universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"Hand Tracking in Virtual Reality"

verfasst von/ submitted by

Dana Marković, BSc

angestrebter akademischer Grad / in partial fullfillment of the requirements for the degree of

Master of Science (MSc)

Wien, 2018 / Vienna, 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 935
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Master Media Informatics
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Helmut Hlavacs
Mitbetreut von / Co-Supervisor:	

Abstract

This work presents a solution of hand gesture recognition by using only image processing techniques. It consists of hand extraction and gesture recognition, as well as modelling the hand in 3D. The proposed field to be applied is Virtual Reality. Main techniques used for hand extraction are background subtraction and Principal component analysis. For gesture recognition a solution using the Levenberg-Marquardt algorithm, and another using an Iterative Heuristic are presented. In the evaluation part different gestures are tested, as well as different surrounding setups.

Abstract

Diese Arbeit präsentiert eine Lösung für die Erkennung von Handgesten nur mithilfe von Bildverarbeitungsmethoden. Die Lösung besteht aus Handextraktion und Gestenerkennung, sowie aus einer Modellierung der Hand in 3D. Das vorgeschlagene Anwendungsgebiet wäre virtuelle Realität. Die Hauptmethoden, die für die Handextraktion benutzt wurden, sind Hintergrundsubtraktion und die Hauptkomponentenanalyse (Principal component analysis). Für die Gestenerkennung werden eine Lösung mit dem Levenberg- Marquardt Algorithmus und eine iterative Heuristik präsentiert. In der Evaluation wurden sowohl verschiedene Gesten, als auch verschiedene Umgebungseinrichtungen getestet.

Keywords background subtraction, image stitching, hand extraction, gesture recogniton, principal component analysis, Levenberg-Marquardt, virtual reality

Contents

List of Figures

1 Introduction

The topic of this work is hand tracking and hand gesture recognition by using image processing techniques without any external sensors. This means a camera is capturing images of user's hand and the system is applying various algorithms to recognize the current hand pose. The result is a 3D hand model. The field of applying the results was intended to be Virtual Reality (VR).

The field of virtual reality has emerged in last few years as very popular and many commercial solutions have been developed mostly for the gaming industry. Oculus[1] and HTC Vive [2] are examples of VR systems with hand handheld controllers and additional sensors for tracking user's movement and interaction with VR objects. Google Cardboard[3] is an example of a simple VR mount where user interactions are only tracked by head movement sensors and an additional button is used for user input. Leap Motion[4] is a sensor that can be attached to a head mount for tracking user's hands motions. Another example of a sensor tracking hands are various hand gloves, e.g. Manus VR Glove[5]. All of them require additional sensors that are either on a user's body or hands to track the hand motions. Our idea was to test a system where a user only needs a phone with a camera, which will track and recognise the hand gestures. This system would be non-invasive and the user would need fewer gadgets to interact with a VR environment.

The system that was developed to test this relies on image processing techniques and can be divided in two main topics: *hand extraction* and *hand gesture recognition*. Hand extraction is done by using background subtraction, which then includes image stitching and other techniques such as Principal component analysis (PCA), filtering, feature detection and extraction etc. The second topic, hand gesture recognition includes implementations of both the Levenberg-Marquardt algorithm and one additional heuristic. The results of both approaches are presented and compared at the end.

The main idea is having a static environment where user is imagined to be sitting at his desk, wearing a VR head mount with a smartphone. The smartphone's camera is capturing images of the surroundings. The images are then fed to a developed solution which then runs hand gesture recognition algorithms. The output is a 3D hand model that represents the recognized hand gesture.

The implemented solution consists of two components, one is called the Image Processing Component and it runs image processing techniques to extract

[1] https://www.oculus.com/

[2] https://www.vive.com/eu/

[3] https://vr.google.com/cardboard/

[4] https://www.leapmotion.com/

[5] https://manus-vr.com/

the hand from the background. Its output is a binary segmentation of a hand. This has been developed using the EmguCv [6] library, which is a C# wrapper of an OpenCV [7] library. The second component is a Unity3D [8] application which contains a 3D hand model. This component executes the gesture recognition algorithm. The output is a 3D hand, which represents the input gesture as closely as possible.

[6] http://www.emgu.com/wiki/index.php/Main_Page
[7] https://opencv.org/
[8] https://unity3d.com/de

2 Related Work

The problem of tracking users' movements has been a topic of many research projects, especially since the field of virtual reality has become very popular in the last few years. The gaming industry has been investing many resources into this, as the users' movement and interactions are of big importance here.

In general, the methods for capturing user input can be divided in two categories: hardware and computer-vision based. The first category includes invasive approaches by mounting different sensors and devices on the user and getting the input from them. The other category relies on camera and computer vision techniques to extract the input. The hardware based ones have higher accuracy, but are mostly invasive and require additional gadgets for user to carry on his body. The computer vision based approaches divide further into ones with markers and without any markers. The first group includes approaches where colour markers or gloves are used for gesture tracking. The second group uses different techniques to recognise either static, pre-defined gestures or dynamic gestures.

Yeo et al. [29] in their work present a computer vision based approach where they perform gesture recognition by using a simple USB camera and a Kinect [9], which is a depth camera. Their solution can recognise a set of gestures. In their setup they have a camera taking photos of both the user's face and hands in the frame, so the first step they do is the background subtraction to extract the user. They use Haar - like features and Canny Edges detector to remove the face from the input. The Kinect depth data is used to remove the background and determine the hand contours. Contour extraction and polygon approximation is used for shape analysis. Further, they calculate the palm centre as the maximum inscribed circle inside the contour and extract the region of interest (ROI) as the area surrounding the palm center. Convex hull and the convexity details are also calculated and used for later analysis. In analysis of the convexity, they define finger tips, their direction and location by several steps, where they calculate angles and depths, as well as the K-curvature of each point. All these steps are a part of a heuristic for determining the resulting gesture recognition on the static set.

Demeulemeester et al. [14] present a teleconference system that includes a so-called Virtual Meeting Room. They simulate human avatars sitting in a meeting room and modelling their gestures and interactions. Following interactions include: participants entering or leaving the room, which participant is speaking or presenting and in which direction thye are looking. Participants are located at different locations. Also additional metadata, such as e.g. company, role of participants is tracked as well. The system consists of multiple cam-

[9] https://developer.microsoft.com/en-us/windows/kinect

eras positioned around the room which send their input to a component that aggregates it. This input is processed by computer vision algorithm such as face recognition, by use of parabolic edge maps, people tracking, gaze detection and hand detection and tracking. The last component visualises this processed data. The location of the participant, the walking speed and the size of each person is used to deduce the behaviour in the simulation. The visualisation of the meeting room has been accomplished with Unity3D.

Ma and Wu [22] propose a model-based approach for hand tracking where they use a time-of-flight camera that captures depth data. The hand model is based on quadratic meshes and provides a simulation of all 26 degrees of freedom. The system is initialised with a hand depth image database on which k-nearest neighbour search is performed. This is done to help automatic initialization and tracking loss recovery. The captured raw depth image is preprocessed, where hand segmentation is preformed to remove unnecessary objects which results in a binary mask of a hand. They developed a tracking system, which is an optimiser based on an improved version of the Particle Swarm Optimization (PSO) [18] to find the optimal motion parameters to be applied on the hand. Thw PSO algorithm consists of having a population of particles in the parameter space. By iteratively moving the particles, the point with optimal objective function value is searched.

Baltzakis et al. [7] in their paper on tracking hands, faces and facial features, proposed an approach for hand tracking which is a blob tracker, specifically trained to track skin-coloured regions. The aim of their work was to support interaction of visitors in museums with an autonomously navigating guide robots. The developed system had the task to classify the face and some facial expressions, as well as the left and right hand correctly. The problems they tackled in the work were: identification and tracking of hands and faces by detecting skin-coloured blobs, classification and identification thereof, and tracking of specific facial features within the recognised blobs. The approach of identifying hand and face based on skin coloured areas relies on building a colour model of human skin and classifying the image pixels based on their fitness to those models. The clustering of skin coloured pixels into solid blobs that correspond to hands and faces was done by using segmentation techniques. The authors describe the process of propagating the information about the location and shape of each blob by means of a set of pixels hypothesis that are initialised at the setup and forwarded between the subsequent frames. Additionally, an incremental classifier was implemented to continuously update the classification of a tracked hypothesis, i.e. whether it belonged to face, or left or right hand. The shape, motion characteristics and relative location of each blob were also tracked.

In their work, Dardas and Patriu [13] used Principal component analysis for real-time hand tracking and recognition. They have a training stage with a set of hand gestures for which they do the training on different lightning, rotations and scales. After that they calculate eigenvectors and training weights by pro-

jecting images onto most eigenvectors. In the test phase, a frame containing hand is projected onto the same eigenspace. By using Euclidean distance, it is then classified to recognise the hand gesture. By using PCA in the training stage they decrease the dimensionality by reducing each NxN image into a vector of length N^2. The eigenvectors of a set of hand gestures form an eigenspace. Each image from the training set corresponds to each eigenvector. Each image from the training set is represented as a linear combination of the eigenspaces. To extract the hand from the frame, authors use skin detection and contour comparison algorithms. Skin detection was done by applying the thresholding, i.e. a pixel is classified as skin if it fits some defined range. Having defined the skin area, contours of that are are extracted and compared to the contours of hand posture templates. If a match is found in those templates, that area will be used for extracting PCA features.

3 Hand Extraction

Hand extraction is a process of importing images from cameras and applying various image processing methods in order to extract an area of the image that contains only a hand. The result of this process is a binary segmentation, desirably containing all black pixels around the hand, which is displayed by white pixels. This segmentation is then further processed for hand gesture recognition. We used background subtraction to extract hand from the image and this process is described in this section.

The algorithm below sums the whole process of image stitching, creating background panos and extracting the hand.

Hand Extraction Algorithm

1. Capture n images of surroundings
2. Apply image stitching to create background pano p

For any following incoming image i:

1. Warp i to p
2. Extract ROI from p
3. Apply filtering to both i and p
4. Calculate difference between i and p
5. Apply thresholding to create binary segmentation s
6. Contour finding, resulting in biggest contour c
7. Apply Principal Component Analysis on c
8. Send s to modelling component

3.1 Background Subtraction

Background subtraction is also called foreground detection and it is widely used in computer vision for motion and object detection. Many computer vision systems use this technique for moving object detection without knowing a priori what those objects are. The most common applications for background subtraction are video-surveillance for detecting persons, vehicles, intrusion e.g. cameras on highways, intrusion detection, and person counting. Work by Xu

et al. [30] classifies background modelling methods into three categories: pixel-based, region-based and hybrid methods, as well as classification in parametric and non-parametric methods.

Authors of [26] present a review of different background algorithms and state that most of them share the following scheme:

Background initialization: a background model is built from a fixed number of frames.

Foreground detection: this is a classification step, where a pixel is defined either as background or foreground. The foreground is computed by comparing background model and the current frame.

Background maintenance: the background model is updated by analysing images for non-moving objects.

The process of creating a background model can also be categorised into two categories: building the static images, the so-called Static Frame Difference and by using previous frames, which is called Frame Difference [26]. In the first approach, a static background is defined and for all subsequent frames an absolute difference is computed. In the second approach, the background is adaptively maintained, by for example, computing the arithmetic mean between successive frames. This approach has the advantage when ambient lights changes, but will fail if moving objects stop suddenly. For foreground detection, most commonly, as in the static background initialization, the absolute difference between background and current frame is computed.

Bouwmans in his survey [9] lists three main conditions for insuring functional background subtraction models: the camera must be fixed, illumination is constant and the background is static. The main issues and challenges that occur at background subtraction are: illumination changes, jittering camera, poor quality image, moved or inserted background objects, foreground aperture, shadows, "ghosting" of objects, etc. [9].

Most common approaches for building a background pano are:

1. Frame differencing
2. Mean filter: series of preceding images are averaged to create a background
3. Running Gaussian average
4. Adaptive Background Mixture Model [27]

3.2 Gaussian Mixture Model

In common probabilistic pixel-based approaches, each pixel has a probability density function. A pixel from new frame belongs to background if its value is well described by its density function.

