Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

A method for objects extraction from recorded lectures.

Bachelor thesis

2016
Ondrej Jariabka

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

A method for objects extraction from recorded lectures.

Bachelor thesis

Study program:	Informatics
Field of study:	2508 Informatics
Department:	Department of Informatics
Supervisor:	RNDr. Zuzana Černeková, PhD.

Bratislava, 2016
Ondrej Jariabka

47689099

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Ondrej Jariabka
Študijný program:	informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: A method for objects extraction from recorded lectures.
Metóda na extrahovanie objektov z nahrávaných prednášok.

Cieľ: Nájsť alebo vytvoriť vhodnú databázu videí, v ktorých sa objavujú školské tabule.
Naštudovať metódy detekcie a sledovania objektov.
Navrhnúť a otestovať metódu na extrakciu školských tabúľ z videí bez oklúzie inými objektmi. Generovať kľúčové snímky s poznámkami z prednášky.

Vedúci:	RNDr. Zuzana Černeková, PhD.
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	prof. Ing. Igor Farkaš, Dr.
Dátum zadania:	28.10.2015

Dátum schválenia: 28.10.2015 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

....
študent vedúci práce

47689099

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Ondrej Jariabka
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science, Informatics
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title:	A method for objects extraction from recorded lectures.
Aim:	Find or create dataset of videos containing white-boards and blackboards. Study detection and objects tracking methods. Propose and test blackboards and white-boards extraction methods from videos without occlusion. Generate keyframes with lecture notes.

Supervisor:	RNDr. Zuzana Černeková, PhD.	
Department:	FMFI.KAI - Department of Applied Informatics	
Head of department:	prof. Ing. Igor Farkaš, Dr.	
Assigned:	28.10.2015	
Approved:	28.10.2015	doc. RNDr. Daniel Olejár, PhD. Guarantor of Study Programme

....

Student Supervisor

Acknowledgement: I would like to thank my supervisor RNDr. Zuzana Černeková, PhD. for her guidance and support throughout writing this thesis. Special thanks goes to my college Marek Šuppa for all his help and encouragement.

Abstrakt

Kvôli stále sa zvyšujúcemu množstvu e-learningových materiálov, ako sú prednáškové videá alebo on-line video kurzy, sme sa rozhodli vytvoriť aplikáciu, ktorá môže pomôcť študentom alebo tvorcom e-learnigového obsahu v ich snahe pripraviť študijné materiály. Hlavným cieľom našej aplikácie je vytvárať slajdy z daného videa, na ktorom je čierna alebo biela tabuľa bez akýchkoľvek objektov, ktoré by bránili uživateľovi vo vyhľade na tabuľu, ako napríklad prednášajúci stojaci pred tabuľou. Slajd by mal obsahovať len platné informácie z kľúčových snímok daného prednáškového videa. Tieto slajdy, za predpokladu, že tabuľa je v statickom videu, sú vyrobené tak že sa získa tabuľa z jednotlivých snímok, ktorá je potom rozdelená na rovnako veľké obdĺžnikové bunky. Tieto bunky sú uložené a sledujeme ako sa informácia mení v ich vnútri. Potom je výsledný slajd vytvorený z uložených buniek, keď sú všetky bunky dostatočne stabilné.

Kľúčové slová: školská tabuľa, prednáška, slajdy, e-learning, generovanie kľúčových snímkov, generovanie slajdov

Abstract

With raising amount of e-learning materials such as lecture videos or on-line video courses, we decided to develop an application, which can help students or content creators in their effort to prepare study materials. Main goal of our application is to create slides from given video, depicting a black or white board without any occluding objects such as lecturer standing in front of this board. Slides will contain valid information from key frames of the given lecture video. Based on the assumption, that the board is static in the video, this is done by extracting the board from video frames, which is then segmented into equally sized rectangular cells. These cells are stored and the change of information inside them is tracked. Afterwards, the final image is created from saved cells when all cells are sufficiently stable.

Keywords: whiteboard, blackboards, lecture, generation of images, slides, e-learning, video presentation perceptual hashes, board extraction, key frame generation, slide generation

Contents

1 Introduction **1**

2 Related Work **3**

3 Fundamentals **5**

3.1 OpenCV 5

3.2 Thresholds 5

3.3 Feature detection 6

3.3.1 Keypoints 7

3.3.2 Edges 7

3.3.3 Main approaches 7

3.4 Edge detection 8

3.4.1 Canny edge detection 9

3.5 Perceptual Hashes 9

3.5.1 Average hash 10

3.5.2 Difference hash 10

3.5.3 Perceptual hash 10

4 Preprocessing **12**

5 Board extraction **14**

5.1 Histogram method 14

5.2 Region growing method 15

5.3 Processing output of board extraction methods 17

6 Occlusion removal **18**

6.1 Initialization 18

6.2 Region growing based method 19

6.3 Absolute difference based method 20

6.4 Slide generation 20

7 Generation of final slide **22**
7.1 Feature detection approach 22
7.2 Perceptual hash approach 23
7.3 Comparing and storing slides 23

8 Dataset and evaluation **24**
8.1 Dataset 24
8.1.1 Board extraction 24
8.1.2 Occlusion removal 25
8.2 Evaluation 26
8.2.1 Board extraction and occlusion removal 26
8.2.2 Change detection 27

9 Results **28**
9.0.1 Board extraction results 28
9.0.2 Occlusion removal results 29
9.0.3 Change detection results 31

10 System limitation **33**

11 Conclusion **35**

Appendix A **39**

List of Figures

2.1 Example of input from Shin, Hijung Valentina, et al. [1]. 4
2.2 Output of Monserrat, Toni-Jan Keith Palma, et al. [2]. 4

3.1 Thresholding operations 7

4.1 Images of original video 13

5.1 These images show main steps in board extraction processing pipeline. Note, in the second image present error area caused by high intensity light. 15

6.1 These images show main steps in objects removal processing pipeline. . 19

7.1 Result of brute force matching between two generated slides. Note, that features were often matched incorrectly based on similarity of individual symbols. 23

8.1 Sample images of ground truths used for evaluation of propossed methods. 25
8.2 GUI of toolkit programs used to create datasets for our methods. . . . 26

9.1 Invalid occlusion masks generated by region growing and absolute difference-based methods. 30
9.2 Example of final slides generated by our pipeline. 32

10.1 Invalid slides generated with our pipeline. 34

Chapter 1

Introduction

With increasing amount of lectures, courses and other e-learning materials available on-line, it is becoming more and more apparent, that students as the primary consumers of such content, lack tools necessary for its usage in an effective fashion. Given the rise of massive open on-line courses [3] a strong push for creation of instructional videos, can also be seen in academic environments. As stated in [4]: "fast expansion of the Internet and related technological advancements, in conjunction with limited budgets and social demands for improved access to higher education, has produced a substantial incentive for universities to introduce e-learning courses". In [5] the authors also claim that "many users stop their on-line learning after their initial experience". They further state that "instructor attitude toward e-Learning, e-Learning course flexibility, e-Learning course quality, *perceived usefulness*, *perceived ease of use*, and diversity in assessments are the critical factors affecting learners' perceived satisfaction".

It is not difficult to find inefficiencies in the ways instructional video content is most often consumed. For example, it is very impractical to always rewind on-line lecture to get to the exact point where specific detail was discussed, pause the video, essentially "copy" the information from the paused video frame and then continue watching. Not only does it break the user's focus but it also requires additional interaction with the video content, that might result in undesired increase of the user's frustration which is certainly not desired.

For this reason, we decided to develop a tool, that could help potential viewers extract information from these presentations or lectures into a more practical format. This could also help content creators to prepare and create content which can be then perceived more easily. As stated in [5] *technology* is one of the main factors affecting user satisfaction. We developed and implemented our system as a tool, that others can insert into their pipelines. For instance, our system can process video directly from the camera and output can be then shown to a student participating in on-line lecture, presentation or course on a website. This also means, that we impose requirement on the speed of our system. Our tool should be able to generate valid slides in real-time. This can also help developers to create better systems for consumption of e-learning content and potentially increase perceived "ease of use" of such a system.

In our work, we focus on whiteboard and blackboard based lectures or presentations. The methods we present select key frames, from which a slide containing the information previously shown on board is created. Another requirement we enforce, is that these slides should contain only information that is on the board, and therefore any occluding objects ought to be removed from the scene and information behind them should also be shown on our final slide. Lastly, we also require that the slide should be generated only if the difference in information, displayed on two different slides, is significant. We do not want to overwhelm user with wasteful amount of generated slides, so the user would essentially have to sort through the generated slides which could increase user's frustration and decrease perceived usefulness.

Chapter 2

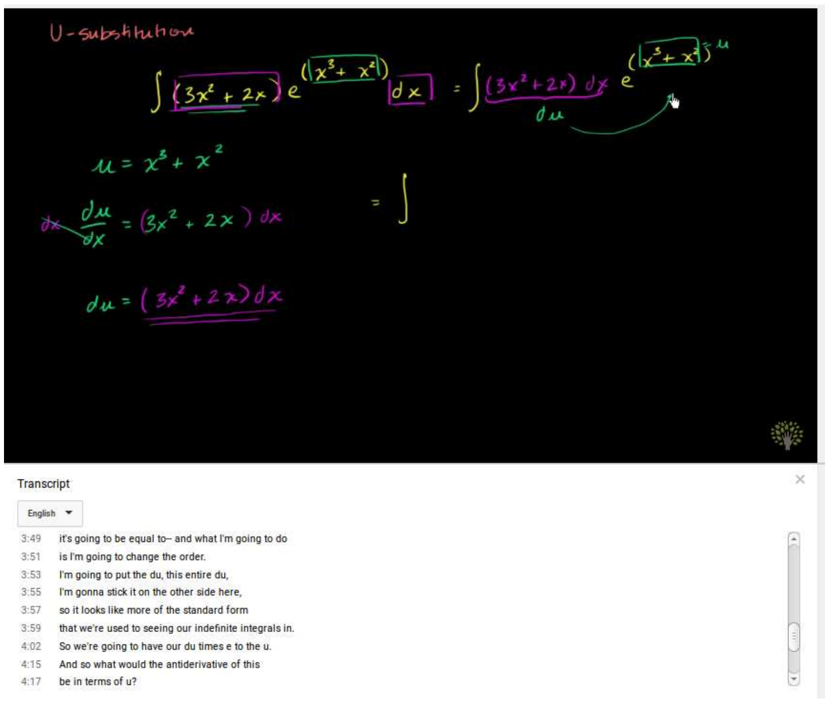
Related Work

While our work tries to solve a very specific problem, there are many similar projects inspired by the increasing amount of readily available video-based lecture material. These projects also try to transform video content into a better format, that can be consumed and perceived by user more easily.

