Tool for the Analysis of Human Interaction with Two-Dimensional Printed Imagery

Anjali K. Jogeshwar
akj5177@rit.edu

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
M.S. DEGREE THESIS

The M.S. Degree Thesis of Anjali K. Jogeshwar has been examined and approved by the thesis committee as satisfactory for the thesis requirement for the Master of Science Degree.

Dr. Jeff Pelz (Advisor)

Dr. Gabriel Diaz

Dr. Guoyu Lu

Date
This thesis is dedicated to

my parents (Ritu Jogeshwar and Kishore Jogeshwar),

and brother (Bhaavin Jogeshwar)

for standing by my side while I quenched my thirst for knowledge.
Abstract

The study of human vision can include our interaction with objects. These studies include behavior modeling, understanding visual attention and motor guidance, and enhancing user experiences. But all these studies have one thing in common. To analyze the data in detail, researchers typically have to analyze video data frame by frame. Real world interaction data often comprises of data from both eye and hand. Analyzing such data frame by frame can get very tedious and time-consuming. A calibrated scene video from an eye-tracker captured at 120 Hz for 3 minutes has over 21,000 frames to be analyzed.

Automating the process is crucial to allow interaction research to proceed. Research in object recognition over the last decade now allows eye-movement data to be analyzed automatically to determine what a subject is looking at and for how long. I will describe my research in which I developed a pipeline to help researchers analyze interaction data including eye and hand. Inspired by a semi-automated pipeline for analyzing eye tracking data, I have created a pipeline for analyzing hand grasp along with gaze. Putting both pipelines together can help researchers analyze interaction data.

The hand-grasp pipeline detects skin to locate the hands, then determines what object (if any) the hand is over, and where the thumbs/fingers occluded that object. I also compare identification with recognition throughout the pipeline. The current pipeline operates on independent frames; future work will extend the pipeline to take advantage of the dynamics of natural interactions.
Acknowledgements

I would like express my heartfelt gratitude to Dr. Jeff Pelz for giving me this amazing opportunity to work with him and on this project. As a mentor, he has taught me something new everyday. It has been an absolute honor to learn and grow in his supervision. He has guided me towards becoming a better scientist.

I am indebted to Dr. Gabriel Diaz for teaching me a lot about vision science and motor guidance, Dr. Guoyu Lu for teaching imaging from various perspectives, and friends in the Chester F. Carlson Center for Imaging Science (CIS) for all their support and encouragement.

I am grateful to Melanie Warren for always being there whenever I needed her and making sure I am safe at all times come what may, Joe Pow and the staff in CIS for being a family away from home.

Special thanks to Nuzhet for taking care of me and boosting me with positivity since the very start of my journey, and Aayush for all the helpful brain storming sessions and support throughout.

Lastly, I would like to thank my family in India. If it weren’t for their love, encouragement and support, this journey (12000 kms away from home) wouldn’t have been possible.
Contents

Abstract iv

Acknowledgements v

Contents vi

List of Figures ix

List of Tables xiii

Glossary xiv

1 Introduction 1

1.1 Per chapter synopsis .................................................. 5

2 Literature Review 10

2.1 Eye tracking and data analysis ........................................ 10

2.1.1 Eye Tracking ......................................................... 10

2.1.2 Automation in data analysis ...................................... 12

2.2 Hand detection .......................................................... 14

2.3 Summary .................................................................. 16
CONTENTS

3 Key Concepts 21

3.1 Color Spaces ................................................................. 21
3.2 Morphological Operations and Structuring Element ................. 22
3.3 Scale Invariant Feature Transform (SIFT) ............................ 25
3.4 Random Sample Consensus (RANSAC) ................................ 28
3.5 Homography ................................................................. 29
3.6 Artificial Neural Network (ANN) .......................................... 36
3.7 Summary ................................................................. 37

4 Data Collection 40

4.1 Lighting ............................................................................. 40
4.2 Apparatus .......................................................................... 41
4.3 Eye Model ........................................................................... 44
4.4 Calibration ........................................................................... 45
4.5 Center Offsets ................................................................. 48
4.6 Task: Mock Transaction .................................................... 49
4.7 Software .............................................................................. 50
4.8 Summary ................................................................. 51

5 Hand-Grasp Pipeline 53

5.1 Skin detection .................................................................... 54
5.1.1 HSV color space selection ............................................. 54
5.1.2 Use of CIE-Lab color space and Dynamic Area Creation (DAC) ........................................... 56
5.2 Hand localization ............................................................... 63
5.2.1 Hand-mask generation .................................................. 63
5.2.2 Hand-mask processing .................................................. 71
5.3 Object identification .......................................................... 75
List of Figures

1.1 Point-of-regard (POR) in the video frame ........................................ 2
1.2 Eye-Gaze Pipeline output ................................................................. 4
1.3 Hand-Grasp Pipeline output .............................................................. 5
1.4 Hand-Grasp Pipeline output split in parts .......................................... 6

3.1 Color spaces .................................................................................. 22
3.2 Various morphological operations ..................................................... 24
3.3 SIFT features marked with a circle on a part of $10 bill and on a part of rotated 10$ bill .......................................................... 26
3.4 Image gradients and key-point descriptor from Lowe [2004](Figure 7) ........ 27
3.5 Model fitting with RANSAC .............................................................. 29
3.6 Homography concept ..................................................................... 30
3.7 Key-point matching ......................................................................... 34

4.1 Lighting in the experiment room ....................................................... 41
4.2 Pupil Labs eye tracker used in the experiment .................................. 42
4.3 Scenes from the eye tracker ............................................................. 42
4.4 Other apparatus in the experiment room ........................................... 43
4.5 Pupil Player software ...................................................................... 44
4.6 Capturing all the views of the eyes .................................................. 45
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7</td>
<td>Various calibration boards used throughout the data collection</td>
</tr>
<tr>
<td>4.8</td>
<td>Participant indicating a fixation with button click during cash-drawer calibration</td>
</tr>
<tr>
<td>4.9</td>
<td>Exchange calibration</td>
</tr>
<tr>
<td>4.10</td>
<td>Exchange calibration from a perspective</td>
</tr>
<tr>
<td>4.11</td>
<td>Center offsets with gaze overlaid as seen from the scene camera during analysis</td>
</tr>
<tr>
<td>4.12</td>
<td>Tasks as seen from the eye trackers</td>
</tr>
<tr>
<td>5.1</td>
<td>HSV thresholding using the trackbar</td>
</tr>
<tr>
<td>5.2</td>
<td>DAC output without filtering</td>
</tr>
<tr>
<td>5.3</td>
<td>DAC workflow</td>
</tr>
<tr>
<td>5.4</td>
<td>Mask generation in DAC</td>
</tr>
<tr>
<td>5.5</td>
<td>The 3-D volumes in CIE-Lab space, corresponding to specified skin range</td>
</tr>
<tr>
<td>5.6</td>
<td>Visualization of DAC</td>
</tr>
<tr>
<td>5.7</td>
<td>The 3-D volumes in CIE-Lab space, corresponding to selected volumes</td>
</tr>
<tr>
<td>5.8</td>
<td>Original frame, masks, filtered and unfiltered outputs</td>
</tr>
<tr>
<td>5.9</td>
<td>Final filtered output of DAC</td>
</tr>
<tr>
<td>5.10</td>
<td>Faster-RCNN outputs</td>
</tr>
<tr>
<td>5.11</td>
<td>Mask-RCNN outputs</td>
</tr>
<tr>
<td>5.12</td>
<td>Mask-RCNN Expectation and reality</td>
</tr>
<tr>
<td>5.13</td>
<td>Deeplab outputs</td>
</tr>
<tr>
<td>5.14</td>
<td>UNet architecture [Ronneberger et al., 2015] (Figure 1)</td>
</tr>
<tr>
<td>5.15</td>
<td>Network input and output</td>
</tr>
<tr>
<td>5.16</td>
<td>Various skin tones in the dataset</td>
</tr>
<tr>
<td>5.17</td>
<td>Images with no skin pixels in the dataset</td>
</tr>
<tr>
<td>5.18</td>
<td>Intersection over Union [Chaudhary and Pelz, 2019] (Figure 4)</td>
</tr>
<tr>
<td>5.19</td>
<td>Accuracy plot and Loss plot while training the model</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

5.20 Network input and output .............................................. 72
5.21 The ROI window drawn around each hand .......................... 73
5.22 Correct hand detection .................................................. 74
5.23 Partially incorrect hand detection .................................... 74
5.24 The refImg with the key-points shown on $1 bank note ......... 75
5.25 Frame with RANSAC inliers projected on template ............. 77
5.26 Homography with a zero determinant .............................. 78
5.27 Good homography projection ......................................... 79
5.28 Homography with a non-zero determinant but a big projection difference .................. 80
5.29 First homography calculation between template and ROI .... 80
5.30 ROI with unclear/incomplete view of the object’s edges ....... 81
5.31 Dynamically found boundaries of the object in the scene and cropped out ...................... 81
5.32 Second homography calculation between cropped object and modified object ........................ 82
5.33 Outputs after warping with first and second homography ...... 83
5.34 Projection of the hand on the bank note template ............... 83
5.35 Occlusion ................................................................. 85
5.36 Pipeline output using UNet predicted mask ...................... 86
5.37 Multiple ROIs with correct hand analysis ......................... 86
5.38 $10 bill with marked icons of freedom ........................... 87
5.39 Pipeline output for false positive .................................... 89
5.40 Repeated analysis ...................................................... 90
5.41 Multiple predictions per ROI ....................................... 91
5.42 Analysis for different hand-masks .................................. 93
5.43 Metric visualization for Han et al. [2006] where G is ground truth and P is the prediction .................. 94
5.44 Metric visualization for Phung et al. [2003] ....................... 95
LIST OF FIGURES

5.45 Metric visualization for IoU ................................................. 96
5.46 Distribution of CDR ............................................................ 98
5.47 Distribution of FDR ............................................................ 99
5.48 Distribution of IOU ............................................................. 100

6.1 Participant sliding the banknote to count ................................. 106
6.2 Interaction analysis ............................................................. 107
6.3 Distribution of performance metrics for the two UNets ............... 110

7.1 Computing SIFT ................................................................. 113
7.2 Frames to be tweened ......................................................... 114
7.3 Hand pose drawn on a RGB image [Zimmermann and Brox, 2017] (Figure 4) .... 115
7.4 Adjusting the experimental setup to the participant’s eye height .... 125
List of Tables

5.1 Performance Summary .......................................................... 97

7.1 Performance Summary showing the mean value of the distribution shown in Figure 5.46, 5.47, 5.48 ........................................... 126
Glossary

ANN Artificial Neural Network. 36, 37

C-SIFT Color-SIFT. 113

CCT Correlated Color Temperature. 41

CIE International Commission on Illumination. 7, 21, 37, 56, 58, 62, 64, 71, 74, 84, 87, 96, 98, 104, 105, 109, 111

CRI Color-Rendering Index. 41

DoG Difference of Gaussian. 25

HSV Hue, Saturation, Value. 7, 21, 37, 54, 56

IRB Institutional Review Board. 4

POR point-of-regard. 2, 13

RANSAC Random Sample Consensus. 7, 28, 29, 33, 37, 76, 77

RGB Red, Green, Blue. 21, 22, 37, 58, 114

ROI Region Of Interest. 3, 13, 14, 73, 75, 76, 79, 84, 88, 90, 92, 112
Glossary

SIFT  Scale Invariant Feature Transform. 7, 25, 26, 28, 33, 37, 75, 76, 79, 82, 112

SVM  Support Vector Machines describe a hyper-plane in n-dimensional space to separate data represented by n-dimensional vector. 14, 15
Chapter 1

Introduction

Interaction with objects has been studied for over half a century [Fitts et al., 1949; Current, 1954; Tinker 1963]. Scientists have been studying interaction to understand the innate nature of objects and behavior of humans as they interact with those objects. There are multiple ways of studying these correspondences. One common practice for understanding interaction, and the simplest, is by observation.

Interaction is often multi-modal i.e., we use multiple senses to interact, for instance, we use our eyes and hands while washing dishes, assembling furniture, or folding clothes. We use our eyes, hands, and legs to drive a manual transmission car or play badminton. We use whatever senses we require to achieve a given task. The nature of human interaction is inherently sequential so the data collection and data analysis should also be sequential [Bakeman and Gottman, 1997]. The choice of what data to collect is dependent on the task or activity under study.

An object like a pen, bag, or keys is usually picked up with the hands. Hands also perform other various functions like opening, closing, holding, grasping, pinching, signing, writing, lifting, etc. In addition to all of these interactions, humans also use their hands to communicate. All of these activities typically require the use of vision working in support of the interaction. One will have to look at the pen to pick it up and look at the paper to see where to write. One will have to see the lock
to put the key in. One will look at a bottle before picking it up. Eye movements in these activities are performed with different motives. Some fixations (the brief periods when the eyes are relatively stationary and high-resolution information can be seen) are made to guide motor movement while some are made to pre-plan the movement. Pelz et al. [2000] termed the first class guiding fixations and the latter look-ahead fixations, though the terminology varies among researcher groups. A lot of the day-to-day activities we do involve gaze and grasp. Other common tasks involving gaze and grasp include reading a newspaper, typing on a keyboard, scrolling on a mobile phone, counting money, driving, making tea, reading a newspaper, etc. and these interactions are studied for purposes such as behavior modeling [Gidlöf et al., 2013], understanding visual attention [Tsai et al., 2012], figuring out motor guidance [Hayhoe 2008], and more.

Observers’ gaze data can be studied with an eye tracker, a device that can monitor and record the orientation of the eye(s) in the head and/or the point-of-regard (POR) on a computer screen [Duchowski, 2007]. If the experimenter wishes to allow free body and head movements, the eye tracker must be a lightweight, wearable system. In such systems, the eye trackers report the orientation of the eyes in the head over time, and record a video of the scene from the observer’s perspective, indicating the POR in the video frame (see Figure 1.1).

