Patterns to Inform a Study Setup for Biometric Image Data Capturing

Artur Lupp*, Alexander G. Mirnig*, Thomas Grah*, Andreas Uhl† and Manfred Tscheligi*

*Center for Human-Computer Interaction, University of Salzburg, Austria
Email: name.surname@sbg.ac.at
†Department of Computer Sciences, University of Salzburg, Austria
Email: uhl@cosy.sbg.ac.at

***Abstract*—This paper presents an application of the contextual user experience (cUX) pattern approach for refining a study concept involving biometric image data. The study was concerned with the acquisition, inspection, and quality evaluation of Near Infrared (NIR) iris biometry images. After creating the initial draft of the study setup and during the design of the detailed study procedure, a number of questions arose, e.g., how to deal with the environmental light during image acquisition, which material to use for 3D printing, how to solve the problem of picking the right questionnaire, how to record high quality videos with mobile phones or even the differences between certain image formats. In order to capture and make these solutions more easily accessible, we used an adapted cUX pattern approach to provide the found solutions in the form of seven study design patterns.**

Keywords–design patterns; pattern reuse; study setup optimization.

I. Introduction

This paper is an extension of a full paper presented at PATTERNS 2017 [1]. Patterns, in general, are a well acknowledged method in Human-Computer Interaction (HCI), providing reliable and reproducible solutions for specific problems. They can be advantageously used to ease the communication between experts with different levels of expertise or even alternate disciplines. This is particularly useful in interdisciplinary areas and academic settings, where often a wide variety of levels of expertise are represented. During the design of an academic study with image recognition, we encountered a number of problems, which we found nontrivial and difficult to solve via standard literature, due to their specific nature. Since knowledge on pattern use and writing was available, we decided to capture the found solutions as patterns, in order to share them in an easy to access format.

The aim of this paper is to provide a more detailed description of the pattern writing process, a greater number of patterns than the original publication and finally an extended discussion. The already provided patterns, "Choosing the Right Light Sources to Examine NIR-Images Differences", "Lens Holder Construction for a Mobile Phone" and "Finding and Adjusting the Right Usability Questionnaire" covered the first steps of the study routine, whereas the more complex image acquisition and processing part had not been covered via patterns at that point. We took the opportunity to provide more solution patterns covering image acquisition and processing topics. Patterns (i.e., local binary patterns) in image application are commonly used to improve face detection and recognition. However, this type of patterns is not comparable to the solution patterns provided in this work. The solution patterns presented in this paper, aim to provide aid and helpful solutions for problems that may occur in the image application domain. With the addition of five new generated patterns covering the image application domain, this work now presents seven patterns in total.

Section II will give some insight into patterns and certain areas in the image application domain. A detailed explanation of the cUX pattern approach is presented in Section III. This section will describe the pattern generation process, starting from the context analysis and problem definition, whilst explaining all in-between steps until the finalization of a pattern. After the explanation of the pattern generation process, Section IV will provide an insight into the to be improved study setup. Section V will illustrate the seven solution patterns with the following *Titles*:

1) Choosing the Right Light Sources to Examine NIR-Images Differences
2) Lens Holder Construction for a Mobile Phone
3) High Quality Video Acquisition with the Nexus 5
4) Extract Media Information from Videofiles
5) Still Image Extraction from h264 Videos
6) Comparison Between Bitmap, Portable Network Graphic, and JPEG Images
7) Extraction of the Eye Area from Frontal Face Images

We will discuss our new findings, especially how to handle the new problems and the associated solutions in Section VI and conclude the paper in Section VII.

II. Related Work

This section will first provide a brief overview of patterns, their history, and pattern approaches. After that, a summary of relevant information and literature on image detection is provided.

A. Patterns

Patterns were first introduced by Christopher Alexander [2][3] as a means to capture working solutions for reoccurring problems in the field of architecture. His initial idea was to consider the act of constructing a building as the sum of a number of many individual problem solutions. These solutions, when described individually, can then be "rearranged" when constructing new or different buildings and not requiring the same problems be solved anew every time a new building is constructed. His ideas and methodology was adopted by Gamma et al. [4] for Software Engineering and related disciplines, and has been used as a tool in these domains since.

Patterns are nowadays considered less as an alternative to guidelines and other general means of guidance, and more as a

supplement. This is because general documentation approaches are often either simplistic or high level [5][6]. Pattern solutions, on the other hand, are firmly embedded in the context their problems occur in. This makes a specific pattern less generally applicable – only when the problem contexts match to a sufficient degree. But it also makes the solutions they describe more specific, as well as practice relevant, and lends them to be used by novices and experts alike [7]. Patterns have been adopted by other domains as well, such as Web Design and HCI [8][9][10] and have also been suggested as a general, discipline-independent knowledge transfer tool [11].

B. Image Application

The mobile phone domain made huge advancements in terms of hardware technology over the last decade. Software, however, does not keep up this trend, especially when looking into image acquisition with mobile phones. The commonly used format for images is JPEG [12], which offers a decent image quality while maintaining a small file size. Videos are saved within a MPEG4 [13] container format housing a H.264/MPEG-4 AVC [14] video stream. H.264/MPEG-4 AVC is typically used for compressed (i.e., lossy) recording to save bandwidth, it is possible to create lossless-coded regions by choosing a special profile. However, there is currently no mobile phone available that is capable of using this special profile, thus all videos recorded with a mobile phone are lossy. Fortunately, object detection in the compressed domain is commonly applied [15][16][17][18][19], thus, face detection on compressed images or videos should not prove too challenging.

Object detection is the basis for face detection, which is picked as the central theme in one pattern, and can also be applied on compressed images and videos [20]. Viola and Jones [21][22] described a method for rapid object detection by using simple distinctive features called Haar-like features and a cascade of classifiers. This method distinguishes the area where the face is detected from the background, thus providing the area of interest. However, as the set of classifiers is pretty simple, the general error rate is high. To improve the detection rates, it is possible to use an extension [23] of the before mentioned Haar-like features in form of an improved cascade set, specially trained to decrease the general error rate. OpenCV [24] offers a good library to utilize this feature set for face and eye detection.

III. Approach

Generating patterns is a process over multiple stages that involves individuals (in this case researchers) working together in collaboration to create high quality patterns. This section will describe our approach, as well as the overall pattern creation process. The first step is the context analysis, followed by the problem definition. During the context analysis, we looked at the underlying study setup we wanted to improve, identifying possible problems that may occur and how the overall process could be refined. In a following discussion session, we collected the results from the context analysis by defining the overall issues and problems. After collecting all problems and ideas, the identified items were rated on a priority scale and compiled into list at the end of the discussion round. Thereafter, the compiled items were arranged in the sequence of the study setup routine to allow a fluid workflow.

1) Initial Pattern Mining & First Iteration: The compiled list serves as a basis for the initial pattern mining and is completed by the previously mentioned researchers. Each researcher is assigned several problems. The number of assigned problems or items per researcher varies and depends on the number of participating researchers, time constraints (if any) and the priority of the problem. Commonly, three to five problems per researcher was the aim, as a problem statement might result in more than one pattern. Therefore, it is advisable to not exceed the recommended number of problems, to ensure that the researchers have a manageable level of workload. Apart from that, the researchers should be competent and well-versed regarding the academic side of the problems they work on. In the case of optimizing a study setup that mostly focuses on video acquisition, image extraction and image processing, this meant that all of the participating researchers should be at least familiar with video recording and processing with image data, with added benefit if they had specialization in face detection and feature extraction.

The next step during pattern mining is the decision whether the problem statement is a high or a low level problem and if its level of granularity is such that it requires a single pattern or needs to be split into several patterns. Each researcher is then instructed to mine publications, presentations, demos, prototypes, books and other useful and available sources for solutions to the problem in question. The next step is the combination of the partial solutions and references into one full solution for the draft pattern. A draft pattern is written, beginning with a self explanatory *Title*. Then, each pattern is divided into six sections.

