

JACOBS UNIVERSITY BREMEN

# Gesture-based Human-computer Interaction

by

Alen Stojanov

A thesis submitted in partial fulfillment for the  
degree of Bachelor of Science  
supervised by Prof. Dr. Lars Linsen

in the  
Electrical Engineering and Computer Science  
School of Engineering and Science (SES)

May 2009

# Declaration of Authorship

I, ALEN STOJANOV, declare that this thesis titled, 'GESTURE-BASED HUMAN-COMPUTER INTERACTION' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at Jacobs University Bremen.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

---

Date:

---

*“The hard work will always find its way to return the favor.”*

Timko Stojanov, my father.

JACOBS UNIVERSITY BREMEN

## *Abstract*

Electrical Engineering and Computer Science  
School of Engineering and Science (SES)

Bachelor of Science

by [Alen Stojanov](#)

In daily life, human beings communicate with each other and use broad range of gestures in the process of interaction. Apart of the interpersonal communication, many hours are spent in the interaction with the electronic devices. In the last decade, new classes of devices for accessing information have emerged along with increased connectivity. In parallel to the proliferation of these devices, new interaction styles have been explored. One major problem is the restrictive human machine interface like gloves, magnetic trackers, and head mounted displays. Such interfaces with excessive connections are hampering the interactivity. This paper proposes a method to develop a robust computer vision module to capture and recognize the human gestures using a regular video camera and use the full potential of these gestures in a user friendly graphical environment.

## *Acknowledgements*

The author wishes to express his gratitude to his supervisor, Prof. Dr. Lars Linsen who was abundantly helpful and offered invaluable assistance, support and guidance. This research project would not have been possible without his support.

The author wishes to express his love and gratitude to his beloved parents Gordana and Timko Stojanovi; for their encouraging power, understanding and endless love, through the duration of his studies.

Deepest gratitude to Prof. Dr. Heinrich Stamerjohanns, his knowledge, advices and discussions invaluable shaped the author's course of work and study.

Special thanks to Jana Giceva, her insights and assistance had a great contribution towards the author's admission in Jacobs University Bremen.

# Contents

<b>Declaration of Authorship</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>List of Figures</b>	<b>vi</b>
<b>Introduction</b>	<b>1</b>
<b>Statement and Motivation of Research</b>	<b>2</b>
<b>Related Work</b>	<b>4</b>
3.1 Getting started . . . . .	4
3.1.1 OpenGL and Qt . . . . .	4
3.1.2 V4L . . . . .	5
3.1.3 V4L2 . . . . .	6
3.2 Application design and dataflow . . . . .	6
3.3 Skin Color Segmentation . . . . .	8
3.3.1 Skin profile creator . . . . .	10
3.3.2 Real-time skin segmentation . . . . .	11
3.4 Feature recognition . . . . .	12
3.5 Gesture tracking . . . . .	13
<b>Discussion and future work</b>	<b>15</b>
<b>A Supporting Algorithms</b>	<b>17</b>
A.1 Flood-fill algorithm . . . . .	17
A.2 Midpoint circle algorithm . . . . .	18
<b>Bibliography</b>	<b>19</b>

# List of Figures

3.1	Design and dataflow in GMove . . . . .	7
3.2	Daylight: 3D Histogram of skin pixel frequencies in HSI color space . . . .	9
3.3	Artificial light: 3D Histogram of skin pixel frequencies in HSI color space .	9
3.4	Skin profile creator . . . . .	10
3.5	Skin color segmentation . . . . .	12
3.6	Distinct regions and bounding box . . . . .	12
3.7	Hand state using concentric circles . . . . .	13
4.8	GMove execution . . . . .	15

# Introduction

Human-computer interaction arose as a field from intertwined roots in computer graphics, operating systems, human factors, ergonomics, industrial engineering, cognitive psychology, and the systems part of computer science. Computer graphics was born from the use of cathode-ray tube (CRT) displays and pen devices very early in the history of computers. The pursuit to create man-machine symbiosis [1] led to the development of many human-computer interaction techniques, starting from primitive text and graphical user interfaces (GUI) to speech and gesture recognition interfaces.

The gesture recognition techniques can be classified into two groups: glove-based and vision-based [2]. The former group of approaches requires users to wear data or color gloves. The glove enables the system to avoid or simplify the segmentation and tracking task. However, its disadvantages are apparent. On the one hand, users have to carry a hardware device, which makes them feel uncomfortable. In comparison, the vision-based methods rely on computer vision techniques without needing any gloves, which is more natural for users. However, one difficulty is how to accurately segment and track hands and face.