![Figure 1.1: Point-of-regard (POR) in the video frame](image-url)
The recorded data is usually analyzed frame by frame which can get very tedious and time consuming. A calibrated scene video from the eye-tracker captured at 120 Hz for three minutes will have 21600 frames to be analyzed. If every frame takes about 30 seconds to be analyzed, the entire three minutes of gaze data will take about 10,800 minutes or 180 hours to be analyzed.

To tackle the problem of time-consuming data analysis process, Drs. Jeff Pelz and Gabriel Diaz created a gaze analysis software tool which takes in a calibrated scene video with the x,y coordinates of the identified fixations, identifies the object being fixated from a small number of objects in an ‘image library,’ and projects the fixation on a high-resolution template of the matched object. Figure 1.2 shows the output of that gaze pipeline. The video frame is on the right side in every figure which shows the fixation in red and a Region Of Interest (ROI) around the fixation in blue. The template is on the left and the goal of the software is to project the fixation accurately on the template. Figure 1.2(a) shows the match points between the template and the video frame. Figure 1.2(b) and 1.2(c) show good projections of the fixation and Figure 1.2(d) shows a bad projection of the fixation on the template. All of this is automated so a researcher won’t have to manually identify the fixation point on the template in every frame.

The Eye-Gaze Pipeline described above dramatically enhanced the analysis of experiments that required the mapping of participants’ gaze onto objects of interest. There was a need, however, for more advanced analysis that incorporated how participants picked up, manipulated, and handled objects with their hands. The primary goals of this MS project were to develop and implement a pipeline to automate the analysis of manual interaction, and to integrate those results with the existing Eye-Gaze Pipeline results.

One study that required such analysis was a study of cashier behavior. The existing gaze pipeline could be used, but manually identifying all the pixels in a video frame that corresponded to the hand took about 5 minutes per frame, requiring over 1,000 hours to perform the hand analysis for each video. This highlighted the need for the project described below.

Hence, the motivation of my thesis was to design a tool that helps researchers analyze large
interaction datasets with relative ease. A tool meant for research purposes must be validated by applying it on research, so this document describes a test of my analysis tool to study cashier behavior while interacting with 2-dimensional printed objects (i.e., banknotes). This behavior analysis included tracking the gaze and grasp of professional cashiers through a series of mock cash-transactions. Sixty-two cashiers participated in the study, that was approved by RIT’s Institutional Review Board (IRB).

The software tool (the Hand-Grasp Pipeline) created automates the process of detecting the hand(s) in video frames, locating their position with respect to objects in the object library, and representing the spatial distribution of those positions. The raw input for the Hand-Grasp Pipeline was the sequence of images captured from the eye tracker scene camera, which captures the environment around the participant from an egocentric point of view. The frames are processed individually with the aim of identifying all pixels that represent the hand in the images, identify those on objects being held by the participant, and to summarize the interactions on the object
templates in the object library.

![Figure 1.3: Hand-Grasp Pipeline output](image)

Figure 1.3 shows the output of the Hand-Grasp Pipeline tool which is made up of a template, video frame and calculated hand projection for every hand. Figure 1.4 shows the output of Hand-Grasp Pipeline split into parts. The Figure 1.4(a) shows the template (with the final hand data overlaid) and video frame (with the hand marked on it). The Figures 1.4(b) and 1.4(c) show the intermediate steps of processing the object in the hand. The object identified in the hand is shown on the upper left corner of both the images 1.4(b) and 1.4(c). The object is then found in the video frame and cropped. The segment at the bottom is where the object exemplar is being simulated to match the object in the video frame. Midway, we see the parts of the hand (i.e., the thumb/finger) that are on the object.

The template is overlaid with the hands that are processed by the pipelines. We also see the object simulation that takes place for both the hands.

### 1.1 Per chapter synopsis

In this section, I have summarized every chapter after Introduction in brief for the reader’s convenience.
Figure 1.4: *Hand-Grasp Pipeline* output split in parts
CHAPTER 2. LITERATURE REVIEW

Keeping the end result of the research-based tool development in mind, I cover some past research about eye tracking and how its data is analyzed, various ways of detecting the hand, and the use of machine learning in eye data analysis and hand tracking.

CHAPTER 3. KEY CONCEPTS

To help understand the hand-grasp algorithm developed for this thesis, several basic concepts are presented here along with the reasoning behind some of the choices made during development. The concept of color spaces is meant to help a user understand why I do some processing in International Commission on Illumination (CIE)-Lab color space and why some processing in RGB color space and what advantage CIE-Lab gave me over Hue, Saturation, Value (HSV) color space. The morphological operations is supposed to help a reader understand what they are and how they function. I have also explained what parameters I use for the cleaning of the image with the morphological operations. The concepts of Scale Invariant Feature Transform (SIFT), Random Sample Consensus (RANSAC) and Homography are explained individually but in the implementation, they go hand in hand because RANSAC is used to select a subset of points for calculating the homography transform. The last concept is an introduction to artificial neural networks, which are used in the project to ‘learn’ the characteristics of pixels that represent skin in video frames.

CHAPTER 4. DATA COLLECTION

The cashier study is explained in detail, describing how the data used to validate the Hand-Grasp Pipeline were collected. Information on instrumentation such as lighting and eye tracking is given, and experimental procedures such as calibration and experimental design are described.
Chapter 5: Hand-Grasp Pipeline

The Hand-Grasp Pipeline is explained which comprises of finding the skin, locating the hands, identifying the object in each hand, simulating the orientation of the object in the scene with the object template and then projecting the hand from the scene onto the template of the held object. The quality control on skin detection, comparison of hand detection with hand recognition for the part of hand localization, quality control on object identification, and multiple data checks on object simulation and hand projection are performed and explained in detail.

Chapter 6: Conclusion

The Hand-Grasp Pipeline’s creation and functioning is concluded in this chapter. Interaction analysis is also explained along with a discussion on how the UNet model performs.

Chapter 7: Future work

This chapter consists of ideas that can be implemented to make any sub-process or the over pipeline perform better.
References


Chapter 2

Literature Review

With the goal of developing a research-based tool for the analysis of gaze and grasp data, the following section reviews past research in eye tracking and how its data is analyzed, various ways of detecting the hand, and the use of machine learning in eye data analysis and hand tracking.

2.1 Eye tracking and data analysis

2.1.1 Eye Tracking

Most of our interaction with objects involve hand movements, for instance, writing, typing, scrolling, flipping through pages of the newspaper or a document, changing gears in a car or on a bicycle, or holding culinary tools. A lot of these tasks also depend equally on eye movements to guide the hands. Just understanding how one holds an object is not enough to fathom the whole aspect of interaction with objects. We also should know where one looks on the object to interact with it. Many tasks require eye-hand coordination and to understand people’s interaction with an object better, we start off with eye movements. Via eye movements, we wish to answer questions like: Where on the object does one focus? Do they make guiding fixations \([\text{Pelz et al.}, 2000]\) to interact with the object? What are the features they notice on an object while they hold it?
CHAPTER 2. LITERATURE REVIEW

By tracking people’s eyes, we see what area/region is of interest to them and what they pay attention to [Duchowski, 2007]. Psychological analysis and physiological analysis of attention form the basis of eye movements. Examining attentional behavior comes under the psychological viewpoint, while understanding the responsible neural mechanisms that drive the attentional behavior comes under the physiological viewpoint. Fixations, scan-paths, smooth pursuit [Duchowski, 2007], etc. helps one understand attentional behavior from the psychological viewpoint. A lot of researchers [Tinker, 1963; Fitts et al., 1949; Card, 1984; Benel et al., 1991; Goldberg et al., 2002] have used eye-tracking as a means to understand how humans interact with objects or systems and use this understanding to design better items.

Fitts et al. [1949] used eye-tracking to understand how pilots used the cockpit controls and instruments while landing an airplane. Their eye-tracking study was conducted on 40 USAF pilots while flying ground-controlled approach (GCA). They used a 35 mm camera, saving 8 images per second of the face and the eyes of the pilots which were reflected off a mirror placed on the instrument panel. They noted the number of fixations along with their duration on multiple instruments in the cockpit and concluded that the rate of eye fixation did not have a significant relation to the years of flying experience. They also wanted to specify the arrangement of the instruments on the panel which is optimum in terms of usability for all maneuvers.

Later on, Tinker [1963] studied the eye movements of readers to try to understand the effect of font face, page layout, font size, etc. as they read newspapers. He used photographic techniques to figure out the patterns their eyes made and the various reading speeds one has, that is dependent on the change in different parameters.

Eventually, the field moved on from improving systems by using eye movements, to making better eye trackers for more accurate eye movement recordings [Jacob and Karn, 2003]. Yarbus [1967] used a lens system for tracking eye movements which were invasive in nature. It prevented blinking which was highly uncomfortable but provided a good accuracy for eye movements. The tracker by itself was big in size, which was set on the table and had a chin-rest attached to it.
Crane and Steele [1981] created the Dual Purkinje Image (DPI) eye tracker, which was non-invasive and tracked the eye based on the 1st and 4th corneal reflection of a light source. Although it was very accurate and precise, it was very expensive, had a small visual field and was a huge contraption with a bite bar. With a claimed accuracy of one minute of arc [Crane and Steele, 1985] and precision of one minute of arc [Stevenson and Roorda, 2010], the tracker must be mounted on a solid tabletop.

Land [1992] created the first wearable eye tracker to study natural tasks (in this case, driving). The data was captured at 50 Hz using a video camera which was mounted on a cycling helmet. There was no additional light source. The accuracy was about 2 degrees for a 40 degree field with the precision of 1 degree.

Babcock et al. [2003] created a self-contained, wearable eye tracker to study high level visual tasks. Bright pupil tracking was used to track gaze. A color CMOS camera was used to record the scene. The system was compact in size, and could be used in off-line mode, meaning the eye tracker could be calibrated before or after observers performed a task.

Kassner et al. [2014] developed head-mountable, portable eye-tracker which was non-invasive and tracked the pupil in infrared lighting. These systems were relatively accurate and precise (in ideal conditions, 0.6 degrees and 0.08 degrees respectively) and small in size.

2.1.2 Automation in data analysis

Over the years, eye trackers have improved in accuracy and speed, increasing the amount of data generated to be analyzed. Manually annotating massive amounts of gaze data can be very tedious and time-consuming. Researchers have been working to automate data annotation for over a decade. Munn and Pelz [2009] created an algorithm (FixTag) to simplify the analysis of data collected by portable eye trackers, by identifying and tagging the fixations. To determine how the eye moved, the pupil and corneal reflection were detected, and the start and end of the fixations were identified.
The fixation was projected in 3D space and a ROI was created around the fixation in 3D space. Keyframes were picked out using Harris corner detection algorithm [Harris et al., 1988]. A POR was picked from the center for every fixation, and was matched to the nearest keyframe. The POR could be on a calibration point, ROI or other which were checked for the fixation points in the keyframes.

Pontillo et al. [2010] designed SemantiCode, semi-automated software which analyzed gaze data to identify the fixated object, region, or material. The program worked with scene video and gaze coordinates in scene-camera space from wearable eye trackers. Pontillo et al. created a window around the gaze point and extracted features from it. They then compared the extracted features to a library of features to determine which item in the library the observer was fixating. The features they looked at were based on color-histogram intersections [Swain and Ballard, 1992]. This paper inspired our eye-gaze analysis pipeline. The key difference was in the features used. We used SIFT [Lowe et al., 1999] features. The eye gaze analysis pipeline using SIFT features inspired the hand-grasp analysis pipeline covered in this thesis.

Pfeiffer et al. [2014] created a software tool (EyeSee3D) to identify the objects (in 3D) at the gaze point automatically. They placed visible fiducial markers in the workspace and created proxies of the objects involved in the search/identification task. With each marker identified in the scene video frame at known 3D coordinates, the orientation and position of the camera in 3D space could be computed. The scene camera location in 3D was combined with gaze information to compute a gaze vector from the observer’s position to intersect a 3D object. Since every object had its own labeled proxy, the annotation process became automatic.

Panetta et al. [2019] created a completely automated software (GoC) for object annotation at the gaze point. They aimed to create software which works with any video with the gaze, represented by a circle, overlaid on it. They converted the input video to individual image frames, then reduce the number of images by checking for similarity between consecutive frames. They then create a ROI around the gaze point and performed object classification on the ROI. Object classification
was based on a bag-of-visual-words [Csurka et al. 2004] made up of SIFT [Lowe 2004] features. The vocabulary size was 100 and they trained a SVM classifier on the vocabulary to detect words in the ROI and auto-annotate the images.

2.2 Hand detection

To understand how humans interact with objects, researchers can look at how humans hold objects. To get such information, one can use image/video-based sensors to capture interaction data and analyze it. The basic steps of such data analysis are finding the hand in the video and then analyzing the grasp actions. There exist many pathways to achieve that goal. One of those is to identify pixels representing skin in the image, then group those pixels to identify the hand(s). Other methods include locating the hand(s) directly. Some include training a neural network to find the skin and then the hand. Some of the networks are trained to find a human in an image or video sequence. Almost all of these methods are capable of obtaining the required data (i.e., the hand) in some form. I will be touching upon some of these methods for skin detection in the following subsection.

Skin detection

Researchers have tried identifying skin in static images and have been on a journey to improve skin segmentation algorithms [Phung et al. 2003; Kaur and Kranthi 2012]. Some tried finding the skin directly in RGB color space and some moved to different color spaces (such as HSV & CIELAB) [Zarit et al. 1999; Patil and Patil 2012]. Since color spaces are mathematical transformations of each other, skin detection is possible in any color space, and the only thing that varies in the detection are the values for respective channels, though some color spaces make specifying color ranges simpler or more intuitive.

Two commonly used techniques to identify skin in images are pixel-based segmentation and region-based segmentation [Ghotkar and Kharate 2012]. In pixel-based segmentation, a given
CHAPTER 2. LITERATURE REVIEW

pixel is identified as either skin or not skin. A basic example of this is thresholding skin in different color spaces. Such pixel-based algorithms are not affected by the information in the neighboring pixels. In region-based segmentation, an entire region or segment will be identified as skin or not skin. Region-based segmentation often requires additional information like the texture of the skin [Shaik et al., 2015] which is derived/dependent on the information in the neighboring pixels. One example of region-based segmentation is region-growing [Adams and Bischof, 1994; Tremeau and Borel, 1997].