One of the most common statistical approaches is probabilistic method by Stauffer and Crimson [27]. It is a Gaussian Mixture Model (GMM) where the distribution of each pixel colour is represented by a sum of weighted Gaussian distributions, which are defined in given colourspace. The distributions are evaluated to determine which pixel is most likely to result from a background. The pixels that do not fit the background distributions are categorised as a foreground until there is a Gaussian that will include them. Multiple Gaussians are necessary due to lightening changes and authors use a mixture of adaptive Gaussians to approximate the process. The value of each pixel represents a measurement of the radiance in the direction of the sensor of the first object intersected by the pixel's optical ray [27]. The probability of observing the pixel value X_t at time t is:

$$P(X_t) = \sum_{i=1}^{N} \omega_{i,t} \times \eta(X - t, \mu_{i,t}, \Sigma_{i,t}) \tag{1}$$

Where N is number of Gaussian distributions, $\omega_{i,t}$ is an estimate of weight of the i^{th} Gaussian in the mixture at time t, $\mu_{i,t}$ is the mean value of the i^{th} Gaussian in the mixture at the time t, $\Sigma_{i,t} = \sigma_k^2 I$ is covariance matrix of the i^{th} Gaussian in the mixture at the time t and $\eta(X_t, \mu, \Sigma)$ is a Gaussian probability density function:

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X^t - \mu_t)^T \Sigma^{-1} (X_t - \mu_t)} \tag{2}$$

Depending on the computational resources, N is chosen between 3 and 5. The on-line N-means approximation is implemented, because an exact expectation maximisation would be too costly. By doing this, every pixel value is checked against N Gaussian distributions. If none of the distributions match the pixel value, the least probable distribution is replaced with a distribution with the current value as its mean value. The prior weights of the N distributions at time t are updated by following equations:

$$\omega_{k,t} = (1 - \alpha)\omega_{k,t-1} + \alpha(M_{k,t}) \tag{3}$$

$$\mu_t = (1 - \rho)\mu_{t-1} + \rho X_t \tag{4}$$

$$\sigma_t^2 = (1 - \rho)\sigma_t^2 + \rho(X_t - \mu_t)^T (X_t - \mu_t) \tag{5}$$

where:

$$\rho = \alpha\eta(X_t|\mu_k, \sigma_k) \tag{6}$$

α is the learning rate and $M_{k,t}$ is simply 1 for models that matched and 0 for non-matches. $\omega_{k,t}$ is the k^{th} Gaussian distribution.

The Gaussians are ordered by the fitness value ω/σ. As the evidence of the distribution increases, its value increases also and decreases as the variance increases. Therefore, to determine which of the Gaussians of the mixture are most likely produced by the background the goal is to find the distribution with highest supporting evidence and lowest variance. The first B distributions are chosen as the background model, where B is estimated as:

$$B = argmin_b(\sum_{k=1}^{b} \omega_k > T) \tag{7}$$

T is a measure of the minimum portion of data that should be accounted for by the background. A pixel is marked as a foreground pixel if it is more than 2.5 standard deviations away from any of the B distributions. If a small value is chosen for T, the model is then unimodal.

An improvement was proposed by using an online expectation minimisation algorithm to update background model. An adaptive GMM was proposed by Zivkovic [31]. Further improvements came from Shimada et al. [25] to improve the accuracy by using a dynamic Gaussian component to control the Gaussian mixture model. Chen et al. [12] proposed a hierarchical algorithm by combining GMM and a contrast histogram. In their work, Zivkovic and van der Heijden [32] review the GMM model of Stauffer and Crimson [27] and improve it further by on-line selecting the number of components.

In his comprehensive study, Bouwmans [9] states that fuzzy concepts were introduced by some authors, to deal with imprecisions and uncertainties in the process of background subtraction. In combination with these, neural network models, which learn to classify each pixel of the image have been proposed: General Regression Neural Network, Multivalued Neural Network, Competitive Neural Network, Self Organising Neural Network, etc. Bouwmans also presents another category of algorithms: Robust PCA models, which separate background and foreground via a robust subspace model which is based on a low-rank and a sparse decomposition.

3.3 Static Frame Difference

The simpler approach to the adaptive frame difference for building the background is the static frame difference. The idea is to define one reference frame

as the background and for every other incoming frame to compute the difference to that background.

The problem can be formulated as following, having:

1. A defined reference frame f_r
2. Current frame f_c
3. Compute the difference: $|f_r - f_c| > T_h$

where T_h is some predefined threshold value.

The result is a binary segmentation which points out non – stationary objects. Compared to the approach described in section 3.2 this one is not nearly as robust, however because of its simplicity there is no need to update the background image. Background subtraction should provide a robust approach, which is insensitive to illumination changes, motion changes and changes in the background geometry. As we will later see, this approach is sensitive to illumination changes, so some filtering and defining the constraints in which the solution works is necessary. However, our idea is to have a static environment setup, e.g. a workplace with an user sitting at a computer desk. The camera is placed on the user's head taking photos of the workplace. Thus, if a camera is capturing what is happening around user all the incoming frames will have hands in it. If the setup is truly static, all that changes in it are user's hands that are moving or changing gestures. That is why we consider any further change as a trigger for hand gesture recognition. The background will not change and as the image for the background is precomputed the difference between this precomputed pano image and the incoming frame is the hand.

3.4 Static Frame Difference vs. GMM

In Fig. 1, the left image represents simple background, whereas the right image represents an incoming frame containing hand in foreground. The difference between these two images can be computed, by simply subtracting the pixel values. Left image in Fig. 2 is the result, a grayscale image where the background is mostly black and the hand is easily distinguishable, although some filtering is necessary. This grayscale image can be further transformed into a binary one. The right image in Fig. 2 shows a clear segmentation of a hand. This can further be used to extract the exact gesture.

Mixture of Gaussians for background subtraction was tested with a stream of input frames from Fig. 3 and results are presented in Fig. 4. The stream starts

Figure 1: Background and hand image

Figure 2: Difference between background and hand image

with an empty background and then different hand gestures were captured in the following frames. OpenCV BackgroundSubtractorMOG [10], which is an implementation of Gaussian Mixture-based algorithm, was used for this. As one can see, the hand is distinguishable and as the hand moves it keeps being recognised as foreground object. In the case if hand would become static for some time, it would fade into the background.

The extractions above were results of using only one image as a background. However, using only one image for a background is not enough. We want to provide more freedom for the user, so if he rotates or slightly moves his head, it should still be possible to extract the hand. To achieve this, we need to take more images of the surroundings. These images need to be stitched into a pano, which will serve as a background image.

3.5 Image Stitching

Image stitching is a process of stitching multiple images into a pano. There are two different methods for doing this: direct and featured based ones. Direct

[10] https://docs.opencv.org/ref/2.4/db/dcf/classcv_1_1BackgroundSubtractorMOG.html

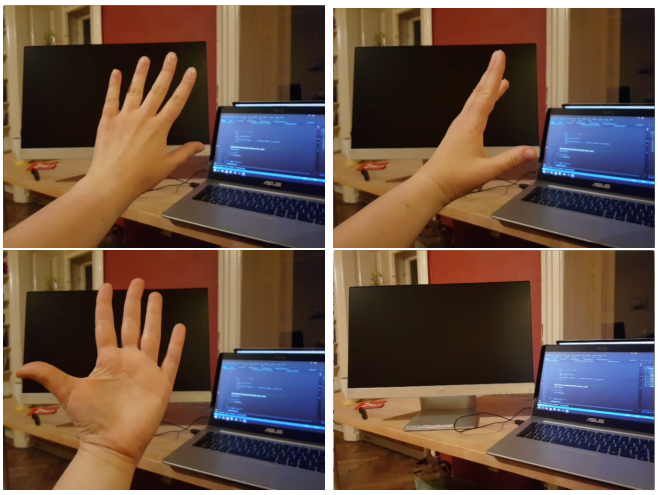


Figure 3: Input images for background subtraction

ones require initialisation, whereas the feature based ones do not. For our implementation, we used OpenCV Stitching namespace, which was implemented based on work by Brown and Lowe [23]. They used an invariant feature based approach for fully automatic panoramic image stitching. As shown in Fig. 5 (see appendix A for larger image), the image stitching pipeline consists of following main steps:

1. Registration: feature extraction and matching
2. Compositing: warping and blending images into a pano

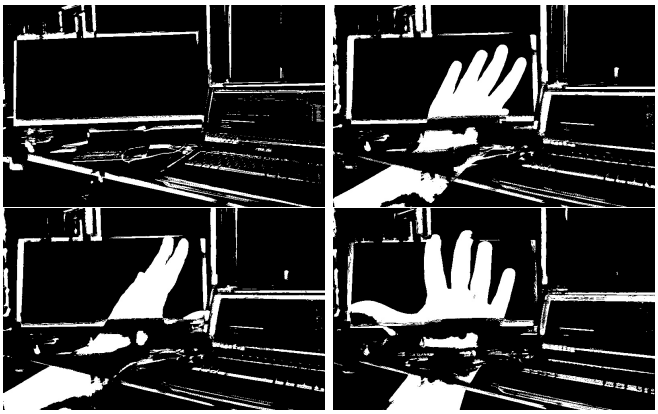


Figure 4: Results from OpenCV BackgroundSubtractorMOG

Algorithm by Brown and Lowe [23]
For n unordered images:

1. Extract SIFT features from all n images
2. Find k-nearest neighbours for each feature using a k-d tree

For each image i

1. Select m candidate matching images with most feature matches for i
2. Find homography by RANSAC
3. Verify matches by probabilistic method
4. Find connected components of the matches
5. For each connected component c:
   (a) Preform bundle adjustment
   (b) Render panorama using multi-band blending

Algorithm above lists all algorithm steps by Brown and Lowe [23] with following main steps:

1. Feature Matching
2. Image Matching

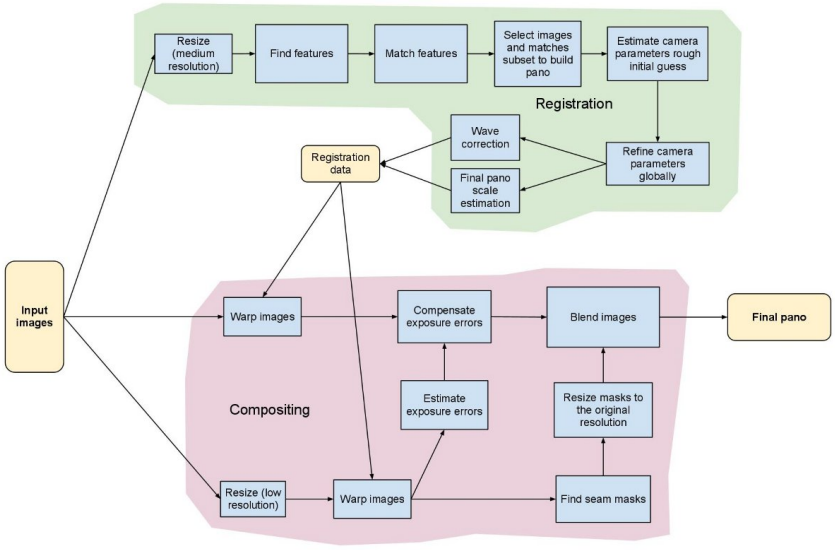


Figure 5: OpenCV Image Stitching Pipeline [5]

3. Bundle Adjustment
4. Panorama Straightening
5. Multi-Band Blending

Feature Extraction and Matching Feature extraction is the process of finding key-points of an image and representing them as a compact feature vector. Key-points are interesting points of an image that describe it. They are used to identify an object and then to compare it to some other image. Thus, feature detection, extraction and matching are combined in image processing for object detection and recognition. The most common features are edges, corners or interesting points, blobs or regions of interest. Edges have one-dimensional structure and represent points that build a boundary of an image. Most commonly, they are defined as sets of points with a strong gradient magnitude. Corners have a two-dimensional structure and are mostly called interest points. Opposed to this, blobs describe images by regions. Commonly used feature detectors for edge features are e.g. Canny [11], Harris and Stephens [17]. The latter is also used for corner detection. Laplacian of Gaussian, as well as Difference of Gaussians are both used for corner and blob detection. The authors of [23] however use Scale Invariant Feature Transform (SIFT) as a feature detector.

Scale Invariant Feature Transform (SIFT) The algorithm uses a SIFT model or an extraction of key points and computes descriptors that were published by Lowe [21]. The approached is named SIFT because it transforms image data into scale-invariant coordinates relative to local features. Generated features vectors of an image are, as the name suggests, invariant to image translation, scaling, rotation, partially invariant to illumination changes and robust to local geometric distortion.

The author presented four steps of the model that compute the set of image features:

1. Scale-space extrema detection
2. Keypoint localisation
3. Orientation assignment
4. Defining a keypoint descriptor

In the first step, a Gaussian function is applied to a scaled space of smoothed and resampled images. After calculating minima and maxima of the result of the difference-of-Gaussians functions, key locations are defined. Dominant orientations are assigned to these key points, but low contrast candidate points and edge response points are discarded. The computed key features are then indexed and matching keys are computed for the next image.