An advantage of gesture is the ability to provide multidimensional input. While most pointing device provide position and orientation information for a single point in space, a gesture interface can input many positions simultaneously since the system tracks multiple features. The aim of the proposed work in this document is to demonstrate this ability, by developing an environment where the gestures are segmented, tracked and recognized. In addition the environment will be able to recognize the position of the palm and each fingertip, generate data to represent the trajectory, acceleration and velocity, as well as the feature of the hand and translate this data on a display, allowing control of a GUI.

# Statement and Motivation of Research

The introduction of the project suggests that there are two main aspects to it gesture recognition system and visualization of the results. Generally, a gesture recognition system should contain three major modules: skin segmentation and tracking, feature extraction, and recognition. The first module is to acquire and locate hands and face across the video frames. The purpose of the second module is to prepare useful features for classification.

Skin color-based segmentation methods are numerous and habitually used as an initial step in face detection applications. The simplicity and low computational cost makes the color-based approach an excellent choice for producing an initial segmentation mask of the picture. After the skin color mask is being produced and all the noise of the pictures has been removed, an edge detection algorithm can be applied, leading to a set of connected curves that indicate the boundaries of skin objects, in this case the hands detected on the video input. Computing the curves of the hands helps in finding the bounding box of the skin objects, tracking their position and occlusions.

According to the means of capturing features, two overall categories are present: Model-based and appearance-based. In a model-based system a geometric model of the hand is created. This model is matched to the (preprocessed) image data in order to determine the state of the tracked hand. In appearance based approaches the recognition is based on a pixel representation learned from training images. However, this also means that differentiating between gestures is not as straight forward as with the model based approach. The gesture recognition will therefore typically involve a (statistical) classifier based on a set of features that represent the hand.

The real-time performance and low computational complexity makes the model based system more appropriate and more precise method in the feature recognition. Therefore a model based system is proposed to count the outstretched fingers using concentrical circles. Similar to a radar system, starting from the center of the palm, the position of

the fingers and the fingertips can be detected and an adequate model of the hand can be built.

Having recognized the different user gestures, a proper environment is needed where the full potential of the gestures is used. Cross-platform APIs as OpenGL and Qt can be used to generate a proper GUI to interpret the gestures. The first step is to implement a functionality of the gestures up to a standard pointing device such as mouse and then extend this functionality to an advanced user interface such as the Zooming User Interface [3].

# Related Work

## 3.1 Getting started

Today's software systems are getting more complex than ever. The complexity comes from difficult functional requirements, and from demanding non-functional requirements, such as very high availability and fast performances. GMove is an interactive system, able to interpret the human hand gestures and apply an appropriate action in a given 3D environment. Although a very small software system, it is facing the same challenges as the modern systems. It is implemented in C++ using additional APIs and frameworks as OpenGL, Qt, V4L and V4L2. The system is fully cross platform, except the module required to obtain the video stream from the video input driver.

### 3.1.1 OpenGL and Qt

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. OpenGL routines simplify the development of graphics software from rendering a simple geometric point, line, or filled polygon to the creation of the most complex lighted and texture-mapped curved surface. GMove uses small amount of the advantages of OpenGL, to present the functionality of the process of gesture tracking.

Qt is a cross-platform application development framework, widely used for the development of GUI programs and also used for developing non-GUI programs such as console tools and servers. Except the basic Qt, two other modules are used, namely: Qthread and QtOpenGL.

A QThread represents a separate thread of control within the program; it shares data with all the other threads within the process but executes independently in the way that a separate program does on a multitasking operating system. This makes it really easy to use and integrate it with the rest of the Qt GUI. The implementation of mutexes and

conditional variables makes it very suitable for synchronizing a highly multi-threaded application.

QtOpenGL provides an OpenGL widget class that can be used just like any other Qt widget, except that it opens an OpenGL display buffer where the OpenGL API can be used to render the contents. The widget ensures a very suitable integration of our 3D environment with our gesture based control interface, as well as with the GUI window.

### 3.1.2 V4L

Video4Linux or V4L is a video capture application programming interface for Linux. It is closely integrated into the Linux kernel and provides support for many USB (Universal Serial Bus) web cameras. Although V4L is currently in its second version and a compatibility mode is included, the support in the newer version is still incomplete and a separate implementation is required for the two modes. As mentioned before, the implementation of the V4L driver is inside the kernel space, therefore the communication with the video device is based on passing different `ioctl`'s. The communication between the user-space process and the kernel device driver is given by the following sequence of commands:

1. Open the video device (`/dev/video*`)
2. Check for the current device capabilities by passing `VIDIOCGETCAP`
3. If the device is able to capture a video stream (defined by the `VID_TYPE_CAPTURE` flag), get the memory map information passing `VIDIOCGETMBUF`
4. Map the device memory to a frame buffer using `mmap`
5. Set the image format and image geometry
6. The device is initialized for capturing. The continuous stream capture is given by:

```
VIDIOCMCAPTURE(0)
while ( capture_enabled == 1 ) {
    VIDIOCMCAPTURE(1)
    VIDIOCSYNC(0)
    /* process frame 0 while the hardware captures frame 1 */
    VIDIOCMCAPTURE(0)
    VIDIOCSYNC(1)
    /* process frame 1 while the hardware captures frame 0 */
}
```