1) The *Intent* provides a short description and is followed by the
2) *Problem* statement, which is, in this case, a question.
3) After stating the problem, a *Scenario* is presented that is used as an example,
4) for which a *Solution* is provided.
5) The solution is backed up by *Examples*, usually illustrated with images.
6) The pattern ends by providing *Keywords*, matching the subject of the pattern.

The draft is then handed to another researcher for a first internal iteration, completing the initial pattern.

2) First Iteration Workshop: The first iteration workshop ideally consists of some or all participants that took part in the context analysis, as well as participants not previously involved, with the aim of introducing new viewpoints. The initial patterns are then read thoroughly by each participant. Each subcategory of a pattern (Title, Intent, Problem, etc.) is rated individually on a 5-point scale via a rating system provided by Wurhofer et al. [25]. If a pattern is rated 3 or lower in any subcategory, it is marked for iteration. After rating each pattern, the participants participate in a discussion session and conclude the workshop. The main goal in this round is to discuss the general pattern quality, as well as the overall impression of the collection, to identity problematic patterns.

3) Second Iteration & Workshop: The feedback and ratings gained from the first iteration workshop are worked into the patterns during the second iteration. This time, each pattern is iterated by at least two researchers, who are versed in the specific topic address by the pattern(s). The focus is on

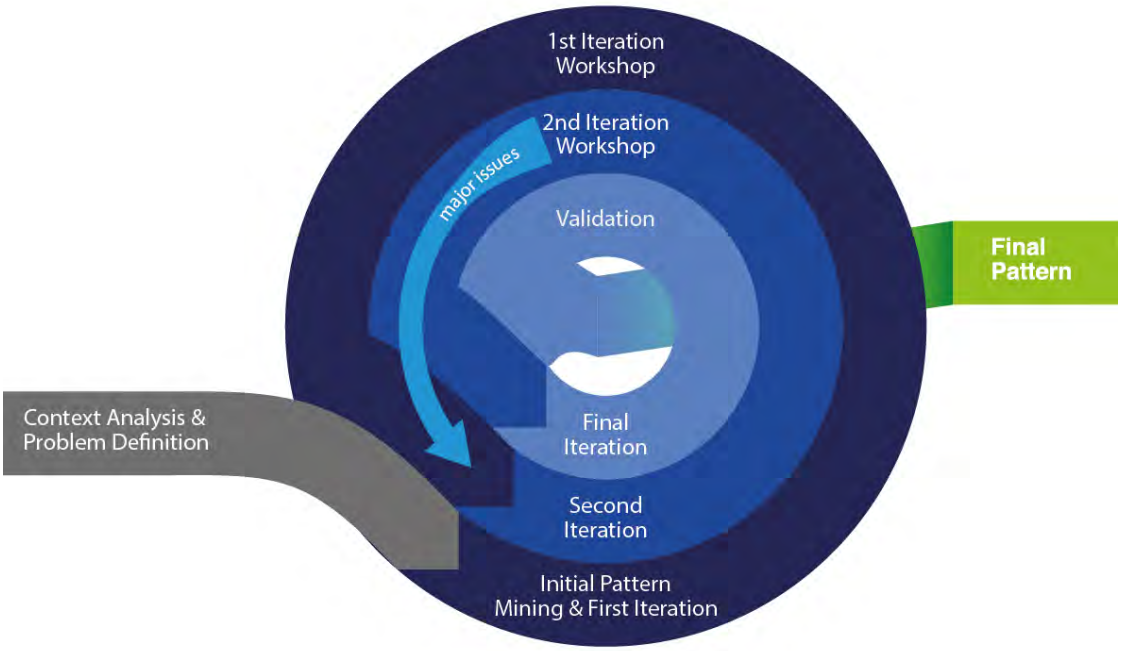

Figure 1. Graphical representation of the Pattern Generation Process adapted from [26].

improving and ensuring the practical relevance of the pattern by providing, e.g., additional implementation examples and best practices. As in the initial pattern creation, each pattern is cross-iterated again by a different researcher for typos, errors, etc., in order to provide a full and complete solution after this iteration. All additional iterations should only be targeting structure improvement, readability, and comprehensibility. Upon completion of this iteration, a second workshop following the same routine as the first one is conducted. It is advised to include new participants who were not part of the first workshop to provide an unbiased view in order to help to identify issues that might have been missed in the first workshop.

4) Final Iteration and Validation: Each pattern is again reworked with respect to the feedback and ratings gained in the second workshop. At this point, most minor issues should have been identified; however, it is still possible to encounter major issues. When a major issue is found in a pattern, it reenters the the reworking loop for another iteration workshop. Usually patterns that reenter the loop are put aside temporarily to ensure a fluid workflow. These patterns are taken up again, if either a new batch of patterns is created, the appropriate iteration phase is reached, or the rest of the patterns are finished.

Patterns with minor issues are corrected accordingly and enter the final validation stage of the pattern creation process. During this stage, each pattern is again rated using the same rating system as before, but without the workshop setting. If a pattern receives a rating of 3 or below in this stage, it reenters the same reworking loop as the patterns with major issues mentioned before. This happens rarely, if ever, as problematic patterns are usually identified before reaching this stage. All ratings above 3 validate the pattern, marking it as finished.

IV. Study Setup

We wanted to improve and optimize an existing study setup, dealing with biometric images. These biometric images had to be analyzed afterwards, with respect to image quality. The setup was divided into several steps. During the first step, study subjects have to capture videos with a customized LG Nexus 5 mobile phone. The IR-blocking filter was removed from the rear camera image sensor, to enable NIR image capturing. The built-in rear camera image sensor is a Sony Exmor R IMX 179. The sensor offers a Red-Green-Blue (RGB) sub pixel layout with $3264x2448$ (8 MegaPixel) pixels and a sensor size of $5.68mm$ $(1/3.2")$, leading to an effective pixel size of $1.4\mu m$. The pixel size is decent for a mobile phone released in 2013. Therefore, taking images or videos in twilight conditions is possible. However, a brighter environment is preferred due to less image noise. Each test subject had to record three frontal face videos using the stock camera lens and two different filters / lenses, which were mounted on the mobile phone. Afterwards, the test subjects had to fill in a questionnaire. Due to the time consuming video capturing process, the questionnaire needed to be short, while still maintaining a decent reliability.

The Nexus 5 was chosen because it was easily available at the time and it allows removing the IR-blocking filter, which is often permanently integrated (i.e., nondetachable) in other models on the market. Removing the filter is necessary for enabling NIR image capturing via the described method. The built-in rear camera image sensor is also integrated in, e.g., the Google PIXEL smartphone as a front facing camera and the approach described here is not limited to only this particular smartphone. While technology changes and advances, in the case of smartphone technology quite rapidly sometimes, the method for capturing images via the described method is likely to stay the same, barring differences in pixel size, pixel matrix

on the image sensor, pre- or post processing. None of these impact the image making process in any significant way. Thus, the described process should be relatively robust to future technology advances, provided the models used allow removal of the IR-blocking filter.

We proposed patterns to refine the study concept using an approach similar to the pattern generation process for car user experience patterns described in detail by Mirnig et al. [27], with some minor changes. The first mandatory step in our approach was to analyze the study concept and the associated setup to extract the problem statements. This was done by organizing a workshop with the person responsible for the study concept and a group of HCI researchers accustomed with the pattern generation process. During the workshop, the study setup was explained as follows. Study participants have to capture three frontal face videos, one for NIR and visible light images without any lens, one with the IR-blocking filter / lens, and one with the NIR-only lens. As it is possible to extract high quality images from high resolution videos, it was decided to capture only videos instead of pure frontal face images. The two different lenses forced the researcher responsible, to change them after every recording, due to the current lens mounting method. To ensure a variety of captured videos, the test subjects had to record the videos in different light environments, which where not yet defined. The final step was the acquisition of data, relating to the usability of the video recording process. As the video capturing procedure was time consuming, the data acquisition had to be fast and reliable. The first workshop brought up the following main problems:

1) Which light sources and ambient environments need to be considered, to ensure a diversity of captured image or video data usually acquired during real life usage?
2) How can the lens / filter changing process be improved?
3) Is it possible to record higher quality videos with a LG Nexus 5?
4) Which tool can be used to extract media information from video files?
5) How can still images be extracted from videos recorded by a mobile phone?
6) Which file format should be used for an image when its extracted from a mobile phone video?
7) How can the eye area can be extracted from a frontal face image?