Visual Transcripts [1] aims to create lecture notes from blackboard-style videos (Figure 2.1). Unlike our work, their system assumes that the video only has a blackboard in it, and that the video is static, except for content which is continually added and a mouse pointer which is used to highlight certain parts of the blackboard. The same type of video is used in another related work [2], in which the authors summarize the input video in form of a single image (Figure 2.2). Parts of the image function as links to positions in the input video and make it very easy for a user to jump to the precise moment where a specific concept was first introduced.

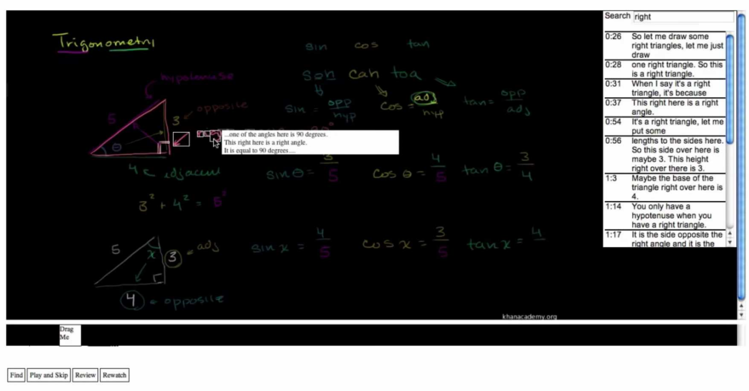
The focus of many authors is mainly on one or few methods for specific subtasks, that represent the respective parts of our system. For example most papers on removing occlusion events from videos focus on 3D objects or use multiple cameras to generate the final image, as described in [6] and [7]. These methods are quite efficient, especially when implemented on GPUs. They are quite accurate but they are not suitable for our purpose.

On the other hand, many approaches are focused on creating fast and efficient methods for recognition of change in images and search for similar or unique objects in images. Various methods were designed in order to solve these tasks, such as using image features to detect similar objects in images or using perceptual hashes [8] to detect significant change. In [9], the authors developed a new method for image hashing which can be used for fast look-ups of similar images. The thesis of Christoph Zauner [10] focuses on implementation and benchmarking of various image hashing algorithms and methods. While the primary aim of these methods was different, they can be easily adapted for the purpose of searching for change, in information in series of



3:49 it's going to be equal to– and what I'm going to do
3:51 is I'm going to change the order.
3:53 I'm going to put the du, this entire du,
3:55 I'm gonna stick it on the other side here,
3:57 so it looks like more of the standard form
3:59 that we're used to seeing our indefinite integrals in.
4:02 So we're going to have our du times e to the u.
4:15 And so what would the antiderivative of this
4:17 be in terms of u?

Figure 2.1: Example of input from Shin, Hijung Valentina, et al. [1].



...one of the angles here is 90 degrees.
This right here is a right angle.
It is equal to 90 degrees....

Search right

```
0:26  So let me draw some right triangles, let me just draw
0:28  one right triangle. So this is a right triangle.
0:31  When I say it's a right triangle, it's because
0:37  This right here is a right angle.
0:54  It's a right triangle, let me put some
0:56  lengths to the sides here. So this side over here is maybe 3. This height right over there is 3.
1:3   Maybe the base of the triangle right over here is 4.
1:14  You only have a hypotenuse when you have a right triangle.
1:17  It is the side opposite the right angle and it is the
```

Figure 2.2: Output of Monserrat, Toni-Jan Keith Palma, et al. [2].

frames generated from a video sequence.

Algorithms for detecting a change in scenes of video sequences were also proposed in [11] and [12]. These algorithms search for "drastical" points of change, such as hard cuts or special editing effects [1]. Even though these methods were quite useful as a model, we could not adapt them because our work is focused more on fine grained change between multiple frames and longer lasting sequences, where change is being slowly added through the span of multiple sets of frames.

[1]An example of thee effects are dissolve or wipe transitions.

Chapter 3

Fundamentals

This chapter introduces the basic concepts and algorithms used in this work. Of course this chapter does not aim for complete coverage on image processing, recognition or any other given subjects. For the basics presented here it is assumed that the reader has basic knowledge about how are images represented in computers and how we can manipulate them, e.g., from a lecture on given subject or from another source, such as book explaining basic concepts [13].

3.1 OpenCV

OpenCV(Open Source Computer Vision Library) is free computer vision library that is written in native C++ and it has C++, C, Python, Java, MATLAB interfaces and supports most modern platforms including Android or Windows. One of the main reasons why we chose to work with this library is because of its diversity. Any method or algorithm can be easily rewritten to most languages or can be deployed quickly to any platform. "OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception." [14] This library contains over 2500 optimized basic computer vision algorithms which can help us identify objects, track camera movements, track moving objects, stitch images together to produce a high resolution image of an entire scene or find similar images. It also supports multiple core computation which is crucial for making sure that our methods and algorithms would run in reasonable amount of time and can process and generate final slides in real time.

3.2 Thresholds

Thresholding is common and simple segmentation method of image processing. We can use thresholding methods for many applications, for example, separate regions of an image, to differentiate the pixels, that we are interested in from the rest or set determined value to pixels so we can identify them. As name suggest these functions use some thresholding value to differentiate some pixels from the rest and then set value to pixels which satisfied *thresholding condition* or to those which do not, as shown in Figure 3.1. Most common thresholding functions are:

Threshold Binary This threshold sets value, to pixels that are higher then thresholding value, to some *maximum value* and others are set to 0.

$$f(x,y) = \begin{cases} maxvalue & \text{if} f(x,y) \text{is} > \text{threshold} \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

where $f(x,y)$ is current value of the pixel.

Threshold Binary, Inverted This function sets maximum value to pixels that did not match binary threshold criteria and 0 to those that did.

$$f(x,y) = \begin{cases} 0 & \text{if} f(x,y) \text{is} > \text{threshold} \\ maxvalue & \text{otherwise} \end{cases} \tag{3.2}$$

Truncate In this function maximum value is exactly thresholding value so pixels that are greater are truncated and pixels that are lower are omitted.

$$f(x,y) = \begin{cases} threshold & \text{if} f(x,y) \text{is} > \text{threshold} \\ f(x,y) & \text{otherwise} \end{cases} \tag{3.3}$$

Threshold to Zero Values for pixels that are lower than thresholding value are set to 0.

$$f(x,y) = \begin{cases} f(x,y) & \text{if} f(x,y) \text{is} > \text{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Threshold to Zero, Inverted Pixels that are lower than thresholding value are omitted while pixels that are higher than this value are set to 0.

$$f(x,y) = \begin{cases} f(x,y) & \text{if} f(x,y) \text{is} > \text{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

3.3 Feature detection

This section aims to be brief overview of *feature detection* and *feature matching* concepts, which are used in this work. This topic is too complex to provide more than brief explanation of these concepts. In depth explanation can be found for example in the book Computer vision: algorithms and applications [13]. Feature detection and

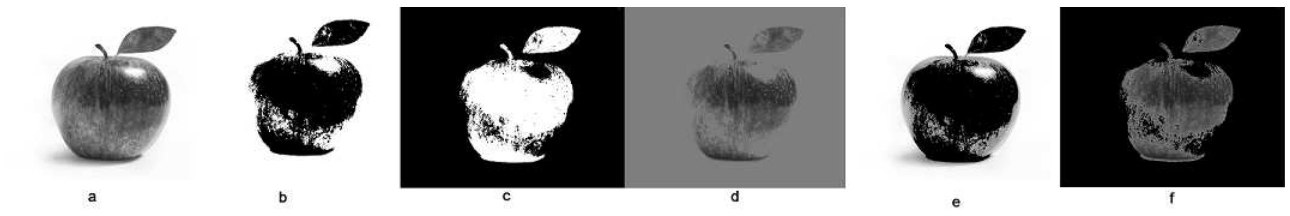


Figure 3.1: (a) Original image; (b) binary threshold; (c) binary inverted threshold; (d) truncate threshold; (e) to zero threshold; (f) to zero inverted threshold.

feature matching are widely and commonly use methods in almost all modern image processing and image recognition applications. These methods are used to abstract image information and make local decisions, at every point of a image, if there is a *image feature* ("interesting parts of image") of given type present at that point or not. We can later use the found *features* to help us align images so we can stitch them together or they can help us construct 3D models of various scenes. Image features are also widely used in object recognition or video stabilization. There are many types of image features, here we will mention just a few of the most frequently used ones.

3.3.1 Keypoints

Keypoint features (*interest points* or *corners*). These terms are used somewhat interchangeably and refer to point-like features which are often described as patches of pixels in area surrounding location of some point. This kind of localize feature is used to detect specific location in the image, like building corners, doorways or interestingly shaped anomalies in the picture. Main advantage of keypoints is that they can allow us to perform matching even when occlusion events or large scale orientation changes are present in a image.

3.3.2 Edges

Another highly used type of a feature are *edges*. Edges are usualy defined as a set of points with strong *gradient magnitude*, that can form almost any arbitrary shape. These features are good indicators of *object boundaries* or *occlusion* of a object by another object.

