REAL-TIME HAND TRACKING USING STANDARD COMPUTER HARDWARE

M.R. FREMOUW

A thesis submitted for the degree of Master of Science in Computing Science

Faculty of mathematics and natural sciences
University of Groningen

November 2009

M.R. Fremouw (maarten@fremouw.nl): *Real-time hand tracking using standard computer hardware,* A thesis submitted for the degree of Master of Science in Computing Science, © November 2009.

SUPERVISOR:
Dr. M.H.F. Wilkinson

SECOND READER:
Prof. Dr. M. Aiello

VERSION:
1.0 (Final)

CREATION DATE:
November 26, 2009 at 22:32

ABSTRACT

In this masters thesis' research is done in the field of hand tracking. The focus lies on designing and implementing a real-time (multiple) hand tracker using Commercial off-the-shelf hardware. The basic idea is to extract a humans' hand from an ordinary webcam image. Several methods are discussed of which some are also implemented in the C programming language and evaluated. This is achieved by combining background subtraction, skin segmentation and connected component labeling. Background subtraction is used to extract only the moving areas of an image, skin segmentation identifies which areas are skin and finally connected component labeling is used to extract the hand boundaries. A frame rate of around 20 frames per second is achieved.

For each component several different algorithms are first evaluated before a decision is made on which to use. For background subtraction, Frame Difference, Approximate Median Filtering and Mixture of Gaussians are evaluated. All evaluated skin segmentation algorithms, i.e., the Intersection approach, Bayesian approach and Artificial Neural Network approach are evaluated thoroughly. Using connected component labeling, a way to detect area's in binary images, the hands are extracted from the image.

This masters thesis is part of a larger project called: "Augmented Reality for Multiuser 3D Interaction" or ARMI. The idea behind ARMI is to create an AR application for interacting with virtual objects in a multiple user environment. Interaction is done without the help of a keyboard and mouse, only a person's bare hands are used for input. Therefore, hand tracking is required. The tracked hands are the base for the next part of the project: hand pose estimation. The result of this effort is a prototype hand tracker application. The main parts of the hand tracker are written in the C language, but ARMI is implemented in Python. Therefore, a Python module is designed and implemented which exposes the most important C functions to Python.

In the final prototype of the hand tracker, Approximate Median Filtering is used for background subtraction and the Bayesian approach for skin segmentation. For the hand tracker Approximate Median Filtering is the right balance between speed and complexity in comparison to the other two. Skin segmentation is evaluated more thoroughly and resulted in a skin detection rate, this rate is calculated using the sum of correctly classified skin and background pixels divided by the sum of total classified skin and background pixels, the Bayesian approach scored best with a skin detection rate of 0.87. Also, the Bayesian approach scored 83.3% of correctly classified skin pixels and 9,4% false positives, i.e., pixels falsely classified as skin.

ACKNOWLEDGMENTS

After a long period of hard labor things finally come together; this masters thesis is finished. There are a few people I would like to express my gratitude to.

First of all, I would like to thank my supervisor from the University of Groningen, Dr. Michael Wilkinson, for supervising me, guiding me and providing me with useful advice for implementing the prototype and writing this thesis during the projects lifespan.

I also would like to show my gratitude to Prof. Dr. Marco Aiello and Dr. Tobias Isenberg for guidance in the start-up phase of this masters thesis project.

Finally, I would like to thank Gijs Boer for coming up with such a great idea for a masters thesis project.

— Maarten Fremouw
Groningen, October 27, 2009

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

1 INTRODUCTION

The general area of this master thesis is hand tracking, focused on Augmented Reality (AR). The general idea of AR is to blend virtual objects and the real environment the user is in seamlessly together. Figure 1 shows an AR example of a television broadcast of an American football game, to help the viewer a yellow line is drawn on top of the field. Using AR brings new problems in the domain of input devices.

Figure 1: AR example, screen capture of a broadcasted American Football game, the yellow first down line (center of image) is computer generated (source: HowStuffWorks [25]).

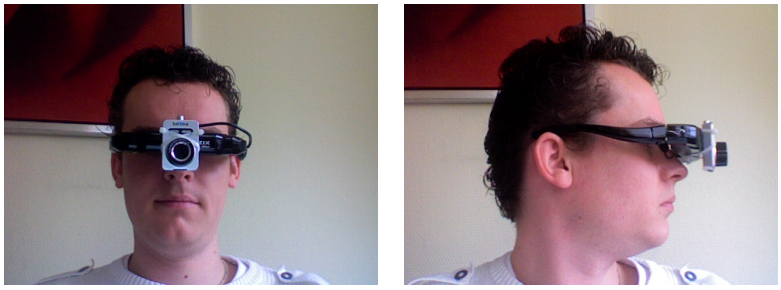
Imagine if you can place anything on top of the real world, but you still have to use your keyboard and mouse combination, this is not very intuitive. Without a clever and intuitive way of interacting with the computer AR makes no sense, it thus is very important to research new intuitive ways for interacting using your hands. Of course there are other possibilities besides the tracking of hands, but people are used to doing things by using their hands; it is very intuitive.

1.1 ARMI

This master thesis is a part of larger project which is called "Augmented Reality for Multiuser 3D Interaction," from now on referred to as ARMI. The idea behind ARMI is to create an AR application for interacting with virtual objects in a multiple user environment. Interaction is done without the help of a keyboard and mouse, only a person's bare hands are used for input. Also, it is a multiple user application, as in an environment can be shared between multiple locations. An example is an architect which designs a building and shares it with a customer. They both see the same virtual building and the architect can show or pinpoint specific design issues by using only his hands. The customer

and architect do not need to travel to each other in order to show and explain the design. Of course it seems easier to just send some pictures or videos of different parts of the building, but with ARMI there is interaction between the architect and the customer; they can both "touch" and rotate the building. Also, they see the building blended with the real world which gives the customer a difference experience as with static pictures or video.

To make all this possible additional hardware is needed besides an ordinary PC. The user of ARMI needs to wear a Head-Mounted-Display (HMD). For this project the Vuzix iWear VR920 [60] is used. The display used for the Vuzix HMD is not a see-through type; HMDs with see-through displays are at time of writing not yet affordable [19]. Therefore a webcam is attached to the front of the HMD to simulate see-through capability. See Figure 2a and Figure 2b for a picture of the HMD with the webcam attached. The webcam used is a Philips SPC1000NC [44]; which is a simple mass market consumer webcam. The camera is capable of delivering frames at a resolution of 640×480 pixels with 30 frames per second.



(a) HMD - front. (b) HMD - right.

Figure 2: The camera and HMD used for ARMI (source: P. Bruining [10]).

There is already much research in the area of AR. ARMI uses ARToolKit [27] to overlay the virtual objects on the real world. ARToolKit does this by recognizing tags from a video feed, any tag can be used; ARToolKit can learn custom tags, this needs to be done beforehand. It is, for example, possible to use the University of Groningen logo as tag and project virtual objects on that tag. See Figure 3a for an example tag and Figure 3b for an example overlay object.

(a) The tag used for tag-recognition. (b) A virtual teapot is displayed on top of the tag.

Figure 3: An example of AR using ARToolKit to detect the tag and display a virtual object (source: H. Lenting [37]).

ARToolKit is able to calculate the angle of the tag from within the image, this makes it possible to walk around it while wearing the HMD. ARMI uses multiple tags to give the user more freedom in movement. Several tags will be placed on a fixed position on a table; multiple tables can be used which share the same virtual objects.

A user is able to move virtual objects using bare hands. To achieve this ARMI uses two webcams which are mounted above a table and are aimed at the tables surface. The feeds from these cameras are used for hand tracking and hand pose estimation. The webcams used are two Logitech QuickCam S 7500's [38], these capture at a resolution of 640×480 and 15 frames per second.

1.2 CONTEXT

ARMI is a rather large project, therefore it is split up into four smaller parts. Every part will be researched and implemented by a master student of the University of Groningen. All group members are Computing Science master students, who are all following the master variant "Software and Systems Engineering." The members (in arbitrary order) are: Gijs Boer, Pieter Bruining, Heino Lenting and, the author of this document, Maarten Fremouw. The four parts are defined as follows:

- Hand tracking, multiple hands are tracked from a video feed using an ordinary webcam.
- Hand pose estimation, the hands tracked are refined more in order to determine an angle and position of the hand (Gijs).
- Concurrent three-dimensional interface, using hands as input device require new ways of interfacing with the user. Also multiple users can work in the same AR environment (Pieter).
- AR object replication of all data and actions to all tables connected to the same AR environment (Heino).

After the hand tracking component, which is covered in this thesis, pose estimation needs to be done. Detailed information on pose estimation can be found in research done by Boer [7]. Hand pose estimation is needed for the three-dimensional concurrent interface. This interface handles the interaction between multiple users of the same environment; more details about this research field is given in the masters thesis by Bruining [10]. Lastly, Lenting [37] has done research in the area of AR object replication. The focus of his research is to keep the replicated environment synchronized and consistent.

See Figure 4 for a graphical (simplified) representation of how all parts are connected to each other and how the data roughly flows through ARMI. In this figure the hatched component is the focus of this thesis. Hands are tracked and coordinates and size of detected hands are sent to the next component: hand pose estimation. An estimate of the pose of the hand is done along with a estimate of the depth. The pose and depth are then sent to the three-dimensional interface. The interface uses the hand pose for interaction with the AR environment. In order

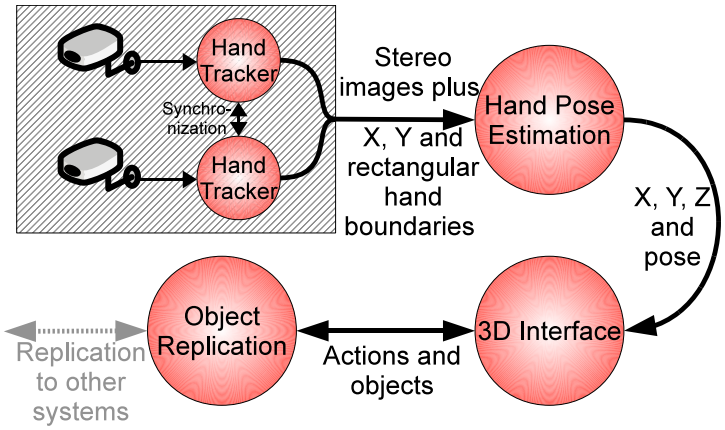


Figure 4: Global system overview of ARMI.

to keep the AR environment synchronized with other environments the data is replicated in the replication part.

1.3 NOVELTY

The field of hand tracking is not new, the novelty is using hand tracking in combination with AR and without using gloves or any other markers; thus tracking bare human hands. Also, current research is mostly focused on the tracking of a hand only, while it is also important to handle dynamic backgrounds in a cluttered environment. For this thesis research is focused on delivering a practical usable solution; without any markers and by using only Commercial off-the-shelf (COTS) components.

1.4 PROBLEM STATEMENT

The hand tracker must be able to track multiple hands by using only COTS components. Also, the system should not require any additional markers, like gloves or stickers, on a users' hand; the tracker must work on bare skin.

The system must be user-friendly and applicable in the real world, ideally without calibration or fine tuning for specific skin. Applicable in the real world is defined as the system should work in environments with more complex backgrounds than a simple concrete wall, for example an office environment.

Furthermore, the setup consists of two cameras which are synchronized for optimal image retrieval and are capable of independently tracking one or more hands. Synchronization of the two cameras is required for the estimating the depth of a hand. In order to increase the accuracy of the depth estimation the two images should differ time wise as little as possible. Depth estimation is part of hand pose estimation by Boer [7] and not the focus of this thesis.

1.5 GLOBAL THESIS OVERVIEW

This paragraph describes the global overview of this thesis. Each chapter after the current chapter will briefly be explained with a short summary of its contents.

In Chapter 2 the current research related to hand tracking is discussed. Some of the previous work done in the field of hand tracking is used for the final hand tracker implementation. Performance both in computational time and detection rate is also discussed in this chapter.

The next chapter, Chapter 3, discusses the theory behind the hand tracker. Several different algorithms are implemented and tested, some of the algorithms are dropped eventually in favor of better performing ones.

Several algorithms are tested prior to the implementation of the final hand tracker. Why specific algorithms are dropped and others survived will be explained in Chapter 4.

The most important implementation details or other details worth mentioning will be explained in Chapter 5. An architectural overview is given. Also, the hand tracker is split up in several frameworks which will be described in detail. Lastly, a few optimizations are done in order to increase the tracking speed, these are described in pseudo code.

In Chapter 6 a conclusion is drawn based on the test results, also a discussion of the predefined goals and requirements and if these are met is given. Lastly, a brief explanation of possible future work to improve the tracker's performance is also done in Chapter 6.

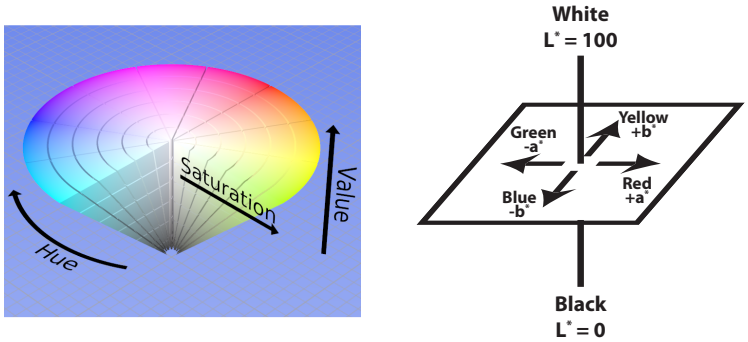
2

STATE OF THE ART

In this chapter an overview of research related to hand tracking and previous work in this field is given. This research can be used as basis for further research towards hand tracking for the ARMI project, research is split up in subfields which each have their own paragraph.

2.1 HAND TRACKING

In the field of hand tracking several techniques are proposed. One relatively straightforward technique is to use histograms to determine the skin color distribution. A histogram in this context shows the frequency of all colors, each color falls in a category. These categories are ranges of colors and are called bins. Most algorithms convert red, green and blue (RGB) values to another color space to minimize the influence of changing light conditions and skin tone diversity. Mostly two-dimensional histograms are used. Commonly used color spaces are hue, saturation and value or intensity (HSV/I) and $L^*a^*b^*$ [16]. HSV/I describe colors as points in a cone, hue represents the pure color, without shades or tints, and ranges from 0 to 360°. Saturation refers to the purity of a color, ranged from 0 to 100%. The third component is value, which also ranges from 0 to 100%, but this component represents the brightness of the color. See Figure 5a for a representation of the HSV color space in a cone. $L^*a^*b^*$ is represented in a cube shape, see Figure 5b. $L*$ ranges from 0 to 100, where the maximum value of 100



(a) HSV color model represented as a cone [24].

