system that requires the orientation or pointing direction of hands.

To allow measurements and human-computer interactions in our augmented reality application, we need more precise information about the hand in the image. For this purpose, we learn the appearance of hands with a random forest that jointly solves classification and regression problems. For the former task, a tree needs to discriminate patches belonging to the hand area due to the additional captured arm or other parts of the scene. The latter task involves the prediction of fingertips and several other joint locations on the hand, yielding the main contribution of our work.

We train a random forest on depth images where split nodes have the objective to separate patches depending on their appearance and the assigned class labels as well as joint annotations. Each leaf node stores offset vectors to vote for joint locations. Our experiments have shown that the result significantly improves when each vote is weighted by its vector length. Therefore, an exponential function limits the influence of votes to a few millimeters. Furthermore, the variance of the set of offset vectors implies that a vote can yield an imprecise prediction of a joint location, hence we eliminate such votes. The patch size is chosen in order to cover several fingers, allowing to learn the relationship between neighboring fingers. Small patches are not distinguishable due to very similar looking fingers but large patches would be too variant because of countless different hand articulations. To achieve further improvements, we

propose to use another forest that generates feature maps for all annotated joint locations, under nearly the same configuration. The maps are forwarded to the main forest that uses them as additional context information in combination with the original depth images for finding the best splitting candidates. This resolves the ambiguities between image patches from fingers due to their self-similarities in the appearance.

Our algorithm is evaluated on different datasets. The provided test sequences have demonstrated that we outperform state-of-the-art algorithms and achieve a precise estimation of joint locations, in particular a low localization error for fingertips.

6.2 Outlook

Currently, a single camera is mounted close to the patient and provides the color as well as depth video stream. However, depending on the performed hand pose, e.g. grabbing objects, some fingers are not captured due to self-occlusions. For that reason, we want to use a multi-camera setup where another camera is placed on the opposite side of the patient. After creating our own annotated dataset for learning the hand appearance, we are able to evaluate the robustness of our algorithm for the rehabilitation system.

Our random forest approach achieves accurate predictions for joint locations, however there are several research topics which can enhance our approach. In the near future, we plan to normalize our binary tests for comparing patch features. The current method only stores the coordinate and the size of the compared rectangles within a patch. When normalizing these parameters with the depth value at the patch center, the comparison of patches gets scale-invariant. Our method works because the datasets contain various hands that are captured in different depths. Hence, this would lead to an improvement of the result and also allow a reduction of the dataset size. Another topic is learning the hand structure or using hand models. In the current approach, each joint location is handled individually, i.e. the algorithm is only based on the appearance of patches and there is no relation between the detected joint locations. Knowing the topology of the hand, allows to eliminate predictions which can not occur in human hands.

References

[1] Personal Page of Danhang Tang (Imperial Computer Vision and Learning Lab at Imperial College London). http://iis.ee.ic.ac.uk/~dtang/hand.html. Accessed on December 18th, 2014. (24)

[2] Intel Perceptual Computing SDK for Creative Interactive Gesture Camera. http://software.intel.com/en-us/perceptual-computing-sdk. Accessed on February 22nd, 2014. (20)

[3] A. Bosch, A. Zisserman, and X. Muoz. Image Classification using Random Forests and Ferns. In *International Conference on Computer Vision*, pages 1–8, 2007. (4)

[4] L. Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001. (27, 35)

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification And Regression Trees*. Chapman & Hall / CRC, 1984. (41)

[6] H.-T. Chen, T.-L. Liu, and C.-S. Fuh. Segmenting Highly Articulated Video Objects with Weak-Prior Random Forests. In *European Conference on Computer Vision*, pages 373–385. Springer, 2006. (4)

[7] Y. Cheng. Mean Shift, Mode Seeking, and Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:790–799, 1995. (38)

[8] A. Criminisi and J. Shotton. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer, 2013. (29)

[9] A. Criminisi, D. Robertson, E. Konukoglu, J. Shotton, S. Pathak, S. White, and K. Siddiqui. Regression Forests for Efficient Anatomy Detection and Localization in Computed Tomography Scans. *Medical Image Analysis*, 17:1293–1303, 2013. (28, 34, 36)

[10] M. Dantone, J. Gall, G. Fanelli, and L. v. Gool. Real-time Facial Feature Detection using Conditional Regression Forests. In *Conference on Computer Vision and Pattern Recognition*, pages 2578–2585, 2012. (5, 31, 36)

[11] M. Dantone, J. Gall, C. Leistner, and L. v. Gool. Human Pose Estimation Using Body Parts Dependent Joint Regressors. In *Conference on Computer Vision and Pattern Recognition*, pages 3041–3048, 2013. (5, 6, 31)