3.3.3 Main approaches

There are two main approaches to finding features in the image. First one is more suitable for video sequences or images taken from nearby viewpoints. First approach finds features in one image and then *tracks* them accordingly using some local search technique, e.g., least squares. The second one is more suitable for object recognition or stitching images together to create panoramas. This method identifies features independently in all images and then *matches* them based on their local appearance. Feature detection and matching is usually split into three main stages:

- Feature detection stage, where images are searched for locations, that will match best in other images.
- Feature description stage, where features are converted into invariant descriptors.
- Feature matching or tracking stage, where images are searched for likely matching candidates. In image tracking stage only small area around detected feature is searched.

So, after we detected features we have to match them. But in most cases features will change by undergoing some transformation in scale or orientation or even deformations. We have to compensate for these changes and we have to make feature descriptors invariant while still preserving difference between patches. For this reason, many view-invariant local image descriptor have been developed. Some of the the most known *feature descriptors* are *Bias and gain normalization (MOPS)* or *Scale invariant feature transform (SIFT)*. After extracting features and their descriptors, *feature matching* or *tracking* is performed. This stage is also divided into two more stages. The first one is to select a *matching strategy* and the second is to find efficient *data structure* and *algorithm* to perform this matching as efficiently and as quickly as possible.

There are many algorithms that performs this entire process such as before mentioned *SIFT* or *SURF (Speeded-Up Robust Features)*. In our work we used probably one of the most known and used one ORB.

ORB (Oriented FAST and Rotated BRIEF) is a good alternative to SIFT and SURF in computation cost and matching performance. This method was first brought up in paper Rublee et al. [15] and while SIFT or SURF are patented, ORB was developed in openCV Labs and is free to use. In our work, we use ORB for various tasks, for example, to detect and compare similar images.

3.4 Edge detection

As mention before edges are defined as set of points with strong gradient magnitude. While interest points are useful for matching images, edges are more common and usually carry semantic information about a image. For example, boundaries of a object, occlusion events or shadow boundaries. We can group isolated edge points into *straight lines*, *curves* or *contures*. But given a image how can we detect and edge? Since edges occur at boundaries of two regions of different colour, intensity or texture we can detect edge only from local information as the biggest change. Book Computer vision: algorithms and applications [13] defines an edge as "location of *rapid intensity variation*." A mathematical way to define this is through gradient.

$$\mathbf{J}(\mathbf{x}) = \nabla I(x) = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})(x) \tag{3.6}$$

The local gradient vector $\boldsymbol{J}$ points to steepest ascent in the intensity function. But taking image derivatives amplifies noise since it accentuate high frequencies. It is therefore necessary first to smooth images with low-pass filter and then compute gradient. Gaussian filter is probably the most used filter for this task.

3.4.1 Canny edge detection

Most known edge detection method is probably *Canny edge detection* algorithm. This detector was developed by *John F. Canny in 1986* and it is also know as *optimal detector*. It was design to satisfy three criteria: *Low error rate*, *Good localization*, *Minimal response*. This method can be described in four steps.

- Filter out any noise.
- Find the intensity gradient of a image.
- Remove pixels, that are not considered to be part of an edge.
- Threshold the image.

For thresholding a image Canny uses two thresholds (upper and lower).

- Pixel is accepted if gradient value is higher then upper threshold.
- Pixel is rejected if it is below lower threshold.
- If a pixel is between these two thresholds it is accepted, if it is connected to a pixel, that is higher then upper threshold.

More information and more complex coverage of this topic can be also found in [13].

3.5 Perceptual Hashes

Perceptual hash is a fingerprint of file derived from various features of its content. This class of hashing functions uses these features of media file to compute *distinct* but *not unique* fingerprints of given file. These fingerprint can later be compared to find same or similar images based on some *similarity measure*, e.g., *Hamming distance* or *cross-correlation* [9]. These hash functions are different from cryptographic hash functions like SHA1 or MD5 because cryptographic hashes use input data as random seed so output data are also random. They rely on so called "avalanche effect", small change in input data will result in drastic change in output of a hash function. Comparing these hashes can only tell us two things: if the hashes are the same the input data are also the same, if they are not then data are different. On the other hand two perceptual hashes can be compared, to give us a some sense of similarity of two files. “Perceptual hash must be robust enough to take into account transformations or "attacks" on a given input and yet be flexible enough to distinguish between dissimilar files.” [16]. These transformations can include rotations, skew, contrast adjustments and different compression or formats. These kind of hashes have many applications, for example, in copyright protection, search for media files or in digital forensics.

3.5.1 Average hash

Also known as *aHash* or *mean hash* is a hashing function which computes hash of a file by firstly reducing a size of image to small square usually of size 8×8 to remove high frequencies. Then reduces a colour of this square to grayscale. Hash is then created by computing mean value of pixels in transformed image which is then plugged into the thresholding function (3.7).

$$f(x,y) = \begin{cases} 1 & \text{if } f(x,y) \text{ is} > \text{mean} \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

3.5.2 Difference hash

Difference hash or *dHash*, similarly to average hash, computes its value by firstly reducing a size and colour of given image but then it calculates difference between adjacent pixels. This identifies *relative gradient direction* in the image. After this it determines resulting value by using thresholding function (3.8).

$$f(x,y) = \begin{cases} 1 & \text{if } f(x,y) \text{ is brighter than } f(x-1,y) \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

While two first algorithms are quick and easy they might be to rigid comparison. For instance, it can generate false-misses if there is gamma correction or colour histogram applied to a image. To reduce this effect we use another hash named after this field of study *perceptual hash*.

3.5.3 Perceptual hash

Main idea behind *perceptual hash*, or more commonly know as *pHash*, is that it uses *discrete cosine transform (DCT)* to reduce the frequencies in the image. Same as previous hashes (Section 3.5.1 and Section 3.5.2) initial step is to reduce size of given image and its colour. Then it computes DCT on given image and subsequently reduces it to keep just lower frequencies of the picture. Next step is to calculate the mean DCT value while excluding the first term. This leaves out completely flat image information (i.e., solid colors) from being included. Lastly it further reduces DCT and computes resulting hash values based on threaholding function (3.9) similar to average hash (Section 3.5.1).

$$f(x,y) = \begin{cases} 1 & \text{if } f(x,y) \text{ is} > \text{mean DCT} \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

These algorithms can be further enhanced to improve their performance, for example, the image can be cropped before being reduced in size. Other variations can track general colouring (e.g., hair is more red than blue or green, background is closer to white than black) or relative location of lines.

Chapter 4

Preprocessing

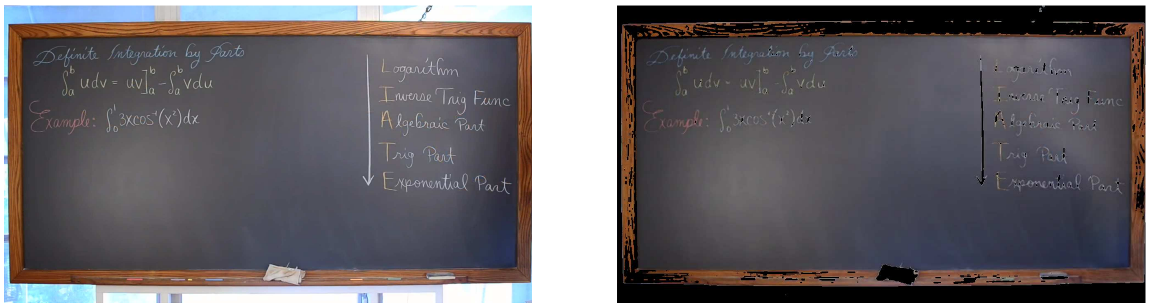
Before we start processing our frames we have to prepare them so they can be later processed by our pipeline. This chapter discuses necessary preprocesing steps such as determining color of a board or preparing frame for board extraction process to increase accuracy of our methods.

At first, we blur the image to get rid of undesirable noise and to reduce colour spectrum of the image. In other words, we quantify colors present in the frame, so in the next step we can detect colour of a board more easily. For this we used simple median blur.

Median blur is frequently used method to blur (smooth) images. Median filtering is widely used because under certain conditions, it preserves edges while still smoothing the image. This is also the main reason why we choose median blur, while still being one of the fastest blurs, it can preserve hard edges of the board, which is very important for a methods used for extraction of the main region of the board described in Chapter 5. The main idea is to run through each element of the image and replace each pixel with median of its neighboring pixels located in square neighbourhood around the given pixel.

This area is also called kernel. In our case, we achieved the best results with 5×5 kernel. This kind of filtering is also particularly effective for impulse noises such as *speckle noise* or *salt and pepper noise*.

Next, we have to decide if we are looking at a white or black board. A mean of color channels is computed to determine *dominant color* in the image. This color is also color of the board under the assumption that the area of a board occupies wast majority of given frame. This colour is then accentuated by thresholding the image to only extract colours that are close to dominant colour as follows



(a) Original image frame from the video.

(b) Output of the preprocesing function applied on the original frame.

Figure 4.1: Images of original video[1] frame and the resulting frame from the preprocesing function

$$f(x,y) = \begin{cases} f(x,y) & \text{if } \begin{cases} I_R(x,y) \in \langle 0.25 \cdot R_{mean}, 1.5 \cdot R_{mean} \rangle \text{ AND} \\ I_G(x,y) \in \langle 0.25 \cdot G_{mean}, 2.5 \cdot G_{mean} \rangle \text{ AND} \\ I_B(x,y) \in \langle 0.25 \cdot B_{mean}, 1.5 \cdot B_{mean} \rangle \end{cases} \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

where $I_R(x,y)$, $I_G(x,y)$ and $I_B(x,y)$ are values of red, green and blue channel of pixel at x and y position respectively. R_{mean}, G_{mean} and B_{mean} are values of red, green and blue channel of obtained dominant color respectively. These values can range from 0 to 255.