For each problem, a draft pattern was created. The draft pattern initially did not provide any final solutions. Thus, it was iterated and reworked until a working solution was found. After that, the pattern was rated and reworked again until it was finally validated. In the next section, we will present the solution patterns we generated. Each pattern provides a solution for a certain problem statement, previously mentioned in this section.

V. Solution Patterns

A. Choosing the Right Light Sources to Examine NIR-Images Differences

Intent: There are several variables one needs to take into account when taking pictures or videos with a mobile phone. Due to the usually small built-in image sensor in mobile phones, sufficient environmental light is a crucial point. Insufficient light leads to higher image noise, which is generally not preferred. However, to analyze a wide area of possible real life conditions, selecting different environments for image capturing is important. This pattern presents three possible scenarios covering the most important lighting conditions. The scenarios were selected to provide images with a quality sufficient for subsequent analysis in mind.

Problem: Which scenarios are needed in order to acquire analyzable data, covering indoor and outdoor lighting conditions that enable NIR image acquisition?

Scenario: The study needed special image acquisition scenarios to reflect actual real life scenarios as closely as possible. Additionally, the ambient light in at least one of the scenarios had to cover the NIR wavelength ($>= 700nm$) spectrum to enable NIR imaging.

Solution: To cover most real life scenarios of possible image capturing conditions, we proposed three scenarios: one outdoor scenario using indirect sunlight (e.g., via a glass reflection in the background) to enable NIR imaging and two indoor scenarios using different light environments to challenge the imaging sensor of the mobile phone.

- **Outdoor (variable ambient light conditions)** - The outdoor scenario is and should be variable. In this condition, the sun is providing the ambient light. Therefore, the image quality is depending on time, weather, and location. To ensure the best possible conditions for NIR image acquisition, daylight is necessary. Therefore, image acquisition in this scenario should be done during the daytime. An example of the outside condition is shown in Figure 3.

- **Indoor (dim light)** - The indoor scenario using a dim light source is intended to challenge the image sensor. The indirect artificial light provides sufficient luminosity for images to be taken, as pictured in Figure 4. Nevertheless, the provided light is dark enough to force the image sensor to use a higher sensitivity setting (this is also referred to as "ISO"), thus, resulting in more image noise. Note that image noise is not desirable in general, but, if the main concept of the study is to analyze the whole range of possible image qualities, it is mandatory to include this unfavorable condition.

- **Indoor (bright light)** - In contrast to the dim light indoor scenario, the bright light indoor scenario uses a very bright artificial white light source to illuminate the frontal face area. This scenario complements the previously mentioned scenarios. The bright artificial light, covers the spectrum visible to the human eye (from about 390 to $700nm$) and provides a decent environment needed to capture regular frontal face images and can be observed in Figure 5. However, conventional light sources are usually not suitable for NIR imaging, as they do not cover the spectrum above $700nm$ (see Figure 2).

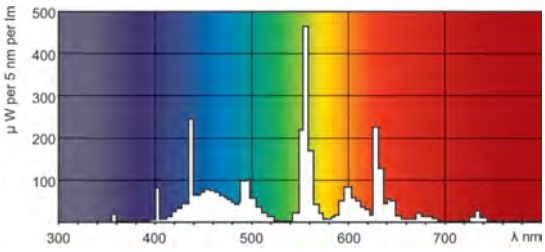

Figure 2. Philips TL5 HO 49W 865 Lamp [28] - Photometric Data.

Examples: This section shows nine sample images. They are grouped by the three proposed scenarios. Each group consists of three images: NIR only, NIR & visible light, and visible light only.

Figure 3. Outdoor - NIR only, NIR & visible light, visible light only (from left to right).

As mentioned in the solution section, the outdoor scenario provides sufficient light. This scenario provides the best NIR image quality, as the sunlight covers a wider spectrum compared to conventional light sources.

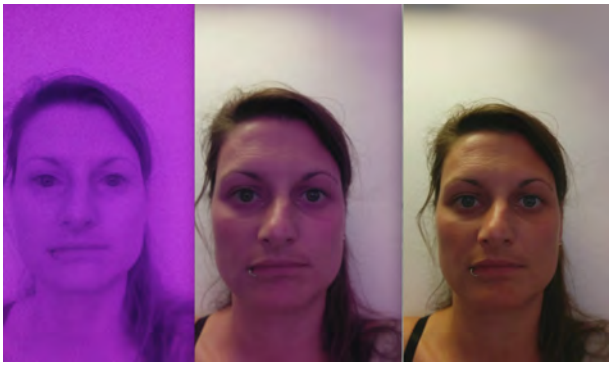

Figure 4. Indoor (dim light) - NIR only, NIR & visible light, visible light only (from left to right).

The indoor scenario with a dim indirect light source tends to induce image noise and is not optimal for NIR imaging.

Figure 5. Indoor (bright light) - NIR only, NIR & visible light, visible light only (from left to right).

The last scenario provides a direct illumination of the facial area. It is very favorable for images captured in the visible spectrum, e.g., due to reduced image noise.

Keywords: NIR, visible light, wavelength, spectrum, image acquisition, illumination

B. Lens Holder Construction for a Mobile Phone

Intent: This pattern describes steps-by-step the construction of a lens holder for the Nexus 5 mobile phone.

Problem: Is it possible to create a method or item to reduced the lens change time and make the whole process more comfortable?

Scenario: Two different filters / lenses are each to be mounted on the mobile phone using a clip. This is very time consuming and elaborate. To ease the transition from one lens to another, they had to be mounted on a movable holder with the possibility to be mounted on the mobile phone.

Solution: A custom made movable lens holder mounted on a hard shell mobile phone case. The following points are describing a step-by-step guide to construct a lens holder for a mobile phone case:

- First, get a hard shell mobile phone case to work with. The case should be made of a robust material, e.g., polycarbonate. The easiest way to obtain a good mobile phone case is either by buying it or by printing one using a 3D printer. Note that the camera lens of the mobile phone should not stick out of the case, when it is mounted on the phone, as it will be tough or impossible to rotate the custom made lens changer afterwards.
- Measure the phone case and the lens width, length, and depth. Measurements should be taken as precisely as possible.
- Sketch the available items (i.e., lenses and phone case) with the measurements from the previous step.
- The sketch is then used to figure out, how to arrange the lenses in a way that allows them to cover the camera lens of the phone when the lens changer is being rotated.
- With the lenses arranged, pick a focus point between them. This is the pivot point of the lens changer. In our case, this point is the small circle in between the two bigger ones, illustrated in Figure 7.

- Craft a paper prototype of the lens holder. Sketch the lens changer with the exact measurements and cut it out. This prototype can be used to simulate the finished product. Try it out, and see if it fits your expectations, as depicted in Figure 6.
- Digitize the sketch and construct a 3D model. Note that it may be beneficial to add some room to move, especially if using a 3D printer that is not 100% accurate. An example of the digitized model is pictured in Figures 7 and 8 (left).
- Print the 3D model with a material that allows editing with tools (i.e., a file or a multifunction rotary tool) later on. In this case, PVC was used.
- Deburr the edges whilst occasionally trying to fit in the lenses. When everything fits accordingly, proceed with the next step. If anything is odd or needs refinement, redo the 3D modeling and print the item again.
- Drill the pivot point holes into the 3D printed item, as well as in the phone case, to combine them later on.
- Temporarily mount the printed lens holder to the phone with a screw, as shown in Figure 8 (right).
- Double check if everything is according to your needs.
- Finally, install the lenses into the lens holder and mount it to the phone case. See Figure 9 for the final result.