[12] K. Dorfmuller-Ulhaas and D. Schmalstieg. Finger Tracking for Interaction in Augmented Environments. In *International Symposium on Augmented Reality*, pages 55–64, 2001. (2)

[13] G. Fanelli, J. Gall, and L. v. Gool. Real Time Head Pose Estimation with Random Regression Forests. In *Conference on Computer Vision and Pattern Recognition*, pages 617–624, 2011. (4, 5, 31)

[14] G. Fanelli, T. Weise, J. Gall, and L. v. Gool. Real Time Head Pose Estimation from Consumer Depth Cameras. In *Symposium of the German Association for Pattern Reocgnition*, pages 101–110, 2011. (4, 34, 40)

[15] E. A. Franz and T. Packman. Fooling the Brain into Thinking It Sees Both Hands Moving Enhances Bimanual Spatial Coupling. *Experimental Brain Research*, 157: 174–180, 2004. (8)

[16] E. A. Franz and V. Ramachandran. Bimanual Coupling in Amputees with Phantom Limbs. *Nature Neuroscience*, 1:443–444, 1998. (1)

[17] J. Gall and V. Lempitsky. Class-Specific Hough Forests for Object Detection. In *Conference on Computer Vision and Pattern Recognition*, pages 1022–1029, 2009. (4, 36)

[18] J. Gall, A. Yao, N. Razavi, L. v. Gool, and V. Lempitsky. Hough Forests for Object Detection, Tracking, and Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:2188–2202, 2011. (32, 34)

[19] J. Gall, N. Razavi, and L. Van Gool. An Introduction to Random Forests for Multi-Class Object Detection. In *Outdoor and Large-Scale Real-World Scene Analysis*, pages 243–263. Springer, 2012. (32)

[20] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003. (18)

[21] T. K. Ho. Random Decision Forests. In *International Conference on Document Analysis and Recognition*, pages 278–282, 1995. (27)

[22] S. Hoermann, L. Hale, S. J. Winser, and H. Regenbrecht. Patient Engagement and Clinical Feasibility of Augmented Reflection Technology for Stroke Rehabilitation. *International Journal on Disability and Human Development*, 13:355–360, 2014. (10, 13)

[23] C. Huang, X. Ding, and C. Fang. Head Pose Estimation Based on Random Forests for Multiclass Classification. In *International Conference on Pattern Recogntion*, pages 934–937, 2010. (28)

[24] D.-Y. Huang, W.-C. Hu, and S.-H. Chang. Gabor filter-based Hand-Pose Angle Estimation for Hand Gesture Recognition under varying Illumination. *Expert Systems with Applications*, 38:6031–6042, 2011. (31)

[25] C. Keskin, F. Kıraç, Y. E. Kara, and L. Akarun. Hand Pose Estimation and Hand Shape Classification Using Multi-layered Randomized Decision Forests. In *European Conference on Computer Vision*, pages 852–863. Springer, 2012. (6)

[26] C. Keskin, F. Kıraç, Y. E. Kara, and L. Akarun. Real Time Hand Pose Estimation Using Depth Sensors. In *Consumer Depth Cameras for Computer Vision*, pages 119–137. Springer, 2013. (28)

[27] P. Kontschieder, S. Rota Bulò, A. Criminisi, P. Kohli, M. Pelillo, and H. Bischof. Context-Sensitive Decision Forests for Object Detection. In *Advances Conference on Neural Information Processing Systems*, pages 431–439, 2012. (38)

[28] L. Lamberti and F. Camastra. Real-Time Hand Gesture Recognition Using a Color Glove. In *Image Analysis and Processing*, pages 365–373. Springer, 2011. (2)

[29] D. Lee and S. Lee. Vision-Based Finger Action Recognition by Angle Detection and Contour Analysis. *ETRI Journal*, 33:415–422, 2011. (31)

[30] T. Lee and T. Hollerer. Handy AR: Markerless Inspection of Augmented Reality Objects Using Fingertip Tracking. In *IEEE International Symposium on Wearable Computers*, pages 83–90, 2007. (31)

[31] V. Lepetit, P. Lagger, and P. Fua. Randomized Trees for Real-Time Keypoint Recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 775–781, 2005. (4, 34)

[32] R. Maree, P. Geurts, J. Piater, and L. Wehenkel. Random Subwindows for Robust Image Classification. In *Conference on Computer Vision and Pattern Recognition*, pages 34–40, 2005. (4)

[33] S. Nowozin. Improved Information Gain Estimates for Decision Tree Induction. In *International Conference on Machine Learing*, pages 297–304, 2012. (29)

[34] I. Oikonomidis, N. Kyriazis, and A. Argyros. Full DOF Tracking of a Hand Interacting with an Object by Modeling Occlusions and Physical Constraints. In *International Conference on Computer Vision*, pages 2088–2095, 2011. (61)

