Yousuf Mohamed-Ahmed

Non-contact heart rate estimation from video

Computer Science Tripos - Part II

Gonville & Caius College

February 21, 2020

Proforma

Name:	**Yousuf Mohamed-Ahmed**
College:	**Gonville & Caius College**
Project Title:	**Non-contact heart rate estimation from video**
Examination:	**Computer Science Tripos - Part II, July 2020**
Word Count:	**0**[1]
Project Originator:	Dr R. Harle
Supervisor:	Dr R. Harle

Original Aims of the Project

Work Completed

Special Difficulties

None.

[1]This word count was computed by the **TeXcount** script

Declaration

I, Yousuf Mohamed-Ahmed of Gonville & Caius College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed Yousuf Mohamed-Ahmed

Date February 21, 2020

Contents

Chapter 1

Introduction

1.1 Motivation

Optical heart rate monitors, the kind of which are standard in modern wearables, work by measuring the amount of light emitted by the surface of the skin. Given this signal, a heart rate can be extracted by analysing the predominant frequencies, for example, through the use of a Fourier transform.

This notion, can be extended to traditional cameras such as those present in smartphones.

1.2 Related work

Chapter 2

Preparation

2.1 Relevant computer vision techniques

2.1.1 Face detection

2.1.2 Optical flow

2.1.3 Harris corner detection

2.2 Approaches to signal processing

2.2.1 Independent Component Analysis

2.2.2 Fourier transforms

2.3 Sensing and photoplethysmography (PPG)

2.4 Languages and tooling

2.4.1 OpenCV

2.4.2 Mobile Vision API

2.4.3 Python

2.4.4 Kotlin

2.4.5 Android

2.5 Starting Point

2.6 Requirements analysis

Chapter 3

Implementation

3.1 Overview

Abstractly, the program consists of three distinct tasks, each of which rely on the result from the previous. Together, forming a kind of pipeline:

- **Face detection**: identify a face in each supplied camera frame
- **Region selection**: given a bounding box around a face, select some set of pixels to consider which are amenable to heart rate detection
- **Heart rate isolation**: given a time series of the mean colour of some region of the face, infer the heart rate

Face detection is largely a solved problem and thereby the project mostly concentrates on the latter two, for which, there is very much ongoing research.

3.2 System design

Each of these tasks occur at different rates and, thereby, have different performance constraints which must be upheld. Face detection and region selection operate on every frame received from the camera and so must run in real-time, or risk dropping streamed frames.

Heart rate isolation, however, is only executed after some adequate number of data points is received and is recomputed after a fixed time. Since none of the prior stages rely on the estimated heart rate, its execution time requirements are less stringent. This is exploited to perform relatively expensive analyses without slowing earlier stages. Crucially, this relies on the assumption that it

can be run concurrently from the earlier stages. Separating these tasks allows for this concurrency to be implemented safely.