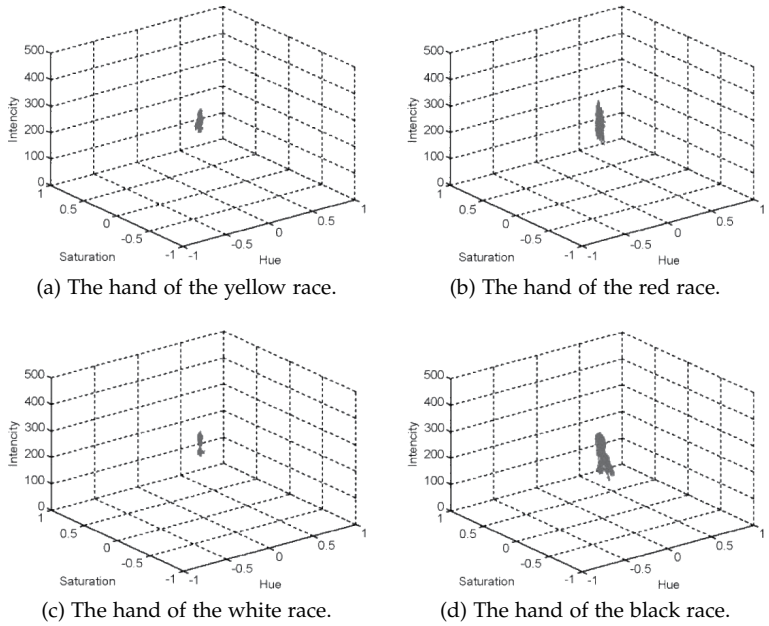
(b) Representation of $L^*a^*b^*$ color space [26].

Figure 5: Two common used color spaces in computer vision.

represents white and 0 black. A positive value of $a*$ represents red, a negative value represents green. For the $b*$ component a positive value represents yellow and a negative value represents blue. Note that the

a* and b* components have no specific maximum values; this depends on the color space used to convert to L*a*b*.

Yin, Guo and Xie [61] have done research in skin color distribution for different ethnic groups of people, when comparing the differences between race, as shown in Figure 6, it is clear that the big difference lies in the Intensity. The other values are grouped quite closely. HSV/I and L*a*b* and in particular the HS and L*a* parts, are better suited for the skin color segmentation [61].



(a) The hand of the yellow race.

(b) The hand of the red race.

(c) The hand of the white race.

(d) The hand of the black race.

Figure 6: Plot of HSI space for different types of human hands (source: Yin, Guo and Xie [61]).

A slightly more advanced technique uses a Bayesian classifier [4, 31, 58], therefore two two-dimensional histograms are used: one for skin color and one for the background color distribution. The histograms represent the probability if a pixel is skin or part of the background [1, 31, 58]. These algorithms are focused on finding an object in a single image and not specifically for sequence of images. An important advantage is that most of the computation is done at training time. After training only a simple look-up in an array is required to get the probability; this is a clear advantage over other algorithms

A comparison is made between histograms and a Mixture of Gaussian (MoG) model, see Figure 7. Even though the histograms are computationally less complex it outperforms the MoG model. This is because the MoG imposes a distribution model on the data while the histograms are distribution free.

Stenger, Mendonça and Cipolla [52] describe a technique for model-based hand tracking using an Unscented Kalman Filter (UKF). The UKF is an extension of the traditional Kalman Filter (KF) by Kalman [33]. The difference is that the KF works on linear systems while UKF is able to handle non-linear systems. Both try to estimate the state of a system from noisy data. A three-dimensional model of a human hand

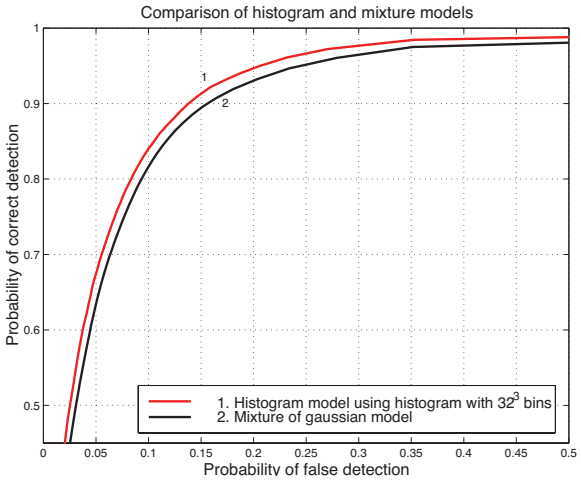
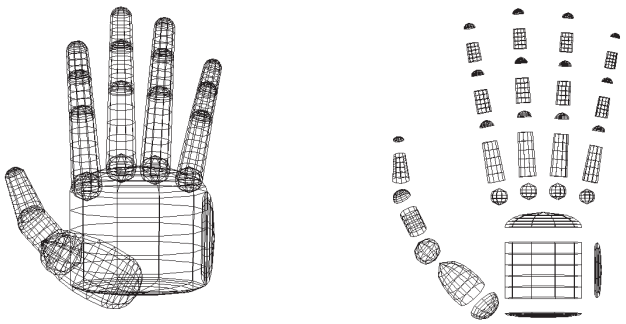


Figure 7: ROC curve comparing histograms with Mixture of Gaussian (source: Jones and Rehg [31]).

is used for generating the contours of a hand and are then compared to the input image; the hand is created using quadrics. The hand model used is a hand model with 27 degrees of freedom (DOF). Six are used for the global hand position, four for the finger poses and five for the thumb pose, see Figure 8 for the three-dimensional hand model. The object tracking is formulated as a nonlinear estimation problem, the model-based hand tracker uses an UKF. UKF is used for estimation of the models' motion parameters.



(a) Front view. (b) Exploded view.

Figure 8: A 27 DOF hand model from 37 truncated quadrics as used in "Model-Based Hand Tracking Using an Unscented Kalman Filter" (source: Stenger, Mendonça and Cipolla [51]).

There are a few advantages over simpler skin classification methods, such as the Bayesian histogram classifier. Using a three-dimensional model it is possible to actually distinguish hands from other body parts, also the model is capable of tracking an occluded hand. However, keeping the goal of this thesis in mind, a simple skin color distribution technique is possibly sufficient; the camera setup is static and focused only on the hands. The disadvantage is that the method requires a lot more computational power in comparison with the simple skin classifiers. Stenger, Mendonça and Cipolla conducted experiments with grayscale images of 360×288 pixels and achieved a frame rate of

three frames per second on an old 433 MHz Intel Celeron. Using more recent hardware this frame rate obviously will increase, but it is an indication of the computational requirements. This method seems more suitable as possible complete approach for both hand tracking and hand pose estimation, but only if a frame rate of 15 frames per second at a resolution of 640×480 pixels is achieved. In Figure 9 a flow chart of the tracking system is given.

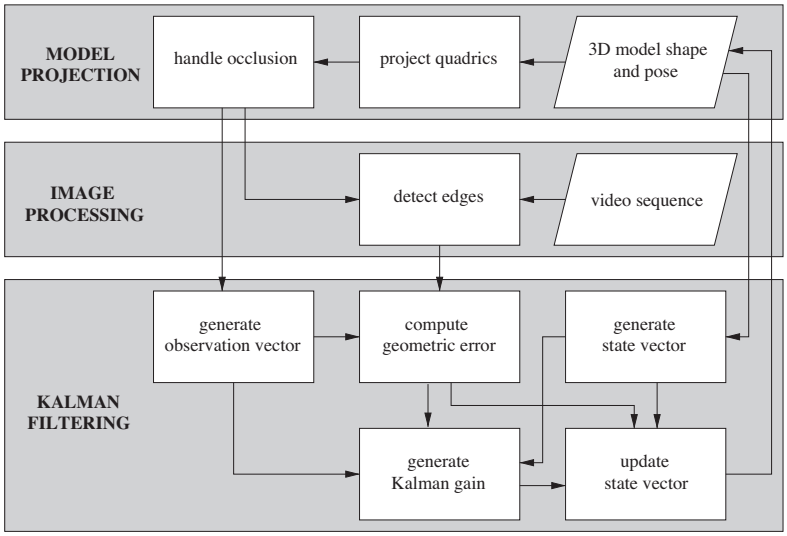


Figure 9: Flowchart of the model based hand tracking system (source: Stenger, Mendonça and Cipolla [52]).

A more sophisticated technique, which does focus on an image sequence, is based on Particle Filtering (PF), also known as sequential Monte Carlo Filtering (MCF). This technique uses a probability distribution over the state of the system. For image processing the state is for example: the location and scale. Some papers use a combination of particle filtering and another algorithm. The idea is to simulate a Bayesian filter by MCF simulations. Estimates are computed based on a set of random samples with associated weights, when the number of samples increase the PF approaches the optimal Bayesian estimate. A common problem with PF is the *degeneracy problem* [3], after a few iterations all but one particle will have a negligible weight which causes that the likelihood and thus the contribution of these particles are also negligible.

The Condensation algorithm [28, 29, 36] is also a form of PF. This algorithm is focused on tracking the outlines and features of foreground objects; it is not specifically created for tracking human hand outlines. The algorithm is a combination of the factored sampling algorithm for static problems with a stochastic model for object motion. A Kalman Filter cannot represent alternative hypotheses simultaneously in contrast to the Condensation algorithm which is able to do this because it uses dynamic models. A disadvantage is that is has problems with highly cluttered environments.

Bray, Koller-Meier and van Gool [8] introduce an algorithm called Smart Particle Filtering (SPF) for three-dimensional hand tracking, this algorithm uses Stochastic Meta-Descent (SMD) in combination with a

PF. SMD is based on gradient descent with adaptive and parameter specific step sizes. Gradient descent is an optimization algorithm used for finding the optimal local minimum of a function. SMD does not guarantee in finding the global optimum, therefore Bray, Koller-Meier and van Gool combined this with a PF. In contrast to the Condensation algorithm SPF is able to handle highly articulated objects with clutter and occlusion robustly.

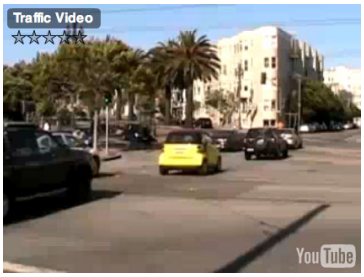
2.2 BACKGROUND SUBTRACTION

To improve hand tracking it is possible to combine hand tracking with background subtraction as preprocessing step. Background subtraction in this context is defined as segmenting out areas of interest, which in this case are the moving areas in a scene; given the assumption that the majority of the moving areas are human hands. As mentioned in Chapter 1 the camera setup is static, so the background does not change greatly over time. Therefore the goal of this step is to remove most of the background pixels in order to improve the skin segmentation, both in accuracy but also in computational time; only the possible non background pixels have to be classified by the hand tracker. Several background subtraction techniques have been researched. One of the simplest techniques is Frame Difference (FD). FD simply subtracts the current from the previous frame, often the frames are first converted to black and white. The result is the difference in pixel values; if the value of the pixel is greater than a certain threshold it is part of the foreground [20].

Another technique is Median Filtering (MF), MF calculates the background over a set of N previous (stored) frames; this consumes a lot of memory. Like FD the current frame is subtracted from the calculated background frame. Approximate Median Filtering (AMF) is a compromise of MF, it uses only one background frame which is updated slowly; changes in pixel values are propagated more slowly than with FD. The difference with MF is that only one background frame is updated. If a pixel of the current frame has a value which is greater than the corresponding background pixel, the background pixel is increased by one, otherwise the background pixel value is decreased by one [39].

Mixture of Gaussians (MoG) can also be used for background subtraction, an approach first developed by Friedman [18]. Unlike the other background models, which are simply a set of values the same size as the image size, MoG uses a parametric model. For every pixel location MoG maintains a density function. The advantage of this approach is that it can handle multiple background distributions. For example, a waving tree will eventually become part of the background; the other algorithms have more problems with this kind of noise [13, 50].

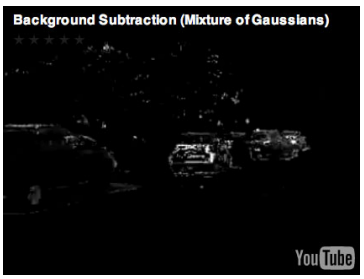
In Figure 10 four screenshots taken from a traffic video are shown, the screenshots of the different background subtraction methods are taken on the exact same time. Figure 10a is the original image. Clearly, as can be seen in Figure 10b, FD performs best with background noise as waving trees, but a slowly moving hand will also disappear quickly.



(a) Orignal.

(b) Frame difference.

(c) Approximate Median Filtering.



(d) Mixture of Gaussians.

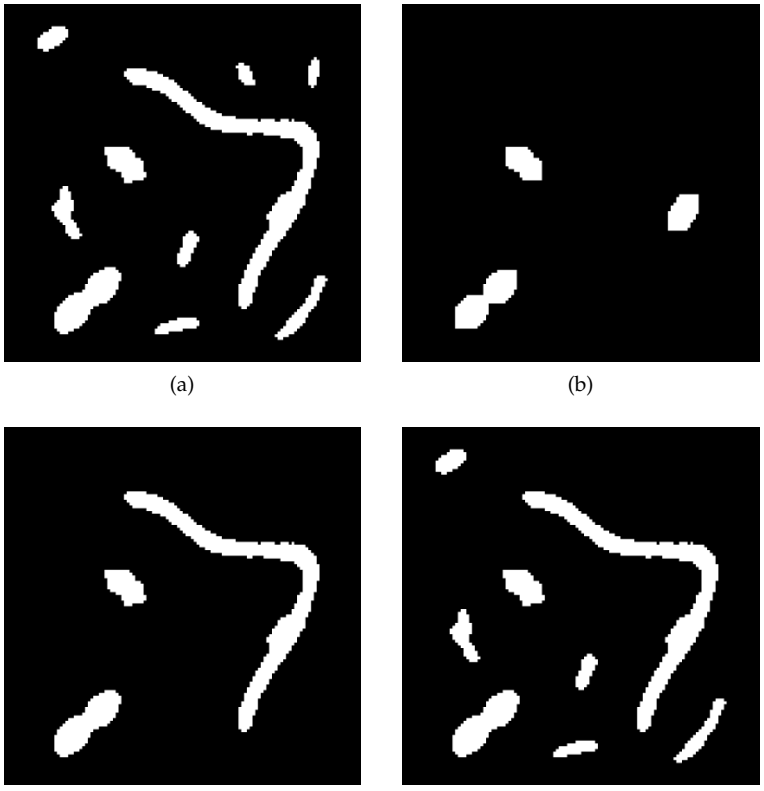
Figure 10: Different background subtraction methods (source: S. Benton [5]).