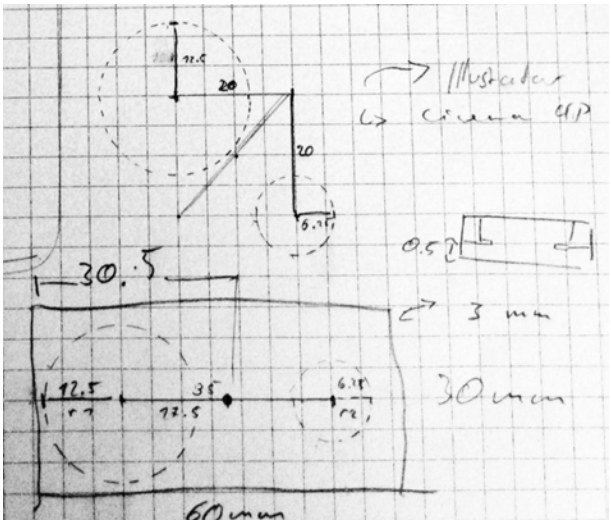

Figure 6. Sketch of the lens holder with exact measurements and radius.

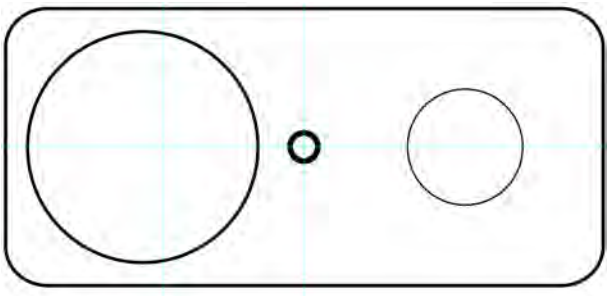

Figure 7. Digitized 2D model of the sketched lens holder.

Figure 8. Lens holder 3D model (left). Printed lens holder with installed lenses/filters (right).

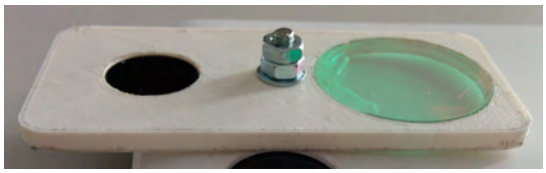

Figure 9. Final lens holder mounted on the phone case.

Examples: Figure 10 holds a QR Code that is linked to a video showing the lens holder in action. Figure 11 is picturing the effect of the different lenses on image acquisition.

Figure 10. YouTube Video - Nexus 5 Lens Holder Case [29].

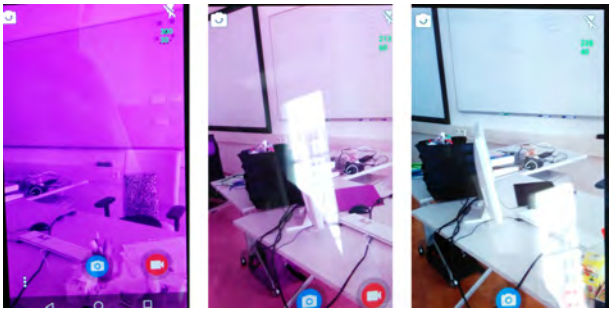

Figure 11. NIR, NIR and visible light, visible light only by using IR-blocking lens (from left to right).

Keywords: NIR, lens holder, phone case, PVC, polycarbonate, 3D modeling, 3D printing

C. High Quality Video Acquisition with the Nexus 5

Intent: This pattern describes the best way to record high quality videos on a Nexus 5 mobile phone.

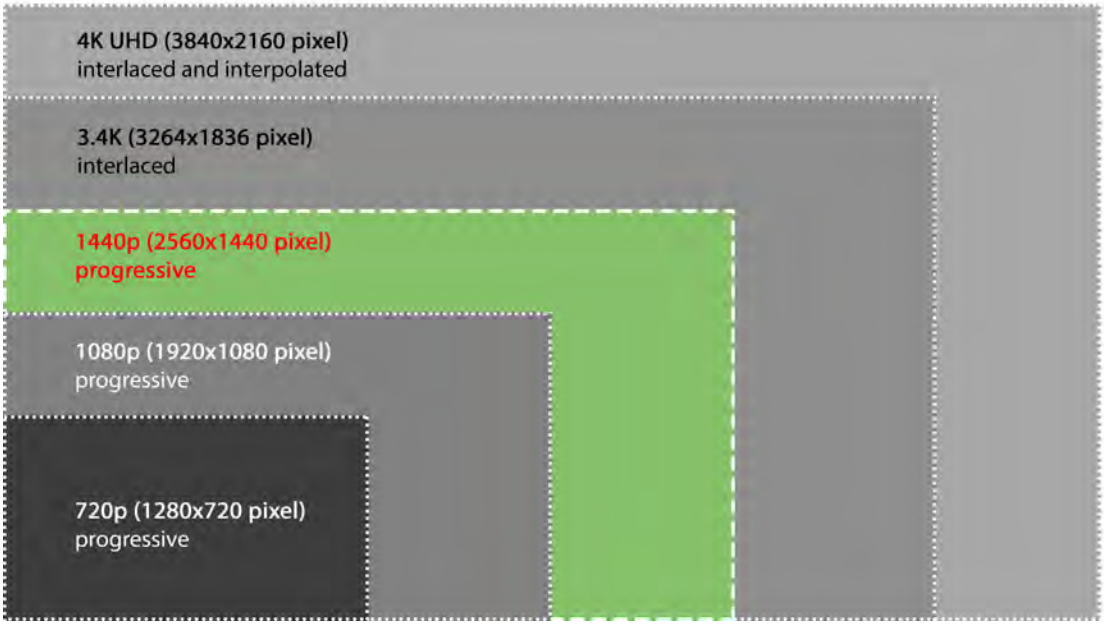

Figure 12. Comparison of available video resolution options for the LG Nexus 5 using SnapCam.

Problem: Out of the box video recording with mobile phones using the pre-installed video recording applications have certain limitations. Usually, the applications offer only a handful of pre defined resolution options to record videos in certain qualities, e.g., $1080p$ ($1920x1080$ pixel) for FullHD or $720p$ ($1280x720$ pixel) for HDReady. However, these options are usually not the highest technically possible video quality options the phones built-in image sensor might provide.

Scenario: For post-processing reasons, high quality still images have to be extracted from recorded video. Thus, the videos have to be recorded in the highest quality possible.

Solution: To enable the best possible video capturing quality on the Nexus 5, it is necessary to use a special application that is capable of exploiting the phones image sensor. Currently, the only app capable to do this using the Nexus 5 mobile phone is Snap Camera [30]. Snap Camera has a feature [31] that enables the recording of higher resolution videos with the built-in Nexus 5 image sensor (Sony Exmor R IMX 179).

The highest resolution with progressive video recording (i.e., each recorded frame is a full picture) provided by the application is $1440p$ ($1440x2560$ pixel).

Choosing the $1440p$ option has certain advantages:

- Higher resolution (compared to $1080p$ or $720p$).
- No interpolation (compared to $4K$).
- Progressive video recording (compared to $3.4K$ or $4K$).

However, there are some points to take into consideration:

- Using a higher resolution during video recording increases energy consumption. The battery will discharge faster.
- Snap Camera is not free; the app needs to be purchased for full use.

Examples: As you can see in Figure 12, the native resolution of the Nexus 5 image sensor is $3264x2448$ pixel. By choosing the available recording options provided by the google stock camera application $1080p$ or $720p$, the video would only use a fraction of the available resolution. The best choice for motion videos is $1440p$. This option records full pictures for each video frame. Thus, it is possible to extract single frames yielding the best possible image quality.

To enable the higher resolution video recording options in Snap Camera, it is necessary to toggle the Google Camera2 API and OpenGLES 2.0 settings in the "Other" menu from the application, as shown in Figure 13. Thereafter, it is possible to select $1140p$, $3.4K$, and $4K$ UHD as recording options (see Figure 14).