Use of machine learning

Machine learning approaches have also been used to identify not only skin but also different parts of the human body or the entire human beings in images. Pixel-based segmentation [Han et al., 2006] and high-level semantic segmentation [Zimmermann and Brox, 2017] were implemented using machine learning.

Han et al. [2006] implemented pixel-based segmentation by thresholding a range of pixel values in RGB color space and trained an SVM classifier to learn actively from a couple of video frames and perform skin-pixel identification. They took the project a step further and implemented the JSEG algorithm [Deng and Manjunath, 2001], an unsupervised algorithm, to incorporate region segmentation. JSEG implements color quantization followed by region segmentation. Three metrics are used to measure the performance of the SVM followed by JSEG. The percentage of correctly classified skin pixels was labeled the Correct Detection Rate (CDR); the percentage of incorrectly classified non-skin pixels was labeled the False Detection Rate (FDR), and the overall Classification Rate (CR) was defined as the ratio of number of correctly classified skin pixels upon maximum of skin pixels in either predicted or ground truth mask. Their proposed model had a 86.34% correct detection rate, a 0.96% false detection rate and a 76.77% classification rate. Even though their classification rate was not very high, the implementation of JSEG on trained SVM data is worthy of note for applying segmentation on the SVM output as opposed to the original input.
Zimmermann and Brox [2017] estimated 3-dimensional hand pose from RGB images by using deep learning. Their entire network comprised of three building blocks, namely HandSegNet, PoseNet, and Pose Prior. HandSegNet segmented the hand in an RGB image. PoseNet then used the hand key-points from the segmented image and created a score map. This map was then passed to the Pose Prior which estimated a 3-dimensional structure that best fit the score map to interpret the pose of the hand. HandSegNet was trained for 40,000 epochs. Zimmermann and Brox [2017] created their own dataset and used multiple other datasets as well. The results reported were for the entire network and not individual part-networks. The network was competitive in performance to the networks trained with depth maps for 3D pose estimation, which implies that their HandSegNet (the first network in the processing sequence) would have performed well for the following networks to perform too. The HandSegNet used convolutional layers with ReLU and maxpooling while downsampling/extracting semantic information, and using a single bilinear upsampling layer to output the mask containing the hand.

Cai et al. [2017] created an automated system for analyzing hand grasp. They first detected hands and extracted hand-based features, to train a grasp classifier to differentiate between various pre-defined grasps types. Using the trained classifier, visual structures of the grasp were obtained and the structures were clustered. Hand-object interactions were recorded from a first-person perspective. To detect the hands, a detector was trained to predict the likelihood of hand for every pixel, and based on the likelihood, the hands were segmented and two candidate regions were retained as hands.

2.3 Summary

Keeping the end result of the research-based tool development in mind, I cover some past research about eye tracking and how its data is analyzed, various ways of detecting the hand, and the use of machine learning in eye data analysis and hand tracking.
References


Chapter 3

Key Concepts

To help understand the hand-grasp pipeline developed for this thesis, several basic concepts are presented here along with the reasoning behind some of the choices made during development.

3.1 Color Spaces

Color spaces or color models are mathematical models that represent color information in channels corresponding to different properties like luminance, hue and saturation. The most widely used color space is *Red, Green, Blue (RGB)* and it is the default color space for saving digitized images [Patil and Patil, 2012]. Any color that can be obtained on a device can be described by a combination of RGB channels.

Linear and Non-Linear transformation of RGB gives us different color spaces [Kolkur et al., 2017]. The *HSV* color space is meant to be a more intuitive color reference for human perception of color than RGB. It separates the color information into three channels, namely hue, saturation and value. The *CIE-Lab* color space is an international standard designed for perceptual uniformity [Omer and Werman, 2004]. The CIE-Lab color space comprises of three channels which are L, a and b. The L channel corresponds to luminance, the a channel describes a red-green opponent axis.
and the b channel describes a yellow-blue opponent axis. Figure 3.1 shows the illustrations of the color models.

Figure 3.1: Color spaces

Every point in RGB space has a corresponding point in all the other color spaces too. But the distance between points in different color spaces differs, as does the ‘direction’ between two points. We take advantage of this fact and find a color space in which normal variation in skin tones due to shadows does not cause a large distance in the color space, while the distance between skin tones and other objects (such as background) are maximized.

### 3.2 Morphological Operations and Structuring Element

Morphological operations are image processing operations that modify the output pixel based on a comparison of the input pixel and a particular set of the neighboring pixels. The set of neighboring pixels that are considered is defined by a structuring element. OpenCV, an open-source computer vision library (see book by Bradski and Kaehler, 2008 or OpenCV documentation) has three built-in structuring element shapes, namely rectangular, elliptical and cross-shaped kernel.

---

* Image was taken from: [https://bit.ly/2XVlCVg](https://bit.ly/2XVlCVg)
It was observed that the arbitrary shape of the object was retained with an elliptical shaped structuring element.

The two most fundamental morphological operations are dilation and erosion. As the names suggest, during dilation the pixel dilates/bleeds with respect to its neighbours [See figure 3.2(c)] and during erosion, the pixel erodes/wears away with respect to its neighbours [See figure 3.2(d)]. If dilation is followed by erosion using the same structuring element, then it is called closing [See figure 3.2(e)] and if erosion is followed by dilation using the same structuring element, then it is called opening [See figure 3.2(f)]. The morphological operations often morph/change the size of the original input. However, closing followed by opening and opening followed by closing retains the size of original input image.

Figure 3.2 shows the original image of size 100x100 and an ellipsoidal element of size 3x3. The figure also shows the various morphological operations performed on the original image with the structuring element. The figure also shows closing followed by opening [3.2(g)] and opening
Figure 3.2: Various morphological operations
followed by closing \([3.2(h)]\).

*Morphological cleaning* is done by performing closing followed by opening and *morphological refinement* is done by performing opening followed by closing. For more information on the structuring elements and morphological operations, see OpenCV documentation.

### 3.3 Scale Invariant Feature Transform (SIFT)

Features are characteristic traits of objects. Features are not limited to the spatial domain like textures or patterns; they can be found in the temporal domain as well. An object can be identified by its associated features. Features should be invariant to maintain consistency. We recognize people because they look the same everyday. We recognize a car to be the exact same one if the number plates match.

Because objects have features, images of those objects will also have features. But these image features are different. You can’t smell an object or listen to it in an image, but you can see its color. You can also see patterns. People can recognize objects if they are held in different orientations. If an object moved away from you, you would still recognize it, which means the features were invariant to scaling as well. The features one uses are task-dependent. The data I deal with varies drastically in orientation and scales. To extract information in these conditions, I use SIFT on images to find features in the image. Figure 3.3 shows the SIFT features marked with circles on a part of the $10 bill and on a part of rotated 10$ bill.

SIFT [Lowe et al., 1999; Lowe, 2004] is scale invariant feature transform. It is also invariant to rotation. First SIFT finds all the key-points and then creates those key-points’ descriptor. To find the key-point, Gaussian filters of various scales (i.e., \( \sigma \)) are applied to every octave of the image and the *Difference of Gaussian (DoG)* is calculated. Octave of an image is basically a down-sampled (by power of 2) version of itself. Once DoGs are computed, local extrema are searched for in various scales. If these extrema don’t change with scale, then they are key-points. To eliminate
Figure 3.3: SIFT features marked with a circle on a part of $10 bill and on a part of rotated 10$ bill

low-contrast key-points [Lowe et al., 1999; Lowe, 2004] the scale space is computed using Taylor series expansion.

After strong key-points are found, their descriptors are computed. The descriptors are based on
the magnitude and direction, which are calculated in every key-point's neighbourhood.

\[
\text{Magnitude} = \sqrt{(A_{i,j} - A_{i+1,j})^2 + (A_{i,j} - A_{i,j+1})^2} \tag{3.1}
\]

\[
\theta = \tan^{-1} \left( \frac{A_{i,j} - A_{i+1,j}}{A_{i,j+1} - A_{i,j}} \right) \tag{3.2}
\]

where $A_{i,j}$ is a pixel in the image.

A window of size 16x16 is created around the key-point and the window is further divided in
sixteen 4x4 blocks. For each block, 8-bin histogram is created which bins the direction ($\theta$). Putting
all the block’s histograms together (16x8), a 128 bin vector is obtained which essentially describes
the orientation of a key-point in its neighbourhood.

Figure 3.4 shows a figure from Lowe [2004] where they show a window of 8x8, divided into
four 4x4 blocks. Each block is summarized by a 8 bin histogram (represented in one sub-block
of key-point descriptor). The 8 bins of the four blocks are put together to obtain the key-point

\[\text{This is an overview of the concept. For a detailed yet simpler explanation, refer to OpenCV’s documentation on SIFT introduction: https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html}\]
Figure 3.4: Image gradients and key-point descriptor from [Lowe, 2004](Figure 7)

descriptor (as shown on the right). This is a conceptual explanation, and is implemented with a window of size 16x16 around the key-point.

Once we have key-points and their descriptors, we can use them for recognition, detection and identification. To compare two key-points, we look at the distance between the descriptor vectors. The lower the distance, the more similar the points. If we want to find a key-point from one image in another image, we look for all the key-points in the target image and compare all of them to the query descriptor. To put a quality check on the matches, we use the Lowe Ratio [Lowe, 2004], calculated by taking the ratio of the distance of query with nearest neighbour to the distance of query with its second nearest neighbour. The lower the ratio, the higher the probability of finding a correct match.

If a query descriptor’s nearest neighboring descriptor is at a distance of 10 units while the second nearest neighbouring descriptor is at a distance of 12 units. The Lowe’s ratio would be 0.833, which means the second nearest neighbour is almost 80% as good as the first neighbour. But, if the first neighbour was at a distance of 6 units, then the ratio would decrease to 0.5, implying that the second nearest neighbour is only half as good as the nearest neighbour. And so, the nearest neighbour in that case would be more likely to be the correct match. This helps control quality and let go of potentially picking up false positives. This ratio is particularly helpful when features repeat.
After identifying good quality SIFT features, additional steps must be taken to eliminate the presence of outliers, and hence, RANSAC [Fischler and Bolles, 1981] is computed to find inliers from all the good SIFT features. The following section explains RANSAC in brief.

### 3.4 Random Sample Consensus (RANSAC)

Fischler and Bolles [1981] came up with a new paradigm called Random Sample Consensus (RANSAC) for model fitting that estimates the inliers and outliers in the data. This algorithm chooses the points which would contribute to a smooth model fit and excludes points that would contribute to an inaccurate model. Let us understand this with an example. Say we have a number of points, and the goal is to find the inliers and fit a line to them. To calculate the slope and intercept of a line, two points are enough. RANSAC will randomly pick two points, calculate the equation of the line, and check the distance of every data point with that line. We can set the threshold that we want the points to be within. Let’s set the max to be 2 units so a point can only be 2 units away from the line, else, if it is more, we don’t want that point included (because that would be an outlier). And so, RANSAC will compare the threshold with calculated distances and find inliers and outliers. We can also set the inlier ratio to be (say) 70%. This means we want a model that fits at least 70% of the data, to be precise, a model that considers the data to have at least 70% of the inliers. If the model calculated by RANSAC doesn’t satisfy the criteria, then RANSAC will again choose two points randomly and repeat the model-making process. One can also define the number of iterations one wants RANSAC to perform because model fitting repeatedly can consume a lot of computer resources. Once a model is calculated which fits the criteria, a mask is generated which shows all the points that lie within the threshold set as inliers and all the ones that don’t as outliers.

We can see from Figure 3.5 how a model fits the data if all the points are chosen vs how RANSAC fits the data, while highlighting the inliers and outliers. One thing to keep in mind is that no point is a bad point. The points are good or bad only with respect to a model. You can choose to
include ALL the points or can exclude some if their behaviour is far from your pool of data points. This is only a model fitting technique which can be applied to any domain. Figure 3.5 shows a line fit on the left and a circle fit on the right. This doesn’t limit RANSAC to only points. I use RANSAC with the features to pick out four points, while excluding error-prone features to calculate homography. See the following section on homography for detailed understanding of homography and how RANSAC is applied.

3.5 Homography

Homography [Kruppa, 1913; Faugeras and Maybank, 1990; Hartley and Zisserman, 2003] is an invertible plane-to-plane mapping technique/transformation which helps to project data from one domain/plane to another. To be precise, homography is a projective transform for point to point correspondences (see Figure 3.6).

The math for homography explained in this section is from the book *Multiview Geometry in Computer Vision* written by Hartley and Zisserman [2003].

Let us consider a pair of corresponding points \(x(x,y,z)\) and \(x'(x',y',z')\), representing a point
X(X,Y,Z) in world space that appears in two different image planes. The transformation of a point from one image plane to another will be given by

$$x' = Hx$$  \hspace{1cm} (3.3)$$

The points $x$ (x,y,1) and $x'$ (x',y',1) would be the inhomogeneous representation of the world [Hartley and Zisserman 2003]. Upon substituting 3D coordinates into equation 3.3 we get

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$  \hspace{1cm} (3.4)$$

Upon multiplying the matrices in 3.4 we get $x'$, $y'$ and $z'$ as

$$x' = h_{11}x + h_{12}y + h_{13}z$$  \hspace{1cm} (3.5)$$

Illustration created by Dr. Jeff Pelz
To remain in inhomogeneous coordinate space, we make \( z' = 1 \), and hence, we divide \( x', y', z' \) by \( z' \):

\[
x' = \frac{x'}{z'} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}
\]

\[
y' = \frac{y'}{z'} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}
\]

\[
1 = \frac{z'}{z'} = \frac{h_{31}x + h_{32}y + h_{33}}{h_{31}x + h_{32}y + h_{33}}
\]

Since the transformation is intra-planar, \( h_{33} \) is equated to 1 and we substitute \( z \) as 1 as well. We get

\[
x' = \frac{h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx'}{h_{31}x + h_{32}y + 1}
\]

\[
y' = \frac{h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy'}{h_{31}x + h_{32}y + 1}
\]

We now have eight unknowns representing 8 degrees of freedom for a transformation. We require at least four pairs of corresponding points (i.e., eight points) to calculate the eight unknowns.