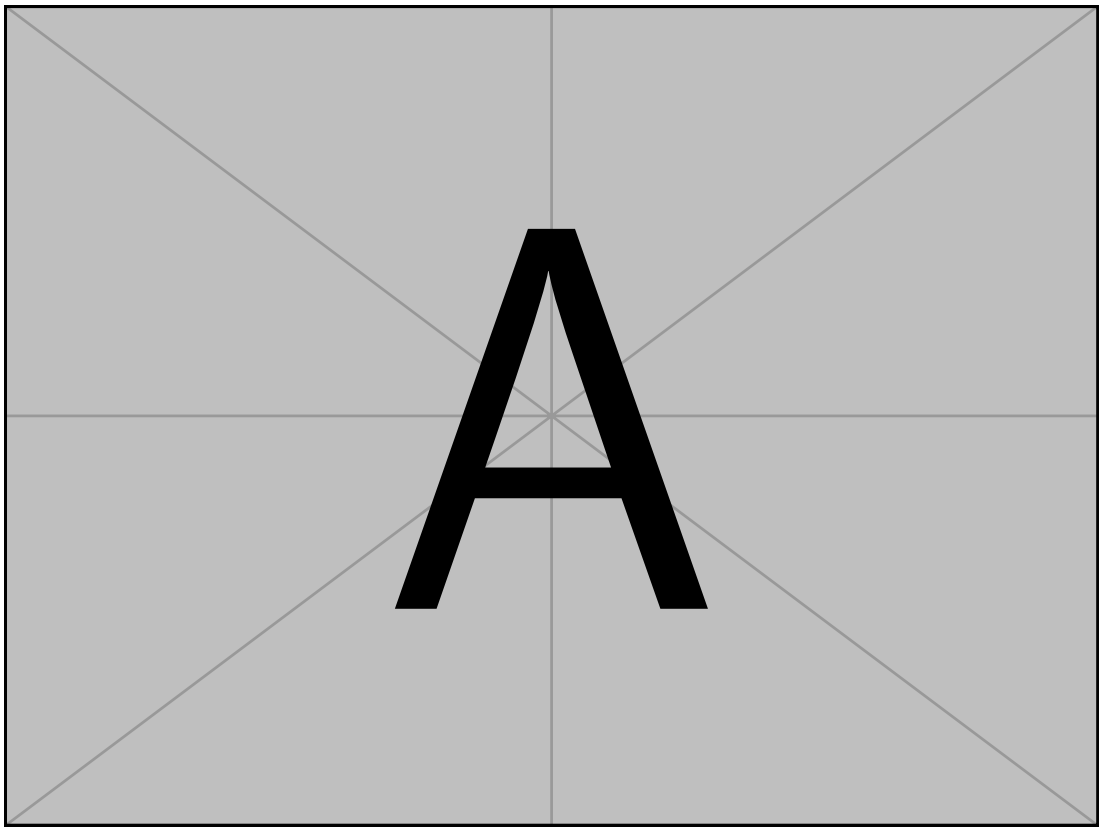

Figure 3.1: A diagram representing the design of the system

The camera streams frames to the **FaceDetector**. The **RegionSelector** then takes the mean pixel colour of the region considered and adds this value to the dataset. Once the appropriate number of values is reached, the **HRIsolator** spawns a new thread, the **IsolationTask**, which attempts to infer the heart rate from the values collected.

Although the region selection could also be executed in a separate thread, the associated setup costs are likely to have an adverse impact on real-time performance. However, one might wish to use more expensive region selection algorithms which cannot run in real-time. In these scenarios, the program could copy the frame and execute the selection in a separate thread from the main face detection loop. This is fundamentally a tradeoff between memory usage and execution time. Furthermore, since there are only a finite number of threads which can be spawned, quickly the limit might be reached without providing

much additional computation time to the **RegionSelector**. As a result, the **FaceDetector** and **RegionSelector** operate sequentially in the same thread.

3.3 Face detection

A face detector is expected to take a single frame and return a bounding rectangle within which a face is present. This could be extended to work on a stream of frames, naïvely, by simply repeated applying this face detector to each frame independently.

3.3.1 Face tracking

Smartphone cameras can readily stream at high framerates, so it is unlikely that a face moves very much between a pair of consecutive frames. At sixty frames per second, frames are recorded only 0.017s apart. The position of the face in the previous frame, gives a strong indication of its subsequent position. Thus, there is opportunity for optimisation beyond simply applying a face detector to each frame individually. This is important since a speedup in this part of the pipeline provides opportunity for more expensive computation in subsequent stages which might improve accuracy. However, it is critical that any optimisations are resistant to motion. Using information from previous frames forms the distinction between face ***detection*** and ***tracking***.

Point tracking

The key principle behind face tracking is to use information from previous frames to reduce the cost of subsequent face detections. To that end, I implemented an optical-flow based algorithm that tracks points on the face rather than repeatedly calling the face detector. The Lucas-Kanade algorithm, a particular variant of optical-flow, tracks a set of points between consecutive frames. Naturally, over time, these points will diverge from their true positions. This is because images contain large numbers of pixels with similar illumination, over time, these cannot be distinguished perfectly. When this occurs, the position of the face should be redetected.

Knowing when the points have diverged is non-trivial, since the true location of the face is unknown. It is crucial for any implementation to act safely enough that the face is not lost track of, whilst simultaneously minimising the

number of redetections. There are two obvious approaches to combat this. The algorithm could redetect the face:

- periodically
- when the tracked face changes in size significantly.