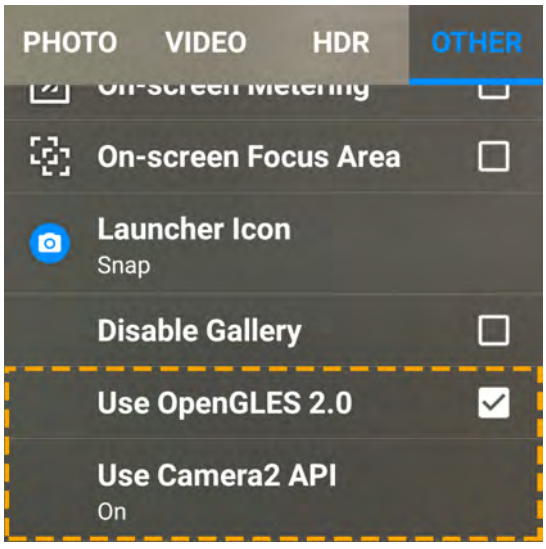

Figure 13. This Figure shows the adjustments that have to be made in SnapCam to enable video recording with higher resolutions on the Nexus 5.

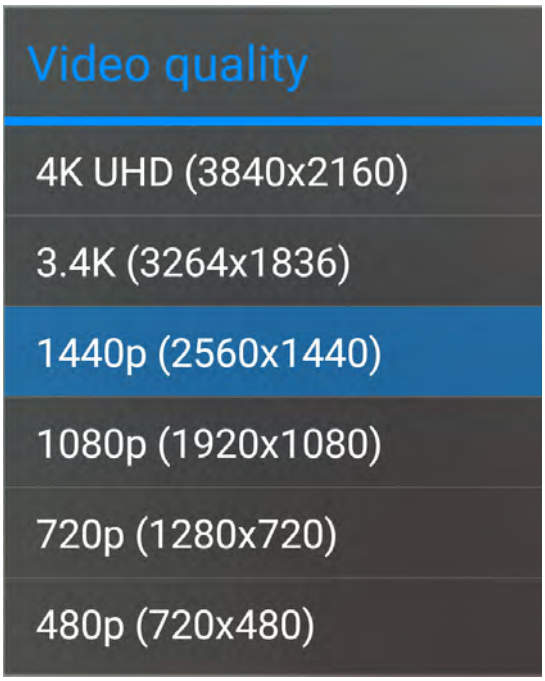

Figure 14. Listing of the available video recording options SnapCam is offering after unlocking higher resolutions.

Keywords: video recording, video acquisition, resolution, Nexus 5

D. Extract Media Information from Videofiles

Intent: This pattern describes the extraction of media information from video files with the help of FFmpeg or FFprobe.

Problem: Working with video files may prove as challenging, particularly if there is only limited knowledge on the settings (i.e., frame rate, codec, interlaced or progressive recording) used. However, this knowledge is vital for post processing video files and should, therefore, be brought to knowledge as soon as possible.

Scenario: In order to work efficiently with video files, the video specifications have to be acquired before even starting the post processing.

Solution: The first step is the installation of the FFmpeg [32] multimedia framework. The framework offers a variety of functions apart from scanning media files or extracting frames from video files and, therefore, is recommended for this task.

To scan a video file recorded with a common device, such as mobile phones or video cameras with FFprobe or FFmpeg, type in the following in a command window:

```
ffprobe <video_filename>
```

or

```
ffmpeg -i <video_filename>
```

Note that using FFmpeg / FFprobe to scan a file may take some time, depending on the input file duration and decoding complexity.

Examples: An example output for scanning the file "video.mp4" with FFmpeg / FFprobe:

```
ffmpeg -i video.mp4

Input #0, mov,mp4,m4a,3gp,3g2,mj2, from
    'video.mp4':

  Metadata:
    major_brand : mp42
    minor_version : 0
    compatible_brands: isommp42
    creation_time : 2016-04-15T12:42:54.000000Z
    com.android.version: 6.0.1
  Duration: 00:00:06.64, start: 0.000000,
      bitrate: 23292 kb/s
    Stream #0:0(eng): Video: h264 (Baseline)
        (avc1 / 0x31637661), yuv420p,
        2560x1440, 23857 kb/s, SAR 1:1 DAR
        16:9, 30.72 fps, 90k tbr, 90k tbn, 180k
        tbc (default)

    Metadata:
      rotate      : 90
      creation_time :
          2016-04-15T12:42:54.000000Z
      handler_name : VideoHandle
    Side data:
      displaymatrix: rotation of -90.00 degrees
    Stream #0:1(eng): Audio: aac (LC) (mp4a /
        0x6134706D), 48000 Hz, mono, fltp, 96
        kb/s (default)
    Metadata:
      creation_time :
          2016-04-15T12:42:54.000000Z
      handler_name : SoundHandle
```

The second line mentions the video file (i.e., video.mp4) used to generate the output. The relevant video information is found in the Stream section of the video Metadata part.

This Stream holds the following important information:

- Stream Number: The first video stream here is declared as #0.0, whereas the audio stream is declared as # 0:1 and followed by the language flag.
- Video Codec: In this case, the video was coded with "h264" using the "baseline" profile. Baseline is commonly applied for lower cost applications with limited hardware resources, e.g., for video conferences or mobile applications. Regarding the information within the parentheses, "avc1" is a different name for the H.264 codec, whereas "0x31637661" is a four character code (Hex to ASCII) equivalent: 0x61 = "a", 0x76 = "v", 0x63 = "c", 0x31 = "1".
- Colorspace: YUV420P is used for storing raw image data at a ratio of 4:2:0, meaning there is one color sample for every 4 luma samples. Thus, the color information is quartered (i.e., saving video bandwidth).
- Resolution: The file was recorded with the resolution of 2560x1440 pixel.
- Storage Aspect Ratio: The SAR defines the ratio of pixel dimensions. Square pixels are 1:1, whereas 1:2 for example would describe a rectangular pixels.

- Display Aspect Ratio: DAR defines the ratio of the width to height of a video file. The ration 16:9 is commonly known as Widescreen.
- Frame Rate: The Frame rate, expressed as frames per second or fps, is the rate at which consecutive frames (i.e., images) are displayed during video playback. In this example, the video has 30.72 frames that are displayed every second.
- tbr, tbn and tbc: These three values are three different timestamps FFmpeg / FFprobe provides.

Keywords: H.264, metadata, FFmpeg, FFprobe

E. Still Image Extraction from H.264 Videos

Intent: This pattern describes one of the best ways to extract high quality still images from H.264 videos.

Problem: There are several ways to extracting still images from a video e.g., with the highest quality possible. One option is taking screenshots by using common video player software (e.g., VLC [33][34]). However, this solution yields a low image quality.

Scenario: For post-processing reasons, high quality still images needed to be extracted from pre-recorded videos. After assuring that the recorded videos had the best possible quality, still images have to be extracted with the least loss of quality.

Solution: The first step is to extract the media information, as explained in Pattern *Extract Media Information from Videofiles*. It is vital to know the video codec and the frame rate, which was used to record the video, in order to extract the images. Finally, FFmpeg is used with the acquired information to extract the still images with the code provided in the example section.