Finally, features must be matched between images. This is done by matching each feature to its k-nearest neighbours in feature space. Lowe and Brown [23] in their algorithm use a k-dimensional tree algorithm to identify nearest neighbours. A k-dimensional tree is data structure used for organising number of points in space with k dimensions. The binary search is performed, recursively partitioning the feature space at the mean in the dimension with the highest variance.

Image Matching The next step is to find all overlapping images. They must be stitched to create a panorama. Here emerges a problem: the cost rises quadratically with the number of images, because it is possible that each image could match every other. The algorithm solves this by identifying images with a large number of matches between them in the feature matching step. Random Sample Consensus (RANSAC) [15] is used to select a set of inliers compatible with a homography between the images and consequently probabilistic method is used to verify the match. RANSAC uses minimal set of randomly sampled correspondences to estimate image transformation parameters to find the solution best suiting the data.

To map the corresponding points of an image to others, a transformation matrix must be computed. This is called *homography* matrix.

The RANSAC Algorithm consist of three main steps:

1. Select a sample of points randomly
2. Fit the model to the sample
3. Check how many points agree: best estimation is with most agreement

By applying this estimation procedure, the homography matrix H is calculated. RANSAC computes inliers and outliers. The former are a set of features that are geometrically consistent, while the latter are a set of features that are inside the overlapping area but not consistent. The probabilistic verification model is used to compare the probabilities that the set of features are generated by a correct image.

Bundle Adjustment Bundle Adjustment is the problem of refining a visual reconstruction to produce a jointly optimal structure and viewing parameter estimates. It is usually a last step in most feature-based 3D reconstruction algorithms. The goal is to obtain a reconstruction that is optimal under certain circumstances regarding the noise in the observed image features. The reprojection error is expressed as a summed square error of several nonlinear, real-valued functions. The minimisation is achieved using least-squares algorithms. The authors of [23] use a Levenberg-Marquardt approach to update the parameters. Each feature is projected onto all the images where it matches and the algorithm is used to minimise the sum of squared image distances with respect to the camera parameters.

Panorama Straightening This step includes improving images so that the *up* vector is vertical. The problem is that 3D rotation to a chosen world coordinate frame is still unknown after all previous steps which would give relative rotations between the cameras. The heuristic used here is that most people usually do not twist the camera relative to the horizon, so the camera's horizontal axis typically lies in the plane. The *up* vector is found by finding the null vector of the covariance matrix of the camera x vectors, and the wavy effect is removed after applying a global rotation such that the *up* vector is vertical. [23]

Multi Band Blending Blending is necessary to prevent image blurring, as some pixels along the ray get more weight (intensity), this leads to some image edges to be more visible due to some effects such as vignetting, mis-registration errors, radial distortion etc. A weight function is assigned to each image and a

weighted sum of the image intensities along each ray is performed using those weight functions. To prevent blurring of high frequency detail due to small registration errors, a multi band blending algorithm of Burt and Adelson [10] is used by authors of [23]. This algorithm blends low frequencies over a large spatial range and higher ones over a shorter range. In their work, spherical coordinates for rendering the panorama are used.

3.6 Frame to Background Warping

After the image stitching is applied, the pano is defined and a new frame containing hand is loaded, it is necessary to fit this frame to the background image so that difference between them would be minimal. To achieve this, it is first necessary to find key points and warp them together. The workflow here is similar to the algorithm applied for image stitching, by again applying feature extraction and matching. Steps are as follows:

1. Detect and compute key features by using SURF
2. Match feature: FLANN based matcher, k-nearest neighbours
3. Use RANSAC to get homography
4. Warp hand image to background by homography using the perspective

Speeded-Up Robust Features (SURF) SURF is a a sped-up version of the SIFT algorithm. The method was published by Bay et al [8]. The algorithm has three main steps:

1. Key points detection
2. Local neighbourhood description
3. Key points matching

As Lowe et al. approximated the Laplacian of the Gaussian with difference-of-Gaussians for finding scale-space, SURF uses a box filter to further approximate the Laplacian of the Gaussian, whereas SIFT uses cascaded filters. The advantage of the box filter is that it can be calculated easily by using integral images. A blob detector based on the Hessian matrix is used to detect key points. The determinant of the Hessian matrix is used both to find the location, as it is used as a measure of local change around the points where the determinant is maximal. It is also used to select the scale. As the key points can be found at different scales, the scale space is represented as an image pyramid. Images

are smoothed with a Gaussian filter and subsampled to get the next higher level of the pyramid. To assign the orientation, Haar wavelets are used in the horizontal and vertical directions for a neighbour of $6s$, where s is the scale at which a key point was obtained. The calculated responses are weighted by a Gaussian function, cantered at the point of interest. A sliding window of size $\frac{\pi}{3}$ is used to calculate the dominant orientation. Two summed responses yield a local orientation and the longest such vector overall represents orientation of a key point. To describe the region around the key point, a Wavelet response is used. Neighbourhood of $20s \times 20s$ window size is taken for every key point and divided in 4 regions. For each subregion, horizontal and vertical responses are calculated and thus give then a SURF feature descriptor with 64 dimensions.

Perspective projection

After the homography matrix is calculated with SURF, we use the OpenCV *warpPerspective* method to apply a perspective transformation to an image. Given a homography matrix H:

$$H = \begin{matrix} \mathrm{H}_{11} & \mathrm{H}_{12} & \mathrm{H}_{13} \\ \mathrm{H}_{21} & \mathrm{H}_{22} & \mathrm{H}_{23} \\ \mathrm{H}_{31} & \mathrm{H}_{32} & \mathrm{H}_{33} \end{matrix}$$

the function applies the following transformation [6]:

$$dst(x, y) = src(\frac{H_{11}x + H_{12}y + H_{13}}{H_{31}x + H_{32}y + H_{33}}, \frac{H_{21}x + H_{22}y + H_{23}}{H_{31}x + H_{32}y + H_{33}}) \tag{8}$$

where *src* denotes source image, which is the hand image and *dst* is output, which is the warped hand image. The hand image is warped to the background in the same matter as all initial frames are matched to be stitched together. Only here the stitching is not done, as we are only interested in computing the image difference. After computing the homography and warping the hand image to the background it is necessary to then extract the ROI from the background so that images of same dimensions can be compared. Here a simple heuristic is used, by cutting all lines which have only black pixels from the warped image. Having this, y and x coordinates with height and width of the ROI in the background image are obtained.

3.7 Filtering

Although hand image warping transforms the image so it fits background, there can still occur some mismatches that when computing the difference will lead to incorporating pixels which should be ignored. In order to discard this noise, we apply median filtering. A median filter is a non-linear digital filtering technique

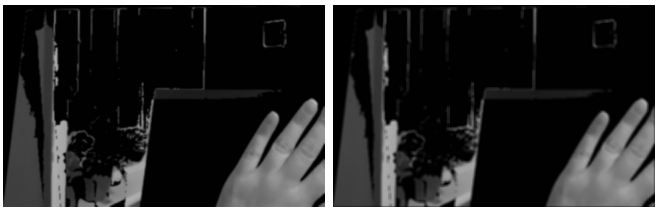


Figure 6: Gaussian and box filter.

that is widely used in image and signal processing to remove noise by smoothing the image. It goes through the image, pixel by pixel by replacing each value with a median of neighbouring entries. A window of some square size is defined, e.g. 5×5, which slides through the image. The pixel value in the centre is replaced by the median value of neighbour pixels. Besides the median filter, a normalised box filter and a Gaussian filter have been tested.

The OpenCV implementation of box filter smoothes an image by using the following kernel [3]:

$$K = \frac{1}{ksize_{width} \times ksize_{height}} \begin{bmatrix} 11.1 \\ 11.1 \\ .... \\ 11.1 \end{bmatrix} \tag{9}$$

where $ksize_{width}$ is the width of box filter and $ksize_{height}$ its height.

Applying a Gaussian filter to the image convolves the image with a Gaussian function:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2+y^2}{2\sigma^2}} \tag{10}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution [1].

In OpenCV [3] an image is convolved with Gaussian kernel:

$$G_i = \alpha e^{\frac{-(i-(k_{size}-1)/2)^2}{2\sigma^2}} \tag{11}$$

where k_{size} is the aperture size, $i = 0, ..k_{size} - 1$ and α is scale factor chose so that $\Sigma G_i = 1$, σ is Gaussian standard deviation, computed:

$$\sigma = 0.3 \times (\frac{1}{2}(k_{size} - 1) - 1) + 0.8 \tag{12}$$

Figure 7: Median filtering

Results of applying a 5×5 median filter to background and hand image are shown in Fig. 7. The upper row contains original images, the lower contains the results of median filtering.

To prevent an unnecessary loss of details, different sizes of filter windows have been chosen for images containing a hand, depending on the image area. The assumption is that hands will mostly be positioned in the central part of the image so in that area the values would not be altered much. The rest of the image might contain unnecessary details that should be removed. Thus, in the central part no filtering was done, whereas in rest of the image the filter was 5×5. The background image does not contain hands in it, so the whole image was filtered by a window of 5×5. Fig. 6 shows on the left Gaussian filter, on the right results from box filter, both of size 5×5.

Erode and Dilate

Erosion and dilation [2] are morphological operations based on image shape, normally performed on binary images. Most common uses are for noise removal, isolation of individual elements and the joining of disparate elements in an image. Dilation consists of convolving an image A with some kernel B, usually of square dimensions. Kernel B has an anchor point, which is usually the centre of the kernel. As the kernel slides over the image, the maximal pixel value overlapped by B is computed and the pixel value at the anchor point is replaced by this maximal value. This causes bright regions in image to expand. Erosion computes the minimum over the area of the kernel and replaces the anchor pixel value with that. By applying erosion, the dark regions in image are expanded. Erosion and dilation were applied on the binarily segmented image, so that the noise originating from the background subtraction would be reduced.

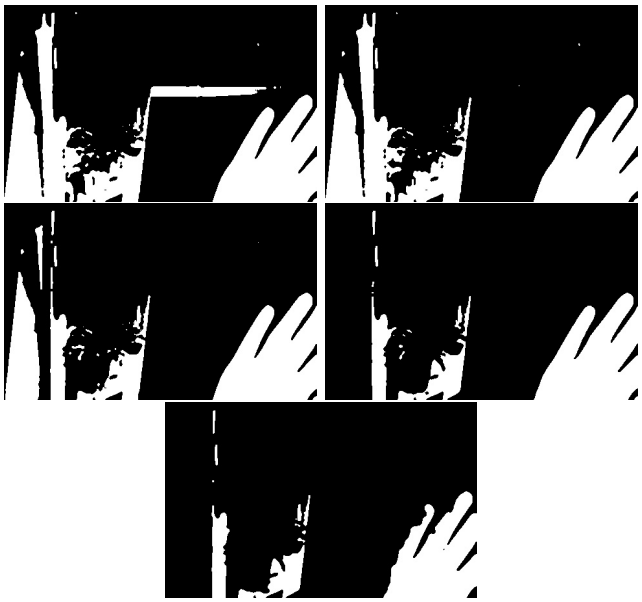


Figure 8: Comparison of different thresholds.

Binary Segmentation

After the filtering is done, the next step is converting the grey image to a binary segmentation. Here again a threshold must be defined. Binary segmentation is achieved by using that threshold value and converting all pixel values above the threshold to black and all below to white. Different threshold values on the blurred hand image from Fig. 6, are applied for comparison in Fig. 8. Beginning from the upper right, images show results from following thresholds: $\{20, 40, 60, 80, 100\}$. It has been decided to use a threshold value of *t=80* to create the binary segmentation for further processing, as this threshold seemed to yield an image which has most unnecessary background details removed and leaves the hand part mostly undamaged.

3.7.1 Contour Finding

Contour finding was done to get biggest contour which will be processed with Principal Component Analysis in order to get the orientation angle. For finding contours OpenCV *findContours* method was used. This method implements an algorithm for finding contours in binary images by Suzuki and Abe [28]. Their algorithm uses border following and topological analysis by deriving a sequence of coordinates from the border between a connected component of

Figure 9: PCA results

1-pixels (1-component) and a connected component of 0-pixels (background or hole). After finding all contours in hand image the next step is to extract the biggest contour, which in the best case is the one defining the hand. This is done by simply comparing sizes of the contours and choosing the biggest one. That contour, i.e. the points of it are then further used for PCA Analysis. An example is given in Fig. 9. Image in the top left corner is input to PCA, image right to it contains the biggest contour in red, the two below show some of the smaller contours also found by PCA.

3.8 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical procedure that extracts important features of a data set by using an orthogonal transformation to convert a set of data points of possibly correlated variables into a set of points of linearly uncorrelated variables called *principal components*.