The MoG method (Figure 10d) handles the background noise quite well but is also the most computationally complex method. The AMF, see Figure 10c, method performs worse than MoG but it requires only little extra computation over FD [5] and others.

2.3 IMAGE FILTERS

Using background subtraction and some type of pixel classifier will likely still leave some small errors in the image. Think of small areas within the image which are falsely classified as skin or vice versa. With connected set openings and closings, such as the area opening method as proposed by Cheng and Venetsanopoulos [12], details from the image can either be completely removed or leave intact. This potentially yields in increased performance as small wrongly classified areas can be removed from the image. Figure 11 shows an example of an image filtered using area openings (Figure 11d), structural openings (Figure 11b) and opening by reconstruction (Figure 11c).

Before continuing with area openings, first a few words on connected set operators. Connected set operators operate on the flat zones of images, flat zones are the largest connected components of constant gray level. A partition of M is any set of disjoint sets $\{S_i\}$ which form the entire set M. Next, γ is a connected set operator if for all sets α_i there exists a β_j such that $\alpha_i \subseteq \beta_j$. Now $\{\alpha_i\}$ is the partition of domain M of image I which is formed by flat zones α_i and $\{\beta_i\}$ is the partition of M created by the flat zones β_j of image $\gamma(I)$. The binary area opening



(a) (b)

(c) (d)

Figure 11: In (a) the original image is shown, (b) uses structural opening with a 7×7 structuring element, (c) is an example of opening by reconstruction (also with a 7×7 structuring element) and (d) uses area openings with $\lambda = 49$. Note that areas in (b) and (c) are changed while (d) leaves them intact. (source: Meijster and Wilkinson [41]).

is based on binary connected openings. The binary area opening Γ^a_λ is defined as follows [41],

$$\Gamma^a_\lambda(X) = \{x \in X | A(\Gamma_x(X)) \geqslant \lambda\} \tag{2.1}$$

where $X \subseteq M$ is a binary image with the domain M and $\lambda \geqslant 0$, λ is a scale parameter. The connected opening $\Gamma_x(X)$ of X at point $x \in M$ results in the connected component of X containing x if $x \in X$ and $\emptyset$ otherwise. The binary area closings are defined as follows [41],

$$\Phi^a_\lambda(X) = [\Gamma^a_\lambda(X^c)]^c \tag{2.2}$$

where X^c is the complement of X in M. Meijster and Wilkinson [41] compare a method based on the Union-Find method with the Pixel-Queue algorithm [9, 59] and the Max-Tree approach [43]. The Union-Find method as proposed is based on the Union-Find algorithm by Tarjan [54]. In contrast to the other two algorithms, this algorithm is able to process multiple peak components simultaneously. A peak component P_h is a connected component of the thresholded image $T_h(f)$, which is defined for binary images as follows,

$$T_h(f) = \{x \in M | f(x) \geqslant h\} \tag{2.3}$$

where f is a grayscale image thresholded at h. Meijster and Wilkinson use the following basic operations originally defined by Tarjan [54],

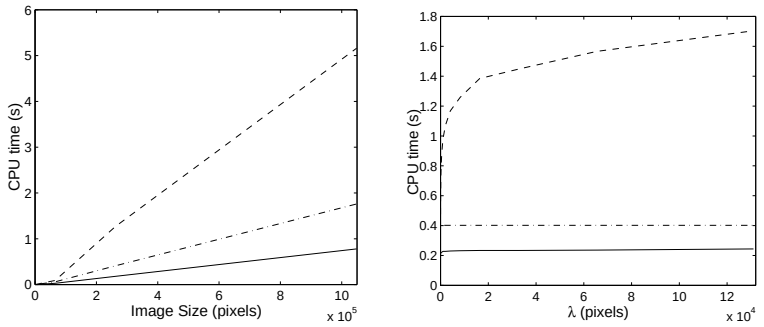
- `MakeSet(p)`, create a new singleton set `p`.
- `FindRoot(n)`, return the root of the tree which `n` belongs to.
- `Union(n,p)`, merge the two sets which contain `n` and `p`.
- `Criterion(r,p)`, determine if `r` and `p` belong to the same set.

The `Union(n,p)` operation uses `FindRoot(n)` to determine the root nodes of the trees containing `n` and `p` and the root nodes are used with `Criterion(r,p)` to determine if they belong to the same set. First a one-dimensional array `parent` equal to the size of the image `I` is created, where `parent[p]` is the parent of pixel `p`. Pixels in this one-dimensional array can be accessed using, $width \times y + x$ where x and y represent the current coordinates. For each processed pixel `p` the function `MakeSet` labels `p` as a singleton set by setting `parent[p]` to -1, a value < 0 indicates that a pixel is the root of a tree and thus has no parent; -1 is used as initial value. Instead of using a separate array `area` to store the area A of each set, `parent[p]` is set to $-A$; this saves memory. Next for each neighbor `n` which already is processed the `Union` function is called. Basically this function first uses `FindRoot` to find the root `r` of `n`, if `r` is not equals to `p` the `Criterion(r,p)` function is called. If this function determines `r` and `p` belong to the same set it returns `TRUE` and the two trees are merged. `TRUE` is returned if `r` is an active root or the gray level `I[r]` equals that of `p`. By making `p` the parent of `r` and adding the area of `r` to that of `p` the trees are merged. If `Criterion` returns `FALSE` the tree already has an large enough area, `p` is made inactive by setting it to $-\lambda$. Finally the area openings can be performed. Again each pixel `p` is processed. For all `p`'s the parent is investigated, if `parent[p]` is negative `p` is the root. `I[p]` is the corresponding gray level for the component. If `parent[p]` is zero or positive it already has set the correct root gray level. In Listing 1 is shown how to perform an area opening using the basic operations: `MakeSet(p)`, `FindRoot(n)`, `Union(n,p)` and `Criterion(r,p)`.

```
1  // S is an sorted array containing all pixels.
2  for(p = 0; p < Length(S); p++):
3      pix = S[p]
4      MakeSet(pix)
5      for all neightbors nb of pix do:
6          if( (I[pix] <  I[nb]) ||\
7              ((I[pix] == I[nb]) && (nb < pix))):
8              Union(nb, pix)
9  // Resolving phase in reverse sort order.
10 for(p = Length(S) - 1; p >= 0; p--):
11     pix = S[p]
12     if(parent[pix] >= 0):
13         parent[pix] = parent[parent[pix]]
14     else:
15         parent[pix] = I[pix]
```

Listing 1: Example of how an area open is performed using the four previously defined operations (source: Meijster and Wilkinson [41]).

Figure 12 shows an overview of how the Union-Find algorithm performs. Figure 12a and Figure 12b show that the Priority-Queue algorithm is strongly dependent on CPU time with increasing area size and image size. Both the Max-Tree and Union-Find method are clearly less dependent on image size and area size. A detailed description of the Pixel-Queue algorithm is given by Breen and Jones [9] and a detailed description of the Max-Three approach is given in a paper by Salembier, Oliveras and Garrido [43].



(a) Computing time of all three algorithms as a function of the image size.

(b) Computing time of all three algorithms as a function of the area.

Figure 12: Computational time over area size and image size. The Priority-Queue is represented by the dashed line, the Max-Tree method by the dashed/dotted line and Union-Find algorithm is the solid line (source: Meijster and Wilkinson [41]).

2.4 CONNECTED COMPONENT LABELING

Connected component labeling is used to connect regions in a binary image. A connected component is defined as a subgraph in an undirected graph. Components are connected if two ore more vertices are connected by its paths. Figure 13 shows an example of a graph containing two connected components. A common used method to determine

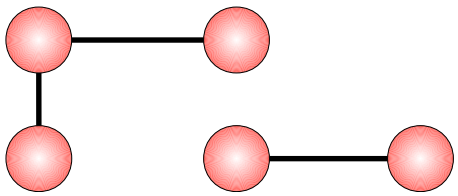
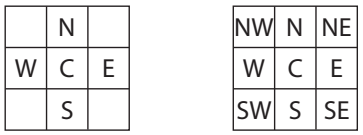


Figure 13: Example graph containg two connected components.

connection of nodes is by using four-connectivity, eight-connectivity or m-connectivity [22]. With four-connectivity a components north, east, south and west neighbor are used for comparison. This can be extended to eight-connectivity by also including its north-east, south-east, south-west and north-west neighbors. Obviously this can be extended to include even more neighbors. In Figure 14 an example of four- and eight-connectivity is shown.



(a) Four-connectivity. (b) Eight-connectivity.

Figure 14: Example of four-connectivity (a) and eight-connectivity (b). C is the current component.

Nodes are given a unique identifier which can be used for further processing; think of for example: finding edges or the position within the image. Several algorithms are available. A classic and probably one of the most well known is the two-pass algorithm [47]; this algorithm processes an image row-by-row. The first pass is used to give all nodes a unique label and determine the equivalence, which is done by using four-connectivity. The second pass connects all equivalent labels with its lowest label identifier. A more detailed description of connected component labeling is given in Chapter 3.

There is a lot of research done in this area and optimization of this classic labeling algorithm. For example there is research done in parallelization of connected component labeling, of which several are based on the Union-Find [54] algorithm [21, 40]. Depending on the achieved frame rate of the hand tracker a single threaded implementation is possibly not sufficient. Also, multi-core CPUs are getting more and more common nowadays, therefore looking into a parallel implementation can be worth the extra effort.

3

HAND TRACKER

This chapter gives an overview of the hand trackers internals. How does the theory work and which methods and algorithms are used? First the requirements are defined followed by an overview of how the different parts of the hand tracker work together. For each part different algorithms are described and compared of which some of them will be implemented and evaluated, evaluation is done in Chapter 4.

3.1 REQUIREMENTS

In the preliminary phase of the project the requirements where set, for the hand tracker the following requirements must hold:

- The system must work using standard computer hardware; any ordinary COTS USB Video device Class (UVC) webcam must be sufficient.
- The hand tracker must be able to track multiple hands within one single frame.
- The system should be able to track hands within 15 frames per second.
- Hand tracking should not require calibration of any sort beforehand.

Clearly computational performance is an important aspect of the hand tracker. With the currently available Central Processing Units (CPU) these requirements should be feasible. A UVC webcam is a webcam which follows a standard defined by the Universal Serial Bus consortium, this will make the webcam less platform dependent. Most modern Operating Systems have support for UVC cameras. For example Linux [56] and Mac OS X [2] drivers are available by default.

3.2 OVERVIEW

The hand tracker requires several steps in order to track hands. While at implementation level some of these steps can be merged, for clarity the different components are shown separately. The complete overview starting from the raw image feed from the webcam ending with a segmented image and hand boundaries is shown in Figure 15. Note that the actual hand tracker uses RGBA, where the A represents the

alpha layer of an image; this does not have impact on the YCbCr conversions as the alpha layer is set to zero and used for labeling in a later stage. A detailed description of how each component exactly works is giving in the next paragraphs.

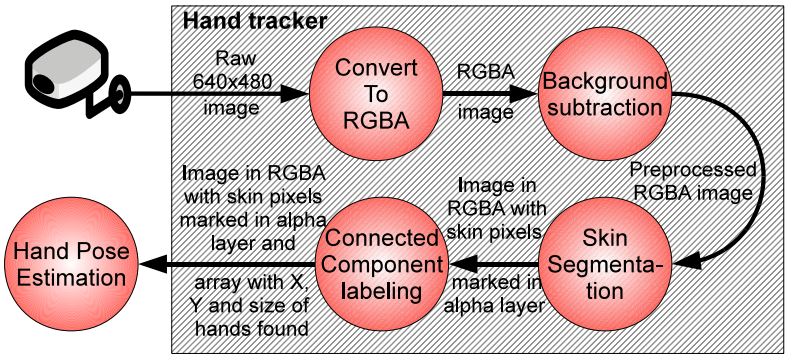


Figure 15: Hand tracker system overview.

The Logitech webcam [38] used for ARMI is not able to deliver images in RGB directly. Most low cost cameras do not support this and the Logitech webcam also does not support this feature. Instead it outputs images in YCbCr 4:2:2 format, Y is the luminance (or brightness) component and Cb and Cr are the blue difference and red difference chrominance components. Note that YCbCr is often confused with the YUV format, but YUV only applies to analog video in contrast to YCbCr which applies to digital video [55]. The following equations are used to convert YCbCr to RGB [30],

$$\begin{aligned} R &= 1.164(Y - 16) + 1.596(Cr - 128) \\ G &= 1.164(Y - 16) - 0.813(Cr - 128) - 0.391(Cb - 128) \\ B &= 1.164(Y - 16) + 2.018(Cb - 128) \end{aligned} \tag{3.1}$$

the R, G and B values are scaled in the range 0 to 255.

The YCbCr 4:2:2 format packs two image pixels into one macro-pixel, see Figure 16. Two luminance components, Y_0 and Y_1, share the same

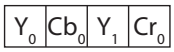


Figure 16: One macro-pixel.

set of chrominance components: Cb_0 and Cr_0. Thus, in contrast to RGB, half of the bandwidth is needed to transport the image from the webcam to the computer. The color space conversion component actually reserves an extra byte for the alpha layer, this layer is normally used for transparency but the hand tracker uses this layer to mark pixels as skin or background.

After conversion the image is fed to the background subtractor. In general the background of the captured image will not change very often; possibly only with changing light conditions. As mentioned earlier the webcam uses a static setup. Therefore using background subtraction a rough selection is made of which pixels require further examination. The background subtractor simply compares every pixel

with the corresponding pixel in the previously stored background image. If the difference is smaller than a certain threshold T it will be counted as background. If the pixel is counted as background the corresponding position in the alpha layer is set to zero, otherwise it is set to 255. The advantage of this approach is that the original image is preserved and only little extra memory is required.