Upon rearranging, we get

\[
x' = h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx'
\]

\[
y' = h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy'
\]

We now write these vectors in terms of matrices and obtain \( D = S*h \) where \( S \) represents the source, \( h \) is the homography matrix and \( D \) is the destination.
We apply the four-point algorithm [4] to obtain the mapping / homography matrix. This algorithm can only be applied if no three points are collinear i.e., any 3 points should not form a straight line.

This can now be extended to N points and the matrix will be

\[
D_{2N\times 1} = S_{2N\times 8} \ast h_{8\times 1}
\]  

\[
S^T D = S^T S h
\]
S and $S^T$ now form a square matrix that can be inverted.

$$h = (S^T S)^{-1} (S^T D)$$  \hspace{1cm} (3.18)

$$h = S^{-1} (S^T)^{-1} S^T D$$  \hspace{1cm} (3.19)

$$h = S^{-1} D$$  \hspace{1cm} (3.20)

We now get a $h$ vector which has to be reshaped into a 3x3 matrix $H$ with $h_{33}$ as 1\textsuperscript{1}.

Homography is used to map one plane to another. This mapping is not limited to two dimensions i.e., x and y. We can also map features from one plane to another. The features can be any dimension as far as they associate to some (x,y) in the plane they belong to. Because ultimately, the x,y point will be used to compute the homography matrix.

I implement homography based on the x,y coordinates of SIFT features where every feature is described with a key-point and descriptor. Using Lowe’s ratio [Lowe et al., 1999], I find good SIFT feature matching pairs. I then find a good homographic mapping based on RANSAC model fitting.

RANSAC randomly picks four pairs of x,y where each x,y, corresponds to a key-point. The homography matrix is calculated based on these four pairs. A threshold for RANSAC is set in terms of pixels. The source is warped with the homography and the distance between a key-point in the new warped plane and the corresponding key-point in the destination plane is checked. If the distance is less than the set RANSAC threshold then that key-point is said to be an inlier, otherwise it is considered an outlier. The total number of inliers contribute to the confidence of the RANSAC on the calculated homography matrix. If the confidence is less than 99.5% then the homography matrix is recalculated by randomly choosing another set of four pairs of matching features.

Figure 3.7 shows a part of the $10 bill and a part of the rotated $10 bill, both marked with circles on the x,y coordinates of SIFT features. The same figure also shows the top 50 key-points

\textsuperscript{1} This is one way of computing homography. Another method is using SVD matrix decomposition.
So far, we have understood that homography is a perspective projective transform, with which we can transform points from one plane and project them onto another. On a higher level, we can view a set of points from a perspective different from the original by using homography. Let us look into some other transforms.

\[
H_S = \begin{bmatrix}
  sR & t \\
  0^T & 1
\end{bmatrix} = \begin{bmatrix}
  s \cos \theta & -s \sin \theta & t_x \\
  s \sin \theta & s \cos \theta & t_y \\
  0 & 0 & 1
\end{bmatrix}
\] (3.21)

\(H_S\) is the similarity matrix. It consists of 4 degrees of freedom and can be determined by 2 pairs of matching points. The angles between the lines and ratios of the distances between the points in two planes remain the same.

\[
H_A = \begin{bmatrix}
  A & t \\
  0^T & 1
\end{bmatrix} = \begin{bmatrix}
  a_{11} & a_{12} & t_x \\
  a_{21} & a_{22} & t_y \\
  0 & 0 & 1
\end{bmatrix}
\] (3.22)

\(H_A\) is the affine matrix. It consists of 6 degrees of freedom and can be determined by 3 pairs of matching points.
CHAPTER 3. KEY CONCEPTS

matching points. The parallel lines, ratios of lengths of parallel line segments, and the ratios of areas of enclosed polygons in the two planes remain the same.

\[
H_P = \begin{bmatrix}
A & t \\
v^T & \mu
\end{bmatrix} = \begin{bmatrix}
a_{11} & a_{12} & t_x \\
a_{21} & a_{22} & t_y \\
v_1 & v_2 & \mu
\end{bmatrix}
\] (3.23)

\(H_P\) is the projective matrix. It consists of 9 degrees of freedom and can be determined by 5 pairs of matching points. The cross-ratio of lengths on a line in the two planes remains the same.

The homography matrix is a perspective projective transform made up of a chain of transforms. We can decompose it as

\[
H = H_S H_A H_P
\] (3.24)

where

\[
H = H_S H_A H_P = \begin{bmatrix}
sR & t \\
0^T & 1
\end{bmatrix} \begin{bmatrix}
K & 0 \\
0^T & 1
\end{bmatrix} \begin{bmatrix}
I & 0 \\
v^T & \mu
\end{bmatrix} = \begin{bmatrix}
A & t \\
v^T & \mu
\end{bmatrix}
\] (3.25)

If we have \(H\), we can identify the components it is made up of i.e., \(A\), \(t\), \(v\) and \(\mu\). Upon exploring equation \(3.25\) we get

\[
A = sR K + tv^T
\] (3.26)

If we solve for \(s\), \(R\) and \(K\), we can get \(A\). To compute \(sRK\), we write the equation as

\[
sRK = A - tv^T = M
\] (3.27)

The \(M\) matrix is a 2x2 matrix which is essentially devoid of translation and perspective projection. It only represents scaling, rotation and shear.

This is important because later on in the hand-grasp pipeline (see Chapter 5), we put conditions on the \(M\) matrix to understand if the mapping calculated has good representation of the amount
of rotation, scaling and shear that the template plane should undergo to simulate the object in the image plane (of the scene camera).

3.6 Artificial Neural Network (ANN)

A human brain is made up of fundamentals units called neurons. The neurons and the massive interconnections between those neurons are often called a neural network and are responsible for tasks like recognizing a person, identifying other people doing some task, etc. repeatedly on a daily basis. It would be an achievement if the computer could perform those tasks and so programmers/researchers came up with the concept of Artificial Neural Network (ANN) [Hassoun et al., 1995; Haykin et al. 2009]. The ANN is essentially a piece of code (or hardware) which conceptually implements a complex network of neurons unlike the way traditional algorithmic for/while loops function. Here, the 'neurons’ are non-linear entities and the network on the whole functions as a complex mapper which creates a non-linear relationship between the dependent data and independent data. Analogous to human brains, the ANN also have activation functions modeled after after a neuron. As human brains can learn a task after observing/performing it repeatedly, an ANN has to be taught in a similar fashion.

The learning process/technique for an ANN can either be supervised or unsupervised. Supervised learning is when the ANN is shown the target variable and its task is to create a relationship which can predict the learnt target. Supervised learning is analogous to regression and is used for the purposes of prediction and classification. Unsupervised learning is when the ANN is aware of the end goal but doesn’t have a target variable. In this case, the network learns the features of the data and groups them. This is similar to clustering algorithms. This is mostly used when one doesn’t know what the data can be grouped into or when one doesn’t have labeled data.

The human brain is a complex entity and computationally implementing it would be a feat. To get closer in performance to the human brain, ANNs were increased in number of layers and
complexity of a neuron. There exist various types of ANN and they all have different kinds of fundamental units. For detailed information on the various neural networks, I highly recommend reading *Deep Learning* [Goodfellow et al., 2016]

One such type of an ANN is the convolutional neural network (CNN) where the layers are made up of convolutional filters. Such a network is often used in tasks involving images. The task could be object recognition or segmentation or detection or any other. I use the CNN to implement semantic segmentation in my hand-grasp pipeline. To be precise, I use the UNet [Ronneberger et al., 2015] to identify the pixels in each video frame that represent a participant’s hand.

### 3.7 Summary

In this chapter, I have explained some concepts which help understand the functioning of the Hand-Grasp Pipeline. The concept of color spaces helps a user understand why I do some processing in CIE-Lab color space and why some processing in RGB color space and what advantage CIE-Lab gave me over HSV color space. The morphological operations help a reader understand what they are and how they are helpful in operations like filling gaps in regions identified as skin. The concepts of SIFT, RANSAC and Homography are explained individually but in the implementation, they go hand in hand to compute the projection of objects between video frames and reference images. The last concept is an introduction to ANNs, which are used to learn the non-linear relationships between dependent and independent data.
CHAPTER 3. KEY CONCEPTS

References


CHAPTER 3. KEY CONCEPTS


Chapter 4

Data Collection

As mentioned in Chapter 1, a study was conducted to understand cashier behavior that required the analysis of both gaze and grasp. But gaze and grasp analysis can only be as good as the data collected. Because the aim was to understand natural behaviour, we required the participants to be as natural as they could be while interacting with the experimenter. The experiment was carried out while the participant was standing throughout, so the participant was free to move their entire body to be comfortable and move naturally while cashing out the experimenter. The data was collected for a total of 62 participants. This chapter describes the data collection procedure in detail.

4.1 Lighting

The experiment was conducted in an office environment. We augmented the regular office lights with ceiling-mounted LED illuminators to provide sufficient illumination across the desktop work surface. The illuminators (see Figure 4.1) provided adequate uniform illumination for the participants to interact with the 2-D printed imagery, and for the eye tracker systems to capture high-definition images of the environment without significant motion blur.

We installed five array illuminators on the ceiling as shown in Figure 4.1. Of the five, three were
CHAPTER 4. DATA COLLECTION

4.1 Figure 4.1: Lighting in the experiment room

Draco Broadcast Dracast model DRSP-500-B and other two were Neewer Dimmable Bi-Color 480 Video Lights. A total of 2,400 high-power LEDs provided an illuminance of 1800 lux on the entire workspace (i.e., over two tables) at a Correlated Color Temperature (CCT) of 4600K and a Color-Rendering Index (CRI) >96.

The illuminators were set to 4600K CCT to ensure that color rendition was adequate for the visibility of all the colored features of the banknote, which might not have been distinguishable under typical fluorescent illumination. To run the experiment on participants of different heights, the table was adjusted to suit individual eye height.

4.2 Apparatus

One goal of the experiment was to collect data under conditions that were as realistic as possible and not reduced in complexity for the sake of experimental convenience. We sought equipment

\* Dracast Silver Series LED500 Bi-Color LED Light with V-Mount Battery Plate, 480 5mm LEDs
\(\dagger\) including 3200-5600K CRI 96+ LED Panel
that could capture natural behaviour data under these conditions while assuring data quality.

Figure 4.2: Pupil Labs eye tracker used in the experiment

We chose the Pupil Labs Binocular Eye Tracker (also known as Pupil Core [https://pupil-labs.com/products/core/][Kassner et al., 2014]) because of its high speed, lightweight, and open-source software. The eye tracker was used to capture video of both eyes at a resolution of 640 x 480 and a temporal resolution of 120 Hz. The high-definition scene camera of the eye tracker was located over the participant’s right eye which captured the scene video at 1280x720 resolution at 60 Hz. Figure 4.3 shows a frame from the scene camera on the eye-tracker. The eye cameras used were relatively small, had fixed focus and could capture images of each eye at 200 Hz. The tracker is shown in Figure 4.2.

(a) Table with the till in front of the participant  (b) Table between the participant and the experimenter

Figure 4.3: Scenes from the eye tracker

‡ Pupil Labs online store: [https://pupil-labs.com/products/core/tech-specs](https://pupil-labs.com/products/core/tech-specs)
The participants were professional cashiers, and the task was to carry out transactions while standing at a mock cash register. The table was painted grey and inclined at 30 degrees so that the cashiers could view objects on the table surface without rotating their eyes too far below 'horizontal.’ The table had a mock cash register consisting of a cash drawer (‘till’) and a monitor which displayed the details of the transaction including the cost, amount tendered, and the change due (see Figure 4.3(a)). The table over which the money was exchanged with the experimenter was also gray so that the banknotes would be clearly separated from the background (see Figure 4.3(b)).

The audio of the experiment was recorded on the data collection laptop by using the built-in microphone. A mock security camera was installed in the room to minimize the risk of theft. There was also a small box with a Led and a push button for the participants to indicate the start of each fixation during the calibration procedures (see Figure 4.4).
Figure 4.5: Pupil Player software

4.3 Eye Model

The eye and scene videos were recorded during each experimental session using Pupil Capture software Version 1.7.42, then processed offline with Pupil Player software Version 1.7.42 [Kassner et al., 2014] (see Figure 4.5). The software synchronizes the eye and scene videos, identifies the pupil border in each eye video frame, and fits an ellipse to the pupil, allowing the eye tracker to be calibrated by fitting a 3D eye model to each eye based on the sequence of pupil fits.

If successive pupil images begin to deviate from the existing model, the model is updated or changed. To minimize tracker deviations due to frequent model updates, it is advisable to expose the system to a large range of eye orientations before calibration begins. The experimenter asked the participant to fixate on a target and moved the target around in the entire workspace to capture a large range of "views" of the eyes to minimize eye-model updates (see Figure 4.6).
CHAPTER 4. DATA COLLECTION

4.4 Calibration

Ideally, calibration would be completed in the same plane as data collection to avoid parallax errors due to the offset between the observer’s eyes and the scene camera [Evans et al., 2012]. While binocular trackers can estimate vergence and partially correct for these errors, errors remain when calibration is performed at a different distance than data collection. To minimize these errors, calibrations were collected at multiple depth planes; one at the till and one at a plane above the surface where the experimenter and cashier exchanged banknotes.

They performed multiple calibrations in multiple planes at the start, middle and end of every session to ensure sufficient accuracy. Figure 4.7 shows the two calibration boards used throughout the data collection. Participants were told to indicate a stable point within each fixation by a button press (see Figure 4.8) which allowed the experimenters to pick fixation frames in the signaled period to use for calibration. Center offset checks were performed before each trial. If the indicated fixation drifted more than 1 degree from the fixation target, the offset was corrected, or the calibration and validation sequence was repeated.