Principal components are calculated by computing eigenvectors of a covariance matrix. The highest n eigenvectors contain the maximum variance in the original data. The principal components are orthogonal to each other and the first components is in the direction of the greatest variance. The size of the eigenvector is encoded in the corresponding eigenvalue and it indicates how much

the data varies along the principal component. All eigenvectors start at the centre of the data points. Applying PCA to a n-dimensional data set yields n-dimensional eigenvectors, n eigenvalues and one n-dimensional centre point. If the variance of the component is small, it can be omitted and by this only a small amount of data is lost. The key point of PCA is *Dimensionality Reduction*. Dimensionality Reduction is a process of reducing the number of dimensions of a data set.

When calculating the PCA, steps are:

1. Compute the d-dimensional mean vector
2. Compute covariance matrix of whole data set
3. Compute eigenvectors and corresponding eigenvalues
4. Sort eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues

Given is a data set containing n observations of p variables. Firstly, data is to be organised in n set of vectors with each representing a single grouped observation of the p variables. An empirical mean vector u with dimensions $p \times 1$, along each dimension $j = 1.., p$ is calculated:

$$u(j) = \frac{1}{N} \sum_{i=1}^{N} x_{(i,j)} \tag{13}$$

The mean vector is then subtracted from each row of data matrix X:

$$B = X - hu^T \tag{14}$$

where h is an $n \times 1$ column vector of all 1s. The next step is to find the covariance matrix C:

$$C = \frac{1}{n-1} B * \otimes B \tag{15}$$

Where $\otimes$ is conjugate transpose operator. Having the covariance matrix C, the next step is to calculate eigenvectors and eigenvalues of it:

$$V^{-1} C V = D \tag{16}$$

where D is diagonal matrix containing eigenvalues of C in the form of an $p \times p$ diagonal matrix:

$$D[k,l] = \begin{cases} \lambda_k, k = l \\ 0, kl \end{cases} \tag{17}$$

λ_k is the k-th eigenvalue of the covariance matrix C. Matrix V, also of dimension $p \times p$, contains p column vectors, each of length p, which represent the p

Figure 10: PCA results with middle point

eigenvectors of the covariance matrix C. The eigenvalues and eigenvectors are ordered and paired. The j-th eigenvalue corresponds to the j-th eigenvector. Eigenvectors $e_1, e_2, ...e_p$ with corresponding eigenvalues λ_1 through λ_p sorted:

$$\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_p$$

The variance of the i-th component is equal to the i-th eigenvalue:

$$var(Y_i) = var(e_{i,1}X_1 + e_{i,2}X_2 + ... + e_{i,p}X_p) = \lambda_p \tag{18}$$

The principal components are uncorrelated with one another:

$$cov(Y_i, Y_j) = 0 \tag{19}$$

The eigenvectors with the lowest eigenvalues carry the least information about the distribution of the data, so they can be ignored. As we are interested in the orientation, we take the first two eigenvectors and corresponding eigenvalues. One other important result of PCA is the determination of the middle point which is used later as the middle point of the hand and thus the hand's position.

The hand orientation angle is calculated by computing the tan of eigenvectors:

$$\alpha = \tan(\frac{e_1}{e_2}) \tag{20}$$

where e_1 and e_2 are largest eigenvectors respectively. Fig. 10 shows the result of PCA applied on the biggest hand contour. The blue axes represent principal components, starting from the calculated middle point of the hand.

3.9 Finger Tip Recognition

In combination with hand contours, one additional heuristic that was tested was to count the number of finger points to make gesture recognition easier. The

idea is to define number of finger points and rotate the fingers of the hand model correspondingly, so that the starting position and rotation of fingers reduces the subsequent process of gesture recognition.

The process includes

1. Describing contours with convex polygons
2. Finding convexity defects
3. Filtering out irrelevant information

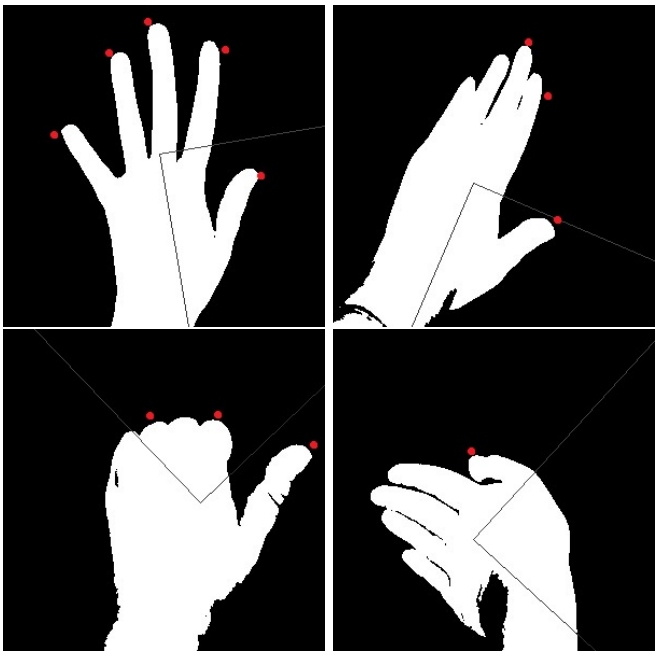


Figure 11: Results of finger tips recognition

Convexity defects are points furthest away from each convex vertex. This heuristic gives additional information about the number of fingers and their position. However, the heuristic did not yield good enough results. Although the number of fingers can be calculated, the exact finger cannot be identified so it can lead to misinformation which is not helping the gesture recognition process. Also, the position of the finger does not tell anything about which finger it is.

In Fig. 11 we can see the results of the fingertip recognition on different hand gestures. The recognised finger tips are marked with red points. As one can see the upper right image is a result of successful recognition, where all five

tips were recognised. This clarity of the convexity defects is helped by the fact that the fingers are spread. However, the remaining three images show that this does not succeed when the fingers are closed or close to each other, so the process could not identify a convexity defect for each fingertip. For testing of this heuristic, we did not use background subtraction to extract hands, but simple binary segmentation of normal hand images.

3.10 Fast Approximate Nearest Neighbour Search

One approach to find hand gestures could be to define a set of classified hand gestures with a fixed number of classes and to try to match every new hand gesture to one of these classes. The disadvantage to this is we need a huge database with initialised classes and to run some classification algorithm for every incoming frame.

We tested this with OpenCV library called FLANN: Fast Library for Approximate Nearest Neighbors [4], which is based on an algorithm from Muja and Lowe [24]. It is a library in OpenCV for performing fast approximate nearest neighbour searches in high dimensional spaces. It is optimised for fast nearest neighbour search in large datasets.

Algorithm

1. Initialize database: images from some input set
2. Initialize mapping list: empty
3. Initialize descriptors matrix: empty

For every image from database:

1. Detect key-points and compute descriptors from key-point locations:
2. Create a mapping with class name and mark start and end index in descriptors matrix : *m.Similarity = 0*
3. Add descriptors to matrix

For any query image q:

1. Calculate descriptor: *d(q)*
2. Initialize FLANN index for database descriptors: f
3. Perform k-nn search: output is
   (a) List of distances to k-nearest neighbours: *S(n,k)*
   (b) List of indices of k-nearest neighbours found: *I(n)*

1. For every index from *I(n)*:
   $if(S(i,0) < 0.6 * S(i,1))$: increase similarity of mapping M(i)
2. Sort the mappings , choose the one with highest similarity as a resulting class

The prerequisite is to create a database of images which contains different hand gestures. The first step is then to calculate feature descriptors on all images from this database. Descriptors from all images are then concatenated in one matrix. Additionally, there is a list of mappings, where one mapping includes the class name and similarity value, as well as start and end indices indicating part of the concatenated matrix that contains the descriptors for that class. This serves as a database and all images for which the gesture must be guessed is going to serve as a query. To define the gesture that yields the biggest similarity for the query image, the descriptor of the query image must be calculated. Here the process is the same as for calculating descriptor for a single database image.

Figure 12: Database images (FLANN)

Having the prerequisites, a query descriptor and a concatenated database of images descriptors, FLANN can be invoked. The OpenCV Index is initialised with database features and a kd-tree parameter, which defines that the constructed index will consist of a set of randomised kd-trees which will be searched in parallel. For our test we used 4 kd-trees. Then a k-nearest-neighbour is performed with 2 nearest neighbours. The FLANN calculates distances to k-nearest neighbours and returns indices of the k-nearest neighbours. If the distance to the first nearest neighbour is less than 60% of distance to second neighbour, we consider the image with the corresponding index from database similar to the query image, so its similarity value in mapping is increased. Finally, the mappings are sorted and the one with the biggest similarity value indicates the class of the image from database.

Database images are shown in Fig. 12 and Fig. 13 shows the results of FLANN on different hand gestures. For every query image we added five images of similar gestures to the database. Fig. 13 includes seven query images of different hand gestures, one row for each query image. Left column are input (query) images, right is the image from database that had highest similarity value. In comparison, one can see that the first query image yields good results, but image with left hand thumb up (second row in the table on the right) yields bad output. This happens because first image had an almost identical image in the database, whereas the images with the thumb up were very different.

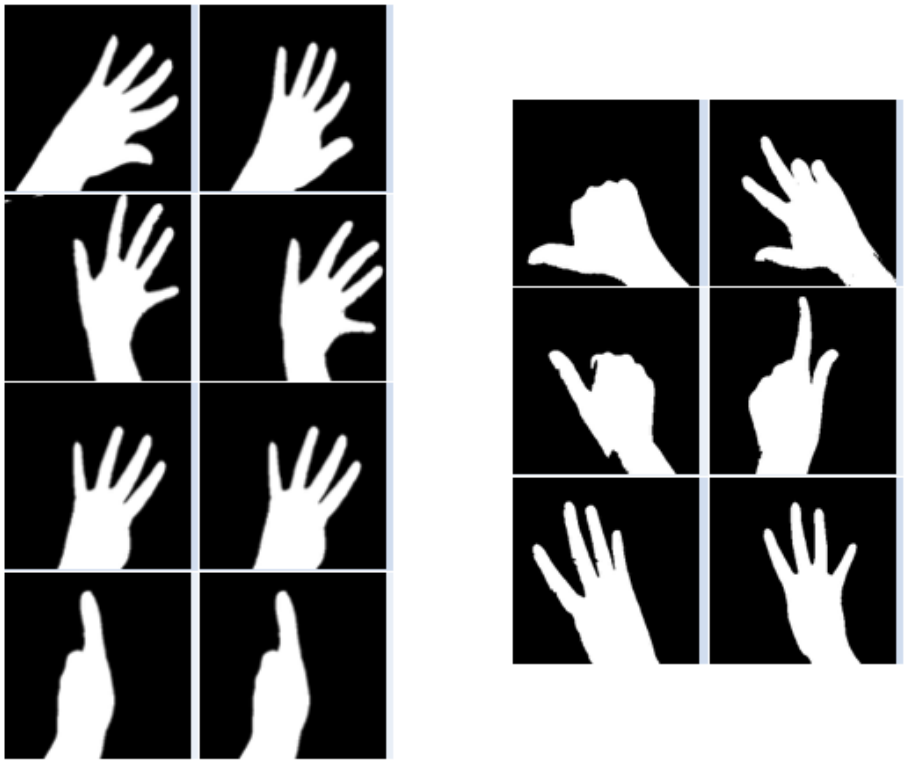


Figure 13: FLANN results

4 Hand Gesture Recognition

This section describes methods used for hand gesture recognition. The result of hand extraction, the binary segmentation of the hand is used in this process in order to define the gesture and to model it in Unity3D. Two different methods were used here, one is an implementation of Levenberg-Marquardt algorithm, the second is a heuristic, named Iterative Heuristic. Both are explained in detail here and the results are compared.

4.1 Levenberg-Marquardt

Levenberg-Marquardt (LM) is a technique used to solve a non-linear least squares problem. Least squares is a method in regression analysis to approximate solution of overdetermined systems. Overdetermined systems are sets of equations where there are more equations than unknowns. By this approach the solution minimises the sum of the squares or the residuals made in the results of every equation. The residual is the difference between an observed value and the fitted value provided by the model. The best fit in the least-squares minimises the sum of squared residuals. There are two classes of least-squares: linear and non-linear. Linear apply to all residuals which are linear in all unknowns.

The non-linear least squares method is used to fit a set of m observations with a model that is non-linear in n unknown parameters. The number of observations m is greater than number of parameters n . The basic approach is to approximate the model by a linear one and to refine the parameters by successive iterations.