7. Unmmap the frame buffer using `munmap`
8. Close the device file descriptor

### 3.1.3 V4L2

The Video4Linux2 standard, as a second version has a lot of implementation improvements, especially in the stability, as well as extended operations and increased control of the hardware. The communication is still defined via different `ioctl`'s, with a slight difference in the naming of the requests. The newer version has a better documentation and hundreds of lines of sample code. The basic concepts to acquire video stream from an input device, applicable to all devices, are the following:

1. Open the video device (`/dev/video*`) and check for capabilities (passing `VIDIOC_QUERYCAP`)
2. Change device properties, select a video and audio input, video standard, picture brightness (these are special `ioctl`'s, not used in the implementation of GMove)
3. Negotiate a data format (using `VIDIOC_S_FMT`)
4. Negotiate an input/output method (one of `O_METHOD_READ`, `IO_METHOD_MMAP` or `IO_METHOD_USERPTR`)
5. Start the actual input/output loop (initialized with `VIDIOC_STREAMON` and using `VIDIOC_DQBUF` to exchange buffers with the driver)
6. Close the device

In practice most steps are optional and can be executed out of order. It depends on the V4L2 device type.

## 3.2 Application design and dataflow

In 3D computer graphics, the terms graphics pipeline or rendering pipeline most commonly refer to the current method of rasterization-based rendering as supported by commodity graphics hardware. The OpenGL architecture is structured as a state-based pipeline. A similar approach is used in the implementation of GMove, such that a rasterized image is being vectorized with the purpose of extracting the semantic of the image. The threaded pipeline of the GMove's dataflow is given on Figure 3.1.

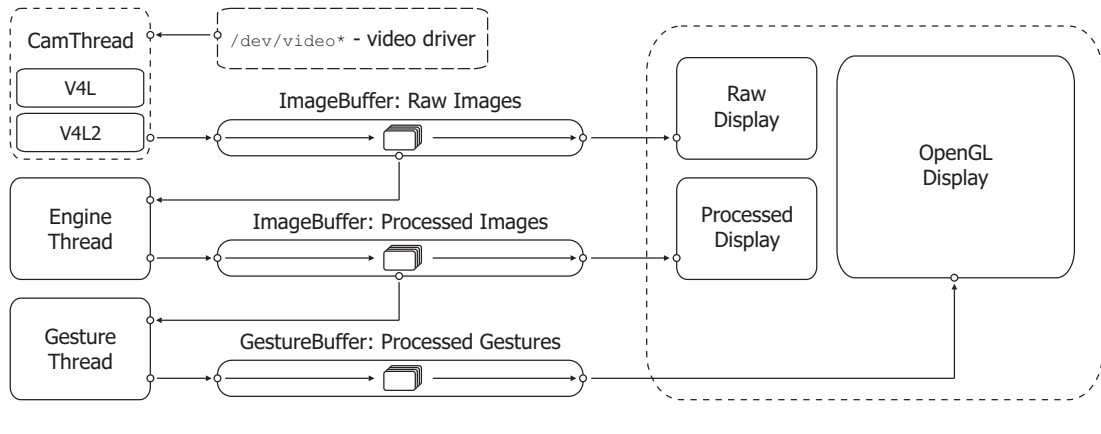


FIGURE 3.1: Design and dataflow in GMove

The processing time, in a system able to understand the human gestures, is essential. Today's standard video input devices (such as web-cams) usually have the frequency of 30 frames per second (fps) with resolution of 320 x 240 pixels for a single frame. This suggests that about 80 000 pixels should be processed for one image or about 2 million pixels in one second. Even if the whole process of extracting the gesture has a linear complexity, the design of the program should be chosen carefully to synchronize the gestures with the input stream. To avoid delayed gestures and buffer overflows in the system, a highly threaded approach has been used. Namely every module of the GMove is working in a separate thread. The data share between the threads is stored in special synchronized buffers, and the command control is done via the callback functions provided by Qt. The system is consisted of 3 threads:

- **CamThread** – responsible for reading the video input stream directly from the driver and provides a general image format in RGB.
- **EngineThread** – the most important module of GMove. This module receives the raw image input from the camera, processes the image and parses the position of the hands and their state.
- **GestureThread** – in this thread, the hand gestures obtained from the **EngineThread** are used in a process where the transitions from the previous image, stored in the gesture buffer, are mapped to the new image frame and an appropriate action is being invoked accordingly.