To improve calibration accuracy, the experimenters came up with a three-plane calibration
(a) Till calibration board  
(b) Fan calibration board

Figure 4.7: Various calibration boards used throughout the data collection

Figure 4.8: Participant indicating a fixation with button click during cash-drawer calibration
procedure in which, every participant was calibrated in three tracking work-areas. The first calibration was the ‘cash-drawer’ calibration where each participant looked at each of nine points on the calibration board over the till, as seen in Figure 4.8. Along with the nine calibration points, the calibration board also had four validation points which were used to as alternate calibration points for difficult calibrations/ dropped calibrations or to grade the quality of the track.

After cash-drawer calibration, the experimenter performed a ‘fan’ calibration in the space below the cash drawer, on the desk area where the counting of banknotes took place. The third calibration plane was a nine-point ‘exchange-calibration’ in which the board was placed in the space between the cashier and the experimenter (the customer), as shown in Figure 4.9. For the two calibration procedures with fan calibration, the participants indicated their fixation by pressing the calibration board with their thumb.

![Figure 4.9: Exchange calibration](image)

Figure 4.9 shows a close-up of the exchange-calibration target. This calibration was designed in a way that when viewed from the participant’s perspective, all the calibration points were equally spaced in degrees of angular subtense from the participant’s perspective. The red lines superimposed on the figure diverge outwards from the participant’s body, giving the desired calibration
layout in perspective.

Figure 4.10: Exchange calibration from a perspective

All the three calibration procedures were performed by the participant thrice i.e., in the beginning, midway through the experiment (after 20 transactions), and at the end of the experiment (after 40 transactions).

4.5 Center Offsets

Small motions of the eye-tracker on the observer’s face can cause drift of the indicated gaze position, degrading accuracy over time even if calibration is accurate and validated. Natural motion such as large head movements, and body movements, etc. will amplify this effect. To minimize these, the experimenters monitored the calibration by having the participant fixate on a small calibration target point printed on a banknote-sized center-offset target (see Figure 4.11). These targets were
Figure 4.11: Center offsets with gaze overlaid as seen from the scene camera during analysis shown at regular intervals throughout the experiment, and between calibrations. If the distance between the fixation target and indicated fixation was more than 1°, then either the calibration and validation sequence was repeated or the offset was corrected.

4.6 Task: Mock Transaction

Figure 4.12: Tasks as seen from the eye trackers
As described above, the mock cash register had till and a monitor. The monitor displayed the following information:

- **Cost**: Amount experimenter/’customer’ was supposed to pay (for a purchase).
- **Tendered**: Amount handed by the customer to the participant/cashier.
- **Change**: Amount the cashier took from the till and gave back to the ’customer.’

For every transaction, the experimenter handed a stack of banknotes to the participants (as if it were for a purchase), the cashier verified that the amount given by the customer was same as the amount displayed on the monitor as *Tendered*. The cashier then made the change and handed it back to the customer.

The entire task included 40 transactions. Center-offset targets were carried out in between every five exchanges. Calibration procedure was carried out at three locations (i.e., the till, below the till and between the customer and the cashier) three times i.e., before the transactions began, after 20 transactions and after 40 transactions. Six practice transactions were carried out after the first calibration so the participant got comfortable with the task and the work area before the first recorded trial. All orientations of the banknotes were random with face-up.

### 4.7 Software

1. To capture the data, we used Pupil Capture (Version 1.7.42) [Kassner et al., 2014].

2. To calibrate the data offline and obtain fixation coordinates in camera space, we used Pupil Player (Version 1.7.42) [Kassner et al., 2014].

3. To develop the Hand-Grasp Pipeline, we used

   (a) Python 3.6.6 [Van Rossum and Drake Jr, 1995]
(b) CSV 1.0

(c) Matplotlib 3.0.0 [Hunter, 2007]

(d) Numpy 1.15.2 [Oliphant, 2006]

(e) OpenCV 3.3.0 with Contrib module [Bradski and Kaehler, 2008]

4.8 Summary

In this chapter, the cashier study which was used to validate the Hand-Grasp Pipeline, was described in detail. Information on instrumentation such as lighting, software and eye tracking is given, and experimental procedures such as calibration, center-offsets, and eye-model creation are described in sufficient detail to allow replication.
References


Chapter 5

Hand-Grasp Pipeline

The end goal was to analyze interaction data which comprised of data from eyes and hand. The existing Eye-Gaze Pipeline was used to analyze eye-data and the \textit{Hand-Grasp Pipeline} was developed for hand-data analysis. The end goal of the Hand-Grasp Pipeline was similar to that of the Eye-Gaze Pipeline; to map the grasp/gaze onto the object template. This could be accomplished manually by having a technician select all the pixels representing the participant’s hand that occluded the object, but that would be extremely time consuming. The Hand-Grasp Pipeline was developed to speed up this process by automation. From a high level, the Hand-Grasp Pipeline can be understood as a multi-stage process where first the object of the interaction is detected, identified, and modeled. Second the participant’s hand (or hands) is (are) detected and projected onto the modeled object. Finally, the detected hand is projected from the modeled object to a 2D template of the object. This high-level outline of the pipeline is parsed into smaller stages below to get a better understanding of the functioning pipeline.

The multiple stages are as follows:

1. Skin detection

2. Hand localization
3. Object identification

4. Simulation of identified object

5. Back projection of the hand onto the object template

6. Drawbacks in the pipeline

7. Output verification

### 5.1 Skin detection

The first step is to determine which regions in the image represent the participant’s hand(s), so we start by detecting skin to answer *What is the hand?* To find all the pixels which represent skin, a number of methods were used to segment ‘skin pixels’ from the video frame.

#### 5.1.1 HSV color space selection

In the first approach, I implemented an interactive track-bar interface (see Figure 5.1) that allowed the experimenter to select a limited range of pixel values, for skin for each participant using six sliders. I created six individual sliders; one each for the upper and lower bounds of the Hue, Saturation, and Value channels in the HSV color space (see Figure 5.1(a)). All the sliders ranged from 0 to 255. Pixel values between the lower and upper bounds set by the sliders in each channel were selected. Using these values of the track-bar, the experimenter thresholds the video i.e., selects all the values that lie within the bounds and visualizes what image regions are accepted and rejected (see Figures 5.1(b)). By adjusting the sliders and monitoring the video, the experimenter selects the appropriate HSV range to isolate the skin for every participant to be used later in post-processing (see Figure 5.1(c)).

* Inspiration was taken from

(a) HSV bar bounds set to accept every color

(b) Adjusting the bar to find the skin

(c) Skin thresholded with other objects/noise

Figure 5.1: HSV thresholding using the trackbar
While the HSV color space allows isolation of skin pixels in the video sequence, the color space is not ideal in the presence of shadows. In addition, the three-channel, high-low range trackbar system allowed only one color ’volume’. When there is shadow cast on the skin, certain pixels are difficult to separate from actual skin in HSV, but are easily separated in CIE-Lab space. If there is even a slight shadow on the skin, it cannot be detected in terms of saturation and value, such a variance is easily detected in the luminance channel of CIE-Lab color space. A limitation of the single-volume track-bar system is that if you attempt to add color for other objects like nail polish, the increased bounds end up adding other objects that lie in between the skin and the nail color in the color volume.

5.1.2 Use of CIE-Lab color space and Dynamic Area Creation (DAC)

Specifying a single color volume in HSV color space resulted in the erroneous identification of many pixels as skin because of shadows, and because the pixels fell into regions of the color volume between skin and other values such as nail polish. Keeping in mind that every individual has a different skin tone and some wear nail polish or rings and some don’t, this created a lot of variability/divergence in the identification of skin pixels in the data.

![Figure 5.2: DAC output without filtering](image)

To deal with divergence, and to incorporate CIE-Lab color space, a new tool was developed that allowed the experimenter to specify multiple, separate color volumes to detect/segment surfaces in
CIE-Lab color space such as skin, freckles, tattoos, wrist bands, etc., and to perform quality checks on the results (see Figure 5.2). In addition to allowing multiple color volumes, the new tool also allowed the experimenter to specify 'voids' within color volumes; small sub-regions within that volume that are excluded.

The primary operations of the custom software tool are:

1. Create a new volume in CIE-Lab color space and add points to define it.
2. Create additional volumes for nail paints, tattoos, etc. (optional).
3. Create voids within color volume(s) (optional).

Figure 5.3: DAC work flow

Figure 5.3 illustrates the workflow of the software. The experimenter begins by creating a
volume, and clicking on pixels in video frames representing the participant’s skin. The coordinates of the point are read and the RGB values at that point are converted to CIE-Lab space. The point’s CIE-Lab values (say \( l, a, b \)) are stored in a point list. With this point, a point with one value higher i.e., \((l+1, a+1, b+1)\), and a point with a value lower i.e., \((l-1, a-1, b-1)\), are also stored. This creates a sphere of radius one unit in CIE-Lab space and the video frame is thresholded on the end bounds of the sphere (see Figure 5.5). This generates a mask where all the points in the frame’s CIE-Lab space corresponding to the skin’s CIE-Lab, are white and rest are black (see Figure 5.4).

When more points representing skin on video frames are clicked, those points are added to the point list, generating an ellipsoidal solid in CIE-Lab space. Hence, the frame gets thresholded for wider range of values in every channel of CIE-Lab space. Figure 5.6 shows more skin as more points are clicked. The skin shown in the figure is essentially the masks overlaid on the video frames.

The researcher can also add multiple 3-D volumes in CIE-Lab space to accommodate different color ranges for things like freckles, nail polish, tattoos, etc. Once a new volume is created, clicking on points will keep adding points to that ellipsoidal volume in CIE-Lab space. Thresholding takes place separately for every volume which generates a mask for every individual volume. To include these volumes, the binary masks are added to give us the volume mask. The CIE-Lab space will have multiple volumes as shown in Figure 5.7.
Figure 5.5: The 3-D volumes in CIE-Lab space, corresponding to specified skin range

Figure 5.6: Visualization of DAC
The researcher can also select pixels in the video frames that have been mistakenly identified as skin because they lie within one of the 3D color volumes, even though they do not represent skin. By selecting those points, the experimenter can create voids; spherical volumes in color space which are then excluded from all color volumes. Binary masks are generated for these spheres as well. All the masks corresponding to the spheres are summed up to obtain a void mask. Finally, the void mask is subtracted from the volume mask to obtain the final mask which has the pixels corresponding to the skin with data. Figure 5.8 shows the video frame, volume mask, void mask, unfiltered and filtered output.

**Quality check**

To clean the image i.e., to smooth the image and to remove small regions of noise, I apply morphological operators to these masks.

To smooth the volume mask and void mask, I perform closing followed by opening with an
Figure 5.8: Original frame, masks, filtered and unfiltered outputs
ellipsoidal structuring element of size 9x9 on the thresholded mask of size 1280x720. Closing first helps in filling up the small gaps and opening after that helps to smooth the shape. A structuring element of 7x7 is too small for an image of size 1280x720. It is almost 103 times smaller which means that the filter operates on about \( \frac{1}{103^2} \) of the image at once. A 7x7 filter only fills up smaller gaps and opening for this big an image is slightly edged. Using a structuring element of 11x11 is a little too big (close to \( \frac{1}{66^2} \) of the image) as it makes the noise points bigger and over-smooths the hand-edges as opposed to retaining the shape.

To remove salt and pepper noise from the final mask and to further refine the shape, opening followed by closing with an ellipsoidal structuring element of size 3x3 was performed on the thresholded mask of size 1280x720. Increasing the structuring element comes with a bargain of adding extra pixels on the hem of the hand. The final filtered output is shown in the Figure 5.9.

For more information on the operations, see Section 3.2.

Smoothing of the image is required else the "skin" pixels will have very sharp edges instead of the smooth hem of the hand. As much as filtering helps get high quality data, it does not filter out the false positives of skin, which are objects that lie in the same CIE-Lab color space as skin.

One key thing to keep in mind is that all the morphological operations performed above are to help a researcher select the skin points and see the effect of the points chosen in DAC. These points shall set the upper and lower bounds of the 3D volume in CIE-Lab color space. For processing of
the frame, a different set of morphological operations is applied.

5.2 Hand localization

Once the bounds of the skin are known in CIE-Lab color space, it is possible to isolate the hands. This process was split into two parts. The first part is to obtain a binary hand-mask where all the pixels corresponding to the hand are white and rest are black. The second step is to find hands in that mask.

5.2.1 Hand-mask generation

To generate a hand-mask, Two approaches are implemented. One is based on detection/identification where I make use of the CIE-Lab color bounds obtained from the prior step. The other approach is based on recognition where I use the UNet convolutional neural network [Ronneberger et al. 2015] to predict the hand-mask.

Detection/identification

The video is looped to process every frame individually. Each frame is thresholded based on the skin CIE-Lab volume boundaries obtained from DAC and then cleaned using morphological processing. The other pixels representing skin in the video frames are the experimenters’ face and hands which are at a larger distance and many fewer pixels correspond to that data. The thresholding technique is susceptible to other objects that lie in the same CIE-Lab space as the skin itself.

To remove the noise points first and then to smooth the volume mask and void mask, I perform opening with an circular structuring element of 5x5 and dilate it with a rectangular structuring element of size 9x9. Opening with a smaller structuring element removes the specs of noise. Dilating with a square structuring element of 9x9 which is about \( \frac{1}{80\%} \) of the image, gives a sharp hem of the hand which is what will be required in further processing.
CHAPTER 5. HAND-GRASP PIPELINE

Recognition

Even though getting rid of false positives like the experimenter’s hands or face is easy in the pipeline, it would be even better to avoid the false positives in the first place. This helped me compare detection/identification with recognition. Since the goal here is to identify and obtain the hand, I pass my images through a few networks to understand what will help me achieve the end result.