Thresholding value is different for every channel and it is approximated to previously found dominant color of our image. The mask obtained by this method is then used to compute bitwise `AND` in separated colour channels in order to boost our colour as shown in Figure 4.1b. Note, that green colour is accentuated while background, white or highly illuminated sections are attenuated. This helps us to increase accuracy of region of interest localization in the frame. This is ran on every frame, that is going to be processed by our pipeline.

[1]We used the video titled "Definite Integration by Parts" by Rob Tarrou which can be found at `https://www.youtube.com/watch?v=6rWG5WPysgE`

Chapter 5

Board extraction

Creators of instructional videos strive to present their content (be it on black or white board) on the majority of visual space the video provides. Despite their efforts, often there are parts of video frames one would not expect to find in a "presentation slide". Moreover, variability in these parts of video frames might cause issues in further stages of the processing pipeline, as it might be mistaken for the actual content. Therefore, in order to create a "presentation slide" from a set of video frames, only the significant parts of the frame need to be considered.

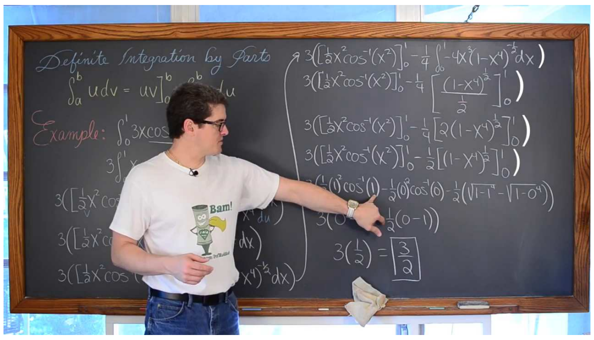
In this chapter, we describe main methods used for extracting board from video sequences. In our research, we focused mainly on videos with one board present in the video or multiple boards not separated by a wider space in between them (Figure 5.1a). After initial preprocessing steps described in Section 4, one of the proposed method is used to obtain main region of the board.

5.1 Histogram method

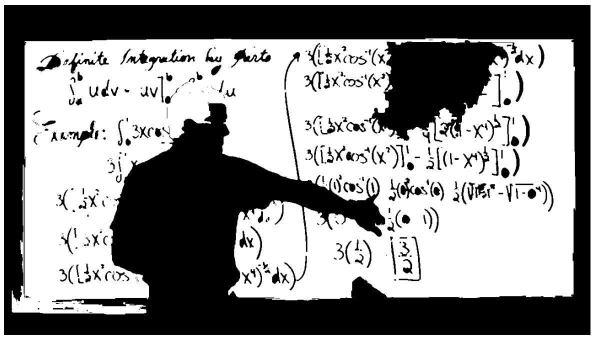
This method is based on simple idea of generating two histograms, for both axis of a input frame, of previously found dominant color. This is done by first reducing a colour spectrum of the image by thresholding the image one more time as follows:

$$f(x,y) = \begin{cases} f(x,y) & \text{if} \begin{cases} I_R(x,y) \in \langle 0.6 \cdot R_{dom}, 1.5 \cdot R_{dom} \rangle \text{ AND} \\ I_G(x,y) \in \langle 0.6 \cdot G_{dom}, 1.5 \cdot G_{dom} \rangle \text{ AND} \\ I_B(x,y) \in \langle 0.6 \cdot B_{dom}, 1.5 \cdot B_{dom} \rangle \end{cases} \\ 0 & \text{otherwise,} \end{cases} \tag{5.1}$$

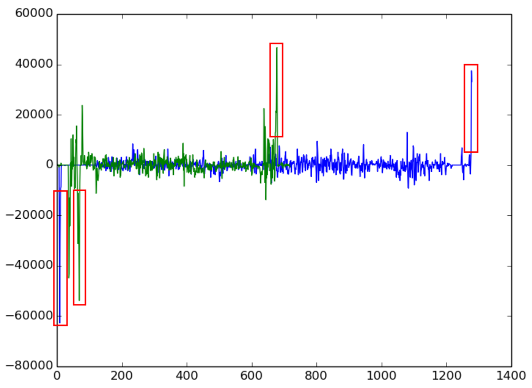
where $I_R(x,y)$, $I_G(x,y)$ and $I_B(x,y)$ are values of red, green and blue channel of pixel at x and y position respectively. R_{dom}, G_{dom} and B_{dom} are values of red, green and blue channel of dominant colour. These values range from 0 to 255.



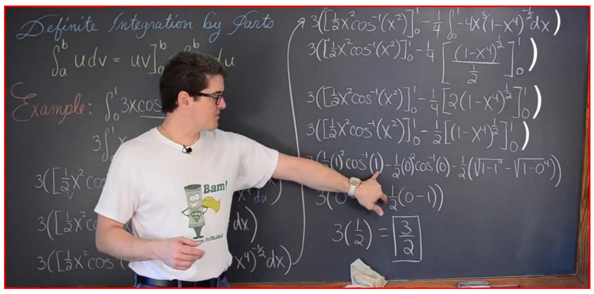
(a) Original image from the video.



(b) Mask of the board generated by region growing method.



(c) Values generated by histogram method. Blue and green color represents x and y axis respectively.



(d) Final image with extracted board and shifted perspective.

Figure 5.1: These images show main steps in board extraction processing pipeline. Note, in the second image present error area caused by high intensity light.

In the next step, all the values in every row and column are summed up separately into two arrays. The first derivation is then computed on the acquired arrays to identify rapid change in dominant colour (Figure 5.1c). Next global maximum and minimum is identify in both acquired arrays. These extremes then specify corners of the board. Finally, a *bounding box* around board is formed from the obtained points.

5.2 Region growing method

This method is based on a simple region growing algorithm. This algorithm consists of two parts. First part is to select a seed for our algorithm. This is done by thresholding to zero (Section 3.2) the dominant colour with almost maximum color value of a board, (5.2) found in advance, as part of the preprocessing step (Section 4).

$$f(x,y) = \begin{cases} f(x,y) & \text{if } \begin{cases} I_R(x,y) \in \langle 0.6 \cdot R_{dom}, R_{dom} \rangle \text{ AND} \\ I_G(x,y) \in \langle 0.6 \cdot G_{dom}, G_{dom} \rangle \text{ AND} \\ I_B(x,y) \in \langle 0.6 \cdot B_{dom}, B_{dom} \rangle \end{cases} \\ 0 & \text{otherwise,} \end{cases} \tag{5.2}$$

where $I_R(x,y)$, $I_G(x,y)$ and $I_B(x,y)$ are values of red, green and blue channel of pixel at x and y position respectively. R_{dom}, G_{dom} and B_{dom} are values of red, green and blue channel of found dominant color respectively. These values range from 0 to 255.

This highlights "blobs" of colour with highest values, so we can be almost sure, that we are starting somewhere in the region of a board and not, for example, on the person standing in front of it. Then, we select largest region where we randomly place our seed.

In the second step a region growing follows. Region growing is *region-based* image segmentation method[1]. The main idea is to segment the image into regions. This approach to segmentation examines neighboring pixels of initial seed point, whether the neighboring pixels property fits some logical predicate P. If pixels satisfies the predicate, it is added to the same region as the initial pixels of that region. This is iterative process. This basic formulation of this method is:

- $\bigcup_{i=1}^{n} R_i = R$.
- R_i is a connected region , $i = 1, 2, \ldots, n$.
- $R_i \cap R_j = \emptyset$ where $i \neq j$ for all $i, j = 1, 2, \ldots, n$.
- $P(R_i) = TRUE$ for $i = 1, 2, \ldots, n$.
- $P(R_i \cup R_j) = FALSE$ for any adjacent region R_i and R_j.

The regions are grown from the seed points to adjacent pixels. In our case, the predicate for including pixel into a region, was color of the pixel. If the pixels are part of the board (satisfy our condition), then they are added as a part of the board. This process also produces the mask of the board (Figure 5.1b). The mask is created separately into different image by undergoing binary threshold function (5.3).

$$f(x,y) = \begin{cases} 255 & \text{if } f(x,y) \text{ is part of the region} \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

Once this is done, and we have obtained the mask of the board, we run series of dilation and errosion to reduce error areas, that can occur at the edges of the board

[1]It is also classified as pixel-based segmentation method since it involves selection of initial seed.

(Figure 5.1b), so the edges of the board are well defined. We search for contours in the generated mask and extract the biggest one. Then we calculate perimeter of selected contour. This perimeter might have irregular shape because of impurities at the edges of found contour in the mask. Then the *Douglas-Peucker line-simplification* algorithm [17] is used to reduce number of vertices in the perimeter to obtain more regular shape. Finally this reduced perimeter is returned as the *bounding box* of our board.

5.3 Processing output of board extraction methods

Output of every methods is submitted to the last test which checks if the area occupied by the the bounding box is at least one third of a given image and if its shape is rectangular. Finally, the image is cropped. If a video or an image is taken from a slight angle, perspective of cropped board is then slightly shifted to compensate for this, so that the resulting slide would look more like an observer is standing in front of a board. This is done by computing transformation matrix based on obtained corners of the board. These corners are used as correspondence points for estimating homography, that best fits all the points. The image is then transformed using the obtained matrix as shown in Figure 5.1d.