Levenberg-Marquardt interpolates between the Gauss Newton algorithm and the gradient descent method. It is more robust than Gauss Newton, as it finds solution more often even it if starts very far off from the final minimum. The goal is to find the parameters that minimise the sum of the squares of the deviations. This is an iterative procedure that starts with an initial guess of parameters that are then updated in each iteration until some of the termination criteria are met.

The least squares problem can be described as an optimisation problem and can be written as follows:

$$min_x \left\| r(x) \right\|^2 \tag{21}$$

where x is a vector of model parameters and r is residual vector, which is formed from individual residuals:

$$r(x) = (r_1(x), r_2(x), ...r_m(x))^T \tag{22}$$

where m is the number of data points (measurements) and each $r_j(x)$ represents the difference between the expected and measured value:

$$r_j(x) = y(\bar{x_j}) - \bar{y_j} \tag{23}$$

y is the model function that returns expected value for the given point $\bar{x_j}$ and the model parameters, while $\bar{y_j}$ value is the observed quantity. The 2-norm from eq. 23 can be rewritten in terms of the objective function in the following form:

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x) \tag{24}$$

The goal is to find x for which the function value is minimal. The model function can be rewritten to following form:

$$y(a) = x_1X_1(a) + x_2X_2(a) + ... + x_nX_n(a) \tag{25}$$

x are model parameters and X are basis functions that can be any functions of x and can be nonlinear. The derivatives of residuals with respect to model parameters are:

$$\frac{\partial r_j}{\partial x_i} = \frac{\partial}{\partial x_i} = \Big[y(\bar{x_j} \mid x) - \bar{y_j}\Big] = X_i(\bar{x_j}) \tag{26}$$

The partial derivatives can be defined in a single (design) matrix: $A_{ji} = X_i(\bar{x_j})$, and the residuals can be defined as:

$$r_j = \Big[\sum_{j=1}^{n} x_i X_i(\bar{x_j})\Big] - \bar{y_j}, j = 1, ...m \tag{27}$$

When dealing with nonlinear functions we are defining Jacobian matrix for residuals

$$J_{j,i}(x) = \Big[\frac{\partial}{\partial x_i} r_j\Big] \tag{28}$$

which is constructed directly from the model function:

$$J(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} y(\bar{x_1} \mid x) & \frac{\partial}{\partial x_2} y(\bar{x_2} \mid x) & . & \frac{\partial}{\partial x_n} y(\bar{x_n} \mid x) \\ \frac{\partial}{\partial x_1} y(\bar{x_1} \mid x) & \frac{\partial}{\partial x_2} y(\bar{x_2} \mid x) & . & \frac{\partial}{\partial x_n} y(\bar{x_n} \mid x) \\ . & . & . & . \\ \frac{\partial}{\partial x_1} y(\bar{x_m} \mid x) & \frac{\partial}{\partial x_2} y(\bar{x_n} \mid x) & . & \frac{\partial}{\partial x_n} y(\bar{x_m} \mid x) \end{bmatrix} \tag{29}$$

Since the objective function is $R^n \to R$, the gradient and Hessian are used:

$$\nabla f(x) = \sum_{j=1}^{m} r_j(x) \nabla r_j(x) = J(x)^T r(x) \tag{30}$$

$$\nabla^2 f(x) = \sum_{j=1}^{m} \nabla r_j(x) \nabla r_j(x)^T + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x)$$

$$= J(x)^T J(x) + \sum_{j=1}^{m} r_j(x)\nabla^2 \tag{31}$$

When solving the nonlinear squares, the second summation term in the Hessian is negligible and can be omitted, which leaves:

$$\nabla f(x) = J(x)^T r(x) = g(x) \tag{32}$$

$$\nabla^2 f(x) \approx J(x)^T r(x) = H(x) \tag{33}$$

$J(x)$ is a Jacobian matrix. It contains first-order derivatives of the residuals with respect to every parameter and every measurement. The gradient $g(x)$ contains first-order derivatives of the objective function with respect to the model parameters. It describes the slopes of the objective function at some point x. $H(x)$ is the Hessian matrix and it contains the second-order partial derivatives with respect to every combination of parameters. It describes the curvature of the objective function at some point x.

In every iterative step, the parameters are updated:

$$x(k+1) = x(k) + p(k) \tag{34}$$

Steepest Descent Method

One of the approaches here for finding the function minimum that can be used is the steepest descent method. This method consists of three steps in each iteration:

1. Select step direction p
2. Select step length a
3. Update parameters with the steepest descent step $x(k+1) = x(k) + p \times a$

The steepest descent step can be expressed as:

$$p_{(k)}^{SD} = \frac{-\nabla f_{(k)}(x)}{\left\|\nabla f_{(k)}(x)\right\|} a \tag{35}$$

or in terms of the Jacobain and residuals:

$$p_{(k)}^{SD} = \frac{-J_{(k)}r(k)}{\left\|J_{(k)}r(k)\right\|}a \tag{36}$$

Gauss-Newton Method

This method contains an improvement of Newton's method for solving the optimisation problem. The Newton step:

$$p_{(k)}^{N} = -\nabla^2 f_{(k)}^{-1}\nabla f_{(k)} \tag{37}$$

Incorporating the Hessian into formula:

$$p_{(k)}^{GN} = -H_{(k)}^{-1}g_{(k)} \tag{38}$$

With highly non-linear functions, the Hessian matrix can get singular or near-singular with the Gauss Newton method. Also, the initial guess is far from the minimiser. Thus, it can converge very slowly or not converge at all. In order to address these problems the Hessian can be modified to improve the convergence. The step direction is a descent direction whenever Hessian is positive definite. As the function can be concave in some points and convex in others, this may not be the case because the Hessian is indefinite. So, to make the Hessian approximation positive definite we can simply add a multiple of the identity to it:

$$H(x) = H(x) + \lambda I \tag{39}$$

Altering of the diagonal elements of $H(x)$ is called damping and factor λ is a positive number and it is thus called the damping term [20]. I is an identity matrix of size $n \times n$. This step is adding positive numbers to the diagonal of the Hessian matrix to make it diagonally dominant and thus positive definite.

If the Hessian matrix from the Gauss-Newton step is replaced with an updated version with the damping parameter, we get:

$$p_k^{L} = -(H_k + \lambda I)^{-1}g_k \tag{40}$$

The smaller the λ is, the more method approaches Gauss-Newton, the larger the λ is, the smaller and safer steps are made in the descent direction. Because λ is scalar, the diagonal elements of the Hessian are scaled equally. As different parameters can be scaled differently, Marquardt updated the step, so that instead of identity matrix, the diagonal is used, which takes scaling into account:

$$p_k^{LM} = -(H_k + \lambda D)^{-1}g_k \tag{41}$$

where $D = Diag(H_k)$

This Levenberg-Marquardt step from eq. 41 can be rewritten in terms of the Jacobian and the residuals to the following:

$$(H_k + \lambda D)p_k^{LM} = -g_k \tag{42}$$

$$(J_k^T J + \lambda D)p_k^{LM} = -J_k^T r_k \tag{43}$$

This leads to the Levenberg-Marquardt algorithm with following steps:

1. Initialise λ
2. Compute current value of function
3. Compute Jacobian and residuals
4. Compute LM step
5. Update parameters by λ factor
6. Compute new value of function
7. If a new value is smaller than the current value, increase λ , else decrease it and update parameters and values with new ones.
8. If the conditions for termination are met, stop the process, else go to step 3.

As this is an iterative process, the conditions for its termination must be defined. The conditions for termination of LM algorithm are [20]:

- The magnitude of gradient $J_k^T r_k$ drops below a threshold ϵ_1
- The relative change in magnitude of p_k^{LM} drops below a threshold ϵ_2
- The error $J_k^T r_k$ drops below a threshold ϵ_3
- A maximum number of iteration k_{max} is reached.

4.2 Levenberg-Marquardt for Hand Gesture Recognition

The hand gesture recognition process can be defined as an optimisation problem with an aim to minimise the objective function. The objective function itself is an image, the binary segmentation of a hand. The residual is the difference between the incoming frame and a current snapshot of the 3D hand model. The snapshot is transformed to binary segmentation, so that compared images contain only black and white pixels. The goal is to find the rotation and translation

Figure 14: 3D Hand model

parameters that transform the model hand so that the difference between these two segmentations is minimal.

In the modelling component which is running in Unity3D, the hand model is located at some initial position in neutral rotation. Every time when this component receives input from the image processing component, the hand is reset to this neutral position and rotation and the LM algorithm is triggered. At every frame, the hand is updated by current parameters, a screenshot of the 3D model is taken and a binary segmentation of it is created. This is then compared to the input image and the error is calculated. The error between two images is simply the number of pixels that mismatch. The goal of the algorithm is to minimise this number of mismatching pixels.

Model function

In order to apply LM to solve a system of non linear equations it is important to identify the model function in the case of recognizing the hand gesture. We are working with binary segmented 2D images, so the best way to compare two of them is to iterate through all pixels and count the number of mismatches.

Other criteria such as histogram comparison, for example, would not be helpful here. The histogram of an image represents the distribution of pixels colours. As in our case we have all black values for the background and white values for hand section, the histogram of two completely different images could be same or very similar.

Our goal is to find a set of parameters, which when applied to our 3D hand

model produce a minimum difference to the input frame coming in from camera.

A snapshot of the 3D hand model used in Unity3D is shown in Fig. 14. The hand joints are defined as data points with its rotation angles as parameters. Specifically, data points are x, y and z coordinates of a joint. At every iteration of the LM algorithm, a joint is rotated by an updated parameter and the difference between the snapshot of this new hand model and input frame is calculated. In order to define objective function in terms of rotation angles for joints we first list basic rotations that can be applied to hand model.

Rotations

Basic rotations around the axes of a coordinate system Around X axis:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{44}$$

Around Y axis:

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{45}$$

Around Z axis:

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{46}$$

Every hand joint can be rotated to some degree around these axes. We use this to construct our objective function and further for the Jacobian matrix and the residual.

Rotation parameters

The implemented LM seeks to find the best parameters for the rotation of fingers around x axis. The parameter is the rotation angle for each finger that must be defined:

$$\theta_{littleFinger}, \theta_{indexFinger}, \theta_{middleFinger}, \theta_{ringFinger}, \theta_{thumb}$$

At the beginning of the process, some initial guess value is assigned to them. For each of the fingers and parameters we define the rotation equations over the X axis:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{47}$$

If we define x, y and z as coordinates for joint position, after rotating the finger around X so the new position will be:

$$\begin{bmatrix} x^{'} \\ y^{'} \\ z^{'} \end{bmatrix} = R_x(\theta) \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x^{'} \\ y^{'} \\ z^{'} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$x^{'} = y$$
$$y^{'} = \cos(\theta)y - \sin(\theta)z$$
$$z^{'} = \sin(\theta)y + \cos(\theta)z$$

By using this formula updated x, y and z coordinates of each finger can be calculated.

Implementation

The parameters needed for the LM algorithm are initialised as follows:

- Minimum Error $e_{min} = 0.0019$: If the calculated error reaches this value, the process is stopped.
- Maximum number of iterations: $k_{max} = 20$
- Lambda initial = 0.1. Initial guess for λ.
- Lambda factor = 0.01. Factor by which λ is updated.

Parameters that are calculated in every iteration k:

1. Minimum Delta Value (*MinDeltaValue*): given the minimum error e_{min} and e_k, which is the error in the current iteration, the delta value is calculated as:
$$\left|(e_k - e_{min})\right|$$
If this value is smaller than the defined *MinDeltaValue*, the iteration stop is triggered.
2. Minimum Delta Parameters (*MinDeltaParams*): this is calculated as a L^2-norm of the parameters vector from current iteration p_k^{LM} and previously retrieved optimum parameters p^{LM}:
$$p = p_k^{LM} - p^{LM}$$

$$|p| = \sqrt{\sum_{i=1}^{n} |p_i|^2}$$

If this value is smaller than MinDeltaParams, the iteration is stopped.

MinDeltaValue and MinDeltaParams are set initialized with value 0.000001. The initial guess parameters for x rotation angle was set to following values:

$$\theta_{littleFinger} = \theta_{indexFinger} = \theta_{middleFinger} = \theta_{ringFinger} = \theta_{thumb} = 10$$

At each iteration, the residual is computed:

- defined joints of the hand model are rotated by the new θ
- a screenshot is taken
- number of pixels that mismatch are counted.