When the background subtraction component is finished, the image is passed to the skin segmentation component. This component only compares pixels which are marked as non-background. Basically each non-background pixel is compared with with a predefined look-up table and when this difference is larger than a certain threshold T, the pixel is counted as skin. If a pixel is counted as skin it is marked in the alpha layer by setting its corresponding value to 255.

After the skin classification the final step is to find all area's marked skin within the image. Using connected component labeling the areas are detected and the X, Y, width and height of the areas are stored into an array. This array, together with the segmented RGBA image is passed to the hand pose estimator [7].

3.3 BACKGROUND SUBTRACTION

In total three different algorithms are discussed in this paragraph ranging from very straightforward to more complex algorithms; both in implementation and computational time. The algorithms explained in more detail are: Frame Difference (FD), Approximate Median Filtering (AMF) and Mixture of Gaussians (MoG).

All algorithms discussed in this paragraph use a single value for a pixel. In reality a pixel consist of more components, usually a red, green and blue component. Of course it is possible to to use all three color components but to save memory every frame is converted to grayscale before it is stored as background frame. For RGB to grayscale conversion the following equation is used [15],

$$Grayscale = 0.3 \times Red + 0.59 \times Green + 0.11 \times Blue \tag{3.2}$$

where $Grayscale$ is the resulting grayscale value. Ideally the luminance Y as output by the camera's should be used, but due to implementation constraints this conversion is done. The hand tracker consists of a separate frame grabber module which outputs directly in RGB, when the system was designed it was not yet known that in a later stage the luminance component could be used again.

The most straightforward algorithm is FD, the difference of each pixel between the current frame F_i and the previous frame F_{i-1} is calculated. If the difference is larger than a certain threshold T the pixel is counted as foreground. The following equation is used,

$$|F_i - F_{i-1}| > T \tag{3.3}$$

where i is the pixel index in a frame F. Obviously, by only using the previous frame FD adapts very quickly to changes in the background.

If a hand stops moving for more than $1/15$ of a second it becomes part of the background [20].

A bit more sophisticated is AMF. Again the difference is calculated,

$$|F_i - F_b| > T \tag{3.4}$$

where F_b is the background frame. But AMF also updates the background frame every n frames. Thus in contrast to FD, AMF propagates changes slowly; this depends on the update rate variable n. The first background frame is simply a copy of the current frame. Every new frame will be compared to the stored frame. The background frame will be updated as follows,

$$F_b = \begin{cases} F_b + 1, & \text{if } F_i > F_b \\ F_b - 1, & \text{if } F_i < F_b \\ F_b, & \text{otherwise} \end{cases} \tag{3.5}$$

where F_b is the background frame [39].

As MoG is a quite complex technique the algorithm is only roughly described here, a more detailed explanation can be found in papers by Cheung and Kamath [13] and Stauffer and Grimson [50]. First the probability function *f* is defined,

$$f(I_t = u) = \sum_{i=1}^{k} \omega_{i,t} \cdot \eta(u; \mu_{i,t}, \sigma_{i,t}) \tag{3.6}$$

where $\eta(u; \mu_{i,t}, \sigma_{i,t})$ is the i-th Gaussian component with intensity mean $\mu_{i,t}$ and standard deviation $\sigma_{i,t}$. The first step is to identify the component of whose mean is closest to I_t. A component $\hat{i}$ is called a *matched component* if the absolute difference between the mean and pixel value is less than the components standard deviation,

$$|I_t - \mu_{\hat{i},t-1}| \leqslant D \cdot \sigma_{\hat{i},t-1} \tag{3.7}$$

where D is a sensitivity parameter. The next step is to update the matched component variables $\omega_{\hat{i},t}$, $\mu_{\hat{i},t}$ and $\sigma_{\hat{i},t}$. If a matched component is not found only $\omega_{\hat{i},t}$ is updated and $\mu_{\hat{i},t}$ and $\sigma_{\hat{i},t}$ do not change. The last step is to normalize the weights again to sum up to one. In order to determine if a pixel is part of the background all components are ranked by their values $\omega_{i,t}/\sigma_{i,t-1}$ and then a threshold is applied to the weights. The background model is the first M components whose weight is above a threshold; M is the maximum number of components. If a pixel value fits within this criteria it is counted as background.

3.4 SKIN SEGMENTATION

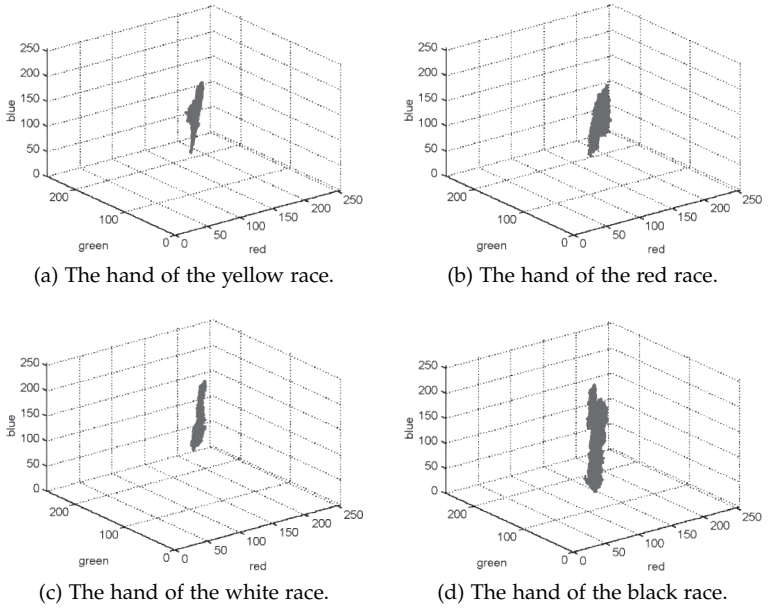
For skin segmentation the following algorithms where implemented and tested: intersection algorithm [1], artificial neural network [46], Bayesian classifier [31, 58]. Each algorithm will be explained in detail in the next sections.

3.4.1 *The intersection approach*

Ahmad [1] describes a technique to do histogram based skin segmentation. The idea is to create a patch from the image and create a two-dimensional histogram of that patch. Not every pixel is checked, instead the image is subsampled; after a patch located at (x, y) is checked the next patch is at location $(x + s, y)$ and at the end of the line $(0, y + s)$ is checked. The subsampling constant s is set automatically using an adaptive technique; this is based on the desired frame rate, if the frame rate is higher as desired, s is decreased and if the frame rate is lower s is increased. Note that for evaluation a static constant s is used. When a histogram is created from a patch it is compared to a predefined histogram. This predefined histogram represents the skin color distribution and is created using a set of training images. Instead of the commonly used RGB color space a normalized two-dimensional variant is used,

$$R' = \frac{R}{R+G+B+1} \text{ and } G' = \frac{G}{R+G+B+1} \tag{3.8}$$

where R' and G' range between zero and one. R' and G' represent the percentages of red and blue. The idea is that this will lower the influence of varying light conditions. Ahmad states that R' and B' are used, this is believed to be an error since one of the papers Ahmad refers to uses R' and G' [53]. Also, research by Yin, Guo and Xie [61] show that the red and green components are grouped together, see Figure 17. Each



(a) The hand of the yellow race.

(b) The hand of the red race.

(c) The hand of the white race.

(d) The hand of the black race.

Figure 17: RGB color distribution of human skin for different skin types (source: Yin, Guo and Xie [61]).

patch needs to be compared to the predefined histogram containing the skin color distribution. This is done using the intersection algorithm by Swain and Ballard [53],

$$M_{p,q} = \frac{\sum_{i,j} \min(H^p(i,j), H^q(i,j))}{\sum_{i,j} H^p(i,j)} \tag{3.9}$$

where $M_{p,q}$ is the resulting match score, i and j the histogram index, H^p is the histogram of the patch created from the current image and H^q is the predefined skin histogram. If the match score is above a certain threshold T, typically a value above 0.9, the patch is classified as skin. A disadvantage of this method is that it does not work on a per pixel base, skin is segmented in the square patch size. It is not as precise as the Bayesian classifier.

3.4.2 *The Bayesian approach*

In contrast to the intersection algorithm the Bayesian classifier as proposed by Jones and Rehg [31] uses two two-dimensional histograms. The histograms typically have a size of 32×32. This method works on a per pixel basis; each pixel will be classified as skin or non-skin. The general idea is that one histogram is trained on a set of human skin images and the second histogram is trained on a set of background images. These histograms can be created by using regular RGB color space converted to normalized RGB as computed in Equation 3.8. But also by converting RGB color space to hue, saturation and value (HSV). The value of HSV represents the lightness and is less important because only hue and saturation are stored. Hue is calculated using the following equation [15],

$$hue = \begin{cases} 0, & \text{if } Max = Min \\ (60° \times \frac{G-B}{Max-Min} + 360°) \bmod 360, & \text{if } Max = R \\ 60° \times \frac{B-R}{Max-Min} + 120°, & \text{if } Max = G \\ 60° \times \frac{R-G}{Max-Min} + 240°, & \text{if } Max = B \end{cases} \tag{3.10}$$

and saturation is calculated using,

$$saturation = \begin{cases} 0, & \text{if } Max = 0 \\ \frac{Max-Min}{Max}, & \text{else} \end{cases} \tag{3.11}$$

The histogram bins correspond to a particular range of color values. Afterwards these histograms are normalized,

$$P_{Skin}(i) = \frac{Skin[i]}{Normalization} \tag{3.12}$$

and the background (or $\neg Skin$) histogram is normalized likewise,

$$P_{\neg Skin}(i) = \frac{\neg Skin[i]}{Normalization} \tag{3.13}$$

where $Skin[i]$ and $\neg Skin[i]$ are the specific histogram bin values. There are some slight deviations in the different Bayes methods. For example Zarit, Super and Quek [4] define $Normalization$ by dividing each bin by the largest bin value. In contrast to Jones and Rehg who normalize the histogram by dividing each bin with the sum of all bins, the second method is used for ARMI.

The method proposed by Jones and Rehg [31] uses the conditional probability $P(c|Skin)$. In other words the probability of observing color

c, given the pixel is skin. A more usable solution is proposed by Vezhnevets, Sazonov and Andreeva [58], $P(skin|c)$, the probability that a pixel is skin given color c. They calculate the probability using the following Bayes rule,

$$P(Skin|c) = \frac{P(c|Skin)P(Skin)}{P(c|Skin)P(Skin) + P(c|\neg Skin)P(\neg Skin)} \tag{3.14}$$

where $P(c|Skin)$ and $P(c|\neg Skin)$ are calculated using their corresponding histograms. $P(Skin)$ and $P(\neg Skin)$ are the prior probabilities that a pixel is skin regardless the color. The prior probability is an estimate based on the number of skin and background samples in the training set. Vezhnevets, Sazonov and Andreeva [58] state that explicitly computing Equation 3.14 is not necessary if only a comparison is made of $P(Skin|c)$ and $P(\neg Skin|c)$. They calculate the ratio as follows,

$$\frac{P(Skin|c)}{P(\neg Skin)} = \frac{P(c|Skin)P(Skin)}{P(c|\neg Skin)P(\neg Skin)} \tag{3.15}$$

The prior probabilities are an estimate of the ratio of skin versus non-skin pixels. It does not directly affect the detection behavior. Also the training data used for the hand tracker implementation will not result in a good estimate; the webcams are pointed at a table. Therefore Equation 3.15 can be further reduced to,

$$\frac{P(c|Skin)}{P(c|\neg Skin)} \geqslant T \tag{3.16}$$

where T is a threshold. This threshold is chosen manually by testing with a range of thresholds.

3.4.3 *The artificial neural network approach*

The last method discussed is based on an artificial neural network. This method is based on the skin histogram based method by Ahmad [1] and is an idea of the author. Ahmad uses a single two-dimensional histogram with normalized RGB color space. The neural network uses the same patch based solution but uses HSV instead of normalized RGB, see Equation 3.10 and Equation 3.11, and feeds the values to the network instead of storing them in a histogram. In research done by Yin, Guo and Xie [61] HSV gave better results for use with skin segmentation. The feedforward artificial network used is called a multilayer perceptron. The multilayer perceptron is able to distinguish non-linearly separable data, in contrast to the original perceptron by Rosenblatt [46] which the multilayer perceptron is based on. Basically the idea is to let the network find a pattern in the input data. See Figure 18 for an example of how the network processes the input data. In this figure x_0 to x_n represents the input data, which in this case is a patch (a square area in the frame). All the weights w_0 ... w_n and Wh_0 ... Wh_m are chosen when the network is trained; how the network is trained is explained in the next subparagraph. θ_0 to θ_m represent the bias values and $outputbias$ is bias used for the output o. Using the following equation the output o is calculated,

$$o = \sigma\left(\sum_{j=0}^{m} \sigma\left(\sum_{i=0}^{n} x_i W x_{i,j} + \theta_j\right) Wh_j + outputbias\right) \tag{3.17}$$

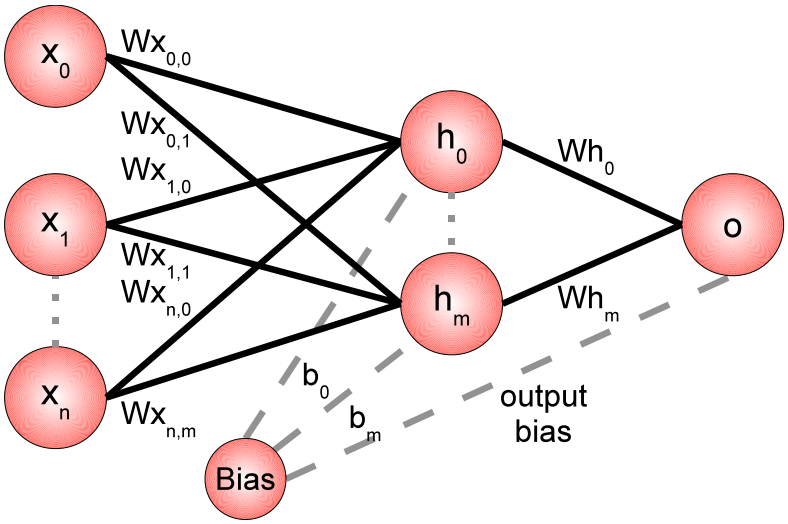


Figure 18: Example of an artificial neural network.

where m is the number of hidden layers and n is the number of input layers. The sigmoid function $\sigma(x)$ is used as activation function and is defined as follows,

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.18}$$

this will give an output value in the range of zero to one, see Figure 19. The general idea is to have a smooth output value which is used as

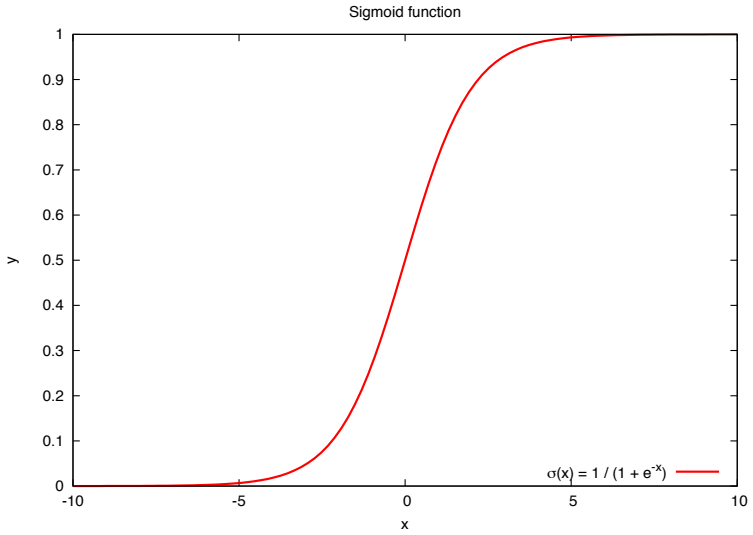


Figure 19: Output of sigmoid function.

percentage, this number represents the possibility the patch used as input for the network is skin. If the output o is larger than a certain threshold T the patch is classified as skin,

$$o \leqslant T \tag{3.19}$$

threshold T is typically a value of around 0.9. In Figure 20d a sample output of skin segmentation using the artificial neural network is shown.