Examples: In this example, the video was recorded on a LG Nexus 5 using the Snap Camera application with the $1440p$ option:

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from
    ''video.mp4'':

  Metadata:
    major_brand : mp42
    minor_version : 0
    compatible_brands: isommp42
    creation_time : 2016-04-15T12:42:54.000000Z
    com.android.version: 6.0.1
  Duration: 00:00:06.64, start: 0.000000,
      bitrate: 23292 kb/s
    Stream #0:0(eng): Video: h264 (Baseline)
        (avc1 / 0x31637661), yuv420p,
        2560x1440, 23857 kb/s, SAR 1:1 DAR
        16:9, 30.72 fps, 90k tbr, 90k tbn, 180k
        tbc (default)

    Metadata:
      rotate      : 90
      creation_time :
          2016-04-15T12:42:54.000000Z
      handler_name : VideoHandle
    Side data:
      displaymatrix: rotation of -90.00 degrees
```

The important variables are:

- Filename: video.mp4
- Duration: 00:00:06.64
- Video Coced: h264

The following code shows how to extract still images from a "video.mp4" file using FFmpeg in a terminal (in this example, the command is executed in the same folder as the video file):

```
ffmpeg -ss 00:00:04 -t 00:00:00.04 -i
    video.mp4 -qscale:v 2 -r 30.72
    frontal\%4d.jpg
```

- [-ss 00:00:04]
  - This part of the command defines the start time of the image extraction. Ideally, this should be done during a frontal face scene with open eyes. In this case, the starting time is 00:00:04.
- [-t 00:00:00.04]
  - This part of the command defines the length of the timeframe in which images will be extracted. Here, the extraction will stop after 0.04 seconds.
- [-i video.mp4]
  - This command defines which input file should be used. It is possible to point the full path. In this case, the video is in the same folder and named "video.mp4".
- [-qscale:v 2]
  - -qscale:v is responsible for the quality of the extracted image. For .jpeg images, it is possible to use values between 2 and 31. The higher the number, the higher the .jpeg compression and, therefore, worse image quality. For best results, values between 2 and 5 should be used.
- [-r 25.0]
  - This part defines the frame rate. In our case, we are using 25.0 frames per second, i.e., one frame every 1/25 seconds.
- [frontal%4d.jpg]
  - This part can be divided in thee parts. "Frontal" is the name of the image, whereas the "%4d" part is a 4-digit automatically incremented number with leading zeros. In the case that multiple images are extracted from a video, this may come in handy. The final part is the file format ".jpg". In this example, ".jpg" image format is used to encode the extracted images.

Keywords: H.264, image extraction, FFmpeg

F. Comparison between Bitmap, Portable Network Graphic, and JPEG Images

Intent: This pattern shortly describes the differences, pros, and cons of .bmp, .png and .jpeg images.

Problem: As described in Pattern *Still Image Extraction from H.264 Videos*, still images can be extracted as .bmp, .png, and .jpeg files from video files. Which format to use, however, depends on certain characteristics the images have to meet, e.g., for post processing.

Scenario: Still images have to be extracted from a video file recoded with an android mobile phone. Now, it is a question of which file format to use for the image extraction.

Solution: Taking three variables into consideration - speed of extraction, image quality and file size, it is possible to quickly decide on a specific file format for the image extraction.

Speed of Extraction: If the speed of extraction is the crucial variable, .bmp is the best choice. Extracting frames as uncompressed .bmp files is the fastest way, due to the minimal processing power needed to extract the images from a vide file. In terms of extraction speed, .jpeg files come after .bmp. The .jpeg extraction is performance intensive, though, still faster than .png. Concluding in terms of extraction speed: $.bmp > .jpeg > .png$.

Image Quality: When image quality (e.g., no or less artifacts) is the main factor for the decision which file format to use, .bmp or .png files are the best choice. Both file formats allow lossless saving of image data. During the extraction process, .jpeg images always produce blocking artifacts, depending on the quality parameter used for .jpeg encoding as pictured in Figure 16. Concluding: $.bmp = .png > .jpg$ in terms of image quality.

File Size If a small image file size is targeted, then .jpeg should be preferred. In general, images using the .jpeg file format offer a small file size due the compression with the tradeoff in terms of image quality. While .png files are compressed as well, the compression is lossless and, therefore, resulting in a bigger file size compared to .jpeg. .bmp files are lossless as well, however, they are not compressed and yield a higher file size. As a rule of thumb in terms of file size: $.jpeg < .png < .bmp$.

Examples: This section shows a comparison of the different file formats .bmp, .png, and .jpeg with respect to extraction time, image quality, and file size. The **extraction time** (user + sys = cpu time used), in seconds, was acquired by inserting the "time" command before the FFmpeg extraction routine, which is a variation if the command presented in Pattern *Still Image Extraction from H.264 Videos* that extracts **n** frames per second from a $7.57s$ long video. The results can be seen in Table I for **n = 1**, Table II for **n = 2** and Table III for **n = 4** respectively.

```
time ffmpeg -i extract.mp4 -vf fps= n
    -qscale:v 2 frontal\%4d.jpeg
```

TABLE I. Extraction time for **n = 1**

	Extraction time **t** in seconds for 8 extracted images
.bmp	$22.216s + 0.478s = 22.694s$
.jpeg	$22.515s + 0.327s = 22.842s$
.png	$27.560s + 0.414s = 27.974s$

TABLE II. Extraction time for **n = 2**

	Extraction time **t** in seconds for 15 extracted images
.bmp	$22.445s + 0.677s = 23.122s$
.jpeg	$22.802s + 0.392s = 23.194s$
.png	$32.100s + 0.486s = 32.586s$

TABLE III. Extraction time for **n = 4**

	Extraction time **t** in seconds for 29 extracted images
.bmp	$22.489s + 0.985s = 23.474s$
.jpeg	$23.525s + 0.395s = 23.92s$
.png	$41.358s + 0.591s = 41.949s$

During encoding, .jpeg images produce blocking artifacts. Depending on the quality settings, the artifacts can be quite present as shown in Figure 16. To visualize the difference in terms of **image quality**, a comparison was done by calculating peak signal-to-noise ratio. Figure 15 shows the difference between two extracted frames, one using .bmp and the other .jpeg file format. The mostly red parts in the middle indicate the variance between the images ,whereas the white dots point out the identical parts. Typical PSNR values for compressed images range between $30dB$ and $50dB$, where higher is better; this comparison has a PSNR of $42.7731dB$ for all color channels.

Figure 15. Representation of image differences (middle) between an extracted .jpeg (left) and .bmp (right) frontal face image.

Figure 16. .jpeg image saved with highest quality settings (left) and with the lowest possible quality settings (right).

These are the **file size** differences of an extracted frame. The image used is the same as in Figure15, from a H.264 coded video with a resolution of $3840x2160pixel$.

- **.bmp**: 24.9MB
- **.png**: 6.2MB
- **.jpeg**: 448kB

Note that the size itself heavily depends on the resolution. Therefore, this example shows the variance in file size between the tree image formats.

Keywords: H.264, still image extraction, extraction time, image quality, file size, FFmpeg

G. Extraction of the Eye Area from Frontal Face Images

Intent: This pattern describes the detection of the eye area in frontal face images with the help of the OpenCV library.

Problem: Detecting eyes in an image is not as simple as it may seem at first. Object detection algorithms with the aim of finding eyes, for example, can not distinguish whether the detected area is a real eye or just something that the algorithm interprets as an eye. Therefore, it is necessary to enhance the detection rate by defining a certain region of interest by detecting the face first, in which the eyes can be found, before starting the eye detection routine.

Scenario: The eye area has to be detected and extracted from frontal face images for post processing.

Solution: Before programming the eye detection and extraction function, it is mandatory to prepare the following things:

- Frontal face image(s) that will be used for eye detection and extraction.
- A working installation of the OpenCV library [24] (installation guides for Windows [35], macOS [36], Linux [37])
- Haarcascade files for Frontal Face and Eye detection [38].

If needed, further information on Haar-like features can be found in Viola and Jones [21][22]. If all the aforementioned things are prepared, implement the eye detection, and extraction routine based on the following code example (C-Code translated from Python with added comments; adapted from [39]):