In eq. 23 the residual has been defined as the difference between the expected and measured value:

$$r_j(x) = y(\bar{x_j} \mid x) - \bar{y_j}$$

$y(\bar{x_j} \mid x)$ is the input frame from the Image Processing Component, $\bar{y_j}$ is the snapshot of the 3D hand model, rotated by the θ parameters from the current iteration k. $\bar{x}$ is a data point, which is in our case is the joint position, j is the current parameter, which is the rotation angle θ. The residual is calculated as a difference between the observed and the modelled image, which is simply a number of pixels whose values do not match:

$$r_j = \frac{count}{width * height} \tag{48}$$

count is the number of pixels that mismatch. The value is normalised by dividing it with the total number of pixels. This residual is squared and summed up for all joints r_j:

$$\frac{1}{2} \sum_{j=1}^{m} r_j^2(x)$$

Jacobian

The number of rows in the Jacobian matrix corresponds to number of data points (joints) , and the number of columns to the number of parameters, which are in our case the rotation angles. Each row in the Jacobian matrix contains a gradient value for parameters that is computed as follows.

Each finger position contains three coordinates: x, y, z. However, we can choose only one as our data point for the model function, because the rotation angle on an axis is the same for all three. For a y coordinate that was chosen and that was previously defined, a new y value in respect to rotation can be calculated:

$$y^{'} = \cos(\theta)y - \sin(\theta)z$$

Gradient in respect to θ:

$$\frac{\partial}{\partial\theta} = -z\cos(\theta) - y\sin(\theta)$$

Hessian

Hessian is calculated by transposing the Jacobian and multiplying it by itself:

$$\nabla^2 f(x) \approx J(x)^T J(x) = H(x)$$

Further, we build the diagonal matrix D from it and finally, the LM step is calculated:

$$(H_k + \lambda D)p_k^{LM} = -J_k^T r_k$$

This system of non-linear equations is solved by using Cholesky decomposition of the left side matrix $(H_k + \lambda D)$. Cholesky factorization decomposes the matrix into the product of a lower triangular matrix and its conjugate transpose. The result is a vector, whose size is number of parameters. The old parameters are updated by subtracting the step value.

With the updated parameters p_k^{LM}, the objective value is calculated and compared to previous value. If the computed objective value with new parameters is smaller than previous, λ is increased by a defined factor, parameters are stored as optimum parameters, and the minimum error is also updated:

$$\lambda = \lambda - lambdaFactor$$

$$p^{LM} = p_k^{LM}$$

$$e_{min} = e_k$$

Otherwise is increased:

$$\lambda = \lambda + lambdaFactor$$

4.3 Iterative Heuristic

In addition to Levenberg-Marquardt, we also implemented a simple heuristic to compare the results. This iterative heuristic consists of defining an initial rotation angle for every finger and then, in a limited amount of iterations updating

the angle for each finger by some angle and calculating the error. At the end, an angle that yielded minimal error is chosen as the best rotation angle for the finger. The results of this heuristic are shown in next section.

The process for finding optimum rotation angle ϕ for each finger is:

1. Calculate the initial error e_s, which is the number of pixels that differ.
2. Define the minimum ϕ_{min} and maximum rotation angle ϕ_{max}
3. Define increase factor u
4. Initialize angle $\phi_1 = \phi_{min}$
5. Initialize $e_{min} = e_s$
6. Initialize $\phi = \phi_{min}$
7. Until ϕ_{max} is reached:
   (a) Stepwise increase angle by a factor u:
   $\phi_i = \phi_{i-1} + u$
   (b) Calculate the current error e_i
   if $e_i < e_{min}$:
   $e_{min} = e_i$
   $\phi = \phi_i$

The resulting rotation angle ϕ is considered to be the angle by which finger should be rotated, so the error between the incoming frame and the current snapshot of 3D model is minimal.

4.4 Hand Model

As this work concentrates on hand tracking and gesture recognition in this section we describe the model of the human hand. We do not concentrate on the skin of the hand, but on the skeletal form and motion. Therefore, we present a kinematic model of a hand. It is kinematic because it describes a set of motions that a hand can perform. The human hand consists of several joints that are flexible in motion and allow several degrees-of-freedom (DoF). The biology of a hand is complex with its bones and muscles, but the model used in this work is simplified by using an skeletal model, which is presented in Fig. 15. A similar model has been used in the works of [19] and [16].

The human hand consists of five fingers: little, index, middle, ring and thumb. Each hand contains four metacarpal (MC) bones, all fingers but the

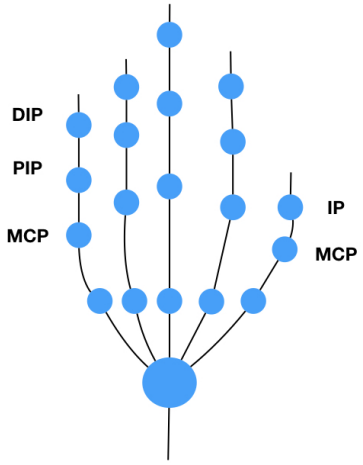


Figure 15: Skeletal model of human hand depicting finger joints.

thumb consist of three bones each: proximal (PP), middle(MP) and distal phalanx (DP). The thumb consists of MC, PP and DP [16]. The joints of a hand represent constraints that define the degrees-of-freedom. The human hand has 27 DoF. The metacarpophalangeal joint (MCP) constrains the pose of PP and MC, the proximal interphalangeal joint (PIP) constrains MP to the PP, the distal interphalangeal joint (DIP) constrains the DP to MP, the interphalangeal joint (IP) constrains DP to the IP in the thumb. These joints are shown in Fig. 15. As one can see, the thumb consists of MCP and IP, all other fingers of DIP, PIP and MCP. The DIP and PIP each have one DoF, MCP has two due to flexion and abduction. The thumb differs, as it has five DoF. The remaining six DoF are due to wrist's rotation and translation DoF.

The recognition of hand gesture by using this model brings a certain complexity, so we decided it has been decided to limit the scope of this work to wrist translation and rotation, as well on rotation of the fingers on the fixed axis. If we imagine a standard coordinate system with three axes, X, Y and Z, where the Z axis implies depth, the rotation of the fingers was done around the X axis. The wrist has been translated on the Y and X axes, with fixed depth. The rotation of wrist was done around the Z axis.

5 Evaluation

5.1 System Overview

The implementation includes two separate components, as shown in Fig. 16, which are:

1. Image Processing Component
2. Hand Modelling Component

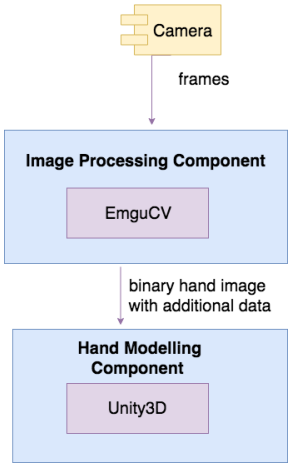


Figure 16: Solution system overview

The Image Processing Component was implemented in C# using the EmguCV library, which is a wrapper of OpenCV. Its processing includes image stitching and background subtraction and PCA on the binary segmented hand image. The result of the process is a binary image that is sent to the second component. The data package sent includes:

- Binary image
- Results from PCA: calculated orientation and middle point position.

The Hand Modelling Component is an Unity3D application, which reads the data sent from the first component, runs the hand recognition algorithm and

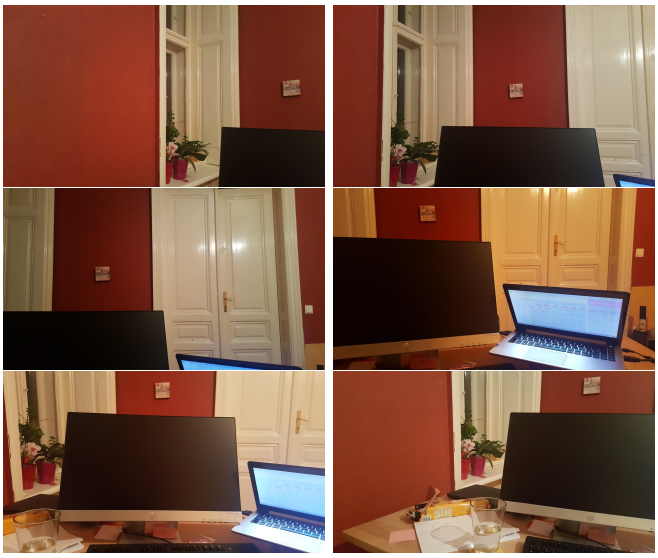


Figure 17: Images used for building background pano

all the approximations on a 3D hand model, to rotate and position the hand in the best possible way.

The data between these two components is sent via User Datagram Protocol (UDP). UDP was chosen because it uses a simple connectionless communication model with a simple protocol mechanism. Although in UDP there is no guarantee of delivery, it is still preferred as it avoids the overhead of handshaking, error checking and correction. The drop of packets, which are camera frames in this case, is preferable to network delays that occur when there are additional checks.

5.2 Hand Extraction

The first step is to take photos of the surroundings. In the setup described in this work, the user is placed at his working desk, the environment is thought to be static, (i.e. minimum changes in object positions are supposed to change). The user rotates his head from left to right and takes some photos of the surroundings. The results are background photos shown in Fig. 17. Taken photos are first stitched to a pano, which will later be used as a background for hand recognition. After applying image stitching the result, the background is shown in Fig. 19. By doing this, background building is done complete and all further steps include hand extraction and gesture recognition.

Figure 18: Image containing hand

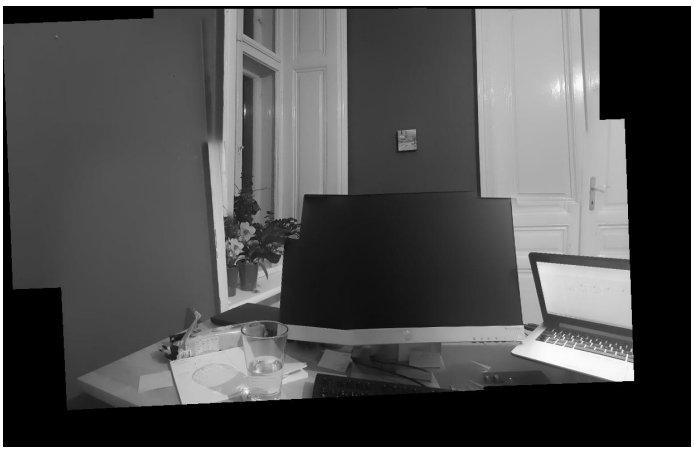


Figure 19: Background pano

Images for building background pano are taken only once at the beginning of the process. All subsequent images are considered to contain hand and are processed for hand extraction and finally, gesture recognition. Fig. 18 is an example image of a hand for which we apply algorithms to recognise a hand gesture and in the final step, the 3D model. The hand image must match the background by homography, so the first step is to warp it to the background. As one can see in Fig. 21, the warped image contains a redundant black area, which needs to be cropped out. By doing this, columns and rows are defined which will be taken as as a ROI from a background pano. After cropping this out the results are images in Fig. 20. The left image is ROI from hang image, the right image from the background. These two images can be compared to then extract the hand.

The next step is to calculate the difference between the ROI from the background pano and the warped hand image. This is done by simply subtracting

Figure 20: Extracted regions of interest (ROI)

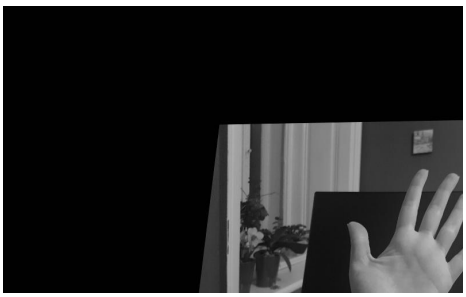


Figure 21: Warped hand image

every pixel value of hand image from background pano. The resulting image is the left image in Fig. 22. In order to create a segmentation image, thresholding

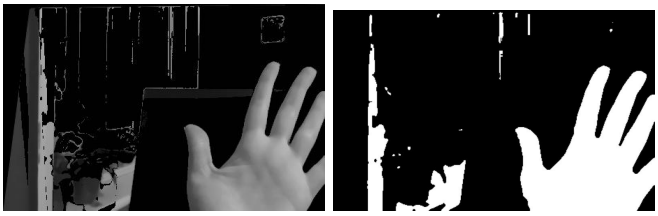


Figure 22: Difference between background and hand image and the binary segmentation

is applied. Different threshold values have been tested here. The threshold value of $t = 80$ seemed to produce the clearest segmentation. The result is the image on the right in Fig. 22. This is the image that is sent via UDP socket to the Unity3D component. The next step is applying Principal Component Analysis to calculate the orientation angle. Having done this, the image can be further sent to the next component, which is applying the hand recognition algorithm and modelling the Unity3D hand model.