A significant increase of the performance is achieved when C++ memory pointers are used instead of actual image frame passing in the memory. To achieve this 3 buffers are used

- `RawImage` – stores the input stream obtained in the `CamThread`.
- `ProcessedImage` – stores the processed output generated from the `EngineThread`
- `GestureBuffer` – stores the gestures provided by the `GestureThread`

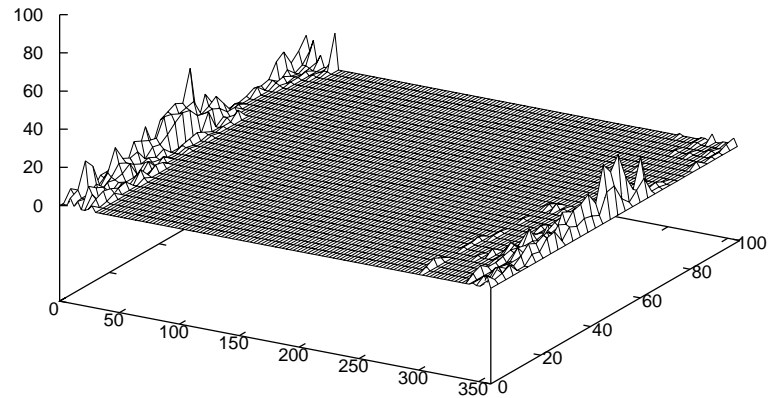
Finally, the GMove GUI is consisted of 3 different widgets:

- `RawDisplay` – where the raw capture video input is shown on the screen
- `ProcessedDisplay`– where the processed image is stored with the semantic data as weighted center of the hand, bounding box and state of the hand is shown grafically.
- `OpenGLDisplay` – a simple 3D scene is shown in this part such that the scene camera is moving accoring the hand gestures extracted from the `GestureThread`

### 3.3 Skin Color Segmentation

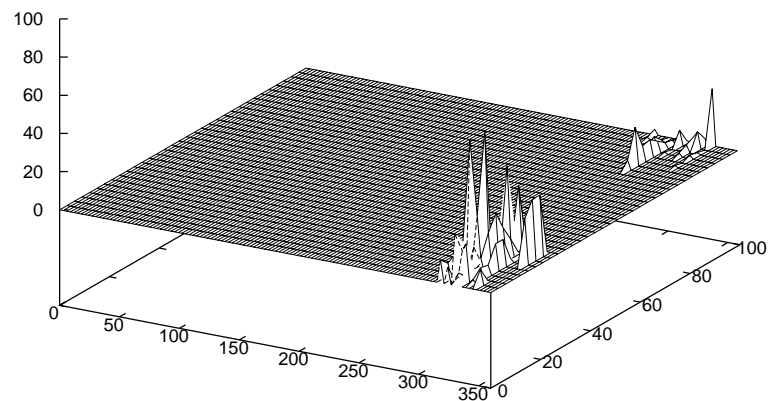
The RGB (Red, Gree and Blue) color space is one of the most widely used color space for digital image representation. However, the RGB color space alone is not reliable for identifying skin-colored pixels since it represents not only color but also luminance. On the other hand, a lot of research activities being based on it and different heuristic rules for classification are developed. On the other hand, HSV (Hue, Saturation, and Value) and HSI (Hue, Saturation and Intensity) color space models emulate the colors in the way that humans perceive them; therefore the luminance factor in the skin determination can be minimized.

Most of the low cost video input devices operate using either Charged Coupled Devices (CCD) or Complementary Metal Oxide Semiconductor (CMOS) as sensors. Both CCD and CMOS sensors follow the similar methods for converting the light intensity into an electric charge. Three sensors are used on each pixel, corresponding to three RGB colors and the image is rendered using the electrical charge values of those sensors. Since this sensors use the light intensity to represent a single frame, there are major differences in the operation of these small devices during daylight and during artificial light, especially in the exposure time. The prolonged exposure time, during artificial light, is a source of image noise and therefore many methods have been developed to minimize this noise [4], including aperture photometry and dark frame subtraction. Many of these methods are already included in the hardware implementation of the input devices, which makes the process of skin segmentation highly erroneous, if a general skin color classifier is



---

FIGURE 3.2: Daylight: 3D Histogram of skin pixel frequencies in HSI color space



---

FIGURE 3.3: Artificial light: 3D Histogram of skin pixel frequencies in HSI color space

being used. To avoid this problem, a “skin profile generator” is being implemented in the program. The skin profile generator finds the skin pixels, selects the most frequent pixels and use them, with a certain threshold, to extract the skin color in real time. Figure 3.2 represents the histogram generated by the skin profile creator according the

skin pixels of an image of one person during daylight and Figure 3.3 represents the histogram generated by selecting the skin pixels on the same person, but on image, captured using artificial light. The differences in the obtained results are obvious. The perception of the video input device is quite different during daylight and artificial light.