Figure 20 gives a clear overview of skin segmentation for all three algorithms discussed in this paragraph.

(a) The original image.

(b) Output of skin histogram based algorithm proposed by Ahmad.

(c) Output of Bayesian classifier as proposed by Jones and Rehg.

(d) Output of artificial neural network based skin segmentation.

Figure 20: Screenshots of skin segmented images using the three described algorithms. In Figure 20a the original image is shown.

Before the network can be used it requires training of the network. This is done using a common used method called back propagation [42].

1. Initialize all weights with random values.
2. Set the input data.
3. Feed the input data forward through the network.
4. Use the difference the desired output and the activation value in order to calculate the network's activation error.
5. Adjust weights to reduce the activation error of the input data.
6. Repeat steps two to five.
7. Repeat this last seventh step until the error is below a certain constant, for example 0.1.

The input data is a manual classified set of skin and background patches, every iteration a new patch a long with its desired output, one for skin and zero for non-skin, is fed in to the network. The weights are adjusted only slightly in order to learn an generalized solution for the input set. Using back propagation it is possible that no solution is found

and the network is stuck in a local optimum. There are probably some sophisticated methods available to overcome this, but the simplest solution is to restart training after, say 5000 iterations, with freshly initialized weights. This is sufficient for skin segmentation.

3.5 CONNECTED COMPONENT LABELING

The goal of this step is to get the dimensions of all areas within the image; the binary areas marked in the alpha layer. The areas are cut in a rectangular shape, the coordinate of the upper left pixel and the width and height of the rectangle is stored. See Figure 21, note that the white rectangle and pink colored pixels are only for demonstrative purposes; in reality the image stays untouched. The outlines are used for the next phase of the project, hand pose estimation. The input of

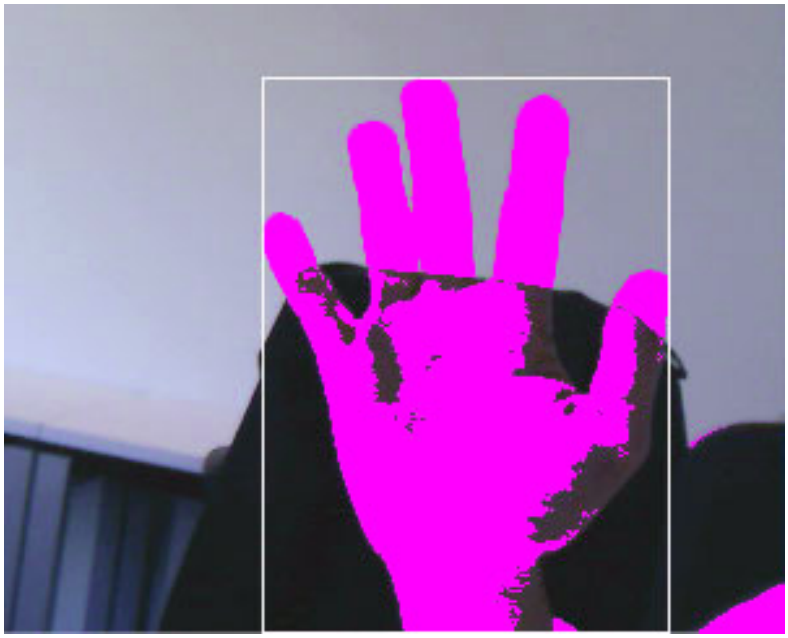


Figure 21: Rectangle outline of area.

this step is a binary image where all skin colored pixels are marked as one and background pixels marked zero. Using a technique called connected component labeling [49] the outlines are calculated, this is a widely used technique for binary image analysis. The idea is to give every area a unique identifier and use that unique identifier for further processing. The outlines are calculated in two passes. The first pass is done simultaneously with skin segmentation. The image is processed starting from the upper right corner and ending in the lower left corner of the binary image. The image is stored as a big one-dimensional array, the position within this array is used as unique identifier.

In order to determine if skin pixels are part of the same area a strategy called four-connectivity [22] is used, see Figure 22. Of every pixel its north, east, south and west neighbor are used. Because of the way the pixels are processed, per row from left to right, it is only necessary to lookup the east and south neighbor.

	N	
W	C	E
	S	

Figure 22: Four-connectivity.

For the hand tracker the classical row-by-row algorithm by Rosenfeld and Pfaltz [47] is slightly modified in order to find the skin areas in an image. The areas are defined as all components which are classified as skin and are connected through four-connectivity. The algorithm works as follows:

1. *First pass*. Iterate through the image and give all pixels marked as skin an unique identifier. Starting in the top left of the image and end at the right bottom of the image.

2. *Second pass*. Iterate through the image the same way as in the first pass, but if the pixel is marked as skin,

   a) Check if the east (right) neighbor is marked as skin, if not skip this step and go the the next step. If the neighboring pixel does not belong to any other component give it the identifier of the current component. If the current component is not yet assigned to another component assign it the identifier of the neighbor else both components are merged into one. The component with the smallest identifier will be used as new component and the upper and lower bounds are updated to the new area boundaries.

   b) Now check the south neighbor, the component one row below the current component. If the south neighbor is marked as skin and has no identifier assigned yet, set the identifier of the neighboring component to the identifier of the current component.

   c) Update the upper and lower coordinate bounds of the area.

The Union-Find algorithm by Tarjan [54] is likely faster, but unfortunately there was no time left to implement and test this solution. In Section 6.1 a more detailed description of an enhancement which uses area openings and incorporates the Union-Find algorithm is given. Figure 23 shows an simplified example of how a binary image is processed using connected component labeling. Note that the area size is left out for clarity.

1	0	1
1	0	1
0	1	0

(a) Binary image.

1	0	3
4	0	6
0	8	0

(b) Assign unique identifiers.

1	0	3
1	0	3
0	8	0

(c) Labeled image.

Figure 23: Example connected component labeling.

This enumeration shows how the algorithm works theoretically, the following snippets of pseudo code are based on the actual implementation in C. First an array of structures (`struct` in C) is created, the layout of this structure is shown in Listing 2.

```
struct ComponentList_Node
    int index, id
    int x1, y1
    int x2, y2
    struct ComponentList_Node* next
```

Listing 2: Structure of connected component node.

The array is of the same size as the to be processed image. In reality the first iteration (first pass in pseudo code) is a bit larger, because the actual classification is also done in this loop. To keep the pseudo code small only the connected component labeling parts are shown, see Listing 3.

```
function labelImageFirstPass(image, width, height, bpp):
    ComponentList_Node cc[width * height]
    ComponentList_Node cc_current, cc_first

    // First pass, give all skin pixels an unique identifier.
    for i = 0; i < width * height:
        if image[(i * bpp) + position_alpha_layer] = 255:
            cc[i] = setConnectedComponent(i, i % width, i / width)
            if cc_current is not NULL:
                cc_current.next = cc[i]
            else:
                cc_current = cc[i]
            if cc_first is NULL:
                cc_first = cc_current
        else:
            cc[i] = setConnectedComponent(-1 , 0, 0)
```

Listing 3: The first pass; label skin pixel with an unique identifier

The `labelImageFirstPass` function uses a small helper function to quickly fill in a new struct with its corresponding values. Note that `x1`, `y1`, `x2` and `y2` are first assigned the same `x` and `y` value. As the area grows these values are updated, see Listing 4.

```
function setConnectedComponent(index, x, y):
    ComponentList_Node newcc

    newcc.id    = newcc.index  = index
    newcc.x1    = newcc.x2     = x
    newcc.y1    = newcc.y2     = y
    newcc.next  = NULL
    return newcc
```

Listing 4: Helper function to fill in a new node.

After the first pass, which uniquely labeled all skin pixels, a second pass is required. This pass will check the neighboring pixels using four-connectivity and will merge regions if needed. The pseudo code is listed in Listing 5 and the node merge function is listed in Listing 6.

```
function labelImageSecondPass(cc, cc_first):
    ComponentList_Node_t cc_current

    cc_current = cc_first
    while cc_current is not NULL:
        i = cc_current.index
        // East neighbor (X-Axis)
        if (i + 1) % width > 0 and cc[i + 1].id != -1:
            // Does not belong to any other component yet.
            if cc[i + 1].id == i + 1
                cc[i + 1].id = cc_current.id
            // Does the current component belong to any component?
            // If not, add current.
            else if cc_current.id == i:
                cc_current.id = c[i + 1].id
            // Merge two components.
            else:
                // Smallest ID will be new merged component ID.
                if cc_current.id < cc[i + 1].id:
                    mergeCComponents(cc[cc[i + 1].id],
                                     cc_current      , cc[i + 1])
                else:
                    mergeCComponents(cc[cc_current.id],
                                     cc[i + 1]        , cc_current)
        // South neighbor (Y-Axis)
        if (i + width) < i_end and cc[i + width].id != -1:
            cc[i + width].id = cc_current.id

        if cc_current.id != i:
            // Update X-Axis coordinates.
            if cc_current.x1 < cc[cc_current.id].x1:
                cc[cc_current.id].x1 = cc_current.x1
            if cc_current.x2 > cc[cc_current.id].x2:
                cc[cc_current.id].x2 = cc_current.x2
            // Update Y-Axis coordinates.
            if cc_current.y2 > cc[cc_current.id].y2:
                cc[cc_current.id].y2 = cc_current.y2
        cc_current = cc_current.next
```

Listing 5: Second pass; merge connected components.

The result of this pass is an array of `structs`. All `structs` which have an identifier assigned are labeled as skin area. All identified skin areas contain the upper left and lower right coordinates, thus the skin area has a rectangular shape. This array is passed to the hand pose estimator by Boer [7]. The marked skin rectangles are used as areas of interest for the hand pose estimator.

```
function mergeCComponents(cc1, cc2, cc3):
    cc1.id = cc2.id
    cc3.id = cc2.id

    if cc1.x1 < cc2.x1:
        cc2.x1 = cc1.x1
    if cc1.x2 > cc2.x2:
        cc2.x2 = cc1.x2
    if cc1.y2 > cc2.y2:
        cc2.y2 = cc1.y2
```

Listing 6: Helper function to merge two nodes.

4

EVALUATION

The evaluation chapter discusses the performance of the various methods and approaches, also a benchmark to measure the frames per second is executed. Not all approaches discussed are implemented. This was not possible due to time constraints; only the researched skin segmentation approaches are implemented. First, the test setup is discussed followed by the actual results, an evaluation of these results and lastly the benchmark to measure the average frame rate. The benchmark only measures the frame rate of the approach which is used in the final hand tracker prototype.

4.1 SETUP

In order to train the three approaches a set of 92 images is used, of which 33 images are images that only contain skin and 59 images contain only background colors. The set consists mostly of images taken inside the Bernoulliborg building, located at the Zernike Science Park, Groningen, The Netherlands. The rest of the images are crawled from the web and selected by hand. An overview of the exact number of images is given in Table 1.

		Count
Actual	Skin	33
	Background	59

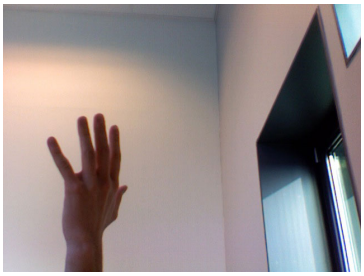
Table 1: Separation of trainingdata.

The skin images are segmented by hand, within these hand images the non skin pixels are marked with a special color: pink (red=255, green=0, blue=255). Note that a set of 92 images may seem like an odd number, but a few images where thrown away because they were not suitable for training. For example a few background images contained people in the background. Figure 24a and Figure 24b show a typical example of a skin and background image used for training.

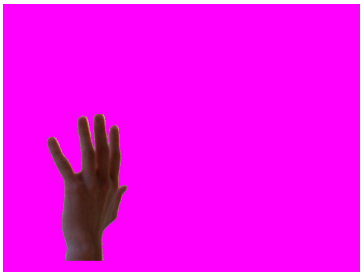
Besides the set of training images also a set of test images is created. This set contains 2×15 images. For every image in the test set there is one original (unmodified) image and one reference image. The first is used by to perform skin segmentation on. The second image is a hand segmented image which is used to determine the score. Figure 24c is an example of a test image. See Figure 24d is an example of a hand segmented reference image. More samples of training and test images are available at the ARMI Project website [17].

(a) Typical skin image used as trainingdata.

(b) Typical background image used as trainingdata.



(c) Typical image used as testdata.



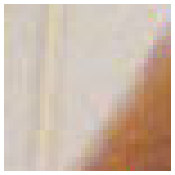
(d) Hand cut image used for determining segmentation score.