To detect a hand, I first tried object detection using Faster-RCNN [Ren et al., 2015]. The output generated by Faster-RCNN network on my data is shown in the Figure 5.10. The output shows that the network detects hand as person with high confidence. The output also shows the bounding box generated by the network for different labels. Person is shown in light green color, apple is shown in off-white color and the dark green is television. However, to obtain a hand-mask, I would have to use my CIE-Lab thresholding on the bounding box, rendering/leaving the use of a network unjustified.

So the next concept I looked into was instance segmentation using Mask-RCNN [He et al., 2017] where a mask is generated with the bounding box. Here, the network essentially differentiates the various instances of objects and can tell if one object is occluded by another object, by generating a mask with the bounding box. The output generated by the Mask-RCNN network on my data is
shown in Figure 5.11. The output shows that the network detects hand pixels as ‘person’ (shown in light green color) with high confidence. It is also observed that one pixel can get classified into two classes. The mask for an object overlaps with other object masks. I was expecting to obtain predictions of the fingers which get occluded by the banknote (see Figure 5.12) but the result obtained was unexpected. The network also predicts other objects like a remote, laptop, and an apple.

Figure 5.11: Mask-RCNN outputs

Figure 5.12: Mask-RCNN Expectation and reality

To test this use of semantic segmentation to give pixel-by-pixel labels, I used Deeplab [Chen et al., 2017]. The output generated by the network on my data is shown in Figure 5.13. The output shows that the network identifies pixels in the hand as ‘person,’ pixel by pixel. The output
doesn’t show the confidence but it can be obtained from the network. However, the output did look promising. The use of transfer learning with Deeplab to predict a pixel either as background or hand could be useful, but re-training a layer of a deep network requires a lot of labeled data which wasn’t available.

Transfer learning one layer requires a lot of images, and training a whole network requires even more. It was suggested\textsuperscript{1} that I use UNet \cite{Ronneberger et al., 2015} which was created to perform image segmentation. The training strategy and the network were designed to rely on data augmentation for enlargement of a limited quantity of labeled data. The advantage of the augmentation is that it helps the network learn with limited training data. The architecture of the network is shown in Figure \ref{fig:unet}. It is based upon a fully convolution network that consists of a contracting path and an upscaling path. Connections are made in between those paths so the network can extrapolate the missing context from the input. Each block on the contracting path consists of two convolutional layers with filter size 3x3 followed by a rectified linear unit (ReLU), batch normalization and max pooling (2x2). Each block on the upscaling path starts with upscaling.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{unet.png}
\caption{Deeplab outputs}
\end{figure}

\textsuperscript{1} By lab mate Aayush K. Chaudhary
the input by convolving with a 2x2 filter, then consists of two convolutional layers with filter size 3x3, followed by a rectified linear unit (ReLU) and batch normalization. Note that the batch normalization was not included in the original network, but it was included in this implementation. Data augmentation usually involves normalizing the data by subtracting the mean of the dataset and dividing by the standard deviation. However, the dataset used is not normalized at the time of input, and to compensate for the same, batch normalization is used in the model, so during training process, the model is exposed to the normalized information.

![UNet architecture](Figure 5.14)

Requiring less data to train was one of the primary reasons to choose the network. To create one ground truth label, the outline of the hand was marked by manually clicking with a mouse on border points along the outline (see Figure 5.15(b)). The resulting shape was filled and saved as a mask (see Figure 5.15(d)). To verify the mask shape, the mask was overlaid on the original video frame as shown in Figure 5.15(c). Creating ground truth took about 7 minutes per image, which is
Figure 5.15: Network input and output
very time consuming. Labels were created for a hundred images which took about three days.

Figure 5.16: Various skin tones in the dataset

The entire UNet implementation was in PyTorch 1.1.0. The network was trained with 440 images and tested with 60 images. The dataset comprised of video frames from four participants having fair, medium, olive and dark skin tones (see Figure 5.16). Most of the images in the training data had two classes i.e., background and hand. The training data also included some ‘negative’ images i.e., images that didn’t contain any pixels classified as participants’ hand (see Figure 5.17 so those images had only one label which was the background).

The ground truth was created for only a hundred images. The data augmentation included flipping the images horizontally and rotating the image by a random angle between +5 and -5 degrees. This augmentation was used to enlarge the dataset by five times giving 440 training images and 60 test images. Both original image and ground truth were down-sampled from 1280x720 to 352x352 with anti-aliasing. The network architecture implemented was a modified

† Current stable release is version 1.1.0
https://pytorch.org/docs/stable/
Figure 5.17: Images with no skin pixels in the dataset

version of [Shah, 2017] with the modification on the size of the input and output. The original input and output of the model was of size 572x572 while my input and output were of size 352x352. The batch size was four. This image size and batch size were small due to memory limitations. In addition to data augmentation, the training strategy also included K-Fold Cross Validation [Stone, 1974] with k = 10.

\[
IoU = \frac{P \cap G}{P \cup G}
\]

Figure 5.18: Intersection over Union [Chaudhary and Pelz, 2019] (Figure 4)

The neural network was trained for 75 epochs on Ubuntu 16.04 with Nvidia Titan X. The training took about 13-15 hours. The loss function used was Cross Entropy Loss in 2-dimensions (i.e., pixel by pixel). Adam Optimizer [Kingma and Ba, 2014] was used for regularization with a learning rate of 0.00001, first exponential decay rate of 0.5 and second exponential decay rate of 0.999. The accuracy was measured in terms of Intersection over Union (IoU) [Everingham et al., 2005] illustrated in Figure 5.18. G is the ground truth and P is the prediction. If the ground truth
and prediction are the same then their intersection and union will be the same which will result in IoU 1. If there doesn’t exist any overlap then the intersection will be zero which will result in IoU of 0.

Figure 5.19 shows the IoU accuracy and loss plots of the model while training. The accuracy of the model on the test images (including the ones which the model has never seen) was 91.63%.

The hand-mask predicted by the network had a thin holed up boundary (see Figure 5.20(b)). This may be because of down-sampling. To clean this, I performed erosion twice using a circular structuring element of size 3x3 and then upscaled this mask to the original input size using nearest-neighbour resampling.

### 5.2.2 Hand-mask processing

After cleaning, all the contours that remain in the hand-mask are now skin. Hand-masks generated by both methods will have fingers as isolated contours and the wrist/hand as large contours. The hand-mask obtained using CIE-Lab thresholding also had some false positive contours. We assume that the hands/wrists will have the most skin, which will correspond to the biggest contours in the
CHAPTER 5. HAND-GRASP PIPELINE

Figure 5.20: Network input and output

(a) Input of the network: original and ground truth (1280x720 each)

(b) Output of the network (352x352)
clean image. So we find the counters with largest area and second to largest area. This assumption is valid for my case of finding hands from egocentric point of view since the object (i.e., hands) closer to the participant will occupy more pixels in the image captured from the scene camera.

Figure 5.21: The ROI window drawn around each hand

Using the circumcenter of the two found contours as the center of 2 hands, we create a region of interest (ROI) (see Figure 5.21).

This step is similar to the eye-gaze pipeline where a window is created of pixel size 161x141 (height x width) around the fixation point. The window was chosen to be rectangle to keep track of correct indexing of row and column.

For the Hand-Grasp Pipeline, window was increased by a factor of 1.5, resulting in an ROI of size 211 x 241 (height x width). The window was kept in the horizontal orientation because the hands were close to the lower edge of the image which shortened the height of the ROI.

Correct detections are shown in Figure 5.22, while a false detection is show in 5.23(a). The assumption of a frame having two hands holds true most of the time but vigorously finding two contours as two hands may result in unexpected results as in Figure 5.23(b), which shows one arm as two hands because of the presence of the bracelet, which broke the hand and forearm into two large contours.

The flawed cases are very easily tackled further in the pipeline.
(a) Two hands in hand-mask generated by CIE-Lab thresholding
(b) Two hands in hand-mask generated by UNet

Figure 5.22: Correct hand detection

(a) Two hands identified as one contour
(b) Forearm bigger than other hand

Figure 5.23: Partially incorrect hand detection
5.3 Object identification

After identifying pixels representing the hand (or hands) and the respective ROIs, the next step is to find what object the participant was holding. SIFT features were used to perform this function. Before finding the object in the hand, all the SIFT key-points of all the objects in the template (which is also known as the reference image or refImg) are first detected and stored with their descriptors in a list. The key-points for the front side of $1 are shown in the Figure 5.24.

![Figure 5.24: The refImg with the key-points shown on $1 bank note](image)

SIFT features are detected in the ROI and the ROI’s key points and descriptors are compared to every object’s key-points and descriptors in the list, giving the number of feature matches per object in the list. Since the number of SIFT features in the ROI varies from frame to frame and is dependent on how much of the object is clearly visible, the number of matching features also change. To set a baseline value for comparison, the highest number of matching points for a ROI...
is set as the maximum value for that ROI.

Initially, only the objects whose number of matching points were in the upper quartile from the maximum value were processed. For instances when the object is clearly visible with distinctive features, the upper quartile resulted in one or two objects that matched the ROI best. However, for other cases, it was observed that 75% was too harsh a threshold as the correct object did not always have the maximum number of matched features. Features common amongst the objects like the seal were the reason for the correct object to not be the one with the highest number of matching points.

To address this, the five objects with the highest number of matching points were identified, and the maximum value was set as the highest number of matching points, then objects with matching points more than 70% of the maximum value were processed. If an object is clearly visible, the 70% threshold is enough to give only two or three objects but if only the common features are matched then all five objects get compared. The criteria of top five was lenient enough to never drop the correct object.

Every ROI in every video frame was processed individually but the processing steps remain the same for all and are independent i.e., no process/step’s output for any ROI depends on another ROI’s processing.

5.4 Object simulation

For every banknote that made through the 70% cut, its orientation in camera space was calculated. This orientation was found using the homography between the match-points in the video frame and the refImg. To be more precise, this orientation was found by calculating the homography between the object SIFT features and ROI SIFT features. RANSAC was used to select the set of matching features. Since RANSAC is random in nature, a simple check was applied to make sure that the points selected were sparse in nature. The template of the object was split into a 5x5 grid. Since we
had the 70% of the best five objects, every object was processed one after another. The normalized
distribution of the selected features in the object template were then found.

The distribution is calculated by creating a histogram of 5x5 and clustering the key-points
(see Figure 5.25). If any single cell of the grid has more than 70% of the key-points, then the
homography is calculated, but because many points are tightly clustered in a small region it may
result in a homography that is only "almost correct" (which will be referred to as almost correct
homography). All the cases of almost correct processing are flagged for manual intervention. This
first homography is actually calculated three times in a loop. This RANSAC check helps flag cases
where only parts of the bank note are visible/accessible for feature extraction. We then add quality
control on the homography to throw out the bad projections.

\[ H_{3 \times 3} = \begin{bmatrix} A_{2 \times 2} & t_{2 \times 1} \\ v_{1 \times 2}^T & \mu_{1 \times 1} \end{bmatrix} \] (5.1)

\[ H'' = H / \mu \] (5.2)

\[ H''_{3 \times 3} = \begin{bmatrix} A''_{2 \times 2} & t''_{2 \times 1} \\ v''_{1 \times 2}^T & 1 \end{bmatrix} \] (5.3)

\[ M_{2 \times 2} = A''_{2 \times 2} - [t''_{2 \times 1} \ast v''_{1 \times 2}] \] (5.4)

The homography matrix can be split into its components where A is affine, t is translation, v is projection and \( \mu \) is the heterogenous plane. We calculate the homography matrix (\( H'' \)) in a homogeneous coordinate system. M is then calculated which is essentially the affine matrix without the translation and projection components. The determinant of M offers insight into whether or not the calculated homography will result in proper rotation, scaling and shear of the template. If the determinant is less than or equal to zero (i.e., \( |M| \leq 0 \)) then the homography will be bad i.e., it will result in unrealistic or impossible projections (see Figure 5.26).

Figure 5.26: Homography with a zero determinant
If the determinant is more than zero (i.e., |M| > 0), then the orientation of the template, after warping, will be close to the orientation of the object in the video frame (see Figure 5.27). In other words, the simulation of the object in the video frame from the template will be successful.

Since a homography is a plane-to-plane mapping, it fails in the cases when an object is not in a plane, as when an planar object bends significantly (or a banknote folds).

The condition |M| > 0 removes bad homographies (e.g., cases of folding objects) but it doesn’t adjust/fix the *almost correct* homography, i.e., the case seen in Figure 5.28 which is not completely wrong but it isn’t good enough to be used. The case of *almost correct* homography occurs because the SIFT features are compared from the template to the ROI (see Figure 5.29) and it is not necessary that the ROI comprises of a clear/complete view of all the edges of the object (see Figure 5.30).

If the |M| > 0, the object template is warped to match the orientation of the object in 3-D space from the scene camera’s perspective. This wrapped object template is called a *modified object*. Using the homography calculated for this, the boundary of the object in the scene is found and cropped. This is a *cropped object* (see Figure 5.31).
Figure 5.28: Homography with a non-zero determinant but a big projection difference

Figure 5.29: First homography calculation between template and ROI
Figure 5.30: ROI with unclear/incomplete view of the object’s edges

Figure 5.31: Dynamically found boundaries of the object in the scene and cropped out
To get more accurate results on the *almost correct* homography, we perform a double homography calculation to get a more precise 3-D orientation of the bank note. I find SIFT features in the *cropped object* and in *modified object* and compare their key-points and descriptors to find a second homography matrix (see Figure 5.32). Based on this second homography, I re-warp the *modified object* to get *simulated object* as shown in Figure 5.33.

![Figure 5.32: Second homography calculation between cropped object and modified object](image)

To get a good *simulated object* along with a good *cropped object*, two binary masks are created; one for the *cropped object* and another for the *simulated object*. The two are compared (i.e., subtract and sum all the absolute values) and discarded if big differences occur between the two because that indicates a bad homography with positive $|M|$ (See figure 5.28).