### 3.3.1 Skin profile creator

The user profile represents a single file filled with color values, representing the user's skin color with the highest probability. The skin profile is created manually, upon initialization of the program. The video input stream can be stopped (or paused) and the skin pixels can be selected with the mouse, selecting small rectangular pieces of the image where the skin pixel are located. A complete skin segmentation is shown in Figure 3.4.

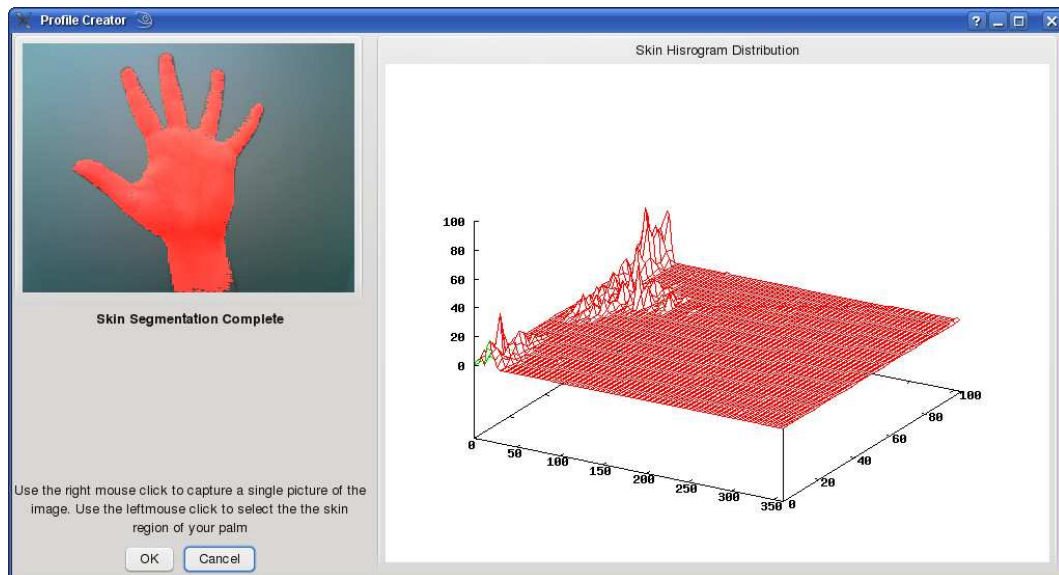


FIGURE 3.4: Skin profile creator

When the user clicks on a certain part of the acquired image, a flood fill algorithm (Appendix A.1) is used to traverse through the skin pixels with a very low threshold in the color pixel difference, to ensure that only similar colors are selected. The skin pixels are converted to HSI color space and the saturation value is being ignored, since this parameter has a low influence on the skin detection[5]. On the right side of the window, a skin histogram[6] is being generated. This histogram represents the frequencies of the pixels with matching S-value (the X-axis denotes the H-value, the Y-axis denotes the I-value and the Z-axis denotes the frequency of the skin having the particular H and I values). When the manual process of skin segmentation is complete, the frequencies of the skin pixels are sorted, and the top 20 frequent pixels are being written in the user

profile. The skin segmentation is the first process of detecting the possible presence of a hand by the video input. This process of finding the skin pixels should be quite fast and therefore using only the most frequent skin pixels for comparison, with a higher threshold, could speed up significantly this process.

### 3.3.2 Real-time skin segmentation

The real-time skin segmentation is a brute-force method, applied to all the pixel of the captured image frame. The pixels are then compared to the 20 values stored in the user profile. If the pixels are in the range defined by the threshold, then the pixel is classified as a skin, otherwise it is ignored. With this implementation a simple binary mask is being generated of skin and non-skin pixels. The issues resulting from the noise generated by the hardware input device will cause many “blank” pixels inside skin region. To fill this gaps, a Gaussian filter is used on the binary mask. In  $R^2$ , Gaussian function has the form:

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

Since the image is stored as a collection of discrete pixels, a discrete approximation to the Gaussian function should be performed before the convolution is applied. Once a suitable mask has been calculated, the Gaussian smoothing can be performed using standard convolution methods. A convolution mask is usually much smaller than the actual image. For example a convolution mask with  $\sigma = 1.4$  and mask width  $k = 4$  has the following form:

$$M(\sigma, k) = \frac{1}{115} \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

The larger the width of the Gaussian mask, the lower is the sensitivity to noise. Finally, the mask is slid over the image, manipulating a square of pixels at a time. The results of this process are shown in Figure 3.5.