Figure 24: Four examples of training and test data images.

The Bayesian classifier segments skin on a per pixel base, thus the segmented image and reference image can be compared one-on-one. For the other two approaches, intersection and artificial neural network, it is less straightforward. These require that a similar patch is created at the same coordinates and size of the reference image, to determine if the reference patch is skin or not the percentage of skin pixels within the patch is calculated,

$$p = \frac{\text{total skin pixels}}{\text{total pixels}} \tag{4.1}$$

If $p > 25\%$ than the reference patch is counted as skin. Changing the percentage value has effect on the trapezoid shaped skin boundaries formed by the patches, see Figure 25. A larger percentage results in that patches located at the boundaries require more skin pixels before they are marked as skin patch. By trial and error with different percentages it was found that a value of 25% gave the best result. Note that the skin training data consists only of images containing 100% skin pixels.



(a) Original image, skin is located at the bottom right.

(b) Image with approximately 10% marked.

(c) Image with approximately 25% marked.

Figure 25: One sample skin patch with two different percentages. The white areas in (b) and (c) are the marked skin areas.

For every patch and pixel, the true positives (TP) and the false positives (FP) are counted. In this context TP are defined as pixels and patches which are correctly classified as skin. FP are the pixels and patches which are counted as skin but in reality are part of the background. The TP are calculated as follows:

$$\frac{\text{true positives}}{\text{total skin pixels or patches}} \tag{4.2}$$

And the FP are calculated as follows:

$$\frac{\text{false positives}}{\text{total non skin pixels or patches}} \tag{4.3}$$

Besides the TP and FP the following terms are also defined: false negatives (FN) are pixels and patches classified as background but are actually skin. True negatives (TN) are pixels and patches which are correctly classified as part of the background.

4.2 RESULTS

For the intersection, Bayesian and artificial neural network approach the optimal threshold T has to be found. This threshold T is used to determine if a pixel or patch is skin or not; if it is smaller than threshold T the pixel is classified as non-skin pixel (background). The optimal threshold T for each approach is a value ranging from 0.05 to 1.00. The step size is set to 0.05. Now, the optimal threshold can be found by running each approach for a range of thresholds values. Each value results in a true positive rate and a false positive rate. True positives are pixels correctly classified; thus it is classified as skin and truly is skin, this also holds for the non-skin pixels. The false positive rate represents the number of pixels and patches that are falsely classified as skin or background.

As mentioned earlier the intersection and artificial neural network approach use patches. In order to make a better comparison with the pixel based Bayesian approach the following additional evaluation is done, for every correctly classified patch (TP) in both patch based approaches the actual pixels of skin within that patch are also counted. It is possible that within that same patch a few pixels are still background pixels, these are counted as FP. To summarize for both patch based approaches a calculation based on complete patches is done and one to calculate the per pixel score. Expected is that these per pixel calculations perform worse, patches on the border of a hand and background (trapezoid effect) still contain background pixels while they are possibly counted as skin.

In Figure 26 the results of all approaches plus two extra plots of the per pixel patch based approaches is shown. A point in the upper left corner represents the best possible prediction of skin and background pixels; thus ideally the true positive rate should be close around one and the false positive rate around zero. Figure 26 shows that the Bayesian approach scores best followed by the intersection approach. The artificial neural network approach performs the worst. The optimal threshold T

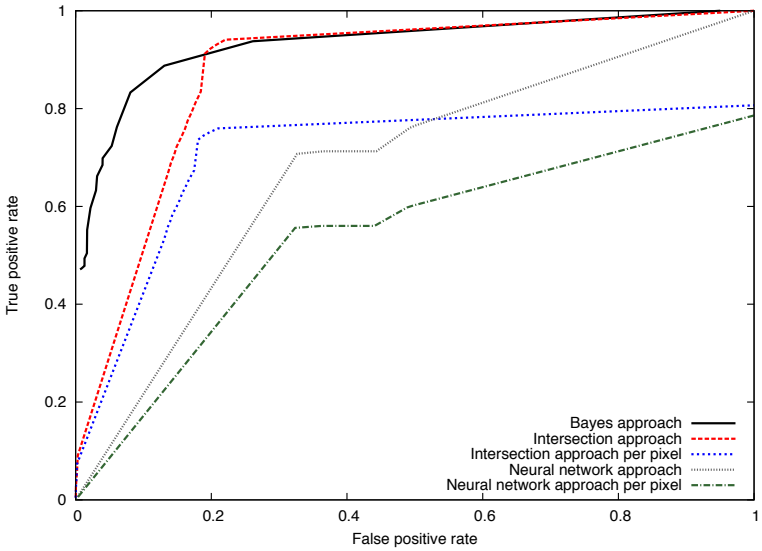


Figure 26: ROC curve for intersection, Bayesian and artificial neural network approach.

can be extracted from the graph by finding the corresponding values closest to one on the y-axis and closest to zero on the x-axis and looking up its corresponding threshold T. These thresholds are: Bayesian $T = 0.85$, intersection $T = 0.85$ and the artificial neural network $T = 0.95$. The per pixel calculations of both patch based approaches have the same threshold as their per patch calculated counter parts. Also, as expected the additional evaluation using per pixel values of the intersection and artificial neural network approach score significantly lower.

		Predicted		
		Positive	Negative	Total
Actual	Positive	TP	FN	TP + FN
	Negative	FP	TN	FP + TN
	Total	TP + FP	FN + TN	

Table 2: Example confusion matrix containing the true positives (TP), false negatives (FN), false positives (FP) and true negatives (TN).

For each approach a confusion matrix is given. A confusion matrix is, like a graph, also used for visualization. The rows represent the predictions while each column represent the actual outcome. This makes it easier to see if an approach is mislabeling pixels (confusing) [35], see Table 2 for an example of the layout. Using a specially constructed test program the TP and FP are calculated for each approach, the test program outputs the TP and FP based on the test data set. Using these two values the remaining FN and TN are calculated and presented in a confusion matrix. The accuracy is calculated using the value of the confusion matrix, Kohavi and Provost define the accuracy as follows [35],

> "**Accuracy (error rate) –** The rate of correct (incorrect) predictions made by the model over a data set (cf. coverage). Accuracy is usually estimated by using an independent test set that was not used at any time during the learning process. More complex accuracy estimation techniques, such as cross-validation and the bootstrap, are commonly used, especially with data sets containing a small number of instances."

the accuracy of all approaches is calculated using the following equation,

$$accuracy = \frac{true\ positives + true\ negatives}{total\ positives + total\ negatives} \tag{4.4}$$

the accuracy ranges from zero to one, where a value of one represents no errors in skin classification, thus a high accuracy indicates that an approach is able to distinguish skin from non-skin accurately (without errors). In the next section for all three approaches the confusion matrix is given first followed by its corresponding accuracy.

The first approach is the Bayesian, see Table 3 for the results with a threshold of $T = 0.85$.

		Predicted		
		Positive	Negative	Total
Actual	Positive	83.3%	16.7%	100%
	Negative	9.4%	90.6%	100%
	Total	92.7%	107.3%	

Table 3: Confusion matrix Bayesian approach with $T = 0.85$.

The accuracy of the Bayesian approach using the values of the optimal threshold is calculated as follows,

$$\frac{83.3 + 90.6}{100 + 100} = 0.87 \tag{4.5}$$

In "A survey on Pixel-Based Skin Color Detection Techniques" by Vezhnevets, Sazonov and Andreeva [58] the Bayesian approach as proposed by Jones and Rehg [31] and used for this evaluation is discussed. These results are presented in Table 4 and also used a histogram size of 32×32.

The results in Table 4 and Table 3 are quite close, TP of 90% versus 83.3% and TN of 14.2% versus 9.4%. Vezhnevets, Sazonov and Andreeva used an image library of 18,696 images, nearly two billion pixels, of which 13,640 images are used for training. Unfortunately this library is not available anymore. It is likely that these deviations are due to slight differences in the implementation and the quite large difference in training data. Possibly, the histograms created from the training data used for this evaluation are not yet optimally generalized; this could be solved by increasing the amount of training data.

		Predicted		
		Positive	Negative	Total
Actual	Positive	90.0%	10.0%	100%
	Negative	14.2%	85.8%	100%
	Total	104.2%	95.8%	

Table 4: Confusion matrix Bayesian approach as tested by Vezhnevets, Sazonov and Andreeva [58].

The next two tables, Table 5 and Table 6, shows the results for the intersection approach with a threshold of $T = 0.85$ a histogram size of 8×8, subsampling constant of 8 pixels and a patch size of 8×8 pixels.

		Predicted		
		Positive	Negative	Total
Actual	Positive	92.2%	7.8%	100%
	Negative	19.8%	80.2%	100%
	Total	112.0%	88.0%	

Table 5: Confusion matrix intersection approach with $T = 0.85$.

		Predicted		
		Positive	Negative	Total
Actual	Positive	74.4%	25.6%	100%
	Negative	18.8%	81.2%	100%
	Total	93.2%	106.8%	

Table 6: Confusion matrix intersection approach per pixel version with $T = 0.85$.

The accuracy of the intersection approach using the values as shown in the table is calculated as follows,

$$\frac{92.2 + 80.2}{100 + 100} = 0.86 \tag{4.6}$$

and next the accuracy of the per pixel version,

$$\frac{74.4 + 81.2}{100 + 100} = 0.78 \tag{4.7}$$

as shown, the per pixel version has a lower accuracy. This is obviously due to the fact that a lot of patches on the boundary of skin and background are falsely classified. Ideally the results are compared with results of the author of the original papers. In "A Usable Real-Time 3D Hand Tracker" by Ahmad [1] there is no evaluation of the actual performance of the approach done, however a typical threshold of $T = 0.9$ is mentioned. This slightly differs from the threshold found in

this paper: $T = 0.85$. Furthermore there is no exact description of how exactly the training of the the pre-calculated skin histogram is done. Is the skin histogram defined using only one patch, or are multiple patches averaged; there are a lot of different ways to create the skin histogram. Also the histogram size is not mentioned by Ahmad.

The last table, Table 7, shows the results for the artificial neural network approach with a threshold of $T = 0.95$ and a patch size of 10×10 pixels with one hidden layer of two neurons.

		Predicted		
		Positive	Negative	Total
Actual	Positive	70.7%	29.3%	100%
	Negative	32.7%	67.3%	100%
	Total	103.4%	96.6%	

Table 7: Confusion matrix artificial neural network with $T = 0.95$.

Table 8 shows the confusion matrix of the artificial neural network per pixel version. The accuracy of the artificial neural network using the

		Predicted		
		Positive	Negative	Total
Actual	Positive	55.6%	44.4%	100%
	Negative	32.4%	67.6%	100%
	Total	88.0%	112.0%	

Table 8: Confusion matrix artificial neural network per pixel version with $T = 0.95$.

values as shown in Table 7,

$$\frac{70.7 + 67.3}{100 + 100} = 0.69 \tag{4.8}$$

and the accuracy of the artificial neural network on per pixel basis.

$$\frac{55.6 + 67.6}{100 + 100} = 0.62 \tag{4.9}$$

The artificial neural network in the additional per pixel evaluation shows a decrease in TP, 70.7% versus 55.6%. So the border between hand and background in an image still seems to confuse the classifier. Also, the artificial neural network is based on an own implementation, thus there is no reference material to compare the scores with, except for the other two approaches. Just like the intersection approach the accuracy and confusion matrix for the artificial neural network show worse results compared to the per patch version.

In Table 9 a summary of all accuracy rates is shown. Overall the Bayesian approach scores best in accuracy compared to the other two

		Accuracy
Approach	Bayes	0.87
	Intersection	0.86
	Intersection per pixel	0.78
	Artificial neural network	0.69
	Artificial neural network per pixel	0.62

Table 9: Summary of accuracies of all evaluated approaches.

approaches. Thus the Bayesian approach scores best in correctly classifying all pixels as skin and in correctly classifying background pixels as background; especially when looking at the per pixel results. Also, a per pixel classifier works best with the hand pose estimation which is done in after hand tracking. To summarize, the Bayesian approach score satisfies the ARMI requirements and is thus used for the final implementation of the hand tracker as part of ARMI.

4.3 BENCHMARK

One of the requirements as mentioned in Chapter 3 is to achieve at least 15 frames per second. Obviously this depends on the hardware used, for this benchmark the authors laptop is used, see Table 10 for the exact specifications of this machine. Also, the benchmark only covers the hand tracker and not the complete solution including the hand pose estimator.

	Apple Macbook Pro
Model identifier	MacBookPro2,2
Processor name	Intel Core 2 Duo
Processor speed	2.16 GHz
Memory	3 GB DDR2 667 MHz
Operating system	Mac OS X Snow Leopard
Operating system version	10.6.1

	Camera
Brand and model	Logitech QuickCam S 7500

Table 10: Hardware used for benchmarks.

For the actual benchmark two small movie files are recorded, the first movie contains only one hand while the second movie contains two moving hands. Tracking two or more hands will increase the processing slightly, for every extra hand the corresponding boundaries have to be calculated. Also, when the image contains a lot of noise, as in falsely as skin classified areas more boundaries have to be calculated

an the frame rate will decrease. This is only noticeable when there are a lot, say 25 or more. Thus the better the Bayesian skin classifier performs the higher the frame rate. The results of these benchmarks are shown in Table 11. For this benchmark the Bayesian approach [31, 58] is used in combination with the approximate median filter background subtraction method [39] and connected component labeling. The benchmark uses a prerecorded movie created with a Logitech webcam [38]. In order to make an accurate estimate of the average frame rate of solely the hand tracker, every frame is cached in memory in RGB colorspace. Every frame is then processed as fast as possible, using the total amount of frames divided by the time it takes to process these frames the average frame rate is calculated.

The main reason for not using the webcam directly as input is because the frame rate is limited, the Logitech webcam is only capable of delivering frames at a maximum rate of 15 frames per second. Also, grabbing a single frame from the (USB) webcam can take up quite an amount of processing time. This mainly depends on the type of camera used. More expensive webcams are capable of outputting frames directly in RGB colorspace, while the webcam used for ARMI will only output frames in YCbCr.

	Frames	Time in seconds	Frames per second
Single hand	210	10.247	20.49
Two hands	206	10.207	20.18

Table 11: Results of performance benchmark.

Table 11 shows that an average of 20.49 frames per second is achieved, which is noticeable higher than required. The current implementation uses only one core, the hand tracker consists mainly of several for loops; it should be relatively easy to make it multi-threaded. The original movie files used for the benchmark and a sample movie of a tracked hand are available at the ARMI Project website [17].