Chapter 6

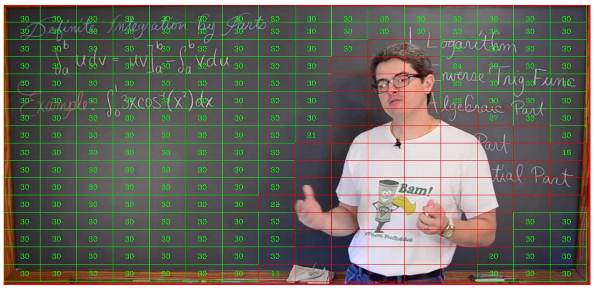
Occlusion removal

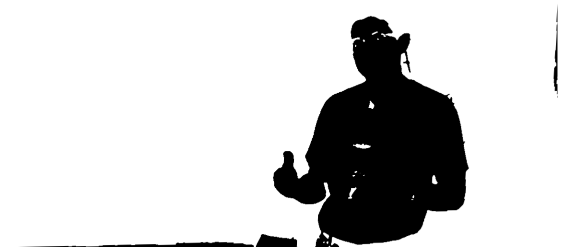
Since we require that our slides should contain only board with written information, we have to remove any occlusion events, that happen in front of the board. This chapter describes algorithms, that we design in order to remove occluding objects from lecture videos. We consider as occluding object anything that is bigger than at least one third of a board and performs some sort of a move. An example of such an object can be a student or a lecturer. These objects should be then segmented out of the frame and replaced by correct section of the board, that these objects occlude. The main idea is to segment the board into smaller sections called cells and then keep track of how information changes in these cells by keeping a simple count of on how many frames we saw individual cell. This process can be described as a sequence of the following steps:

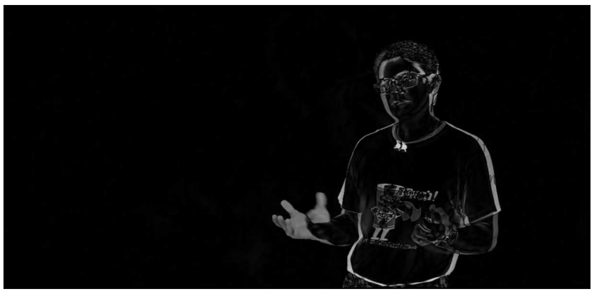
- Divide board into smaller cells
- Initialize each cell
- Compute mask of occluding objects
- Check each cell to see whether it is occluded by another object
- Change “seen” counter for each cell based on occlusion events
- Stitch individual cell into final slide

6.1 Initialization

We divide our cropped image of the extracted board into equally sized rectangular cells based on width and height of the input image. Individual cells are overlapping by values spanning between 10 to 20 pixels based on the size of a given frame. Each cell is then initialized. This is done by setting “seen” counter to 0 and its value is


(a) Input image segmented into grid of rectangular cells. Note, that overlap of the individual cells is not shown on the image.


(b) Occlusion mask produced by inverting mask of a board generated by region growing method in board extraction process.


(c) Result of computing absolute difference between current frame and last saved slide.

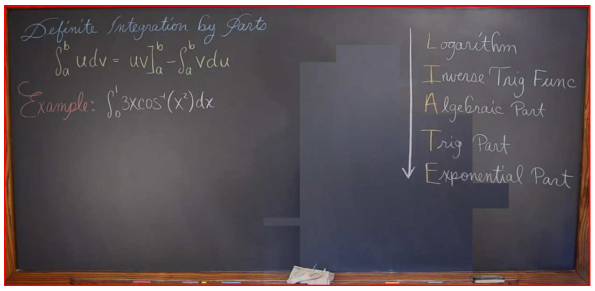

(d) Final slide generated by "stitching" algorithm without any further postprocessing.

Figure 6.1: These images show main steps in objects removal processing pipeline.

stored in an array. When cells are initialized, then we evaluate each of them to check if it is partly or fully occluded by another object[1]. This is done by computing bitwise `AND` between section and occlusion mask (see Section 6.2 and Section 6.3). If there is overlap detected the counter is not increased, otherwise it is incremented. Occlusion mask might contain a lot of noise, for instance, from fast changing high intensity pixels. This can cause a problem, if there is a lot of noise in the section, the section might be marked as occluded. We try to mitigate this effect by detecting size of occluded area between cell and occluding object. The cell is marked as occluded only if the occluded area occupies majority of a cell (Figure 6.1a). For obtaining occlusion mask, we proposed two methods.

6.2 Region growing based method

The first method is the same as our region growing algorithm (Section 5.2), since this method only considers pixels, that are part of the board. Pixels representing skin,

[1]Note, that if a cell is occluded throughout the whole video, we believe, that it is save to assume, that there is no valid information behind it.

cloths, hair or other objects are omitted. This can then specify any objects that are in front of the board. If region growing algorithm was previously used for board extraction, the inverted mask produced by this method, from the board extraction process, is then considered as mask of the occluding objects. The inverted mask is shown in Figure 6.1b and series of dilations and erosions is performed to remove error areas and to accentuate edges. Finally, we search for contours and filter out those, that are smaller than one third of the image under a assumption, that the lecturer or student occupies wast majority of the board.

6.3 Absolute difference based method

The second method is based on computing absolute difference between images. This is done by subtracting each pixel value of the first image from the corresponding pixel value in the second image. Since we are interested only in the amount of how much these pixels differ, the absolute value is calculated for these results. Our method uses the last slide, which was saved and therefore we assume, that this frame contains only valid information and it is without any occlusion events and current frame. At first, both images are converted into grayscale which speeds up the computation of the mask, since we are not interested in color values of pixels. We can ignore them and focus only how intensity of the pixel changes. The absolute difference is then calculated between these images (Figure 6.1c). This generates mask of events that changed from slide to slide. Next, we threshold this mask with a binary threshold (6.1) to remove pixels, that arose from slight light changes for example person casting a shadow onto a background.

$$f(x,y) = \begin{cases} 255 & \text{if } f(x,y) > 50 \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

Finally, similarly to the previous method we search for contours and filter out those that are not at least one third of given image.

6.4 Slide generation

The final slide, shown in Figure 6.1d, is then "stitched" together from the last saved slide and currently stored sections, where we threshold each section's counter with two values. If the counter value is higher then upper threshold, we saw that section enough times to be sowed into the image. If the value is under the lower threshold, section is rejected and corresponding part of the old image is used. If the value is between these two thresholds one of the methods discussed in the Chapter 7 is used to detected how much old and new section differ. If this similarity measure is higher then our threshold, then section is rejected because sections were too similar and old part of the slide is used. If it is lower, the section changed enough so it can be sowed into the image.

Chapter 7

Generation of final slide

This chapter focuses on how are we actually going to choose key frames from the lecture video. The main problem, that we faced was how we can choose key frame in the video or presentation. When we can say, that enough information was added or subtracted from a board, so we can actually create slide of the board. Also we don't want to overwhelm user with huge amount of slides where information changed insignificantly. On the other hand if we choose only few frames of the video as our keypoints we can loose substantial amount of relevant information. So, we want to reliably compare changes to the information in last created slide and current slide that is being processed. For this we developed two main approaches: first one uses feature detection and second computes perceptual hashes with one of the specified methods in Section 3.5.

7.1 Feature detection approach

To detect change with this method we used ORB detector for feature detection (Section 3.3). First step is to detect features in both last generated valid slide and current slide. Next, brute force matching with *Hamming norm*, between features of these two image, is performed (Figure 7.1). This binary descriptor matching takes significantly less time then vector descriptor matching [18]. But nearest neighbor matching will always return a match. This can lead to a number of false matches. For this a cross check filter is applied at feature matching stage. As the name suggests it tries to increase accuracy by matching features in both directions. After this matcher returns only consistent pairs of descriptors, that are closest to each other in matcher's collection. Number of matched features is then compared to maximum of found matches. This ratio is then returned to be later compared as our similarity measure for further tests.

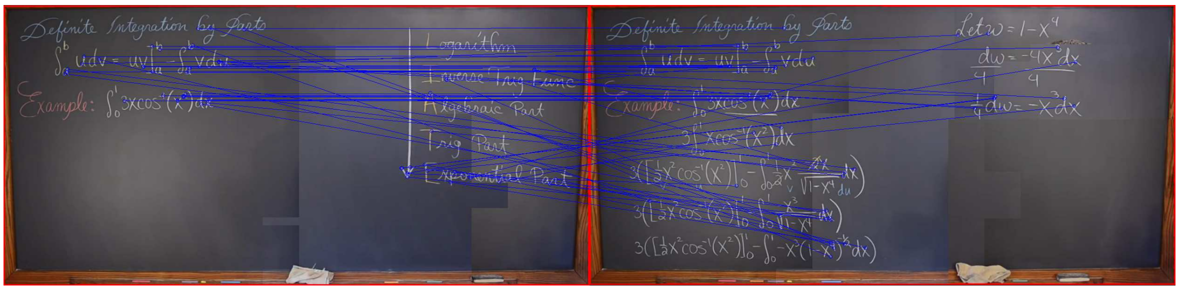


Figure 7.1: Result of brute force matching between two generated slides. Note, that features were often matched incorrectly based on similarity of individual symbols.

7.2 Perceptual hash approach

Another approach to detect change in our slides is through perceptual hashes (Section 3.5). We implemented and compared probably three most known functions: aHash, dHash and pHash. These hash functions are used to compute hashes of the last saved slide and the currently processed slide. Hamming distance is then computed between these hashes. To unify the output from hash functions and feature detection function we compare length of last hash and the obtained hamming distance. This ratio is then returned.

7.3 Comparing and storing slides

Output from these similarity functions is then compared to our similarity threshold. The value of the threshold is by default set to 60%. This value can be adjusted by user to fit their preference. If our measure is lower than this threshold, slides differ and information on the board has to be different, so slide is created. If it is greater, then slides are similar. Current slide is thrown away and all counters are reset. With the slide, we also store mask of the board and number of last generated valid slide, so we can resume from specific point in the video, if the process of generating slides is suddenly terminated.

Chapter 8

Dataset and evaluation

One of the main problem we faced, was how to actually evaluate our designed methods, since at this time, there is not known any publicly available dataset with on-line lecture videos or images of the lectures, on which we could test and evaluate our methods. Thankfully, there is plenty of content available on-line issued under standard licenses which means, that they are save to use for educational and research purpose. An example of such a source can be, for instance, videos of lectures from Massachusetts Institute of Technology publicly available on-line[1]. In this chapter, we describe our approach of creating a dataset of videos and presentations from various on-line lectures and courses and methods for evaluating our designed algorithms based on this dataset.