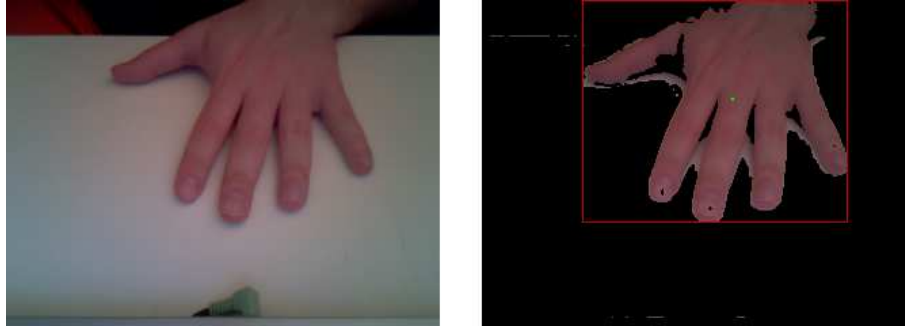


FIGURE 3.5: Skin color segmentation

### 3.4 Feature recognition

GMove allows two or more hands to be captured by the video stream. The feature recognition can be applied to one hand only, and a method for identifying the the distinct hands is required. The skin mask generated by the skin segmentation process gives a binary mask of skin/non-skin pixels. Since one hand represents a continuous region of skin pixels, a flood fill algorithm (Appendix A.1) is applied on the binary mask and the different regions are calculated.

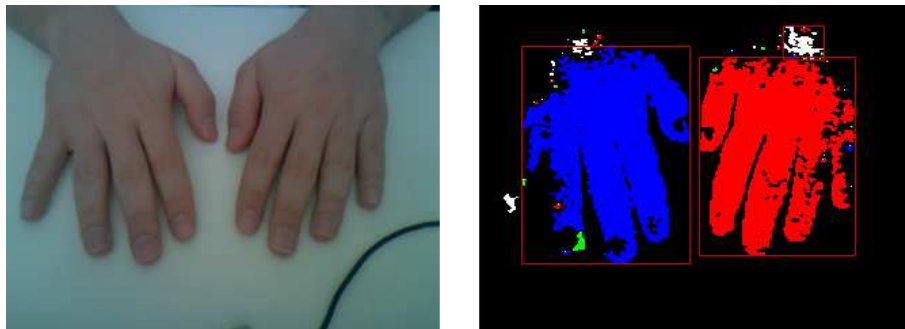


FIGURE 3.6: Distinct regions and bounding box

The flood-fill algorithm (Appendix A.1) is able to determine the minimal and maximal coordinates of the regions and those coordinates form the bounding box of the region. The bounding boxes with low number of pixels are ignored, and also the regions with hight to width ratio which does not correspond to the human hands are ignored. The center of the bounding box is also calculated. If there is intersection between two bounding boxes, such that the center of one of the boxes lies inside the other bounding box, the boxes are joint into one larger bounding box.

The hands can have any orientation in the video input stream and the method to determine the state of the hand should be robust towards all cases. Many classifiers are

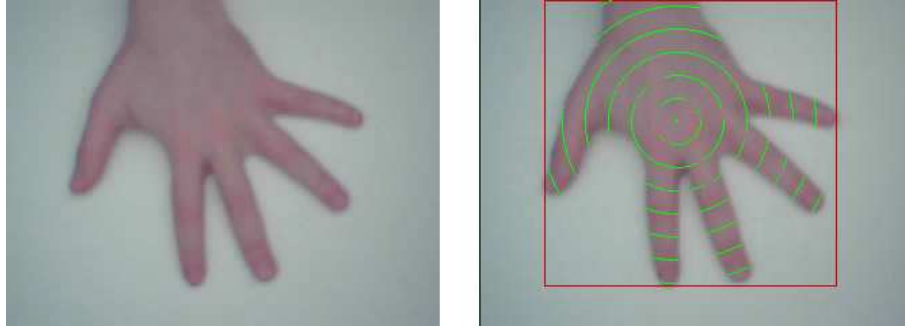


FIGURE 3.7: Hand state using concentric circles

implemented such that a certain training data is given and then the acquired image is being compared towards the predefined training data. This approach is quite effective, when only one orientation of the hand is being analyzed. To handle the many cases of the hand orientations, a geometric model of the hand is being generated. The geometrical model is derived by analyzing the skin pixels intersections when concentric circles are drawn in each of the bounding boxes. The center of the concentric circles should lie in the center of the palm; therefore it is sufficient to determine the average of all the skin pixels in the individual bounding box, ensuring that the center point lies within the hand skin pixel region. Then, the concentric circles are drawn and the intersection points with the edges are determined. The far most intersection points define the fingertips of the hand.

The representation of the image frame is still given as a matrix of pixels. Therefore the intersections of the concentric circles should be analyzed on pixel level, requiring a rasterized circle representation. Since the intersections of the concentric circles should work in real time, a midpoint circle algorithm (Appendix A.2) is used to draw the circle. The method is based on the Bresenham's line algorithm and works in linear time.