```
# Pseudocode example for the extraction of
    the eye area
program eye_extraction
# Load a frontal face image and haarcascades
    for face and eye detection
LoadFiles()
  image =
      cv.LoadImage('frontal_face_image.png')
  faceHaarCascade =
      cv.Load('haarcascade_frontalface_alt.xml')
  eyeHaarCascade =
      cv.Load('haarcascade_eye.xml')
# Face and Eye Detection
DetectFaceAndEyes(image, faceHaarCascade,
    eyeHaarCascade)
  # Convert the color image to grayscale for
      post processing
  grey = cv.CvtColor(image, gray)
  # Detect face
  face = cv.HaarDetectObjects(image,
      faceHaarCascade)
  # If faces is found
  if face:
  for ((x_pos, y_pos, width, heigh), n) in
      face:
  # Create a bounding box around the face area
  point1 = (int x_pos, int y_pos)
  point2 = (int (x_pos + width), int (y_pos +
      heigh))
  cv.Rectangle(image, point1, point2)
  # Estimate the eyes position by setting the
      region of interest and remove the lower
      part of the face image to reduce the
      probability for false recognition
  # The removal of the lower part can be seen
      in the last devision 'int((point2[1] -
      point1[1]) * 0.6))'.
  # The '0.6' in the last devision indicates
      that approximately 1/3 of the lower
      part of the face is cut out
  cv.SetImageROI(image, (point1[0],
              point1[1],
              point2[0] - point1[0],
          int((point2[1] - point1[1]) * 0.6)))
  # Detect the eyes
  eyes = cv.HaarDetectObjects(image,
      eyeHaarCascade)
  # If eyes were found
  if eyes:
  # For each eye found
  for eye1 and eye2 in eyes:
  # Draw a rectangle around the eyes (code
      applies if eyes are horizontally
      aligned)
  point1 = eye1_x_pos, eye1_y_pos)
  point2 = (eye2_x_pos + eye2_width,
      eye2_y_pos+ eye2_heigh)
  cv.Rectangle(image, point1, point2)
  # Reset the image region of interest for
      the image to be drawn correctly
  cv.ResetImageROI(image)
  # Extract the eye area and save it
  eye_area = cut.Out(image, point1, point2)
  save.Image(eye_area, 'eye_area.png')
```

Examples: A frontal face image like the one depicted in Figure 17 should be loaded into the the program described by the code example.

Figure 17. Frontal Face Image used for face and eye detection and feature extraction.

Figure 18 shows a) a blue rectangle for the detected face area, b) two orange rectangles for each detected eye, and c) the final rectangle around both eyes covering the to be extracted eye area. The final output after extraction should look like the example image pictured in Figure 19.

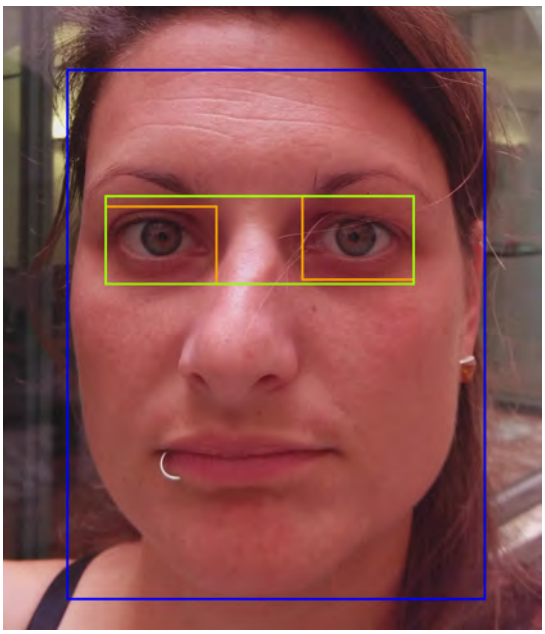

Figure 18. Extracted Frontal Face Image.

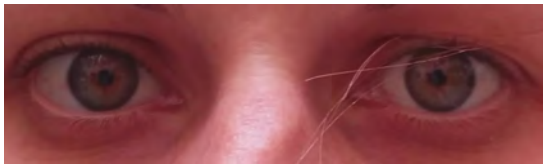

Figure 19. Extracted Frontal Face Image.

Keywords: H.264, image extraction, FFmpeg

VI. Discussion

Using the cUX pattern approach to create easy-to-use solutions allowed us to adjust and improve the overall study concept and setup in several ways. Apart from that, we also acquired a deeper insight into the pattern creation process overall. This gave us a chance to notice certain weak points in the creation process, which, when improved, would help to generate better patterns.

A. Pattern Generation Process

As mentioned at the end of Section III, each iteration and the following rework phase refines the pattern. The pattern is increasing in quality, with every feedback received during the iteration process. Bottom line, the more iterations processes a pattern runs through, the better it gets. In our case, we had a constant collaboration during the creation process of the patterns, which enabled us to get on demand feedback when necessary. Due to active collaboration, we had the possibility of continuous iterations, allowing us to interplay between problem statements and solutions. Usually, problem statements are defined in the beginning and changes can only be made during workshops. Solutions, however, are provided during the fist iteration, at the very earliest. Therefore, modifications can be made only after receiving feedback. Until then, the work on the pattern is on hold.

The interplay showed us a huge advantage, due to the possibility to refine the problem statement while simultaneously adjusting the solution. This induced the improvement of both the problem statement and the related solution leading to a higher quality pattern. The problem, however, was the recurring chance to rephrase the problem statement at any time. Thus, it was tempting to rephrase the problem statement to fit a certain solution, even when it was only covering a part of the statement. This behavior is not desired at all. Patterns are supposed to provide proven solutions. In the beginning, after describing the problem statements, we did not know if we could cover these criteria with our suggested solutions. However, we evaluated our patterns regarding that point through trial and error. Each and every solution we provide in our patterns was tested before it was adopted into the patterns. This was only possible due to the interplay and instant feedback and, therefore, can not be generally applied. However, we found that this way of verification improved the provided solutions to a high degree.

B. Pattern Sections

The next discussion point is the use of a *Topics* section proposed by the cUX pattern approach. Topics, in this case, are predefined keywords used to show the scope of the problem and, additionally, address one or more user experience factors.

We willingly omitted that section, as we saw no need for them in our created patterns. Topics may be beneficial to organize a collection of patterns, providing a variety of solutions for a large main field. Each pattern can be assigned to at least one of the topics. However, in our case, we only had a limited amount of problem statements that we wanted to address. Thus, creating a system in which we want to organize our patterns seemed unnecessary. Therefore, it was sufficient enough to provide keywords only at the end of the patterns. The keywords provide research topics and fields that may be related to the pattern and may be used to get more insight into certain areas covered or not sufficiently covered in the patterns.

C. More than one Solution

Patterns are by no means always the one and only possible solution for a certain problem. This is especially noticeable in problems concerning programming questions. The two main issues regarding programming questions in patterns are, for one, the programming language and, second, the implementation. Taking the pattern "Extraction of the Eye Area from Frontal Face Images" as an example, there were several ways to solve this problem. It is be possible to extract the eye area from the frontal face images by hand, image by image. This is very time consuming and inefficient, but it is a working solution. Thus, to optimize the procedure, the detection of the region of interest and the extraction needed to be automated, ideally by a program. One of the more versatile tools to accomplish that is the OpenCV library, which is supporting the most operating systems, but only a handful of programming language interfaces(i.e., C++, C, Python, and Java).

The problem is either to choose a certain language, preferably the most popular one, to provide a low-level solution or to provide a high-level description of the solution using pseudocode. Naturally, a low-level solution would be predestined to provide a copy and paste implementation that could be used right away. However, this would limit the usage of the pattern. To broaden the usage of the pattern, the decision was made to use a high-level description utilizing pseudocode without going far into the exact implementation. This is only one example from many that shows that a) there are many ways to solve a problem and b) the solution chosen heavily relies on the pattern creation team.

D. Aid & Explanation instead of direct Solutions