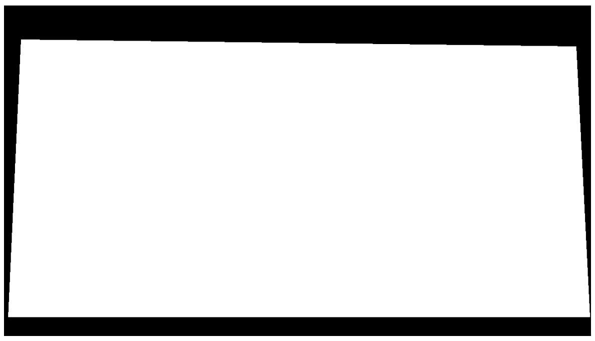
8.1 Dataset

Since, in our work we combine multiple methods, each design to performed specific task in our pipeline, we had to also create specific datasets for each step in pipeline. So, we can properly evaluate and compare our methods.

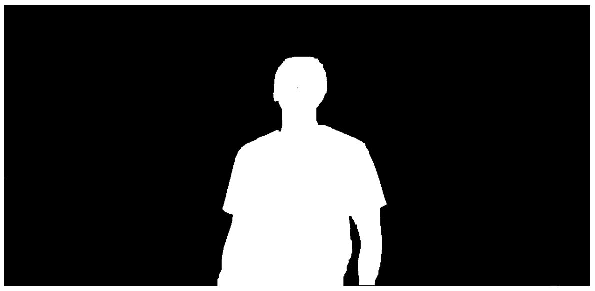
8.1.1 Board extraction

For finding main region of interest in the images, where board is located, we created a dataset of 19110 images. For these images the bounding box of the board was marked by a human expert (Figure 8.1a). This was done by creating toolkit program, that experts used for specifying the corners of the board. These images were extracted from individual videos and presentations. The process of generating the images can be described as follows: User selects a video from our dataset of videos. He is presented with a single window where chosen video starts playing (Figure 8.2a). Every frame of the video is processed by histogram method proposed in Section 5.1, to approximate

[1]Videos of various topics can be found at `http://ocw.mit.edu/courses/audio-video-courses/`.



(a) Ground truth marked by human expert for board extraction.



(b) Ground truth marked by human expert for occlusion removal.

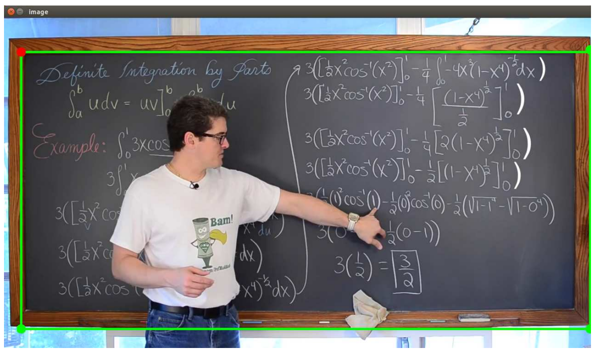
Figure 8.1: Sample images of ground truths used for evaluation of propossed methods.

corners of the board. User has the ability to pause the video and adjust found points, so they better fit corners of the board and resume playing the video or can also rewind or go through the video frame by frame. Coordinates of corner points are gradually saved to file in CSV format. From these values we can later reconstruct bounding box of the board. This should serve as the "ground truth" in our experiments [19].

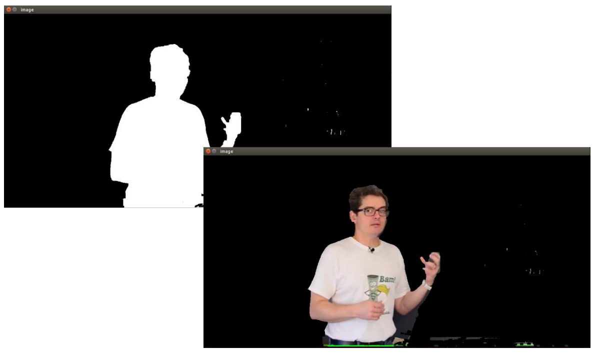
8.1.2 Occlusion removal

For comparing how well our methods were able to create occlusion mask from individual frame, we also had to create "ground truth" for these methods so we can later evaluate and compare their results. Essentially, we had to manually create occlusion mask of objects, that ought to be remove from the scene, in specific frames. We created a hand-curated dataset of 150 images for which the ground truth was also provided by a human expert (Figure 8.1b). This was done by creating another toolkit program. The process of creating an occlusion mask can be described as follows: User selects one frame from previously randomly selected set of frames. This frame is compared to the "0th slide", that was manually chosen as frame containing only the board without any occlusion events. Both frames are then converted to grayscale. Then algorithm similar to method proposed in the Section 6.3 follows. Algorithm computes absolute difference between grayscale images of given frame. Resulting image is then thresholded with same function as Equation 6.1. A series of errosions follows. Next, we search for the contours in the image. Mean value for every colour channels is calculated, in the area, that is occupied by individual contours, in both original images. For these values *Euclidean distance* is calculated as follows

$$\sqrt{\sum_{I} \Big(src_1(I) - src_2(I)\Big)^2}, \tag{8.1}$$



(a) GUI of toolkit program used for labeling board extraction dataset.



(b) GUI of toolkit program used for labeling occlusion removal dataset.

Figure 8.2: GUI of toolkit programs used to create datasets for our methods.

where $src_1(I)$ is computed mean value for specific channel of "0th slide" and $src_2(I)$ is mean value of selected frame for specific color channel.

Lastly, we compare these distances with user defined value of *distance threshold*. If the value is higher, then the contour is considered as occluding object and all pixels belonging to contour are set to the maximum value 255. If it is lower all pixels are set to 0. This identifies how scene changed compared to "0th slide". However this process also produces a lot of noise, for instance, bigger objects written to the board can be also identified as occlusion. To deal with this problem, user is presented with two windows (Figure 8.2b). In the first window generated occlusion mask is shown. In the second window an parts of the original image are shown based on previously generated mask. User can then adjust the image to get rid of the noise or to mark parts of the image that should also be considered as occlusion. The final mask is then saved and should server as the "ground truth" for evaluating occlusion removal methods.

8.2 Evaluation

This section describes how we evaluated and compared proposed methods. Every method was tested for performance from 300 method calls by measuring its individual speed in milliseconds in our processing pipeline. The tests run on single desktop workstation, using the following hardware: NVIDIA Corporation GeForce GT 650M, Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz.

8.2.1 Board extraction and occlusion removal

While speed might be interesting comparison, we are more interested in their performance: essentially the answer to the question how well were proposed methods able to extract the board from input images or how well they were able to generate occlusion mask (how well they identified occluding objects). To compare these methods we computed *precision* and *recall* for each of them. We define *precision* as:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{8.2}$$

and recall as:

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{8.3}$$

In both of these equations *true positives* are defined as the area of the image which was marked by the method as a board or as a occlusion event and the “ground truth” agrees with that. *False positives* is defined as the area, that was marked by the method as a board or occlusion object but the “ground truth” did not mark it as a board or a object. Lastly, *false negatives* is the area which was marked by “ground truth” but the method does not agree with that. These two values can be put together into a single metric, that is called *F1 score* and it is defined as the harmonic mean of *precision* and *recall*:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{8.4}$$

Note, that while these methods are traditionally more often used in information extraction, they are also considered a well defined metric in computer vision and image processing [19].

8.2.2 Change detection

It was difficult to design a measure, that would express how well is function performing in terms of selecting key frames in video. This was due to the fact, that even though we can have individual frames in sequence tagged the resulting value, if the selected key frame is in the right place, is very perception dependent. The number of slides, that should be generated can vary vastly from user to user. This also corresponds with reality, where some students are able to remember more from the lecture, therefore they need to write less information. This is why we chose to only measure how many slides will a method create and how long it will take, the whole pipeline (with a given measure), to do so.

Chapter 9

Results

In this chapter we analyze our proposed methods used in our pipeline. We also present performance values in terms of speed as well as precision and recall values for proposed methods obtained by evaluating each method on specific dataset (Section 8). Preprocesing of every frame in a video, took on average with 300 method calls - 5.89 ms (+- 0.47 ms). Results of individual methods were compared and the best methods were selected to be incorporated into our final pipeline. Lastly, we present final slides generated by our system.

9.0.1 Board extraction results

Method name	speed in milliseconds	standard deviation
Histogram method	4.75	0.65
Region growing method	**2.03**	0.57

Table 9.1: Average performance values in milliseconds with standard deviation for individual board extraction methods.

As we can see in Table 9.1, the Histogram algorithm was on average about two times slower than the region growing algorithm. This is understandable as the histogram algorithm is much more complex and needs to perform more operations than a simple region growing algorithm. The slowest part of the histogram algorithm is actually summing up color values of individual axis, which means we have to go through every pixel in the frame.

Given values in Table 9.2, we can observe some interesting statistics about the proposed methods. We can see, that while the histogram algorithm has a very high recall and therefore produced quite few false negatives its precision is quite low on the other hand. This suggests, that it produced quite a lot of false positives which might not be desired for the final processing pipeline. Since this would mean, that

Method name	precision	recall	F1 score
Histogram method	0.4754	**0.9738**	0.6389
Region growing method	**0.9854**	0.9567	**0.9708**

Table 9.2: Precision, recall and F1 score values of board extraction methods.

parts of a frame that are part of a background, would be marked by our method as a board and would be included into final slide. This can be caused by the fact that a lot of the lecture rooms has similar colored background (especially whiteboards). While region growing stops at the edges of the board, which are usually well defined, histogram considers color information for entire image, which might increase number of false positives. This could also later decrease accuracy of change detection methods.

The region growing algorithm on the other hand shows balanced values for both precision and recall. This suggests that it is a robust method, even though it produced an increased amount of false negatives which might not be desirable, since it would mean, that parts of the board would be lost. This can be caused by pixels with higher intensity values, which might be produced by light pointing at one spot on the board. These error areas may be considered as different regions and that can lead to loosing parts of the board, if they are located at the edges. Difference between recall values of histogram and region growing method are not significant.