5 IMPLEMENTATION

The implementation chapter consists of the most important implementation details. A few components required optimization in order to increase tracking speed, these optimizations are also discussed here. Furthermore snippets of pseudo code are given.

5.1 ARCHITECTURE OVERVIEW

The hand tracker is implemented in various programming languages. Most code is written in C [34], some specific Apple OS X parts are written in Objective-C and a custom Python module is implemented in order to communicate with ARMI. While most of ARMI is written in Python, the hand tracker required a more efficient language with less overhead; thus C is chosen. Note that C++ is also sufficient, but because of personal preference and experience with the C language it was not chosen over C. In Figure 27 an overview of the architecture is shown.

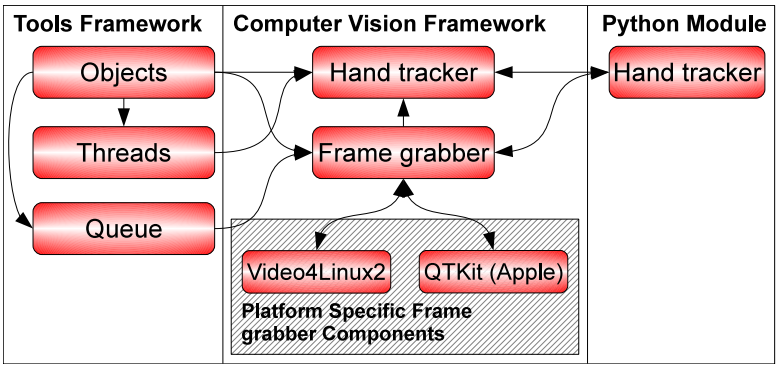


Figure 27: Architectural overview of the hand tracker.

Basically the hand tracker consists of three components, a common Tools Framework (TF), the Computer Vision Framework (CVF) and the Python Module (PM). The source code of the hand tracker is also available for download at the ARMI Project website [17].

5.2 TOOLS FRAMEWORK

The Tools Framework is mainly used to implement the Object-Oriented Programming (OOP) programming paradigm [48] in C. Basically a C `struct` called Object is created, see Listing 7, this can be seen as the base class; every other class should inherit it. This base class contains a reference counter and a pointer to a destroy function. Obviously C does

not support inheritance, this is simulated by clever usage of defines and the C preprocessor.

```
typedef void (*f_destroy_ptr)(void** obj)
#define OBJECT_REQUIRED int referenceCounter \
                        f_destroy_ptr destroyPointer
#define FREE_OBJECT(o) if o != NULL:         \
                           free(object); object = NULL
#define OBJECT_CREATE(o) (o##_t*)Object_Create(              \
                         (f_destroy_ptr)&o##Destroy, sizeof(o##_t))
typedef struct
{
    OBJECT_REQUIRED
} Object_t
```

Listing 7: Base object class.

By simply including the define into a structure a new class can be created. See Listing 8 for a small example of how the OOP pattern as implemented can be used.

```
typedef struct
{
    OBJECT_REQUIRED
    char* string
    int   length
} StringObject_t

StringObject_t* StringObject_Create(char* string, int length):
    StringObject_t* self = OBJECT_CREATE(StringObject)
    if self != NULL:
        strncpy(self->string, string, length)
        self->length = length
    return self

void StringObjectDestroy(StringObject_t** self):
    if ObjectCanRelease(self):
        free((*self)->string)
        FREE_OBJECT(*self)

void StringObjectGet(StringObject_t* self, char** str, int* len):
    if self != NULL:
        *str = self->string
        *len = self->length
```

Listing 8: Example, implementation of a string class.

Also four default Object functions are defined:

- `ObjectCreate(size, destroyptr)`, allocate `size` memory and set pointer to destroy function.
- `ObjectDestroy(self)`, a pointer to the destroy function.
- `ObjectRetain(self)`, increase reference counter by one.
- `ObjectCanRelease(self)`, decrease reference counter by one and return `True` if the counter is equal to zero, otherwise set self to `NULL`.

This creates the possibility to have multiple instances of one object. For example two frame grabber instances are required, one for each webcam instance. This adds some slight extra overhead but allows multiple instances of an object with only little extra effort. Lastly, the Tools framework contains a debug module which keeps track of all allocated and freed memory. It can pinpoint the exact location in the source file if some allocated memory is not freed. The framework also contains some common used data structures such as a queue and a few helper modules for easy creation and destruction of threads.

5.3 COMPUTER VISION FRAMEWORK

The second framework is the Computer Vision Framework (CVF). This framework is responsible for all computer vision related code. It contains the actual hand tracker implementation and the frame grabber interface.

The frame rate is set to a fixed number, for ARMI it is set to 15 frames per second. The hand tracker will maintain this rate independent from the CPU time. Thus a faster CPU will have more processing time left for other resources. In case the CPU is not fast enough to maintain the frame rate it will throw warnings to the console and continues tracking at the maximum rate possible.

Two hand tracker instances should be able to run simultaneously. The two frames of the webcams are used to determine the depth of an object in a later stage of ARMI. The more the frames differ in time the less precise the depth estimation. Therefore blocking condition variables [23] are used. The second instance of the hand tracker is passed to the first instance. The first instance will now signal when the second instance should grab a frame. Thus the second hand tracker simply waits on a condition from the first. The hand tracking takes place in a separate thread, for sake of clarity the pseudo code is listed in Listing 9.

```
1  char frame [width * height]
2  char buffer[width * height]
3  char currentFrame
4
5  function mainHandTrackerThread():
6      char background[width * height]
7
8      while TRUE:
9          if not isSynchronizeThread:
10             ConditionWait(threadCanContinue)
11         startTime = GetCurrentTime()
12         frame     = GetFrameFromFrameGrabber()
13         if wait_counter++ > (1 / fps) * 5 && background == NULL:
14             for i = 0; i < width * height; i++:
15                 background[i] = ConvertToGrayscale(frame, i)
16         for i = 0, i_offset = 0; i < i_end; i++, i_offset += 4:
17             [image_h, image_s, image_v] = ConvertRGBToHSV(frame, i)
18             skin    = MFHistogram_FindValueInHistogram(
                   histogram_Skin   , image_h, image_s)
```

```
19              nonskin = MFHistogram_FindValueInHistogram(
                    histogram_NonSkin, image_h, image_s)
20
21              if background != NULL:
22                  frame[i_offset + position_alpha] = 0
23
24                  foreground_bw = ConvertToGrayscale(frame, i)
25
26                  if abs(foreground_bw - background[i]) > 50:
27                      if skin / nonskin >= threshold:
28                          frame[i_offset + position_alpha] = 255
29
30                  DoFirstPassOfConnectedComponentLabeling()
31
32                  if background_update_counter > 15:
33                      if foreground_bw > background[i]:
34                          if (background[i] + 1) < background_max:
35                              background[i] += 1;
36                      else if foreground_bw < background[i]:
37                          if (background[i] - 1) >= 0:
38                              background[i] -= 1;
39
40          if background_update_counter > 15:
41              background_update_counter  = 0
42          else:
43              background_update_counter += 1
44
45          if isSynchronizeThread:
46              delay = (1 / fps) - GetCurrentTime() - startTime
47              if delay < 0:
48                  print "Warning, frame rate requirements not met."
49              else:
50                  sleep(delay)
51          currentFrame = frame
52          frame        = buffer
53          buffer       = currentFrame
54          currentBLOBs = blobs
55          blobs        = blobsBuffer
56          blobsBuffer  = currentBLOBs
57
58          ConditionSignal(frameReady)
59
60          if isSynchronizeThread:
61              ConditionSignal(threadCanContinue)
```

Listing 9: Main hand tracker function.

Line 13 to 15 show how the background image is filled for the first time. This is done only once, but note the counter. Sometimes the webcam initializes with a completely blank screen. Thus the hand tracker waits a few frames before the first background image is set. Line 26 shows how the difference between the background frame and current frame is calculated using the absolute value. Line 26 to 28 show how the background filter and skin classifier work together. The background subtractor returns the difference between the current and previous frame, a value close to zero indicates no difference between the current and background pixel; thus it is likely the pixel is part of the background. If this difference is larger than threshold T_b the

pixel is also checked by the skin classifier. If the probability the pixel is skin is also larger than a certain skin threshold T_s it is marked skin by setting a value of 255 in the alpha layer of the current frame. Lastly, line 32 to 38 update the background frame. This is only done every 15 iterations. This way changes are slowly propagated to the background frame. When the Python module is initialized the hand tracker it only needs to call the two functions listed in Listing 10.

```
1 function HandTrackerGetImage():
2     ConditionWait(frameReady)
3     Lock(currentFrame)
4     return [currentFrame, currentBLOBs]
5
6 function HandTrackerReleaseImage()
7     UnLock(currentFrame)
```

Listing 10: Hand tracker functions used for Python module.

a call to `HandTrackerGetImage()` will block until a frame is ready, which is returned immediately. When done processing the frame should be released right away by calling `HandTrackerReleaseImage()`. ARMI will call these functions 15 times a second (15 frames per second).

5.4 PYTHON MODULE

The Python module is only a wrapper which combines everything in one easy-to-use module. It takes care of creating and cleaning up the frame grabber instances as well as the hand tracker instance. In order to maintain the performance only a pointer to the location of the frame in memory is passed. Therefore only little extra overhead is added; only a few extra function calls. The following six functions are defined for the Python module,

- `new(devicename, width, height, framerate, skinHistogram, nonSkinHistogram)`, create a new hand tracker instance.
- `start(trackerEnabled)`, start the tracker thread and begin with hand tracking. If `trackerEnabled` is set to `True` it will simply pass-through the image without tracking. Returned is the image along with an array containing the location of the hands.
- `stop()`, stop the hand tracker.
- `keepSynchronizedWith(handtrackerInstance)`, synchronize two hand tracker instances. The current instance is responsible for maintaining the pace.
- `getImage()`, this function blocks until an image is ready.
- `releaseImage()`, release the image so the tracker can use it again.

To emphasize the ease of use of the hand tracker Python Module a very simple example of how the tracker can be used in Python is shown

in Listing 11. Only a few lines of code are needed to get started and retrieve the images from the hand tracker.

```
#!/usr/bin/env python
import handtracker

if(__name__ == '__main__'):
    htinstance = handtracker.new(None, 640, 480, 15,  \
                                 "skinHistogram.txt", \
                                 "nonSkinHistogram.txt")
    if htinstance is None:
        print("Error creating hand tracker instance.")
        exit(0)

    err = htinstance.start(True)
    if(err != 0):
        print("Error while starting the hand tracker.")
        exit(0)

    while(True):
        [image, blobs] = htinstance.get_image()

        # Do custom processing on tracked hands.

        htinstance.release_image()
```

Listing 11: Simple example of how to use the hand tracker with Python.

The two histogram files are generated using a small command-line utility. Note that at line five the device name is set to `None`, this means the default webcam will be used; the iSight on Apple systems and the first plugged in webcam in Linux. Between line 18 and 22 further processing can be done using your own code.

5.5 OPTIMIZATIONS

In order to maintain a frame rate of 15 frames per second some crucial parts of the hand tracker are optimized by hand. The most noticeable optimizations are discussed in this section.

Every single frame grabbed from the webcam is in YCrYCb format and needs to be converted to RGB before further processing can take place. This function is called 15 times a second therefore any optimizations here potentially improve performance. The original YCrYCb to RGB conversion function used floating point operations which is defined in Listing 12.

```
function YCrYCbRGB(y1, Cr, y2, Cb):
    Cr = Cr - 128
    Cb = Cb - 128

    y1 = 1.164 * (y1 - 16)
    y2 = 1.164 * (y2 - 16)

    r =  1.596 * Cr
    g = (0.813 * Cr) + (0.392 * Cb)
```

```
    b = (2.017 * Cb)

    r1 = Clip(y1 + r)
    g1 = Clip(y1 - g)
    b1 = Clip(y1 + b)

    r2 = Clip(y2 + r)
    g2 = Clip(y2 - g)
    b2 = Clip(y2 + b)
    return [r1, g1, b1, r2, g2, b2]
```

Listing 12: Pseudo code of the float YCrYCb to RGB conversion.

The optimized bit shift version is defined as follows, Listing 13.

```
function YCrYCbRGB(y1, Cr, y2, Cb):
    Cr = Cr - 128
    Cb = Cb - 128

    y1 = y1 - 16
    y2 = y2 - 16

    y1 = y1 + (y1 >> 3) + (y1 >> 5) + (y1 >> 7)
    y2 = y2 + (y2 >> 3) + (y2 >> 5) + (y2 >> 7)

    r =  Cr + (Cr >> 1) + (Cr >> 4) + (Cr >> 5)
    g = (Cr >> 1) + (Cr >> 2) + (Cr >> 4) + (Cb >> 2) + (Cb >> 3) +
         (Cb >> 6)
    b =  Cb + Cb + (Cb >> 6)

    r1  = Clip(y1 + r)
    g1  = Clip(y1 - g)
    b1  = Clip(y1 + b)

    r2  = Clip(y2 + r)
    g2  = Clip(y2 - g)
    b2  = Clip(y2 + b)
    return [r1, g1, b1, r2, g2, b2]
```

Listing 13: Pseudo code of the bit shift YCrYCb to RGB conversion.

A disadvantage of the bit shift version is that some precision is lost, but that does not matter for the hand tracker. To test the difference in conversion speed a small benchmark utility is created, see Table 12 for the results, the values are the time it takes in milliseconds to convert a single frame. The integer version is per frame 0.6 milliseconds faster, which shows only little gain on a per frame basis; but remember that this conversion routine is done 15 times per second.

		Time in milliseconds
YCrYCb	Float	3.22
	Integer	2.62

Table 12: Comparison of float and integer YCrYCb to RGB conversion.

For the final version of the hand tracker Approximate Median Filtering (AMF) is used. This filter uses a background image in grayscale which is slowly updated. The original version used floating point operations, the grayscale value of a pixel is calculated as follows,

$$grayscale = 0.3 * r + 0.59 * g + 0.11 * b \tag{5.1}$$

And the optimized version,

$$\begin{aligned} grayscale &= ((0.3 * 255) * r + (0.59 * 255) * g + (0.11 * 255) * b)/8 \\ &= (77 * r + 150 * g + 28 * b) >> 8 \end{aligned} \tag{5.2}$$

The optimized version only uses one bit shift, the real optimization lies in using integers instead of floating point numbers. The grayscale image is stored in an unsigned char that ranges from zero to 255, the values are discretized.