[35] R. Okada. Discriminative Generalized Hough Transform for Object Dectection. In *International Conference on Computer Vision*, pages 2000–2005, 2009. (4)

[36] J. Park and Y.-L. Yoon. LED-Glove Based Interactions in Multi-Modal Displays for Teleconferencing. In *International Conference on Artificial Reality and Telexistence*, pages 395–399, 2006. (2)

[37] V. S. Ramachandran and D. Rogers-Ramachandran. Synaesthesia in Phantom Limbs induced with Mirrors. *Proceedings of the Royal Society of London*, 263:377–386, 1996. (1)

[38] H. Regenbrecht, E. Franz, G. McGregor, B. Dixon, and S. Hoermann. Beyond the Looking Glass: Fooling the Brain with the Augmented Mirror Box. *Presence*, 20: 559–576, 2011. (1, 9, 10, 13)

[39] H. Regenbrecht, G. McGregor, C. Ott, S. Hoermann, T. Schubert, L. Hale, J. Hoermann, B. Dixon, and E. Franz. Out of reach? A Novel AR Interface Approach for Motor Rehabilitation. In *Symposium on Mixed and Augmented Reality*, pages 219–228, 2011. (10, 11)

[40] H. Regenbrecht, S. Hoermann, G. McGregor, B. Dixon, E. Franz, C. Ott, L. Hale, T. Schubert, and J. Hoermann. Visual Manipulations for Motor Rehabilitation. *Computers & Graphics*, 36:819–834, 2012. (17)

[41] H. Regenbrecht, S. Hoermann, C. Ott, L. Muller, and E. Franz. Manipulating the Experience of Reality for Rehabilitation Applications. *Proceedings of the IEEE*, 102: 170–184, 2014. (1, 8)

[42] C. Robertson, L. Vink, H. Regenbrecht, C. Lutteroth, and B. C. Wünsche. Mixed Reality Kinect Mirror Box for Stroke Rehabilitation. In *International Conference of Image and Vision Computing New Zealand*, pages 231–235, 2013. (1)

[43] F. Schroff, A. Criminisi, and A. Zisserman. Object Class Segmentation using Random Forests. In *Proceedings of the British Machine Vision Conference*, pages 1–10, 2008. (4, 28)

[44] J. Shotton, M. Johnson, and R. Cipolla. Semantic Texton Forests for Image Categorization and Segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008. (4)

[45] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-Time Human Pose Recognition in Parts from a Single Depth Image. In *Conference on Computer Vision and Pattern Recognition*, pages 1297–1304, 2011. (5, 6, 31, 35)

[46] J. A. Stevens and M. E. P. Stoykov. Simulation of Bilateral Movement Training Through Mirror Reflection: A Case Report Demonstrating an Occupational Therapy Technique for Hemiparesis. *Topics in Stroke Rehabilitation*, 11:59–66, 2004. (1, 8)

[47] M. Sun, G. Bradski, B.-X. Xu, and S. Savarese. Depth-Encoded Hough Voting for Joint Object Detection and Shape Recovery. In *European Conference on Computer Vision*, pages 658–671. 2010. (4)

[48] M. Sun, P. Kohli, and J. Shotton. Conditional Regression Forests for Human Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*, pages 3394–3401, 2012. (28)

[49] D. Tang, T.-H. Yu, and T.-K. Kim. Real-Time Articulated Hand Pose Estimation Using Semi-supervised Transductive Regression Forests. In *International Conference on Computer Vision*, pages 3224–3231, 2013. (61)

[50] D. Tang, H. J. Chang, A. Tejani, and T.-K. Kim. Latent Regression Forest: Structured Estimation of Articulated Hand Posture. In *Conference on Computer Vision and Pattern Recognition*, pages 3786–3793, 2014. (6, 7, 24, 55, 61)

[51] J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon. The Vitruvian Manifold: Inferring Dense Correspondences for One-Shot Human Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*, pages 103–110, 2012. (58)

[52] Z. Tu and X. Bai. Auto-Context and Its Application to High-Level Vision Tasks and Brain Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:1744–1757, 2010. (41)

[53] P. Viola and M. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. In *Conference on Computer Vision and Pattern Recognition*, pages 511–518, 2001. (34)

[54] R. Y. Wang and J. Popović. Real-time Hand-tracking with a Color Glove. *Transactions on Graphics (ACM)*, 28:1–8, 2009. (2)

[55] H. Yang and I. Patras. Sieving Regression Forest Votes for Facial Feature Detection in the Wild. In *International Conference on Computer Vision*, pages 1936–1943, 2013. (28)

[56] Z. Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334, 2000. (17)