The workflow and techniques we described previously used for hand extrac-

Figure 23: Different illumination settings: input

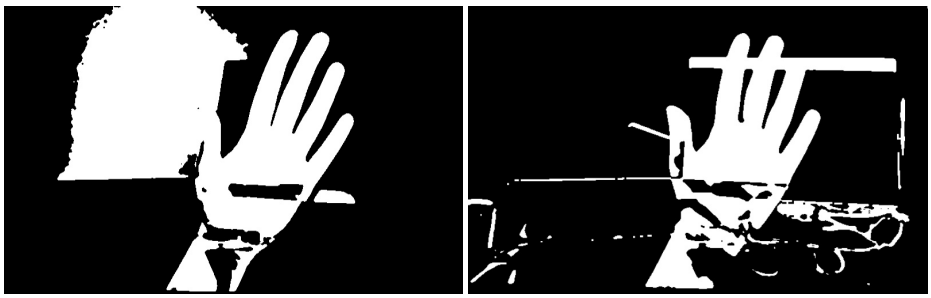


Figure 24: Different illumination settings: output

tion have some shortcomings and limitations. The process we described in this work relies on the assumption that the result of background subtraction will be a hand segmentation that can be fed into the Hand Modelling Component. However, if the background subtraction does not produce an usable result, it cannot be proceeded to the other step of gesture recognition. The limitations of the background subtraction method are:

1. Illumination
2. Camera angle
3. Changes in background

Illumination The background subtraction approach is sensitive to sudden illumination changes. This means that if the background pano is created under one lightening condition, but hand images come from a differently lit setting, most often the resulting segmentation will contain noise due to this illumination change. To analyse the limitation on different lighting settings in the surroundings, we created one background image with normal lightening conditions, i.e. room was lit up by bulb and then took two images of hands, one without camera

Figure 25: Different camera angles: input

flash and one with. Fig. 23 shows image taken with flash on the left and image with normal illumination on the right. The resulting hand segmentations, produced by background subtraction are presented in Fig. 24. The image on the left is the result of an input image taken with flash, image on the right is result of an image taken without flash. The flash image produced a segmentation containing an additional area marked white, due to the big difference in pixel intensities. Image on the right does not contain such artifacts. The conclusion is that when creating a background pano and later taking the hand images, the lightening setting in surroundings must not change suddenly and dramatically, otherwise the segmentation will include additional redundant areas.

Camera angle One important step at extracting the hand from the background is warping the hand image to the background pano. This is described in subsection 3.6. In that process, the hand image is warped to the background by a homography matrix using the perspective transformation. In case the camera orientation angle changes too much, the warping will not be successful, and it will not be possible to extract the hand. The image on the left in Fig. 25 is an example of an input image, where camera angle did not change much, whereas image on the right is an example where camera changed the orientation angle. It appears as the camera translated to the left and then rotated it so that it is more orthogonal to the black monitor than to the hand itself. This resulted in warped images in Fig. 26 in first row and hand segmentations in the second row. As one can see this result is useless and cannot be used for further gesture recognition. Thus, when taking images of the hand, the camera orientation and position cannot be changed to an extent where the incoming images can no longer be matched to the background pano.

Changes in background The prerequisite for using static background subtraction, as it is used here, is to have objects that do not change their position during the recognition process. This means that once the background pano had been created, objects that are part of it must not change their positions, because they might be falsely recognized as the hand. Further, no new objects other

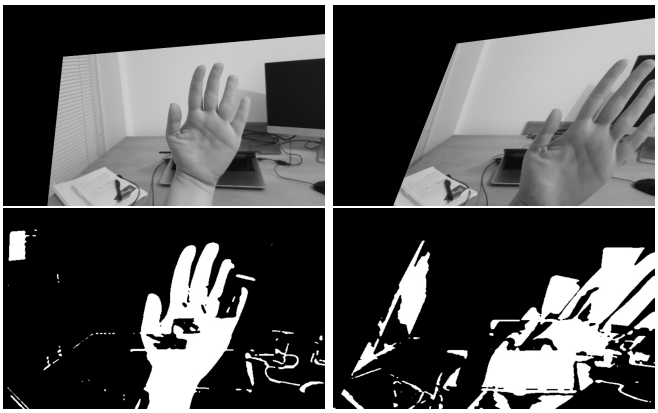


Figure 26: Different camera angles: output

Figure 27: Background objects: input

Figure 28: Background objects: output

Figure 29: Background objects moving: input

Figure 30: Background objects moving: output

than hand itself must be part of the incoming frames that are used for hand extraction. Fig. 27, left contains an image which has the same objects as the background pano. The image on the right in this figure has the difference that the laptop lid is open. Because of that, the resulting segmentation presented on the right in Fig. 28 does not contain clear distinction of the hand and cannot be used for further processing. An example with existing background objects, which are moving is presented in Fig. 29. The laptop in image on the right has changed the position and because of that the segmentation in Fig. 30 on the right does not show clear distinction of hand, whereas the one on the left does.

5.3 Hand Gesture Recognition

The Unity3D application waits for an input image to be received via UDP socket. Once the component receives an image, it starts the preparation for gesture recognition, which consists of the following steps:

1. Reset fingers to neutral rotation

   All finger joints have their defined neutral position, to which they are reset

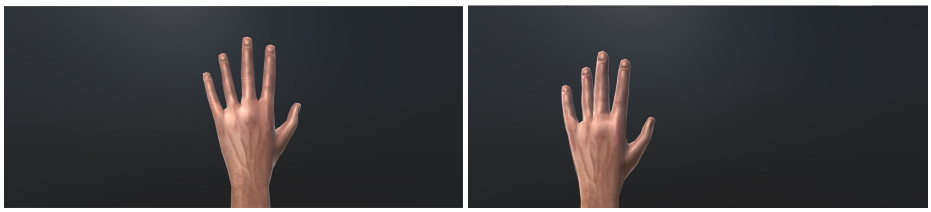


Figure 31: Hand in neutral position and hand translated according to PCA

at the beginning of the recognition process.

2. Translate hand to neutral position

The Image Processing Component sends the position of the hand, which was computed via PCA. These are X and Y coordinates to which hand is translated. The hand is shown in neutral position in Fig. 31 on the left. The hand, translated according to PCA mean position coordinates, is in Fig. 31 on the right side.

3. Estimate hand translation on X axis iteratively to improve the initial setup, hand is additionally translated by an estimated distance, which is iteratively computed.

Iterative translation and rotation

Both for improving the initial position and rotation of the hand with the aim of reducing the error, the process of iterative estimation of the start parameters was calculated. The process for each finger is:

1. Calculate the initial error, which is the number of pixels that differ.
2. Define minimum and maximum parameters and the increase factor

For rotation:

1. Define the minimum angle ϕ_{min} and maximum rotation angle ϕ_{max}:
   For fingers these angles have values:
   $\phi_{min} = 0$
   $\phi_{max} = 100$

   For wrist:
   $\phi_{min} = -50$
   $\phi_{max} = 50$

Figure 32: Input image and result from iterative translation estimation.

2. Start with angle $\phi_1 = \phi_{min}$, then stepwise increase angle by a factor u:
   $u = 6.0$
   $\phi_i = \phi_{i-1} + u$

For translation:

1. Define min and max distance points for a given x coordinate:
   $x_{min} = x - diff_{max}$
   $x_{max} = x + diff_{max}$
   where $diff_{max} = 0.2$
2. Start with $x_1 = x_{min}$ and stepwise increase until $x_i = x_{max}$ by the factor u:
   $u = 0,012$
   $x_i = x_{i-1} + u$
3. In each step translate/rotate by new parameter and calculate error: e_i
4. When the maximum parameter(angle ϕ or x position) is achieved, stop and choose the parameter that yielded minimum error

For the input image from Fig. 32 on the left, the error for the initial position e_s was 0.317475. The iterative estimation yielded position shown in Fig. 32 on the right. The calculated error for this position was $e = 0.17158$.

Estimate wrist rotation by X axis

The optimum hand rotation angle is also computed iteratively. The result of translation is taken as input for estimating hand rotation by X axis. The initial error was $e_s = 0.37599$. The resulted best rotation with error $e = 0.3746$ is by angle $\phi = 26.0$, result in Fig. 33.

After this preparation, either the Levenberg-Marquardt algorithm is started

Figure 33: Result from iterative rotation

or the Iterative Heuristic, which again tries to iteratively rotate the finger joints to find the best rotation angles.

Levenberg-Marquardt vs. Iterative Heuristic

To compare the Levenberg-Marquardt and the Iterative Heuristic, a set of gestures has been defined and images of the gestures were taken. The test set consists of hands showing one, two, three and five fingers in different orientations. In total the set includes 25 images.

Each image has been loaded and run through the whole process, that has been described previously in this section. This includes first running the background subtraction, and afterwards, for each image separately the Levenberg-Marquardt and the Iterative Heuristic. The result of each algorithm was a rotated 3D hand model with an error value. Fig. 35 contains a scatter plot with error values of the Iterative Heuristic and the Levenberg-Marquardt approach for different gestures. (See appendix B for larger image). The Y axis is the error value, the X axis contains gesture labels. E.g. label `one_side1` describes a gesture containing one pointed finger like in the image in Fig. 34 in the lower right. Blue dots represent the error values of the Iterative Heuristic and red ones of the Levenberg-Marquardt. For the gestures where there is only one colour depicted, that means the values were same, so the other dot is occluded. To better understand the results, Fig. 36 displays the results of the recognition process for the following gestures:

- `five_rotated1`
- `three1`
- `two_rotated1`
- `five2`
- `two_side2`

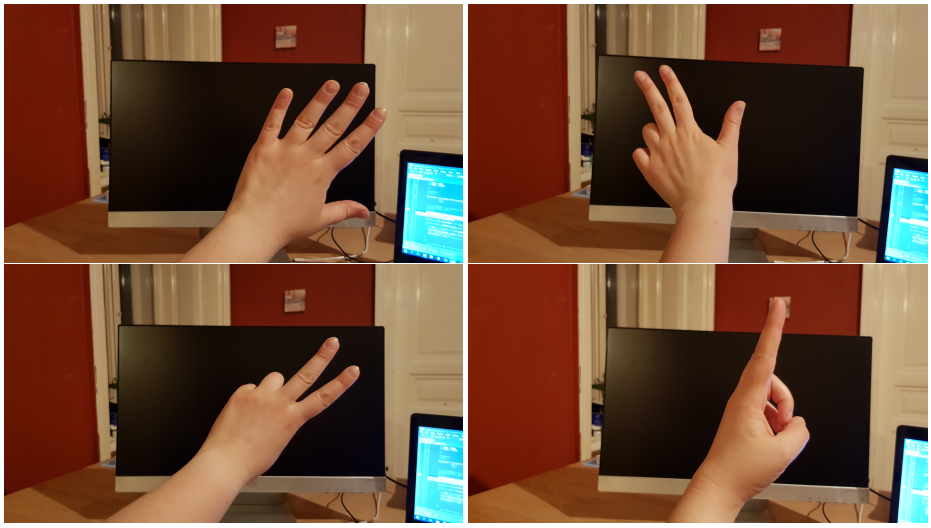


Figure 34: Gestures from test set

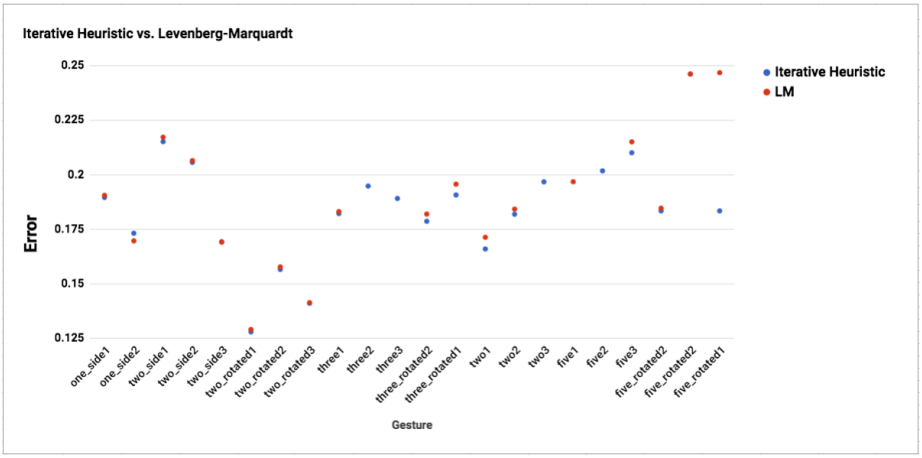
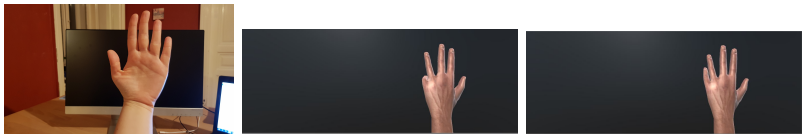
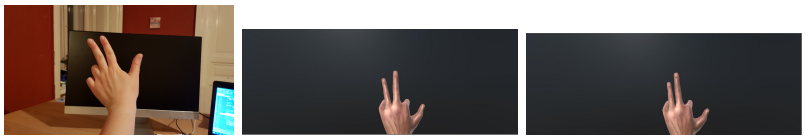