Apart from having difficulties to chose the best way to solve a problem, there are problems that can only be solved by providing a couple of possible ways to handle certain difficulties. "Comparison between Bitmap, Portable Network Graphic, and JPEG Images", for example, offers a solution in the way of providing the reader with vital information as a basis for deciding how to handle the problem of choosing a certain file format for image extraction. Each image format has its advantages and disadvantages. When compared to .png and .bmp files, .jpeg files are smaller, but offer the worst quality, as .jpeg files are automatically compressed. The extraction speed is decent, but by far not as fast as .bmp extraction; .png and .bmp files are lossless and offer the best image quality with the tradeoff of file size and, in case of .png files, extraction speed as well. Thus, there is no optimal solution without knowing the actual terms of use. The pattern can provide a solution in the form of aid and explanation to ease the decision of which file format to utilize for extraction; however, there is no one way solution for that kind of problems.

VII. Conclusion and Future Work

In this paper, we presented seven patterns to help design and refine a study setup for biometric image data acquisition analysis. By adapting an existing cUX design patterns approach, we were able to successfully document the study setup and its optimization in question and document these for future applications. The pattern structure and modular nature allows for further expansion and setup variations in the future. The presented patterns cover the most immediate problems for the specific setup but should not be considered a full pattern collection for biometric image data analysis. Nevertheless, the goal of creating additional solution patterns to improve the study setup, focusing on the image application domain, was fulfilled. We provided four additional patterns answering common problems in the image application domain.

Future work will have to focus on, not only reapplying these patterns and refine them further, but also expand towards related problems that could only be touched in the patterns above (e.g., further details on compression formats and artifacts, a wider range of file formats, more phone types or image acquisition devices in general, etc.). The existing patterns can already be used to inform future study setups with solutions regarding (a) choice of the right lighting conditions, (b) construction of a custom lens holder, (c) high quality video acquisition with a mobile phone, (d) the extraction of media information from video files, (e) extract still images from H.264 videos, and (f) the extraction of the eye area with the help of Haar-like features.

Further expansion will focus on providing more thorough solutions and suitability for more instances and broader contexts, towards a more profound knowledge base on biometric image analysis, suitable for an even wider range of users.

References

[1] A. Lupp, A. G. Mirnig, A. Uhl, and M. Tscheligi, "A Study Setup Optimization – Providing Solutions with Patterns," in PATTERNS 2017, The Ninth International Conferences on Pervasive Patterns and Applications, 2017, pp. 11-16.

[2] C. Alexander, "A Pattern Language: Towns, Buildings, Construction," Oxford University Press, New York, USA, 1997.

[3] C. Alexander, "The Timeless Way of Building," Oxford University Press, New York, USA, 1979.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software." Pearson, 1994.

[5] M. J. Mahemoff and L. J. Johnston, "Principles for a Usability-Oriented Pattern Language," In Proc. Australian Computer Human Interaction Conference OZCHI '98, IEEE Computer Society, 1998, pp. 132-139.

[6] A. Dix, G. Abowd, R. Beale, and J. Finlay, "Human-Computer Interaction," Prentice Hall, Europe, 1998.

[7] D. May and P. Taylor, "Knowledge management with patterns," Commun. ACM 46, 7, July 2003, pp. 94-99.

[8] J. Borchers, "A Pattern Approach to Interaction Design," AI & Society, 12, Springer, 2001, pp. 359-376.

[9] A. F. Blackwell and S. Fincher, "PUX: Patterns of User Experience," Interactions, vol. 17, no. 2., NY, USA: ACM, 2010, pp. 27-31.

[10] M. Obrist, D. Wurhofer, E. Beck, A. Karahasanovic, and M. Tscheligi, "User experience (ux) patterns for audio-visual networked applications: Inspirations for design," in Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries, ser. NordiCHI10. New York, NY, USA: ACM, 2010, pp. 343-352.

[11] A. G. Mirnig and M. Tscheligi, "Introducing a General Multi-Purpose Pattern Framework: Towards a Universal Pattern Approach," International Journal On Advances in Intelligent Systems, vol. 8, 2015, pp. 40-56.

[12] G. K. Wallace, "The JPEG still picture compression standard," IEEE transactions on consumer electronics, 1992, 38. Jg., Nr. 1, S. xviii-xxxiv.

[13] DRAFT, I. T. U. T. recommendation and final draft international standard of joint video specification (ITU-T Rec. H. 264— ISO/IEC 14496-10 AVC). Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050, 2003, 33. Jg.

[14] H264, I. ISO/IEC 14496-10 AVC. Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification.

[15] S. Rakshit, D.M. Monro, "An Evaluation of Image Sampling and Compression for Human Iris Recognition," IEEE Transactions on Information Forensics and Security 2(3), 2007, 605-612.

[16] M.A. Figueroa-Villanueva, N.K. Ratha, R.M. Bolle, "A comparative performance analysis of JPEG2000 vs. WSQ for fingerprint compression," In: Kittler, J., Nixon, M.S. (eds.) AVBPA 2003. LNCS, Springer, Heidelberg 2003, vol. 2688, pp. 385-392.

[17] R.C. Kidd, "Comparison of wavelet scalar quantization and JPEG for fingerprint image compression," Journal of Electronic Imaging 4(1), 1995, pp. 31-39.

[18] L. Granai, J.R. Tena, M. Hamouz, J. Kittler, "Influence of compression on 3D face recognition," Pattern Recognition Letters, 30(8), pp. 745-750.

[19] K. Delac, S. Grgic, M. Grgic, "Image compression in face recognition - a literature survey," In: Recent Advances in Face Recognition, I-Tech, 2008, pp. 236-250.

[20] P. Elmer, A. Lupp, S. Sprenger, R. Thaler, and A. Uhl, "Exploring compression impact on face detection using haar-like features," in Scandinavian Conference on Image Analysis. Springer, 2015, pp. 53-64.

[21] P. Viola, M.J. Jones, "Rapid object detection using a boosted cascade of simple features," In: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2001, vol. 1, pp. I-511-I-518.

[22] P. Viola, M.J. Jones, "Robust Real-Time Face Detection," International Journal of Computer Vision, 57(2), 2004, pp. 137-154.

[23] R. Lienhart, J. Maydt, "An extended set of Haar-like features for rapid object detection," In: Proceedings of the 2002 International Conference on Image Processing, vol. 1, 2002, pp. I-900-I-903.

[24] OpenCV. Available: http://opencv.org [Accessed: 20 - Aug - 2017]

[25] D. Wurhofer, M. Obrist, E. Beck, and M. Tscheligi, "A quality criteria framework for pattern validation," International Journal On Advances in Software, vol. 3, no. 1 and 2. IARIA, 2010, pp. 252-264.

[26] A. Mirnig, T. Kaiser, A. Lupp, N. Perterer, A. Meschtscherjakov, T. Grah, and M. Tscheligi, "Automotive User Experience Design Patterns: An Approach and Pattern Examples," International Journal On Advances in Intelligent Systems, vol. 9, 2016, pp. 275-286.

[27] A. G. Mirnig et al., "User Experience Patterns from Scientific and Industry Knowledge: An Inclusive Pattern Approach," in PATTERNS 2015, Seventh International Conference on Pervasive Patterns and Applications. IARIA, 2015, pp. 38-44.

[28] Philips MASTER TL5 HO 49W/865 UNP/40 - Product Page. Available: http://bit.ly/2kDgmOV [Accessed: 11 - Jan - 2017]

[29] YouTube Video - Nexus 5 Lens Holder Case. Available: https://youtu.be/J3dParRRQJg [Accessed: 11 - Jan - 2017]

[30] [App][2.3+]Snap Camera. Available: http://bit.ly/2iCujgO [Accessed: 31 - Jul - 2017]

[31] [App][2.3+]Snap Camera - 4k Video on Nexus 5 Feature. Available: http://bit.ly/2tQQW47 [Accessed: 31 - Jul - 2017]

[32] Download FFmpeg. Available: http://bit.ly/2v6qkNm [Accessed: 21 - Aug - 2017]

[33] VideoLAN. Available: http://www.videolan.org [Accessed: 20 - Aug - 2017]http://www.videolan.org

[34] VLC HowTo/Take a snapshot. Available: http://bit.ly/2wJ6AU5 [Accessed: 20 - Aug - 2017]

[35] OpenCV Installation - Windows. Available: http://bit.ly/2wmqx0f [Accessed: 20 - Aug - 2017]

[36] OpenCV Installation - macOS. Available: http://bit.ly/2ukmjoE [Accessed: 20 - Aug - 2017]

[37] OpenCV Installation - Linux. Available: http://bit.ly/2xvIzNE [Accessed: 20 - Aug - 2017]

[38] OpenCV Haarcascade Files. Available: https://github.com/opencv/opencv/tree/master/data/haarcascades [Accessed: 20 - Aug - 2017]

[39] DETECTING EYES WITH PYTHON & OPENCV. Available: http://bit.ly/2vHNjhc [Accessed: 20 - Aug - 2017]