### 5.5 Back-projection of the hand onto the template

To back-project the hand holding the object, the hand has to found on the object. Since we have the hand-mask generated in Section 5.2.1 we crop it using the same boundary as we used to obtain the *cropped object*. If no skin pixels are found in the cropped hand-mask, the next frame is processed.
Figure 5.33: Outputs after warping with first and second homography

Figure 5.34: Projection of the hand on the bank note template.
Otherwise, it is referred to as thumbs on note. The inverse of the second homography and the inverse of the first homography are then calculated. The thumbs on note is warped twice: first with the inverse of the second homography and then with the inverse of the first homography. This takes the thumbs on note from camera space/modified object space to cropped object space to reference image/template space [See figure 5.34].

Once the skin is identified on the reference banknote template, it is called hands on template. To further refine and verify the identification of pixels representing skin in the image, a downsampled version of the hands on template is processed. The original 100x239 template is downsampled to 25x59 per object. The size is reduced to save computing resources such as time and space as every 'skin' pixel on the template will be written out to CSV file.

Since the aim is to find the thumbs and fingers on the bank note, the area of the target can be estimated with respect to the downsampled banknote. Constraints are added to the contours to limit them to regions that can reasonably represent hands (or portions of hands): their area should be more than 2 pixels i.e., > 0.14% of the bank note and less than 590 pixels i.e., < 40% of the bank note.

This removes salt and pepper noise while taking care of the case where a bill is in the background of the hand, getting occluded by the hand but showing up in the ROI window around the center of the hand, a relatively common situation (see Figure 5.35).

Figure 5.34 shows the processed result of the hand-mask that was generated using CIE-Lab thresholding, the Figure 5.36 shows the processed result of the hand-mask generated by UNet.

The pipeline does process well to give multiple different object detections, as seen in Figure 5.37.
Figure 5.35: Occlusion
Figure 5.36: Pipeline output using UNet predicted mask

Figure 5.37: Multiple ROIs with correct hand analysis
5.6 Drawbacks in the pipeline

The end result of obtaining the hand pixels that held an object, on the correct object template is met. Multiple good matches are also found implying the Hand-Grasp Pipeline is not limited to processing only one instance of object or hand. However, the pipeline also has some flaws which are elaborated below.

5.6.1 False positives

The pipeline using the hand-mask generated by CIE-Lab thresholding still suffers from false positives of the skin. Since the objects I have used are banknotes, the Icons of Freedom (IoFs) (see Figure 5.38) on the $10 bill correspond to the same CIE-Lab values as most participants’ finger nails, because regardless of the various skin tones of all the participants, finger nails in CIE-Lab color space is nearly same for everyone.

![Figure 5.38: $10 bill with marked icons of freedom](image)

To process these false positive contours, we take advantage of the fact that the hand always extends beyond the edge of the object. Thus, if the contours in a cropped object are floating islands/not attached to the edge of the object, then those contours cannot be hand/fingers/thumbs. Islanding was implemented to eliminate floating contours such as caused by the IoFs. To implement this concept, I create a 2-pixel boundary on the inner side of the $10 bank note and then look for contours that touch the boundary. If the contour touches the boundary then the contour started
from the edge, thus, it was a finger or a thumb. Islanding is implemented for every bank note after finding the *thumbs on note* and before finding the area of the contours. Keeping it in that sequence eliminates floating contours earlier in the pipeline.

*Islanding* works as long as the false positive is not 'attached' to the hand, because then that false positive passes through the pipeline as part of the hand. Figure 5.39 shows an IoF to be thresholded and *islanding* does eliminate it. UNet was implemented to generate the hand-mask which would be free of false positives and it does exclude all false positives more than 90% of the time.

### 5.6.2 Repeated analysis

Since every [ROI] gets processed individually, if the hands are too close to each other (usually when one is about to fan out the bills) then the prediction per [ROI] is same, which means we obtain dual instances of the same piece of data. Figure 5.40 shows the dual instances of the same data being calculated. This is because the hands are so close to each other, that each [ROI] takes in information of the other [ROI] as well.

To tackle this issue of repeated analysis, I make use of two flags namely, flagForCenter0Success and areaOfOverlap. If the area of overlap between the two regions of interest is more than 20% then the flag sets. flagForCenter0Success sets when the any one [ROI] gets analyzed successfully. At the end of analyzing second [ROI] for the one video frame, if areaOfOverlap and flagForCenter0Success both are set then there is a possibility of result repetition and this should be confirmed by manual intervention.

### 5.6.3 Multiple predictions per hand

For bank notes underneath the top bank note, multiple predictions are made per [ROI] Figure 5.41 shows multiple bank notes analyzed for one [ROI]. It also shows how the same piece of information gets processed twice. This occurs because the [ROI] is big and fixed in size. To tackle this, per [ROI]
(a) $10 bill with icon of freedom as false positive

(b) Zoomed in part of the hand-mask on the $10 banknote which includes the IoF

Figure 5.39: Pipeline output for false positive
a counter is set which counts the number of predicted objects that were successfully analyzed. A note is made of the counter if it is more than one.

5.7 Output verification

Because I was aware of the pipeline having some flaws, I created the *fail safe* which saves the ENTIRE processing information. The x,y coordinates of every pixel in the final *hands on template* image that represents fingers/thumbs is saved in a CSV file. All the images generated during processing to cross check if everything is thresholded and calculated properly are also saved.

Multiple flags are set for every frame to allow a researcher to inspect a potential flaw in the analysis of a frame. And based on these flags, we can fathom how well the pipeline performed and how capable is it of running immaculately.

For the final CSV, I write out some additional parameters and they are: the area of overlap between the two [ROI] windows of the two hands and a *definite* binary flag which corresponds to the final result being definitely correct or may requiring human supervision. The definite flag is then based off on multiple other flags namely *flagForCenter0Success, flagForArea, flagHIsDicey* and *noOfSaved*. The *flagForCenter0Success* corresponds to hand 0 being successfully processed. If hand 0 is successfully processed, only then we calculate the area of overlap between the two [ROI] windows. The *flagForArea* corresponds to the area of overlap being more than 20%, and if this flag
Figure 5.41: Multiple predictions per ROI
is set, this may result in multiple predictions of the same bank note in 2 hands as the ROI window will have the same key-points and descriptors that used to find the bank note from the template. The noOfSaved flag informs us of multiple bank notes being correctly predicted for one hand and saved onto the CSV. This is when human intervention is required to check if all the bank notes that got predicted to be in one hand were correct or not. If only one bank note was predicted per hand with an area of overlap being less than 20%, then this frame is a definite case. flagHIsDicey was set if RANSAC chose 70% of points from one area (as discussed in Section 5.4). However, it was noticed that all the frames flagged by flagHIsDicey, didn’t make through the quality checks in the pipeline.

5.8 Visualization

Once I have the x,y coordinates of all the pixels that belong to the finger/thumb, I use 2-dimensional histogram to see where on the bank is the highest concentration of grasp, there-by completing my analysis of hundreds and thousands of frames in just a few days.

The Figure 5.42 shows a summary of ONLY a 100 images that were passed to the pipeline. Points are normalized within each banknote face and then the heat map is created. If there is sharpness in the heat map, it may be caused because of similar grasps to one location on the banknote. In this case, smoother the blur, more area of the pixels have been repeated detected by the pipeline. For instance, the 20$ bill looks very smooth implying more area being detected repeatedly. The 50$ bill looks very sharp which could mean, a very small region was detected repeatedly.
5.9 Performance

To understand the performance of the system, let’s understand some important terms. Ideally, we would want all the skin pixels to be classified as skin and all non-skin pixels to be classified as non-skin. However, in reality, the system makes some errors where some skin pixels are detected as non-skin and some non-skin pixels are detected as skin. To quantify these values, let us look at some metrics used in the past.

Han et al. [2006] used three metrics to understand the performance of the system namely Correct Detection Rate (CDR) which was the percent of correctly classified pixels, False Detection Rate (FDR) which was the percent of wrongly classified non-skin pixels and Classification Rate (CR) which is the percent of skin pixels correctly classified as skin pixels from the maximum of the total number of skin pixels in either the ground truth or the prediction. I have created an illustration to interpret this better (see Figure 5.43).
Figure 5.43: Metric visualization for [Han et al. 2006] where G is ground truth and P is the prediction

The ground truth is in black and the predicted mask is in red. The blue dotted lines highlight the area of correct detection while the green dashed lines highlight the area of false detection. The rate of missed skin pixels unclassified would be 100 - CDR. The interesting point here is the classification rate which depends upon the skin pixels that are correctly classified and the number of skin in the bigger of the two masks (i.e., either the ground truth or the prediction mask). Row one of Figure 5.43 shows ground truth and prediction of same size and row 2 of the same figure shows prediction of double the size of ground truth. Column-wise, the Figure shows CDR = 100%, 50% and 0%. The metrics together help understand the performance of the system.

If the entire ground truth is predicted (Figure 5.43(a), 5.43(d)), then the false detection and classification rate are dependent on the size of prediction. The same notion also holds true if half of the ground truth overlaps with the prediction (Figure 5.43(b), 5.43(e)), or if no part of ground truth is in the prediction (Figure 5.43(c), 5.43(f)).

[Phung et al. 2003] used two metrics to understand the two errors the system could make. They
used False Detection Rate (FDR) where a false detection was a non-skin color identified as skin color and False Rejection Rate (FRR) where a skin color was detected as non-skin color. I have created an illustration to interpret this better (see Figure 5.44).

The ground truth is in black and the predicted mask is in red. The green dashed lines highlight the area of false detection while the purple dash-dotted line shows the skin color that was classified as non-skin. The rate of missed skin pixels correctly classified would be 100 – FRR. The metrics used for here help us understand how bad can the system perform. Row one of Figure 5.44 shows ground truth and prediction of same size and row 2 of the same figure shows prediction of double the size of ground truth. Column-wise, the Figure shows FRR = 0%, 50% and 100%. Looking at both the metrics together, we can understand the performance in terms of error.

If the entire ground truth is predicted (Figure 5.44(a), 5.44(d)), then the false detection rate is dependent on the size of prediction. The same notion also holds true if half of the ground truth overlaps with the prediction (Figure 5.44(b), 5.44(e)), or if no part of ground truth is in the prediction (Figure 5.44(c), 5.44(f)).

The field of computer vision often uses the metric of IoU to compare the predicted semantic
segmentation mask with ground truth. IoU is a ratio of the intersection of the ground truth with the prediction upon the union of the two. Figure 5.45 shows the IoU computed for some cases to better understand the metric. Row one of Figure 5.44 shows ground truth and prediction of same size and row 2 of the same figure shows prediction of double the size of ground truth. Here, if the IoU is 50% (Figure 5.45(b), 5.45(d)), one cannot get an estimate the hit rate which could be 50% or 100%.

To evaluate my approaches for hand-mask generation namely CIE-Lab thresholding prediction mask and UNet prediction mask, I use the following metrics.

- Correct Detection Rate (CDR)

- False Detection Rate (FDR)

- Intersection Over Union (IOU)

\[
CDR = \frac{100 \times \#(Skin_{GT} \cap Skin_P)}{\#Skin_{GT}} \quad (5.5)
\]
\[
FDR = \frac{100 \times \#(\text{Skin}_P - (\text{Skin}_{GT} \cap \text{Skin}_P))}{\#\text{Skin}_{GT}} \quad (5.6)
\]

\[
IOU = \frac{\#(\text{Skin}_{GT} \cap \text{Skin}_P)}{\#(\text{Skin}_{GT} \cup \text{Skin}_P)} \quad (5.7)
\]

Here, \#\text{Skin}_P is the number of skin pixels in the predicted mask and \#\text{Skin}_{GT} is the number of skin pixels in the ground truth mask. Using these metrics, I summarize the performance of the two algorithms used in the pipeline in Table 5.1.

<table>
<thead>
<tr>
<th></th>
<th>CDR(%)</th>
<th>FDR(%)</th>
<th>IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIE-Lab</td>
<td>79.16</td>
<td>4.80</td>
<td>0.74</td>
</tr>
<tr>
<td>UNet</td>
<td>98.61</td>
<td>2.96</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table 5.1: Performance Summary

We can see that CIE-Lab does come close to picking out about 80% of the actual skin pixels but with a bargain of picking almost 5% of false positives, reducing the overall performance for the images to 0.74. On the other hand, UNet was trained on skin from four participants and performs well on the skin it was trained on. The network can correctly detect the skin it was shown with a rate of almost 99% and a false detection of about 3% which is close to CIE-Lab, however, the overall performance is at 0.96. Key thing to note here is that the table shows median of all the metrics computed and not the mean because there were some extreme predictions. The distribution of the metrics computed for a 100 images whose ground truth was manually made, is shown in Figure 5.46, 5.47, 5.48. The median in the table is marked in the figure with an orange line. In the figure, we can see that for some cases, CIE-Lab performance drops really low and it also detects five times the amount of ground truth as false positive for one instance. This is the reason median is chosen instead of mean as mean would include the extreme cases. However, the table with mean values is reported in the Appendix.

Looking at the metrics, I can say with confidence that if UNet has seen the skin tones, then it
predicts better masks than CIE-Lab thresholding. See Section 6.1 for more discussion.

![Distribution of CDR](image)

(a) Correct Detection Rate  
(b) Correct Detection Rate zoomed in on 85-100%

Figure 5.46: Distribution of CDR

### 5.10 Summary

In this chapter, the Hand-Grasp Pipeline was explained which comprised of finding the skin, locating the hands, identifying the object in each hand, simulating the orientation of the object in the scene with the object template and then projecting the hand from the scene onto the template of the held object. The quality control on skin detection, comparison of hand detection with hand recognition for the part of hand localization, quality control on object identification, and multiple data checks
Figure 5.47: Distribution of FDR
Figure 5.48: Distribution of IOU
on object simulation and hand projection were performed and explained in detail. Visualization
and the performance of the approaches were also discussed.
CHAPTER 5. HAND-GRASP PIPELINE

References


Chapter 6

Conclusion

The motivation for the Hand-Grasp pipeline development was speeding the analysis process. If I already have the CIE-Lab space for a participant’s skin established, then each video frame gets processed in about 6-7 seconds which includes time for thresholding the hands, identifying the object, simulating it, back-projecting the hand and writing out the image with flags for output verification. If I use the network’s predicted mask for the hand to process the data then a video frame takes about five seconds (excluding the time taken by the network to predict the hand-mask). Both these processing techniques are faster than manually marking the pixels that correspond to the hand. However, they require some extra steps which may be time consuming.