The former wastes computation time for stationary videos where the redetections might be unnecessary. Simultaneously, the time between redetections must be short enough to deal with videos with significant movement. This static approach, therefore, was not considered. For this reason, the latter was implemented. We track the face and if the size of the box surrounding it changes significantly, indicating that our confidence in the tracked points has reduced, then we recompute the true position of the face.

```
points = []
last_detection = None
def face_tracker(frame):
     if redetect or points is empty:
         face = face_detector(frame)
         last_detection = face
         points = select_new_points(face)
     else:
         points = track_points(points)
         face = bounding_box(points)
     redetect = change_in_size(last_detection, face) > threshold
     previous_face = face
     return face
```

Figure 3.2: A simplified pseudocode representation of an optical-flow based face tracker

Impact of the rate of redetections It is important to reason about precisely when face tracking provides a performance boost. To understand this let us consider a sequence of frames which the face tracker has been applied to. The cost of tracking is compared with that of repeatedly detecting the face in each frame independently. Under the following notation:

- W: number of consecutive frames considered
- R: the total number of redetections by the face tracker

- n: size of each frame in pixels
- p: number of points tracked
- $f(n)$: cost of a face detection on a single frame of size n
- $s(p, f)$: cost of selecting p points to track in a face of size f
- $g(p, n)$: cost of tracking p points in a frame of size n

The cost of the repeated face detector is $Wf(n)$. Since, for each frame in the sequence, it detects the face independently and incurs a cost of $f(n)$.

The point tracking approach instead has a cost of $Wg(p, n) + R(f(n) + s(p, f))$. For each redetection it calls the face detector and selects a new set of points. It also tracks the set of points for every frame, hence the term $Wg(p, n)$. In the worst case, $R = W$, so there is no saving in terms of computational complexity. Instead let us investigate for what values of R there is a cost saving.

$$\begin{aligned} Wf(n) &> Wg(p, n) + R[f(n) + s(p, f)] \\ R &< \frac{W[f(n) - g(p, n)]}{f(n) + s(p, f)} \\ \frac{R}{W} &< \frac{f(n) - g(p, n)}{f(n) + s(p, f)} \end{aligned}$$

Notice that R/W represents the percentage of frames for which a redetection occurs. There is only a performance saving when this percentage falls below the value on the right hand side. Furthermore, this value itself is based on the relative costs of face detection and of selecting and tracking points. Implicitly, the algorithm assumes that the cost of tracking and selecting points is less than that of face detection, otherwise, this endeavour would be useless. This assumption is validated and the limit is evaluated experimentally in section 4.1. From this it is shown that even for videos with lots of movement, the value R/W falls below this limit and the inequality is satisfied.

Details To maximise the performance of the optical flow algorithm, selecting points randomly will not suffice. That is because, points which are particularly different to their neighbours, for example, are much easier to track. To achieve this, the Shi-Tomasi corner detection algorithm is used which returns some number of points satisfying this condition. Furthermore, for face tracking to work properly, the points selected must span the entire face of the user. This is

relatively easily enforced by mandating a minimum distance between each of the points selected and by selecting enough points.

It is conservative enough that the true position of the face is not lost, whilst still providing a 3x performance boost on the naive face detection approach. This is fully evaluated in section 4.1.

Camshift

3.3.2 Face alignment

3.4 Region selection

Face detection systems, typically, return a bounding box, within which it is believed a face is present. However, naturally, the box will also contain pixels from the background of the image since faces are not, in general, perfectly rectangular.

These background pixels will not contain any information as to the underlying heart rate of the user. As a result, considering the entire bounding box will add unecessary noise to the resulting signal which consists of the mean colour from each region considered in the sequence of frames. One approach might be to only consider skin pixels, however, robust, pose-invariant skin detection is a somewhat unsolved problem. Furthermore, it is unclear whether all skin pixels are equally useful. For example, it might be that cheeks contain greater predictive power than the nose. Several approaches are presented regarding this problem with associated algorithms that are evaluated in section 4.2.

3.4.1 Skin detection

Suppose that we wish only to consider skin pixels. An obvious approach might be to apply an edge detection algorithm to isolate the boundary between the face and the background. However, edge detection algorithms, like the Canny edge detector, tend to produce large numbers of irrelevant edges and so were not implemented.

Colour-based filtering

Considering that skin tones tend to fall within a certain range of colours, one might encode this information in a primitive skin detector. For example, it is

known that green is not a valid skin tone but brown might be. If a large enough number of skin tones are investigated then a range within which a skin pixel might lie could be defined.

A dataset of ∼250000 skin and non-skin pixels was collected by Bhatt et al. [1] and was sampled across a variety of skin tones, genders and ages. One could define the range of skin tones present in this dataset as a rudimentary skin detector. Clearly this will fail in many scenarios but serves as a useful baseline for comparison with more advanced techniques.