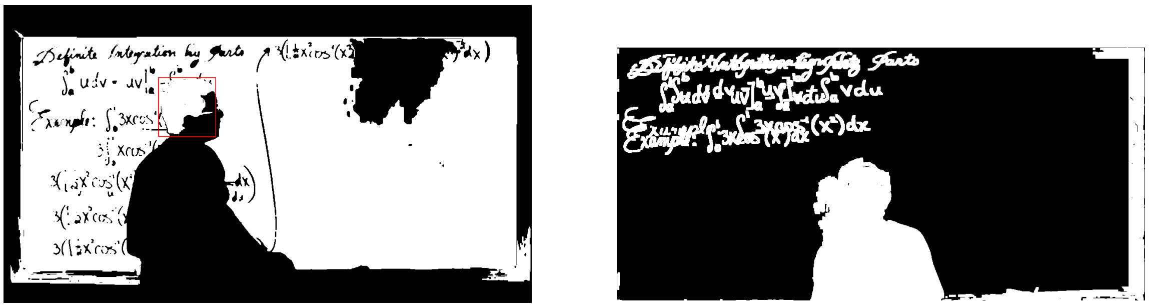
As we can see from the comparison of F1 scores, the region growing algorithm seems to be a more robust method and therefore we can conclude that it might be a better choice than histogram based method.

9.0.2 Occlusion removal results

Method name	precision	recall	F1 score
Region growing-based	0.2531	0.5595	0.3485
Absolute difference-based	**0.4767**	**0.9877**	**0.6430**

Table 9.3: Precision, recall and F1 score values of occlusion removal methods.

The precision, recall and F1 score value for both of the suggested methods can be found in Table 9.3. As we can see, the scores of the absolute difference-based method are better. This can be explained by the nature of the region growing-based method: given its randomized seeding, it might easily start growing the board's region from a point in the middle of the object, that should be removed. This essentially means, that it does the inverse task and produces incorrect result. Another problem, from which this method suffers and can produce incorrect results, is connecting regions that should be disjoint. This can be caused by increase amount of noise in the image. Since region



(a) Highlighted part of the image should be considered also as occlusion.

(b) Raw output of abs. difference-based method with text being considered as occlusion.

Figure 9.1: Invalid occlusion masks generated by region growing and absolute difference-based methods.

growing considers neighboring pixels, these “noisy” pixels can act as some sort of path between unrelated regions. Typical example of this problem is shown in Figure 9.1a, where hair of the lecturer is considered to by part of the board as they have similar color to the color of the board. This is also caused by our condition for adding pixels into regions being too general.

Name	speed in milliseconds	standard deviation
Region growing-based	**160.47**	0.91
Absolute difference-based	195.68	0.84

Table 9.4: Average performance values in milliseconds with standard deviation for individual occlusion removal methods.

As we can see in Table 9.3 precision values for both methods are quite low, which suggest that both methods produced quite a lot of false positives. This can be explained by the fact that content written on the board can get recognized as occlusion. Region growing method can consider insides of the letters or larger objects drawn on the board as different regions, since they usually have strongly defined edges, which produces quite a lot of noise in the final occlusion mask (Figure 9.1a). This means, that cells which are not actually occluded are being marked as occluded. This leads to substituting wrong parts of the frame and result into incorrectly generated slides. Although absolute difference-based method shows slightly better values for precision, it also suffers from the same problem, recognizing written content as part of occlusion (Figure 9.1b). These better results can be explained by incremental nature of the algorithm and the fact that the information is slowly added to the board throughout multiple frames.

Looking at the performance values in Table 9.4 we can see that while region growing-based method is faster, the difference is not substantial. Based on this analysis we chose the absolute difference-based method to be used in the final pipeline.

9.0.3 Change detection results

Name	speed in milliseconds	standard deviation
ORB algorithm	13.13	0.85
pHash algorithm	7.52	0.76
dHash algorithm	5.32	0.43
aHash algorithm	**3.23**	0.59

Table 9.5: Average performance values in milliseconds with standard deviation for individual methods for change detection methods.

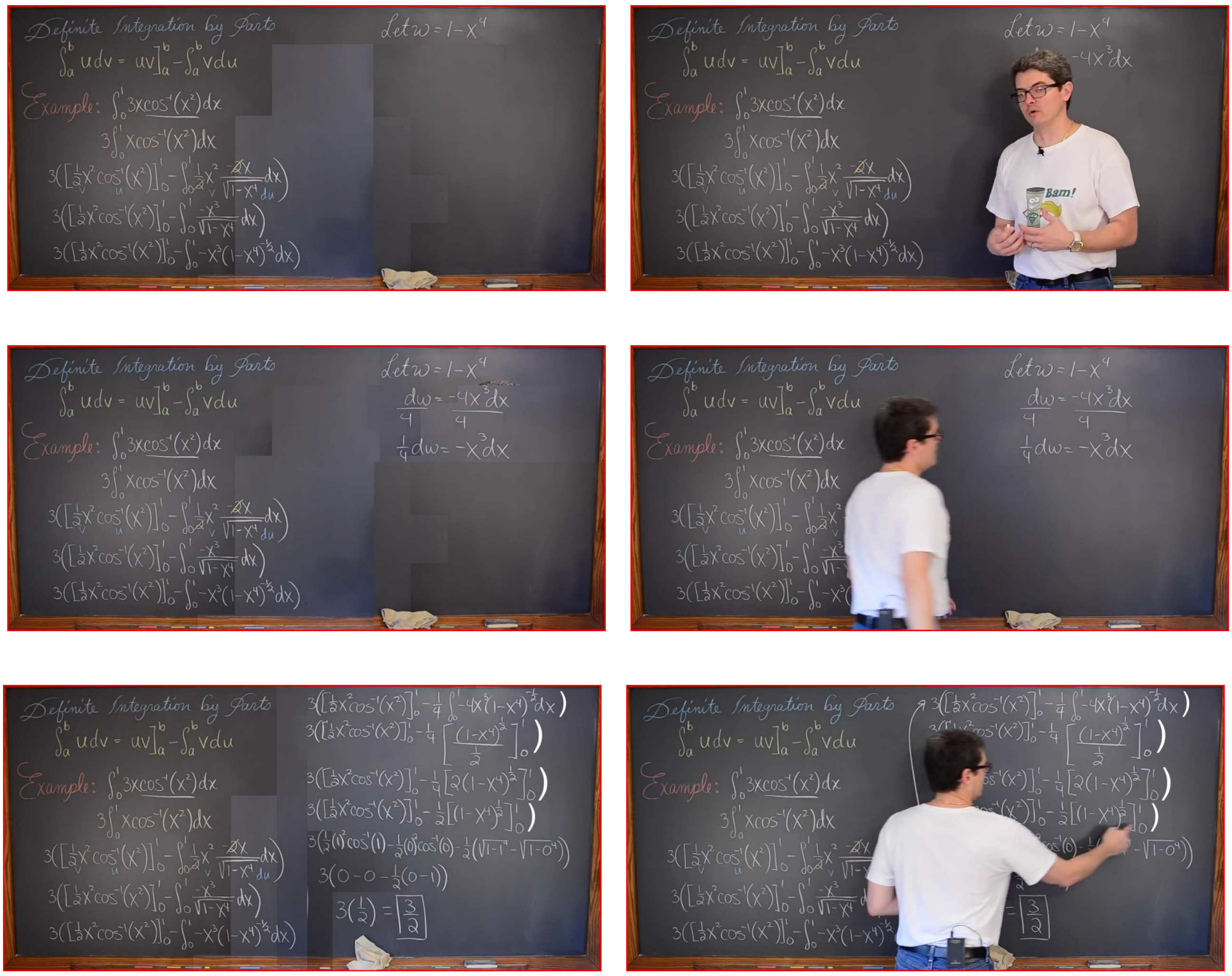
As we can see in Table 9.5, the ORB algorithm was the slowest which is understandable as it is also the most complex algorithm. Performance of hashing methods was comparable with the fastest one being the aHash algorithm[1]. This can be also closely related to values in Table 9.6 where, aHash not only has the best performance values in terms of speed but it also managed to create most slides. This can be associated with aHash being one of the the simplest methods, that is very vulnerable to even slight light intensity change in the scene. In a similar fashion, dHash suffers from the same problem, as it also created the same number of slides.

Name	speed in seconds	number of slides
ORB algorithm	1783	7
pHash algorithm	1117	**6**
dHash algorithm	1115	8
aHash algorithm	**1112**	8

Table 9.6: Average performance values in seconds for individual runs of entire pipeline with given method on the whole input video and the number of slides created using these methods.

From values in Table 9.5 and 9.6 we can conclude, that pHash is the best choice for detecting change in our video sequences. It provides the best ratio between speed of individual methods, speed of the entire run of a pipeline and number of created slides. While feature detection did reasonably well in comparing images, its performance could not compare to hashing algorithms. Original and output images obtained by running the final processing pipeline on an example video are shown in Figure 9.2.

[1]Which is not very surprising since it performs simple mean on given images.