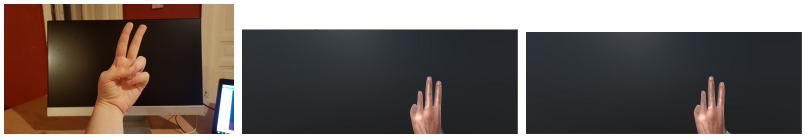
Figure 35: Resulting error values of Levenberg-Marquardt and Iterative Heuristic



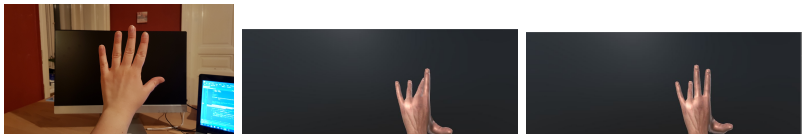
(a) `five_rotated1`



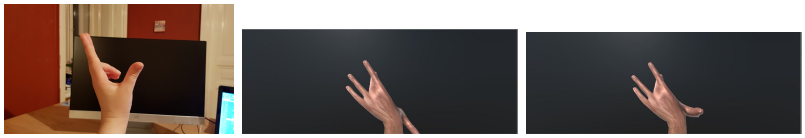
(b) `three1`



(c) `two_rotated1`



(d) `five2`



(e) `two_side2`

Figure 36: Resulting gestures from the Levenberg-Marquardt and the Iterative Heuristic

In Fig. 36 the first column consists of input gestures, the second one contains the resulting gesture of the Levenberg-Marquardt algorithm and the last column is the resulting Iterative Heuristic gesture. Under each gesture the label is given. The first gesture, labelled as `five_rotated1` is a hand with five fingers but rotated so that camera is capturing the palm. As the recognition process is not performing the rotation around the Y axis, the complete recognition would not be possible. In the second column, the result of the LM algorithm shows that the little finger is rotated around the X axis by an angle so large that it appears hidden. Interesting is that the error value that the LM algorithm yielded was 0.2467825, whereas the error from the IH is 0.183415. For a human eye it is easy to see that LM result is better, as the little finger is hidden, so it appears shorter and more like the thumb.

Gesture labelled as `three1` shows an almost perfect match where both algorithms recognized three fingers. However, instead of showing middle, index and thumb, both algorithms determined the left ring, middle and thumb in the original rotation, but rotated the index instead of ring finger.

Gesture `two_rotated1` shown in Fig. 36c yielded the smallest error value for both algorithms. The error value for LM was 0.1290525, for IH 0.127965. Considering that rotation over the Y axis was not a part of the process, the better match would be if one additional finger was rotated, so the resulting gestures would also show only two fingers.

The most basic gesture is the one labelled `five2` from the test set. Both algorithms rotated a thumb too much, but still the complete number of fingers in the modelled hand corresponds to the input gesture. The last gesture `two_side2` is a hand rotated over the Y axis showing two fingers. Due to the occlusion, it is hard to say whether the middle finger is bent or standing straight. Both algorithms produce pretty much the same result, where the thumb and middle fingers are rotated. Even though the number of fingers does not match, the rotation of the hand is pretty similar to the original one.

5.4 Conclusion

The aim of this work was to explore the possibilities of using only image processing techniques to recognize human hand gestures. Although this seems very natural to us humans, as we engage in this process everyday when communicating with each other, it is not that easy for a computer to handle this task. That is why current solutions in virtual reality mostly include different controllers, so that user can interact with the environment.

The first part of our solution, the background subtraction and principal compo-

nent analysis show that it is possible to extract the hands from the surroundings under some limitations.

The second part, the recognition itself included two approaches, the Levenber-Marquardt and the Iterative Heuristic. Comparing the results of them, one can conclude that the Iterative Heuristic yields very similar results to the Levenberg-Marquardt approach. The IH has an advantage of being a simple approach where the fingers are only rotated iteratively in the defined range and in the end the angle which produced the smallest error is chosen as the best one. As it is simple heuristic, it was also used for the initial setup of the location of the hand and the translation of the wrist. Compared to it, the advantage of LM is that it can stop processing anytime one of the conditions is met. Further, additional parameters, e.g. rotation around another axis could be included into the model. This would of course lead to an increase in computing time but could also lead to better results.

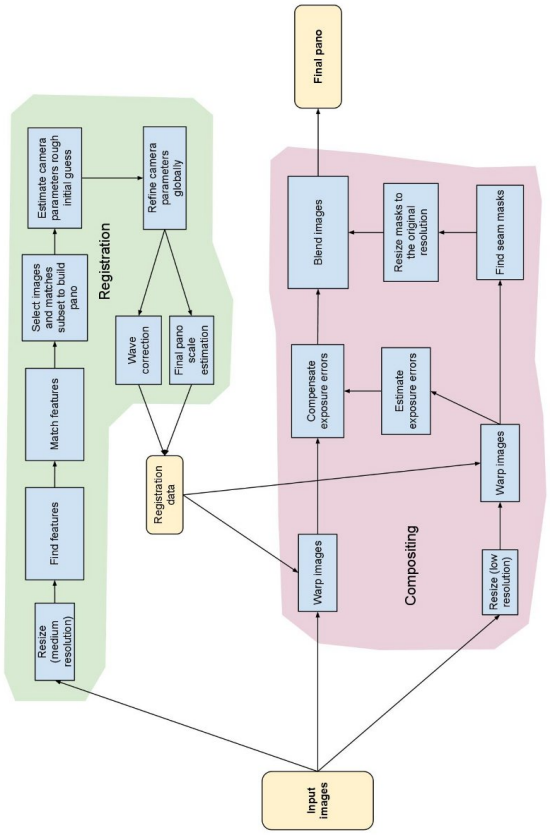
The results have shown that both algorithms, IH and LM, could recognize the gesture to some extent, depending on the gesture's complexity. The standard gestures that include only finger rotation would yield to a similar 3D hand model, as shown in the figures of the previous section. However, as the model used here includes only a rotation parameter for the X axis, different rotations of the hand itself, which produce an occlusion of the fingers, would not lead to satisfying results.

We have seen as well that the mathematical error which was computed does not necessarily correspond to that what a human eye sees. The error measure that was computed returns the number of pixels that differ, but no information about the location of the pixel.

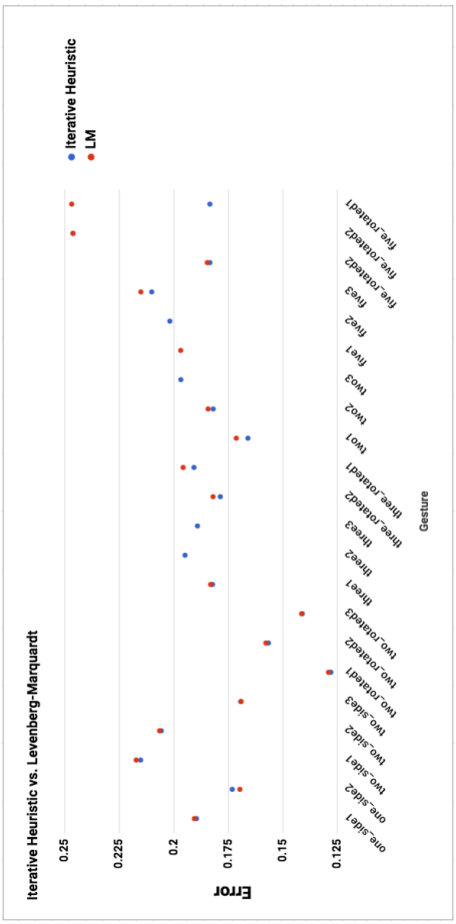
Even though the presented flow has limitations such as difficulties with illumination changes, camera orientation and static objects, it still has the advantages that it is not limited to some predefined model of images of gestures. It could be applied to anybody's hands as far as the surroundings correspond to the background subtraction's limitations. However, there are definitely different ways in which the system could be improved. One additional optimization could be to add some pre-defined gestures in order to speed up the process. Some very common gestures could be stored in some database and then the input could first be compared to them. One possibility here would be to use the FLANN algorithm that was described in previous sections.

The solution we designed, relies only on image processing techniques and thus has the advantage of being non-invasive to the user and not limiting his interactions. Another advantage as well is that the number of external gadgets is minimal, as only a camera is necessary, thus making it also inexpensive.

A OpenCV Image Stitching Pipeline



B Results of Levenberg-Marquardt and Iterative Heuristic



References

[1] Gaussian blur. https://en.wikipedia.org/wiki/Gaussian_blur.

[2] OpenCV eroding and dilating. http://docs.opencv.org/2.4/doc/tutorials/-imgproc/erosion_dilatation/erosion_dilatation.html.

[3] OpenCV filtering. http://docs.opencv.org/2.4/modules/imgproc/doc/-filtering.html.

[4] OpenCV FLANN. https://docs.opencv.org/2.4/modules/flann/doc/-flann_fast_approximate_nearest_neighbor_search.html.

[5] OpenCV Stitching Pipeline. https://docs.opencv.org/2.4/modules/stitching-/doc/introduction.html.

[6] OpenCV warpPerspective. http://docs.opencv.org/2.4/modules/imgproc/-doc/geometric_transformations.html.

[7] H. Baltzakis, M. Pateraki, and P. Trahanias. Visual tracking of hands, faces and facial features of multiple persons. 2012.

[8] H. Bey, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF).

[9] T. Bouwmans. Traditional and recent approaches in background modelling for foregrounded detection. 2014.

[10] P. Burt and E. Adelson. A multiresolution spline with application to image mosaics. 1983.

[11] J. Canny. A computational approach to edge detection. 1986.

[12] T. Chen, C.-S. Chen, C.-R. Huang, and Y.-P. Hung. Efficient hierarchical method for background subtraction. 2007.

[13] N.H. Dardas and E.M. Petriu. Hand gesture detection and recognition using principal component analysis. 2011.

[14] A. Demeulemeester, K. Kilpi, S.A. Elprama, S. Lievens, C.F. Hollemeersch, A. Jacobs, P. Lambert, and R. Van de Walle. The icocoon virtual meeting room: A virtual environment as a support tool for multipoint teleconference systems.

[15] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. 1981.

[16] A. Gustus, G. Stillfried, H J. Vasser, and P. Van der Smag Jörntell. Human hand modelling: kinematics, dynamics, applications.

[17] C. Harris and M. Stephens. A combined corner and edge detector.

[18] J.F. Kennedy, J.Kennedy, and R.C. Eberhart. *Swarm intelligence*. Morgan Kaufmann Publisher, 2001.

[19] J. Lin, Y. Wu, and T. Huang. Modeling the constraints of human hand motion.

[20] M.I.A. Lourakis. A brief description of the levenberg-marquardt algorithm implemened by levmar. 2005.

[21] D. Lowe. Distinctive image features from scale-invariant keypoints. 2004.

[22] Z. Ma and E. Wu. Real-time and robust hand tracking with a single depth camera. 2013.

[23] M.Brown and D.Lowe. Automatic panoramic image sttiching using invariant features.

[24] M. Muja and D. G. Lowee. Fast approximate nearest neighbours with automatic algorithm configuration.

[25] A. Shimada, D. Arita, and R.I. Taniguchi. Dynamic control of adaptive mixture-of-gaussians background model. 2006.

[26] A. Sobra and A. Vacavant. A comprehensive review of background subtraction algorithms evaluated with senthetic and real videos. 2013.

[27] C. Staufter and W.E.L. Grimson. Adaptive background mixture models for real-time tracking.

[28] S. Suzuki and K. Abe. Topological structural analysis of digitized bianry images by border following. 1983.

[29] H.S. Yeo, B.G. Lee, and H. Lim. Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware. 2013.

[30] Y.Xu, J.Dong, and D.Xu B.Zhang. ackground modelling methods in video analysis: A review and comparative evaluation. 2016.

[31] Z. Zivkovic. Improved adaptive gaussian mixture model for background subtraction. 2014.

[32] Z. Zivkovic and F. van der Heijden. Efficient adaptive estimation per image pixel for the task of background subtraction. 2006.