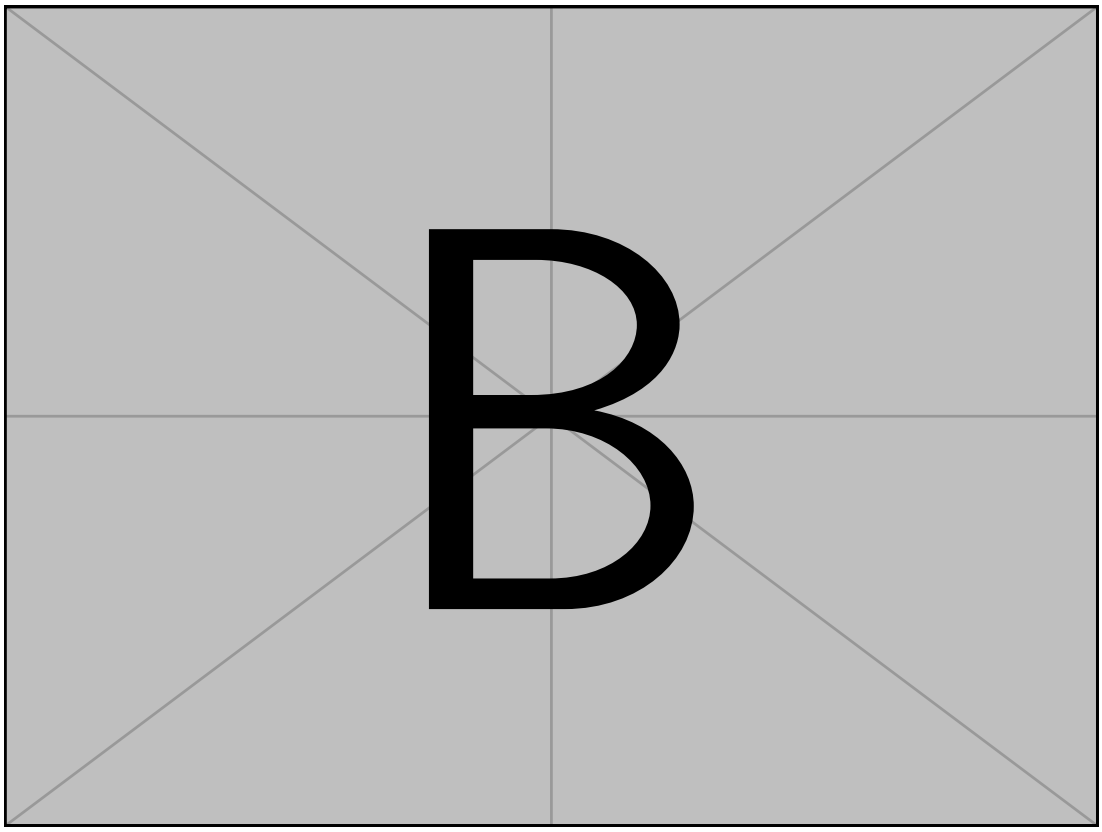

Figure 3.3: Skin dataset represented in YCbCr space

Issues This approach takes no consideration of the particular face that it is considering. Since it is not contexualised, it can fail in several circumstances. For example, there are some hair colours which could be a skin tone on another individual but are clearly not on the particular example in question. Instead, an

