Yousuf Mohamed-Ahmed

Non-contact heart rate estimation from video

Computer Science Tripos - Part II

Gonville & Caius College

February 18, 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 18, 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
- **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.

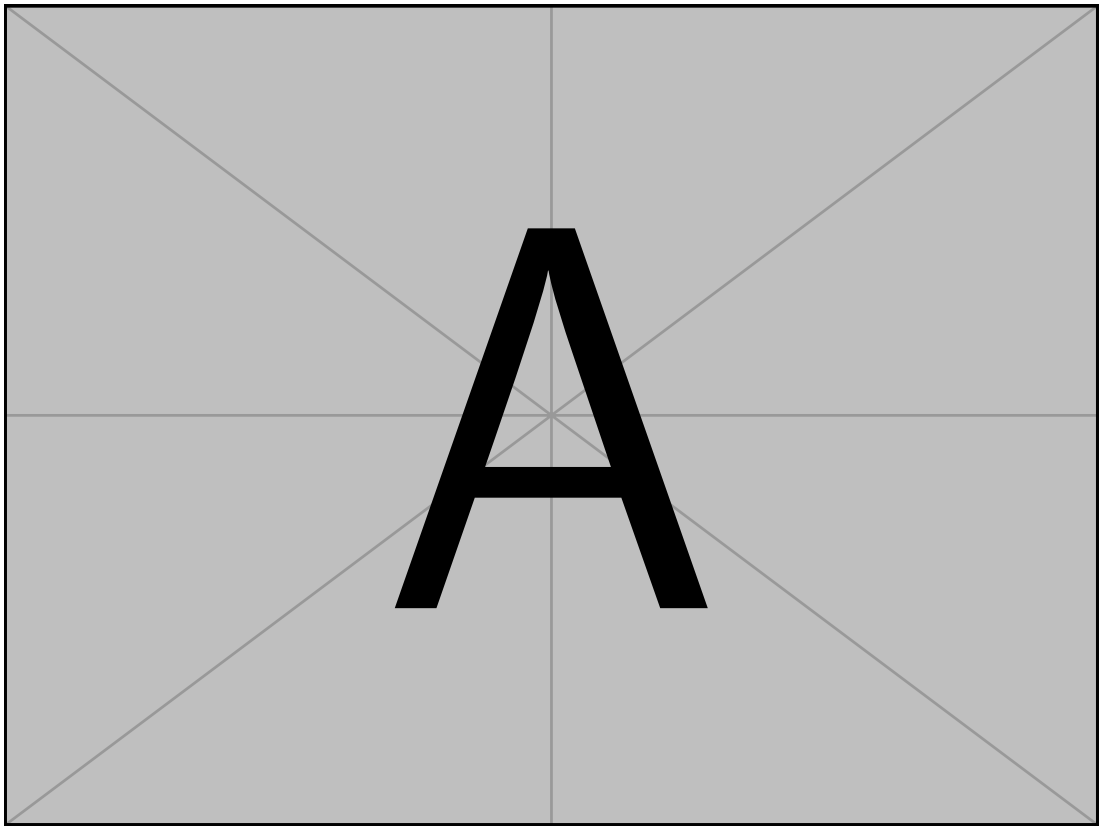

Figure 3.1: A diagram representing the design of the system

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.

```
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
- $s(p, f)$: 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 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 would be useless. This assumption is validated and the limit is evaluated experimentally in section 4.2. From this it is shown that even for videos with lots of movement, the value R/W falls below this limit.

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. This approach is fully evaluated in section 4.2.

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. As well as any imperfections in the face detector.

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. The ideal algorithm would only consider skin pixels, however, robust, pose-invariant skin detection is a somewhat unsolved problem. However, not all skin pixels are equally useful

3.4.1 Facial landmarks

3.4.2 Skin detection

K-Means

Colour Space Filtering

3.5 Heart rate isolation

3.5.1 Blind-source separation

3.5.2 Respiration rejection

3.5.3 Optimisation

3.6 Repository overview

Chapter 4

Evaluation

4.1 Evaluation method

4.2 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

4.3 Sensing power

4.4 Performance

4.5 Summary

Chapter 5

Conclusion

Appendix A

Project Proposal

Computer Science Tripos - Part II - Project Proposal
Non-contact heart rate estimation from video

Yousuf Mohamed-Ahmed

Project Originator: Robert Harle
Project Supervisor: Robert Harle
Director of Studies: Timothy Jones and Graham Titmus
Overseers: Rafal Mantiuk and Andrew Pitts

Introduction and Description of the Work

Heart rate measurements from smartwatches are notoriously spurious and can, especially during exercise, provide inaccurate measurements. Optical heart rate monitors work by measuring the amount of light emitted by the monitor that is reflected by the surface of the skin. From this, the volume of blood flowing beneath the surface can be inferred, known as a plethysmogram. Tracking this value through time allows the heart rate of the user to be estimated. However, large amounts of movement, as typical during exercise, generate motion artifacts which degrade the accuracy of the measurements from smartwatches by decreasing the signal-to-noise ratio of the plethysmogram.

Papers have shown that an accurate heart rate value can be extracted from a video recorded by a camera [1]. This is because slight changes in the colour of the face, as well as slight movements of the face, allow a plethysmogram to be inferred. Any technique which can measure a plethysmogram optically via a non-contact method is known as remote photoplethysmography (rPPG) and is an active area of research. This project is concerned with the implementation of such a system which should be accessible through an Android application using a phone camera.

Since, as some experiments have shown, the face contains a greater PPG signal than the wrist [2], I would like to investigate the extent to which rPPG could be used as a replacement for smartwatches during exercise. This will involve investigating the effects of distance on accuracy as well as comparisons of different algorithms for computing the heart rate.

It is expected that some may perform better than others in certain scenarios such as amount of movement or lighting conditions and, as a result, there is scope for combining algorithms or using heuristics for selecting appropriate implementations at runtime. In order to develop these heuristics, the effects on accuracy of numerous conditions should be examined such as, potentially, the effects of the resolution and framerate of the camera.

This could be critical since many current rPPG systems do not run in realtime and thus, any reductions in the amount of processing required, for example, by downsampling images or not considering every frame, could help to maintain accuracy whilst improving performance.

Furthermore, recent reductions in the price of smartphones means that high-quality cameras are much more ubiquitous than heart-rate measuring equipment. An abnormal heart rate can be indicative of wider medical problems and, as a result, the ability to measure heart rates easily could be helpful for communities without regular access to healthcare. To that end, this project may be suitable for medical uses in remote areas.

Starting Point

I have no previous experience developing Computer Vision applications and nor in Digital Signal Processing. However, I believe, Part 1A Introduction to Graphics and Part 1B Further Graphics will prove useful precursors to understanding the Computer Vision aspects of the project. Likewise, I am currently studying Part II Information Theory which I hope will help provide a grounding in some of the mathematics behind Digital Signal Processing, which in turn will help with understanding the fundamental algorithms in the field.