### 3.5 Gesture tracking

The complete computer to human communication requires also a gesture tracking module. The recognized feature will only denote the state of the hand, and the hand movement can denote the amount of action that is influenced to the current state. In the current implementation of the feature extraction, the system is able to differentiate two hand states:

- Fist (the palm has all the fingers “inside”) which denotes the idle mode
- The fingers of the palm are stretched outside which denotes the active mode

The feature recognition will only provide informations concerning the hand state, the bounding box and the weighted center of the hand for a single frame of the video input. However, no information is available about the transition from one frame to the other. The transition between one frame to the other can be defined with simple links between the bounding boxes, defining the hands. The challenge in this module is to provide accurate transitions between the bounding boxes in the consecutive frames. The limitation of the current standard video input devices force them to operate with a frequency of 30 fps, which can be erroneous process, especially if fast movements are performed.

The transition from one frame to another suggest that in most cases, sudden change would not be present in the image, meaning that the new frame will probably contain the same amount of bounding boxes as the previous, the distance of these bounding boxes will be relatively low and the size of the bounding box will not change much. The gesture buffer has a constant memory of several gestures. Initially, this gesture buffer is filled with the data provided from the feature recognition process. For every next gesture obtained, a match is being performed. The matching is defined with the following criteria:

- Distance between the weighted centers of two bounding boxes
- The difference of the amount of skin pixels between the bounding boxes
- The difference of the size of the bounding box

If the last gesture stored in the gesture buffer has  $N$  bounding boxes and the new gesture contains  $M$  boxes, a matrix with  $N \times M$  elements is initialized. The evaluation algorithm discussed above is performed, filling the matrix with the result of matching every two bounding boxes. The two lowest values are then selected and a new gesture is being performed.

# Discussion and future work

Developing a system such as GMove is a challenging task, especially for all the technical implementations required to get started and create a basic environment where the logic of the program is applied. Currently, the system is able to support all the input devices, compatible to either V4L or V4L2 driver standards able to provide a continuous video stream. Relatively low amount of work is required to port the program to any other operating system supporting OpenGL and Qt.

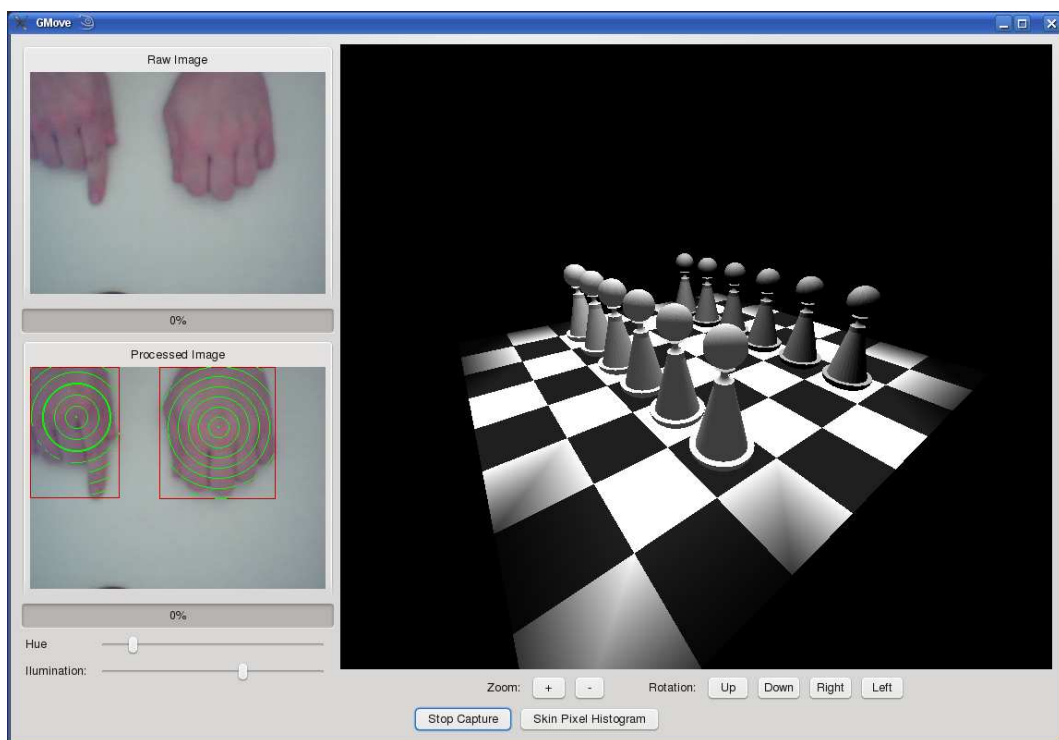


FIGURE 4.8: GMove execution

GMove is able to understand basic hand gestures and is able to track this gestures in real-time. The whole process of computer to human interaction is implemented in almost linear time, therefore very good performance is expected on the standard computers today. The methods for skin color segmentation and feature extraction are robust towards

different video input devices as well as different skin color or race. The methods for noise removal and the threaded design promises a relatively stable environment.