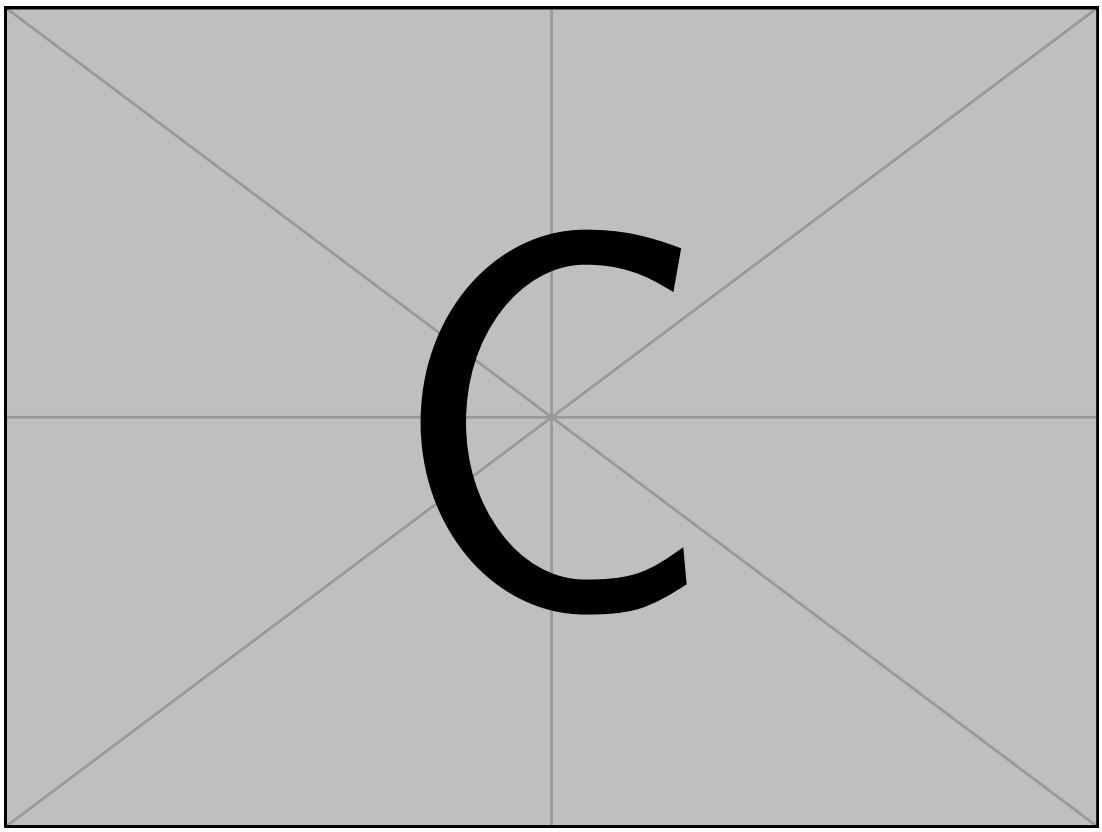

Figure 3.4: An example of the hair failure case

algorithm should attempt to identify skin on the particular face in question.

K-Means

If we consider how a human might identify skin, it could involve initially identifying the skin tone of the person and then assuming all parts of the face of a similar colour are in fact skin. This reduces the problem to identifying the skin tone in a face and then measuring the colour difference between each pixel and the skin tone. If the colours are within some specified threshold, then we can consider them to be skin pixels.

We might suppose that the image consists of clusters of pixels, some of which belong to the skin and others which don't. The center of the cluster of skin pixels represents the skin tone of the individual. Implicitly, this approach assumes that the Euclidean distance between points in our colour space is representative of the perceived colour difference. This property is known as perceptual uniformity and is not a property of all colour spaces.

The colour space YCbCr is an approximation of perceptual uniformity and hence the image is converted from RGB before the application of clustering. Under the assumption that our image consists of two clusters of pixels, skin and non-skin, we could simply apply the k-means algorithm to identify these clusters of pixels. Under the further assumption that the majority of pixels are skin pixels, we return the largest cluster as our set of skin pixels.