		Time in milliseconds
Grayscale	Float	2.00
	Integer	1.59

Table 13: Comparison of float and integer RGB to grayscale conversion.

For the RGB to grayscale conversion a small benchmark utility is created, the results are shown in Table 13, the times are based on a single frame conversion. The improvement is again very small, 0.41 milliseconds, but this conversion is like the YCrYCb to RGB conversion also done at a rate of 15 frames per second.

6

CONCLUSION AND FUTURE WORK

This thesis presents an approach for real-time hand tracking using inexpensive COTS hardware; as part of a larger project called ARMI. The goal was to built an AR environment using standard hardware: a Head-Mounted-Display and several webcams. The hand tracker is the first component in the chain of processing user input. After tracking (multiple) hands the pose is estimated, the action is determined and the data is distributed.

From the three different approaches of skin segmentation tested the Bayesian approach scored best. The intersection and artificial neural network approach performed worse. The problem statement in Chapter 1 states that no calibration or fine tuning whatsoever should be required, this requirement is only partially met, the current hand tracker is far from optimal. It still requires some manual fine tuning, for example selecting the right skin threshold or update rate of the Approximate Median Filter. However the histograms used for skin segmentation are usable for all skin types.

As stated in Chapter 1 two webcams are used which try to track multiple hands independently, the webcams are positioned above a table. Due to this setup mostly the users hands are visible and other parts, the users face for example, hardly interfere. Another requirement for the hand tracker is to be able to handle cluttered and complex backgrounds, the example movie on the ARMI project website show that this also works sufficient [17]. In Chapter 3 a frame rate of minimal 15 frames per second is set; the benchmark in Chapter 4 resulted in an average of around 20 frames per second for two hands, which is sufficient. Finally, there is room for improvement, but as a proof of concept the results are acceptable; most importantly results are good enough for the next part of ARMI: the hand pose estimator [7].

Another problem was the default settings of the webcams. Most webcams set brightness, contrast, saturation and white balance to automatic. Unfortunately these controls where not always set correctly. For example disabling these controls did not always work as it should. Using the automatic controls sometimes results in an over-exposed image.

6.1 FUTURE WORK

The result of this thesis is a proof-of-concept implementation of the hand tracker. Afterwards, when most of the writing and implementation was done, potential improvements always pop up. Due to time constraints it is not always possible to implement the improvements.

The current implementation first converts a frame from YCbCr to RGB and finally, for background subtraction, to grayscale. Instead of converting a frame from YCbCr to RGB to grayscale, pass the Y directly to the background subtraction component. As the luminance Y represents the brightness or the achromatic portion of the image, the color information is stored in the Cb and Cr components. This would save the extra RGB to grayscale conversion and potentially increase tracking speed.

Currently background subtraction marks pixels as skin, in the next step skin segmentation does the same. More research needs to be done in how to combine these two components in a better way to increase the overall tracking performance. Only the skin segmentation component is evaluated, a better approach is to evaluate the complete chain, background subtraction along with skin segmentation. This would give a better approximation of how the hand tracker performs in practice.

In the current setup two webcams are placed in an angle above a table and pointing to the table surface. The current hand tracker cannot determine if skin is part of a hand or other parts of the human body, for example a human face. Due to the current setup the hand tracker does not need to know this information. This makes the setup of the system quite static. One improvement is to make the setup more dynamic, so the two webcams can be positioned freely. This can be done by explicitly detecting hands or exclude specific non hand body parts [11, 14, 45, 57]. Another possibility is to have better integration with the hand pose estimation done by Boer [7]. For example, getting feedback if a hand pose is found to create specific search regions or adapt the skin threshold. The hand tracker works on a per frame basis, information, for example about hand positions, is not propagated throughout a sequence of images. Using a non-linear estimator such as the Unscented Kalman Filter (UKF) [32] an estimate can be done on the position of the hands in the next frame.

The current implementation is single threaded, i.e., a single thread per hand tracker instance and therefore one thread per webcam. Although a frame rate of 20 frames per second is already achievable, a higher rate could be easily achieved by creating a multi threaded implementation; as in multiple threads per hand tracker instance and deployed on, for example, a quad core processor. This could for example be realized using the OpenMP framework [6], this framework is a cross platform application programming interface (API) for shared memory multiprocessing programming.

A pixel is classified as skin or non-skin, this result is stored as a binary image in the alpha layer. Instead of using a binary image another approach is to use the probability a pixel is skin, discretize the value and assign it the corresponding pixel in the alpha layer. Now by using grayscale area openings [12, 59] with a predefined threshold the image is filtered and classified. Also, the area opening step needs to be extended to store the area bounds. This eliminates the separate connected component labeling step and can yield in an increase in performance; obviously this needs to be developed in more detail first.

BIBLIOGRAPHY

[1] Subutai Ahmad. A usable real-time 3d hand tracker. In *In Proceedings 28th Asilomar Conference on Signals, Systems and Computers*, pages 1257–1261. IEEE Computer Society Press, 1995.

[2] Apple. Mac os x leopard. http://www.apple.com/macosx/specs.html; Last accessed: July 07, 2009.

[3] M. Sanjeev Arulampalam, Simon Maskell, and Neil Gordon. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50:174–188, 2002.

[4] Boaz J. Super Benjamin D. Zarit and Francis K. H. Quek. Comparison of five color models in skin pixel classification. In *In ICCV'99 Int'l Workshop on*, pages 58–63, 1999.

[5] S. Benton. Background subtraction, part 1: Matlab models. http://www.dspdesignline.com/210000460; Last accessed: October 04, 2009.

[6] OpenMP Architecture Review Board. Openmp. http://openmp.org/; Last accessed: October 05, 2009.

[7] Gijs Boer. An incremental approach to real-time hand pose estimation using the gpu. Master's thesis, University of Groningen, 2009.

[8] Matthieu Bray, Esther Koller-meier, and Luc Van Gool. Gool. smart particle filtering for 3d hand tracking. In *In AFGR04*, pages 675–680, 2004.

[9] Edmond J. Breen and Ronald Jones. Attribute openings, thinnings, and granulometries. *Comput. Vis. Image Underst.*, 64(3):377–389, 1996. ISSN 1077-3142. doi: http://dx.doi.org/10.1006/cviu.1996.0066.

[10] Pieter Bruining. A 3d interface for synchronous collaboration in distributed augmented reality environments. Master's thesis, University of Groningen, 2009.

[11] Roberto Brunelli and Tomaso Poggio. Face recognition: features versus templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(10):1042–1052, 1993.

[12] F. Cheng and A.N. Venetsanopoulos. An adaptive morphological filter for image processing. *T-IP*, 1:533–539, 1992.

[13] Sen ching S. Cheung and Chandrika Kamath. Robust techniques for background subtraction in urban traffic video. volume 5308, pages 881–892. SPIE, 2004. doi: 10.1117/12.526886. URL http://link.aip.org/link/?PSI/5308/881/1.

[14] James Davis, James Davis, Mubarak Shah, and Mubarak Shah. Gesture recognition. pages 101–106, 1994.

[15] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990. ISBN 0-201-12110-7.

[16] Adrian Ford and Alan Roberts. Colour space conversions, August 1998.

[17] M.R. Fremouw. Armi website. http://armi.fremouw.nl; Last accessed: October 16, 2009.

[18] Nir Friedman and Stuart Russell. Image segmentation in video sequences: A probabilistic approach. pages 175–181, 1997.

[19] Tek Gear. Advisor 150. http://www.tekgear.ca/index.cfm?pageID=90&prodid=355&section=83&nodelist=1,83&function=viewproducts; Last accessed: October 23, 2009.

[20] Rafael C. Gonzalez and Paul A. Wintz. *Digital Image Processing*. Addison Wesley, 2008. ISBN 9780131687288.

[21] Yujie Han and Robert A. Wagner. An efficient and fast parallel-connected component algorithm. *J. ACM*, 37(3):626–642, 1990. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/79147.214077.

[22] Robert M. Haralick and Linda G. Shapiro. *Computer and robot vision*, volume 1. Addison Wesley, 1992. ISBN 0-201-10877-1.

[23] C. A. R. Hoare. Monitors: an operating system structuring concept. *Commun. ACM*, 17(10):549–557, 1974. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/355620.361161.

[24] Michael Horvath. Hsv cone. This file is licensed under the Creative Commons Attribution ShareAlike 3.0; http://commons.wikimedia.org/wiki/File:Color_solid_comparison_hsl_hsv_rgb_cone_sphere_cube_cylinder.png; Last accessed: August 12, 2009.

[25] HowStuffWorks. Ar first down line. http://www.howstuffworks.com/first-down-line.htm; Last accessed: October 20, 2009.

[26] HunterLab. Cie l*a*b* color scale. http://www.hunterlab.com/appnotes/an07_96a.pdf; Last accessed: August 12, 2009, 2008.

[27] ARToolworks Incorporated. Artoolkit. http://www.hitl.washington.edu/artoolkit/; Last accessed: June 30, 2009.

[28] Michael Isard and Andrew Blake. Contour tracking by stochastic propagation of conditional density. pages 343–356, 1996.

[29] Michael Isard and Andrew Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28, 1998.

[30] Keith Jack. *Video Demystified: A Handbook for the Digital Engineer, 5th Edition*. Newnes, Newton, MA, USA, 2007. ISBN 0750683953, 9780750683951.

[31] Michael J. Jones and James M. Rehg. Statistical color models with application to skin detection. In *International Journal of Computer Vision*, pages 274–280, 1999.

[32] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the kalman filter to nonlinear systems. pages 182–193, 1997.

[33] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[34] Brian W. Kernighan, Dennis Ritchie, and Dennis M. Ritchie. *C Programming Language (2nd Edition)*. Prentice Hall PTR, March 1988. ISBN 0131103628.

[35] Ron Kohavi and Foster Provost. Glossary of terms. *Machine Learning*, 30(2):271–274, 1998. doi: 10.1023/A:1017181826899. URL http://www.springerlink.com/content/x105525142v51t20.

[36] Takeshi Kurata, Takekazu Kato, Masakatsu Kourogi, Jung Keechul, and Ken Endo. A functionally-distributed hand tracking method for wearable visual interfaces and its applications. In *In IAPR Workshop on Machine Vision Applications*, pages 84–89, 2002.

[37] Heino Lenting. Replicating augmented reality objects for multi-user interaction. Master's thesis, University of Groningen, 2009.

[38] Logitech. Quickcam s 7500. http://www.logitech.com/index.cfm/webcam_communications/webcams/devices/4225&cl=NZ,EN; Last accessed: June 30, 2009.

[39] N. J. B. McFarlane and C. P. Schofield. Segmentation and tracking of piglets in images. *Machine Vision and Applications*, 8(3):187–193, 1995. doi: 10.1007/BF01215814.

[40] A. Meijster, J. B. T. M. Roerdink, and W. H. Hesselink. A general algorithm for computing distance transforms in linear time, 2000.

[41] Arnold Meijster and Michael H.F. Wilkinson. A comparison of algorithms for connected set openings and closings. *IEEE Trans. Patt. Anal. Mach. Intell*, 24:484–494, 2002.

[42] T. Mitchell. *Machine Learning*. McGraw-Hill Education (ISE Editions), 1997. ISBN 0071154671.

[43] Albert Oliveras Philippe Salembier and Luis Garrido. Anti-extensive connected operators for image and sequence processing, 1998.

[44] Philips. Spc 1000nc. http://www.consumer.philips.com/consumer/en/sg/consumer/cc/_productid_SPC1000NC_00_SG_CONSUMER/Webcam+SPC1000NC-00; Last accessed: June 30, 2009.

[45] James M. Rehg and Takeo Kanade. Digiteyes: Vision-based human hand tracking. Technical report, Pittsburgh, PA, USA, 1993.

[46] F. Rosenblatt. The perceptron: probabilistic model for information storage and organization in the brain. *Psychological Review*, 65: 386–408, 1958.

[47] Azriel Rosenfeld and John L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13(4):471–494, 1966. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/321356.321357.

[48] Stephen R. Schach. *Object-Oriented and Classical Software Engineering*. McGraw-Hill Pub. Co., 2001. ISBN 0072554509.

[49] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Prentice Hall, 2001. ISBN 0-13-030796-3.

[50] Chris Stauffer and W. Eric L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:747–757, 2000.

[51] B. Stenger, P. R. S. Mendonça, and R. Cipolla. Model-based 3d tracking of an articulated hand. *cvpr*, 2:310, 2001. ISSN 1063-6919. doi: http://doi.ieeecomputersociety.org/10.1109/CVPR.2001.990976.

[52] B. Stenger, P. R. S. Mendonça, and R. Cipolla. Model-based hand tracking using an unscented kalman filter. In *In Proc. British Machine Vision Conference, volume I*, pages 63–72, 2001.

[53] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7:11–32, 1991.

[54] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/321879.321884.

[55] Emma L Tonkin and Greg Tourte. Video streaming of events. Technical Report CSTR-06-018, Ariadne Issue 49 (ISSN: 1361-3200), October 2006.

[56] Linus Torvalds. Linux uvc. http://linux-uvc.berlios.de/; Last accessed: July 07, 2009.

[57] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–86, 1991. ISSN 0898-929X. doi: http://dx.doi.org/10.1162/jocn.1991.3.1.71.

[58] Vladimir Vezhnevets, Vassili Sazonov, and Alla Andreeva. A survey on pixel-based skin color detection techniques. In *in Proc. Graphicon-2003*, pages 85–92, 2003.

[59] Luc Vincent. Morphological area openings and closings for greyscale images. In *Proc. of the Workshop Shape in Picture*, pages 197–208. Springer, 1992.

[60] Vuzix. iwear vr920. http://store.vuzix.co.uk/euro/acatalog/iWear_DV920.html; Last accessed: June 30, 2009.

[61] Xiaoming Yin, Dong Guo, and Ming Xie. Hand image segmentation using color and rce neural network. *Robotics and Autonomous Systems*, 34(4):235 – 250, 2001. ISSN 0921-8890. doi: 10.1016/S0921-8890(00)00125-1. URL http://www.sciencedirect.com/science/article/B6V16-42M1JBB-4/2/24de90c6741d11038af5299cb157ee1f.