Although the application has already a handful of nifty features to offer, it is by no means a full-edged system able to be used in the daily human to computer communication. There are only limited number of gestures supported in the system with action attached to those gestures. The program operates in its limited environment and the features are currently available to that environment only. One ideal extension to this system would be writing a small implementation of a kernel driver for an additional input devices (maybe a module which is able to control the mouse). The system is relatively precise, but it is limited to a 2D gestures, since it has only the video input device as a perceptual module and does not have any notion of the 3D. One idea of further development is extending the system to use two or more cameras, such that a stereoscopic view is created and a real 3D model of the hand is being created from the input.

The ultimate challenge of this project would be a design of a very intuitive human interaction system, available in the futuristic science fiction movies such as “Minority Report”. My personal belief is that this kind of a system design is not far from reality. Following the example from the movie, a great feature to this system would be adding a module, able to track the human eyes and determine the human gaze [7]. As humans, we tend to point our gestures towards the person that we communicate or the machine that we’re working on. The gaze recognition system will be able to fully determine the field of interest and all the actions can be concentrated on that particular field of the screen or output device.

# Appendix A

## Supporting Algorithms

### A.1 Flood-fill algorithm

Flood fill, also called seed fill, is an algorithm that determines the area connected to a given node in a multi-dimensional array. It is used in the magic wand tool of paint programs to determine which parts of a bitmap to fill with color. It is used as a “magic wand” tool in many painting program, to select and color a continuous area of pixels with similar color. A pseudo code of the algorithm is given below.

---

**Algorithm 1** Flood-fill – starting point  $(x, y)$

---

```
initialize_queue()
enqueue(x, y)
while queue_non_empty do
  dequeue(x, y)
  if visited[x, y] ≠ true then
    if traverse_cond(x, y) then
      visited[x, y] ← true
      enqueue(x, y + 1)
      enqueue(x, y - 1)
      enqueue(x + 1, y)
      enqueue(x - 1, y)
    end if
  end if
end while
```

---

The traverse condition specified in the algorithm above ensures that a transition from the initial position is possible to the position with coordinates  $x$  and  $y$ . The condition can check for color similarity, ensure that the values of the coordinates do not exceed the bounds of the image frame etc.

## A.2 Midpoint circle algorithm

The algorithm starts accordingly with the circle equation  $x^2 + y^2 = r^2$ . Only the first octant is considered and the algorithm starts to draw a curve which starts at point  $(r, 0)$  and proceeds upwards and to the left, reaching the angle of 45 degrees.

---

**Algorithm 2** Midpoint-Circle – center point  $(x_0, y_0)$  and *radius*

---

```

f ← (1 − radius)
ddFx ← 1
ddFy ← (−2 · radius)
x ← 0
y ← radius
setPixel(x0, y0 + radius)
setPixel(x0, y0 − radius)
setPixel(x0 + radius, y0)
setPixel(x0 − radius, y0)
while x < y do
  if f ≥ 0 then
    y ← (y − 1)
    ddFy ← (ddFy + 2)
    f ← (f + ddFy)
  end if
  x ← (x + 1)
  ddFx ← (ddFx + 2)
  enqueue(x − 1, y)
  f ← (f + ddFx)
  setPixel(x0 + x, y0 + y)
  setPixel(x0 − x, y0 + y)
  setPixel(x0 + x, y0 − y)
  setPixel(x0 − x, y0 − y)
  setPixel(x0 + y, y0 + x)
  setPixel(x0 − y, y0 + x)
  setPixel(x0 + y, y0 − x)
  setPixel(x0 − y, y0 − x)
end while

```

---

# Bibliography

- [1] J.C.R. Licklider. Man-Computer Symbiosis. In *IRE Transactions on Human Factors in Electronics*, pages 4–11, 1960.
- [2] George Awad and Junwei Han and Alistair Sutherland. A Unified System for Segmentation and Tracking of Face and Hands in Sign Language Recognition. In *ICPR (1)*, pages 239–242, 2006.
- [3] Chris Harrison and Anind K. Dey. Lean and zoom: proximity-aware user interface and content magnification. In *CHI*, pages 507–510, 2008.
- [4] Grzegorz Wrochna. CCD image enhancement techniques for high noise devices. 2003.
- [5] Guo Yu Kong Li-tao Chang Xiao-qi Shan Xiao-you Lu Yi-xing, Liu Ying. Features of human skin in hsv color space and new recognition parameter. *Optoelectronics Letters*, 3(4), 2007.
- [6] Michael J. Jones and James M. Rehg. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46(1):81–96, 2002.
- [7] Rowel Atienza and Alexander Zelinsky. Intuitive human-robot interaction through active 3d gaze tracking. In *ISRR*, pages 172–181, 2003.