- For using thresholding-based pixel segmentation, the experimenter has to select skin for every participant. If a participant has nail polish or tattoos on/around their fingers, then they have to be explicitly selected or else those areas won’t be processed by the pipeline. This can be very time consuming if you have a lot of participants as this will have to be done manually for every participant.

- The same is not a concern when using UNet and performing semantic segmentation. The network (i.e., UNet) is trained to identify hands which means that it will make use of the
semantics in the image and then threshold the hand. The thresholded hand will include the nails, tattoos, etc. However, the training of the network takes about 13-14 hours. Creating ground truth for the network to train, takes additional time. One label takes about 7 minutes to create. A hundred images took about three days. This too is a heavy time investment. If later in time we find out that the data was biased towards one skin tone or didn’t have enough images in the training set to differentiate between red nail polish from the experimenter’s red shirt, then the dataset will have to be adjusted and the network will have to be re-trained.

The end result of the pipeline, using thresholding in CIE-Lab space or UNet for hand-mask generation, are similar except for the Icon of Freedoms being detected as hand while pixel segmentation. The Hand-Grasp Pipeline does have some shortcomings which can be overcome in future research.

One key functioning of the Pipeline is that it processes every frame individually. There is no piece of information from prior processed frames being passed on to the current frame for processing. This makes the Pipeline very useful for processing spatial information. But because there is no information being grabbed from adjacent frames, the pipeline cannot be used for spatio-temporal analysis. In other words, one cannot analyze the sliding movement of the fingers on the object as the participant did in Figure 6. Hence, there is scope for improvement by utilizing information from prior frames and incorporating the dynamics of the interaction that will be useful for spatio-temporal analysis.

The Hand-Grasp Pipeline processes only hands. To analyze interaction data, one will require processed information for both eyes and hands. If the Eye-Gaze Pipeline can be incorporated with the Hand-Grasp Pipeline, then there is scope for the spatio-temporal analysis of interaction. Figure 6.2 shows spatio-temporal analysis conducted for a participant from another study, where the task was to sort sixteen banknotes into stacks of like denomination. The results in this figure are generated by manually analyzing the video frames. The study’s data collection was similar to
the one explained in this thesis. The analysis involved observing the temporal relationship between information gathering and manual interaction. This was defined by measuring the intervals between:

- Fixation(s) and picking up a banknote (*Pickup*)
- Fixation(s) and dropping the banknote onto its stack (*Release*)

**Pickup**

Figure 6.2(a) shows a sample distribution of *Pickup* interaction where $\Delta t = t_{\text{pickup}} - t_{\text{fixation}}$. The bimodal distribution is made up of two sub-populations,

1. A group of *Guiding Fixations*, made immediately before picking up the banknote. The mean of *Guiding Fixations* is 0.41 seconds with a standard deviation of 0.21 seconds.

2. A group of *Look-Ahead Fixations*, made at an earlier time, with an intervening event between that fixation and the *Guiding Fixation*. The mean of *Look-Ahead Fixations* is 0.94 seconds with a standard deviation of 0.33 seconds.
CHAPTER 6. CONCLUSION

(a) Pickup Fixations

\[ \Delta t = t_{\text{pickup}} - t_{\text{fixation}} \]

(b) Release Fixations

\[ \Delta t = t_{\text{release}} - t_{\text{fixation}} \]

Figure 6.2: Interaction analysis
The *Guiding Fixations* made up 61% of the Pickup fixations.

**Release**

Figure [6.2(b)] shows a sample distribution of *Pickup* interaction where $\Delta t = t_{\text{release}} - t_{\text{fixation}}$. The bimodal distribution is made up of two sub-populations,

1. A group of *Guiding Fixations*, made immediately before placing the banknote in a stack on the desk. The mean of *Guiding Fixations* is 0.5 seconds with a standard deviation of 0.3 seconds.

2. A group of *Look-Ahead Fixations*, made at an earlier time, with an intervening event between that fixation and the *Guiding Fixation*. The mean of *Look-Ahead Fixations* is 0.82 seconds with a standard deviation of 0.25 seconds.

In the *Release* interaction, the *Guiding Fixations* made up over 79% of the total fixations, and there was more overlap between the sub-populations.

The participant data analyzed was a recording of about one minute and it took over four hours to analyze it by hand. It is clear that integrating the two pipelines would offer a large benefit in such analyses.

### 6.1 Discussion

We can see that the pipeline saves time to process the data which was the end goal. To process 45 participants, it took about 3 days which would have otherwise taken months (or maybe years) to process. About 55% of the frames get analyzed of which, about 63% get analyzed without getting flagged. Of the 55% data, only 37% of the frames need to be manually checked. The 45% of the data that doesn’t get analyzed, comprises of frames that may not have the hand or may not have a bank note in the hand (i.e., the frames in between the trials where the cashier doesn’t have any
money in his/her hand) or may have folded banknotes which the homography would not process. If the object simulation used an algorithm that can perform 2D to 3D mappings, unlike homography which can perform only 2D plane-to-plane mappings, then the amount of processed data would be higher. This shows that the pipeline on the whole does have a good scope of improvement.

However, the processing approaches come with a bargain. To generate the hand mask, I can either use CIE-Lab thresholding or UNet. Selecting a participant’s skin in CIE-Lab space may at the max take two minutes. However, if one has to process data for a 1000 participants, selecting the skin may not be ideal. UNet maybe a good choice then. But UNet too comes with a trade-off. The network was only trained with four skin tones namely fair, medium, olive and dark. Its performance on unseen skin tone was still a question and to answer this, I trained UNet with data from three subjects and tested it together with data from fourth unseen subject.

The network trained on data from four subjects is called UNet (train 4, test 4) and the network trained on data from three subjects (fair, olive and dark skin tones) is called UNet (train 3, test 4). The performance of UNet (train 4, test 4) on unseen data from the same participants it was trained on was 91.64%. And the performance of UNet (train 3, test 4) on unseen subject was 89.32%. If the network hasn’t "seen" a subject then the accuracy drops by 2.32%. To understand more about the performance, I use my metrics to compare these two networks as well. From Figure 6.3, we can see that the network trained with data from three subjects does better at correctly detecting skin from the unseen subject, however, the rate of detecting false positives also increases. Because the network has not seen the skin tone from one participant, it may not differentiate in features from the unseen subject and other objects. Hence, the IOU is lower than the IOU of the network trained with four subjects. The network that has seen more skin tones does have less correct detection as opposed to the other network trained with three subjects, but its false positive rate is also low, thereby increasing the overall performance measured by IOU.

If one has to use this pipeline to process data for 20 participants, it is easier to pick out the CIE-Lab skin values but the same is not feasible for processing data from a 1000 participants. The
(a) Correct Detection Rate (CDR)  
(b) False Detection Rate (FDR)  
(c) Intersection Over Union (IOU)  
(d) CDR zoomed in on 85-100%  
(e) FDR zoomed in on 0-20%  
(f) IOU zoomed in on 0.75-1

Figure 6.3: Distribution of performance metrics for the two UNets
network does perform better than the CIE-Lab thresholding but at the cost of creating ground truth and training on as many skin tones as possible to perform better. We can see that the network gracefully improves as it sees more skin tones. To use the pipeline for a 1000 participants, it is possible that only 20 frames per skin tone is required by the network and it will perform at par. Or there may be a possibility that the network only requires 100 frames for certain skin tones and it can interpolate well for the missing ones.
Chapter 7

Future work

The strengths and limitations of the Hand-Grasp pipeline have been highlighted in the previous chapters. Following are some ideas/methods proposed that might improve on the system.

- Currently in the Hand-Grasp Pipeline, the center of the hand is the circumcenter of the contour that is assumed to be the hand and a ROI of fixed size is created around the detected centre. The area of skin pixels in the ROI varies depending on the pose of the hand, the number of SIFT features detected in the ROI also varies. The more hand in the ROI, the less area of the banknote in the ROI. A relationship between the area of hand and the area of the ROI can be created so the size and centre of the ROI can be dependent on the contour detected as hand. This would be dynamic ROI creation which would help with the drawback of detecting the entire arm as the hand.

- SIFT works by converting the image to grayscale and then detecting the features on the grayscale image. Figure 7.1(a) shows a part of the $10 bill in grayscale and Figure 7.1(b) shows the same cropped bill in color with circles marked on the x,y, coordinate of the SIFT features detected in the respective grayscale image. Since the template I used only had one kind of the banknote (i.e., the latest series), SIFT worked well and didn’t clash much with
other denominations. But if one is to add older series of the banknotes into the task, then **Color-SIFT (C-SIFT)** would be helpful as it would incorporate color information to compute the features and it would be able to differentiate a $10 bill of 2003 series from 2006 series.

![Gray image and SIFT visualization](image)

(a) Gray image  
(b) SIFT visualization

**Figure 7.1: Computing SIFT**

- I currently use UNet that I trained on video frames from four participants’ data which had different skin tones. The trained network was aimed to segment hands from egocentric views captured in the scene video, pixel by pixel. However, this is biased as the network has only seen limited images like a tattoo on a fair person and a nail polish on a dark skin person. The network has not learnt about light skin with nail polish and dark skin with tattoos. To tackle this problem of network’s unknown performance on unseen data due to training with limited research-centric data, one can use a neural network that was already trained with a million images of the hand from an egocentric point of view. This doesn’t limit to using images in which the hand holds something, it could also be an egocentric view of someone communicating in sign language. The advantage of such a network would be that it would have learnt what a hand of various skin tones under various lighting conditions looks like from an egocentric view for various purposes such as writing, signing, clapping, and so on.
• To train the neural network, a lot of data is required along with ground truth and to create an individual video frame’s ground truth, it took about seven minutes which prevented a creation of a large dataset. To make ground truth creation a little smoother, tweening was looked into. An example of a tweening animation is a face morphing into another face. The movement of the common features from start frame to end frame is computed and implemented along with alpha blending. Figure 7.2 shows three frames in sequence, apart from one another by eleven frames. We can almost guess the motion of the hand. If we use tweening animation to generate the transition frames for two frames shown in the Figure 7.2 we would have created ground truth for 10 frames within five minutes as opposed to investing 70 minutes. If the motion of the hand can be tweened like this, temporal analysis like the sliding motion of the finger could also be analyzed. I implemented alpha blending which generated the transition frames in less than a minute. However, I couldn’t implement the motion of features in time which would be the next immediate step for creating more ground truth with this method.

![Figure 7.2: Frames to be tweened](image)

• The data collected is from egocentric point of view, hence, the entire interaction is recorded from the same view and if we fit a skeleton to the hands in the image, more information about the interaction can be obtained. [Zimmermann and Brox (2017)] has created a deep network to estimate the 3D pose of a hand from a static [RGB] image. This is helpful as most pose estimation networks use depth information but this network only uses the [RGB] image. Figure 7.3 shows the skeleton drawn on a [RGB] image. If we obtain the information of the two
hand poses, we can quantify the poses and can easily estimate if the two hands are touching the same bill to avoid repeated analysis. Using the simulated object and two hand poses, the hand grasp on every object can be quantified for every hand. We can also estimate the point of contact of the hand with the banknote on the back side of the note which is away from the camera. This may be difficult but could be tried.

![Hand pose drawn on a RGB image](Zimmermann and Brox 2017) (Figure 4)

Figure 7.3: Hand pose drawn on a RGB image [Zimmermann and Brox 2017](Figure 4)

References

Bibliography


Appendix

Procedure

This is the procedure we followed while collecting the data.

• Participants were greeted upon arrival at the laboratory, the basic premise and protocol for the study were explained, and they were informed that they may end participation at any time, for any reason, with full compensation.

• If the participant agreed to proceed after reading the informed consent form and having the opportunity to ask questions, they then signed a consent form approved by the Rochester Institute of Technology Institutional Review Board.

• If the participant was wearing mascara, which interferes with the eye tracker, they were asked to remove it, but were provided with a complimentary pack of mascara for re-application at the end of the experiment, or to take home.

• The participant was also asked if they would volunteer to use a curling tool to curl their eyelashes, which helped to mitigate occlusion of the pupil during tracking.

• Eye height was measured by asking the participant to stand in front of a measuring tape (see Figure 7.4(a)), and the measurement was used to set the height of the table (see Figure 7.4(b), 7.4(c), 7.4(d)).
• The participant then sat in a chair (i.e., immobile chair in Stage Gaze and temporary chair in Stage Grasp), and the eye tracker was placed on their head, and adjustments were made to the camera position to improve eye-image quality.

• The height of the table was adjusted so the field of view (FOV) of the scene camera covered the entire workspace (see Figure 7.4(e)). If the FOV didn’t cover it, further minor adjustments were made in the height of the table.

• The calibration procedure was explained, and the participant engaged in a practice calibration procedure. Instructions for respective tasks were also given. After a final inspection, the experimenters initiated data recording.

• The participant was asked to complete multiple calibration procedures. If necessary, a calibration procedure was repeated until a satisfactory calibration was achieved.

• Recording was halted at the end of data collection, the tracker was removed, and the participant asked to fill out a demographic form, financially compensated for their time, and offered a copy of the signed consent form before the end of the experimental session.
(a) Measuring the eye height of the participant
(b) Adjusting the table’s height
(c) Side view of the adjusted table
(d) Participant looking at the center of the workspace
(e) Field of view from the tracker

Figure 7.4: Adjusting the experimental setup to the participant’s eye height
# Performance Summary

<table>
<thead>
<tr>
<th></th>
<th>CDR(%)</th>
<th>FDR(%)</th>
<th>IOU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIELab</td>
<td>76.77</td>
<td>15.77</td>
<td>0.66</td>
</tr>
<tr>
<td>UNet</td>
<td>98.11</td>
<td>3.66</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Table 7.1: Performance Summary showing the **mean** value of the distribution shown in Figure 5.46, 5.47, 5.48