Definite Integration by Parts

$\int_a^b u\,dv = uv\Big]_a^b - \int_a^b v\,du$

Example: $\int_0^1 3x\cos^{-1}(x^2)dx$

$3\int_0^1 x\cos^{-1}(x^2)dx$

$3\left(\left[\underset{v}{\tfrac12 x^2}\,\underset{u}{\cos^{-1}(x^2)}\right]_0^1 - \int_0^1 \underset{v}{\tfrac{1}{\cancel{2}} x^2}\,\underset{du}{\frac{-\cancel{2}x}{\sqrt{1-x^4}}dx}\right)$

$3\left(\left[\tfrac12 x^2\cos^{-1}(x^2)\right]_0^1 - \int_0^1 \frac{-x^3}{\sqrt{1-x^4}}dx\right)$

$3\left(\left[\tfrac12 x^2\cos^{-1}(x^2)\right]_0^1 - \int_0^1 -x^3(1-x^4)^{-\frac12}dx\right)$

Let $w = 1-x^4$

$\frac{dw}{4} = \frac{-4x^3dx}{4}$

$\frac14 dw = -x^3dx$

$3\left(\left[\tfrac12 x^2\cos^{-1}(x^2)\right]_0^1 - \frac14\int_0^1 -4x^3(1-x^4)^{-\frac12}dx\right)$

$3\left(\left[\tfrac12 x^2\cos^{-1}(x^2)\right]_0^1 - \frac14\left[\frac{(1-x^4)^{\frac12}}{\frac12}\right]_0^1\right)$

$3\left(\left[\tfrac12 x^2\cos^{-1}(x^2)\right]_0^1 - \frac14\left[2(1-x^4)^{\frac12}\right]_0^1\right)$

$3\left(\left[\tfrac12 x^2\cos^{-1}(x^2)\right]_0^1 - \frac12\left[(1-x^4)^{\frac12}\right]_0^1\right)$

$3\left(\tfrac12(1)^2\cos^{-1}(1) - \tfrac12(0)^2\cos^{-1}(0) - \tfrac12\left(\sqrt{1-1^4} - \sqrt{1-0^4}\right)\right)$

$3\left(0 - 0 - \tfrac12(0-1)\right)$

$3\left(\tfrac12\right) = \boxed{\frac32}$

Figure 9.2: Example of final slides generated by our pipeline.

Chapter 10

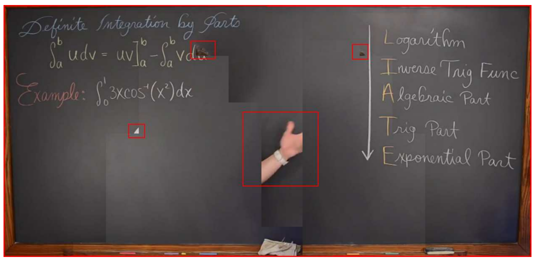
System limitation

While our system can successfully generate slides from the video lectures or presentations, the task we try to solve is rather complex in the sense, that the input videos or presentations may vastly vary. For this reason we had to make some assumption in terms of input videos. This leads to some restrictions that our system poses and to the reason why our system still fails to perform in certain cases. This chapter focuses on these restrictions and limitation of proposed system.

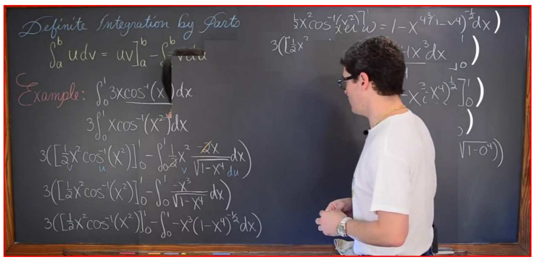
One of the major restrictions of the system is on the color of objects in front of the board. This might be major problem, if for instance lecturer has clothes of same color as the board color, parts of his clothes might get recognized as valid region of the board throughout the video. Which would result in invalid information being displayed on our slides as these parts would still be occluding information on the board as shown in Figure 10.1a.

Current methods responsible for generating final slide also do not performed any tests, if information stitched from multiple frames of the video is in any way relevant. In other words, if parts of the board that were stitched together should be actually displayed on one slide, if the information is in any way connected. This might lead to the loss of information in the slide which would result in the whole slide being invalid. This is also a problem as some of our methods, for example, absolute difference-based method, responsible for generating occlusion mask (Section 6.3), are heavily reliant on the fact that previously generated slides contain only valid information and are without any occlusion events. This relation can in the end result in whole sequence of frames, from the point where the first invalid slide was generated, being invalid (Figure 10.1b).

Another restriction that our system poses, is that videos have to contain only one or multiple boards but not separated by wider space as these boards can still be considered and are processed as one bigger board. If there are multiple boards present in the video or presentation, our board extraction methods proposed in Section 6.2, picks one that occupies the biggest area of the frame. As both of our board extraction methods select



(a) Invalid slide with highlighted error areas.



(b) Generated slide from whole sequence of invalid slides.

Figure 10.1: Invalid slides generated with our pipeline.

the biggest contour of rectangular shape present in the frame.

This can also cause problems with videos that use multiple cameras and frequently switch between them, which can lead to different board suddenly becoming bigger and being selected by our methods as the biggest one. This would most certainly cause new slides being frequently generated, as the amount of how much information changed detected by our change detection algorithm discussed in Chapter 7, would be sufficient. This inconsistency might not be desirable. But more importantly this inconsistency can also lead to wrong information being displayed on the slides as consequence of incremental nature of some of our algorithms.

Chapter 11

Conclusion

In our work, we present and compare methods for detecting and extracting boards from white or black board video based presentation. This setup can later be abstracted to detecting and extracting large scale object in image or video, that matches some sort of a predicate. In our case, it was colour and shape of an object. We got sufficient results with regards to board extraction with the best method being region growing.

We also tested and evaluated functions for comparing how similar are two images in order to generate key frames in the video sequence. Finally, we proposed methods for extracting information from given presentation even with occluding events present in the sequence. In this category, we also managed to achieve sufficient results with hashing functions with the best one being the pHash algorithm.

We proposed two methods for removing objects in front of our board. The performance of the absolute difference-based one was found to be better overall and chosen for the final pipeline. The slides created by our final processing pipeline can be seen in Figure 9.2.

Many of these algorithms can be improved. For example, after function that creates final slide, we can use post processing method to smooth transition between sections of the old image and the new one. Also we can add test to the "stiching" process of the image if the information, that is being stitched to the frame, is actually in any way connected to the rest of the slide. Such a test, could also help us to find the best matching slide where we could stitch the new information to the slide. This could partially solve our issue with incremental nature of our process. Or it might be possible to further shift perspective in order to better fit the current slide, if for instance, camera moved during the presentation. Support for multiple boards with bigger gaps between them can be added as well as support for multiple cameras, which can provide us with additional information such as depth of the scene. This can drastically improve accuracy with which occlusion mask is generated, since we could almost precisely identify, which objects are in front of the board. Also based on colour of a board, text on the slide can be further enhanced for latter use for example in OCR algorithm.

Given our focus on simplicity, performance and speed, we also believe, that the proposed algorithms might serve as a basis for a system that would produce slides as images from white and black board based videos in real time and could be integrated into bigger and more complex e-learning systems.

Bibliography

[1] H. V. Shin, F. Berthouzoz, W. Li, and F. Durand, "Visual transcripts: lecture notes from blackboard-style lecture videos," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, p. 240, 2015.

[2] T.-J. K. P. Monserrat, S. Zhao, K. McGee, and A. V. Pandey, "Notevideo: facilitating navigation of blackboard-style lecture videos," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1139–1148, ACM, 2013.

[3] L. Pappano, "The year of the mooc," *The New York Times*, vol. 2, no. 12, p. 2012, 2012.

[4] T. Volery and D. Lord, "Critical success factors in online education," *International Journal of Educational Management*, vol. 14, no. 5, pp. 216–223, 2000.

[5] P.-C. Sun, R. J. Tsai, G. Finger, Y.-Y. Chen, and D. Yeh, "What drives a successful e-learning? an empirical investigation of the critical factors influencing learner satisfaction," *Computers & education*, vol. 50, no. 4, pp. 1183–1202, 2008.

[6] B.-G. Lee, H.-H. Kang, and E.-S. Kim, "Occlusion removal method of partially occluded object using variance in computational integral imaging," *3D Research*, vol. 1, no. 2, pp. 6–10, 2010.

[7] T. Hosokawa, S. Jarusirisawad, and H. Saito, "Online video synthesis for removing occluding objects using multiple uncalibrated cameras via plane sweep algorithm," in *Distributed Smart Cameras, 2009. ICDSC 2009. Third ACM/IEEE International Conference on*, pp. 1–8, IEEE, 2009.

[8] T. Yeh, K. Tollmar, and T. Darrell, "Searching the web with mobile images for location recognition," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2, pp. II–76, IEEE, 2004.

[9] C. De Roover, C. De Vleeschouwer, F. Lefèbvre, and B. Macq, "Robust image hashing based on radial variance of pixels," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 3, pp. III–77, IEEE, 2005.

[10] C. Zauner, *Implementation and benchmarking of perceptual image hash functions.* na, 2010.

[11] J. Meng, Y. Juan, and S.-F. Chang, "Scene change detection in an mpeg-compressed video sequence," in *IS&T/SPIE's Symposium on Electronic Imaging: Science & Technology*, pp. 14–25, International Society for Optics and Photonics, 1995.

[12] T. Miyatake, S. Yoshizawa, and H. Ueda, "Method for detecting change points in motion picture images," Jan. 28 1992. US Patent 5,083,860.

[13] R. Szeliski, *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[14] Evan Klinger, "The open source vision library," 2015. [Retrived 2016-02-06] from `http://www.opencv.org/`.

[15] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2564–2571, IEEE, 2011.

[16] Evan Klinger, "The open source perceptual hash library," 2015. [Retrived 2016-02-06] from `http://www.phash.org/`.

[17] J. E. Hershberger and J. Snoeyink, *Speeding up the Douglas-Peucker line-simplification algorithm.* University of British Columbia, Department of Computer Science, 1992.

[18] Z. Peng, "Efficient matching of robust features for embedded slam," 2012.

[19] V. Y. Mariano, J. Min, J.-H. Park, R. Kasturi, D. Mihalcik, H. Li, D. Doermann, and T. Drayer, "Performance evaluation of object detection algorithms," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 3, pp. 965–969, IEEE, 2002.

[20] Evan Klinger, "The open source perceptual hash library," 2015. [Retrived 2016-02-06] from `http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html`.

Appendix A

Attached CD contains source code of our application as well as example video used in this thesis.