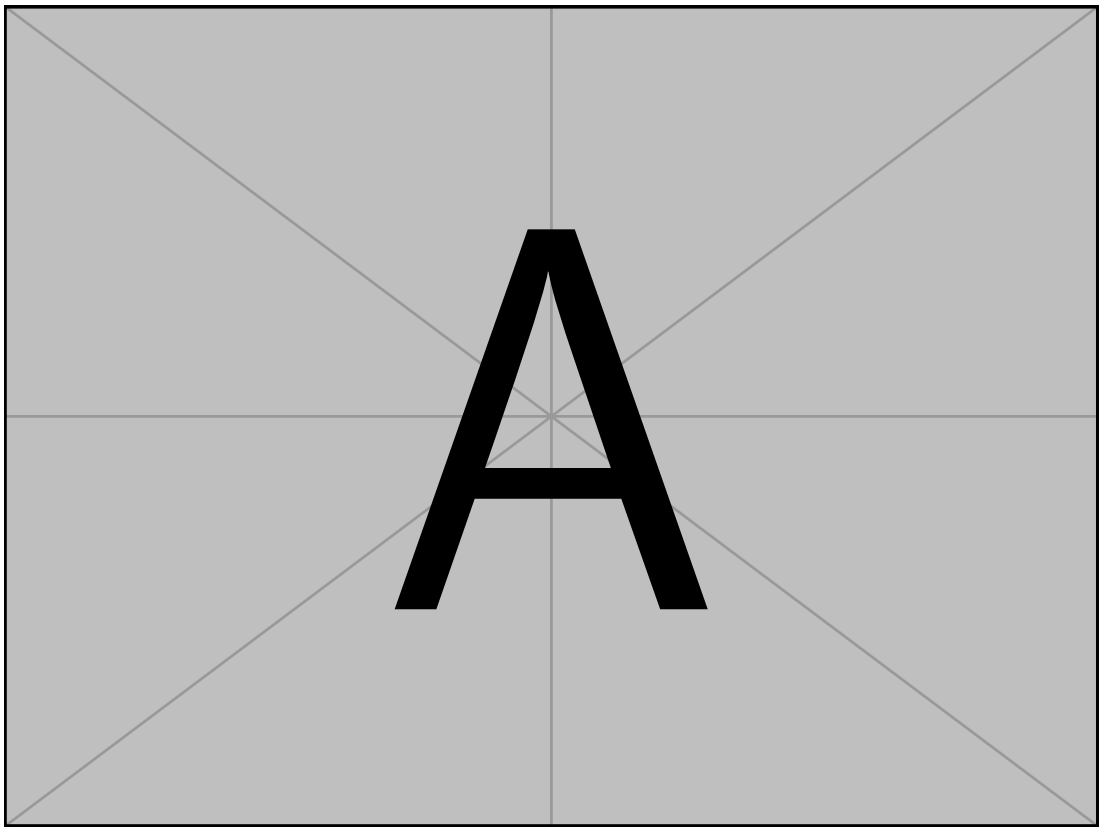

Figure 3.5: An example of the application of the k-means algorithm to skin detection

Issues

Complexity

3.4.2 A hybrid solution

The rudimentary colour-filtering approach provides a good baseline as to the

3.5 Heart rate isolation

Given a time series of mean colours in the region of the frame considered, inferring the heart rate might, naively, be taken as the prevalent frequency. That is, the largest peak in the Fourier transform. However, it is important to consider that the colours observed are not only the result of the underlying biological phenomenon of interest. Thereby this naive assumption is prone to returning, instead, the frequency of some other factor that impacts colour of the face. For example, respiration or movement of the face will have an impact, as well as any repetitive changes in lighting such as flickering. Isolating, the heart rate signal from the observed colour of the face, is the crux of this project.

3.5.1 Blind-source separation

Suppose that our observed colour signal, $\mathbf{I}(t)$, a vector-valued function, consists of a mixture of several underlying signals, $x_i(t)$ one of which is the pulse, $p(t)$. If we consider $\mathbf{I}(t)$ as a matrix of dimension $n \times T$ where n is the number of colour dimensions and T the number of timesteps, then we might suppose that $(x_1(t), x_2(t), ..., p(t), ..., x_{n-1}(t))^T = AI$. That is That is: $\mathbf{I}(t) = (x_1(t), x_2(t), x_3(t))$, for constants λ_i. Alternatively we can consider this as finding the set of basis vectors v_1, v_2, v_3 such that $I(t) = (I(t) \cdot v_1)v_1 + (I(t) \cdot v_2)v_2 + (I(t) \cdot v_3)v_3$. The key problem is finding the basis v_1, v_2, v_3 such that $p(t) = (I(t) \cdot v_i))v_i$ for some i and further identifying the value of i.

The nature of this task is to identify and return $\mathbf{p}(t)$ from only $\mathbf{I}(t)$. This is known as the blind-source separation problem.

The above formulation is not enough to isolate the signal $\mathbf{p}(t)$. Thus assumptions must be placed on the nature of each $\mathbf{x}_i(t)$ and $\mathbf{p}(t)$ in order to turn this into a tractable problem.

We can reframe this as finding some basis for our vector space $I(t) = \sum_i^n I(t) \cdot (v_i)v_i$

Independent components analysis

A possible assumption might be that each of the $\mathbf{x}_i(t)$ and $\mathbf{p}(t)$ are statistically independent. That is, informally, one cannot gain any information about one of these signals given another.

Principal components analysis

3.5.2 The split peak issue

3.6 Repository overview

Chapter 4

Evaluation

4.1 Face tracking

Performance cost The performance of the above algorithm, clearly depends on the proportion that each of the two branches are executed.

Motion resistance

Correctness

4.2 Region selection

4.3 Blind source separation

4.4 Evaluation method

4.4.1 Experimental setup

4.4.2 Ethics

4.5 Sensing power

4.6 Performance

4.6.1 Energy consumption

4.6.2 Time cost

4.7 Summary

Chapter 5

Conclusion

Bibliography

[1] Abhinav Dhall Rajen Bhatt. Skin segmentation dataset. *UCI Machine Learning Repository*.

Appendix A

Project Proposal