Work to be Done

Face detection

The face is typically the most exposed region of the body during exercise and is believed to be the easiest region to extract a reliable PPG signal from [2]. Given this idea, the system, from videos, will have to be able to identify faces within the videos. At present, this is likely to involve a sweep of the frame using the Viola-Jones algorithm to box any faces in the frame. This initial stage may then be followed by picking out the exact area of the face from within the box.

Region of interest selection

Within the region of the face itself different areas contain differing amounts of information regarding the pulse. For example, eyes give no information about the pulse and it is speculated that some regions of the face such as the forehead and cheeks give more information than other regions like the nose. Several algorithms will be developed to locate these regions and their performance will be compared.

Region tracking

Once a region is selected, it must be tracked between frames. This is because the colour of the face itself is of no real concern but the frequency at which the colour changes is the means by which we can infer the heart rate. Thus if different regions in each frame are tracked then this frequency will begin to diverge from the true value. This is likely to use some kind of Optical Flow algorithm, the exact nature of which will be investigated.

Signal processing of RGB signals

The resulting signals extracted from the region of interest (ROI), will be very noisy and will require the use of signal processing techniques before applying a Fourier transform to extract the prominent frequency. It is expected that this will require the use of a Blind Source Separation technique such as Independent Component Analysis to separate the independent sources contributing to the signal.

Android application

An Android application will be developed for easy testing of the system by allowing users to record videos of themselves and estimate their own heart rate.

Interaction between Android application and video analysis

The provision of signal processing and computer vision libraries isn't particularly strong in the JVM languages which are how Android applications are typically programmed. As a result, it is likely that the the video analysis software will be written as a separate program, most likely in Python or Julia, which will then interact with the Android application via pipes.

Evaluation

I hope to conduct my own experiments on the performance of the system relative to a smartwatch when compared to a known ground truth. As well as experiments across a variety of skin tones and lighting conditions.

Test bench application

In order to test the developed system, in comparison with a traditional smartwatch, an application for extracting the heart rates measured by the smartwatch will have to be developed. This will take the form of a logging application which will be able to run in the background on the watch.

Possible Extensions

- Tracking multiple faces in a single video
- Measuring heart rate whilst exercising
  - **Increased tracking**: the core program is only expected to deal with minor movements such as limited head movement, an increase in the stability of the tracking algorithms will almost certainly be required to deal with exercising users.
  - **Dealing with increased distance from camera**: an upscaling algorithm might be required to be able to extract heart rate from a distance, since the face region of the user might be small which would lead to a decrease signal-to-noise ratio, upscaling may help to increase this.
  - **Increasingly noisy signals**: more rigorous signal processing will be required to remove additional noise caused by movement from the signals. This is because the frequency of movement may fall within the same range as the expected frequency of the heart [3], hence simple band-pass filters will no longer work.
- End to End Deep Learning Approach: Recent papers [4] have shown that we can use models based on Convolutional Neural Networks taking a pair of frames to estimate the change in pulse. An implementation of this system could then be compared to the core program.
- Most systems outlined in the literature carry out off-line processing of the video frames, however, many provisions exist on Android for parallelized computation on images [5]. These could be utilised to develop a real-time application.

Success Criteria

- Develop an Android application that allows users to estimate their own heart rates
- Stationary users in appropriate lighting conditions should be able to measure their heart rate with reasonable accuracy

Timetable

I have created a timetable consisting of 12 2-week periods starting on 26/10/2019 and finishing on 25/4/2020.

1. **26/10/2019 - 09/11/2019**
   I will begin by conducting research into the OpenCV library for Computer Vision and assessing the provisions already present in the library. I should also be prototyping, most probably in Python, the face detection.
2. **09/11/2019 - 23/11/2019**
   The process of researching and prototyping should continue, with a rough structure of the analysis software in place, meaning that the program should be able to receive streams of frames and detect faces in each frame.
3. **23/11/2019 - 07/12/2019**
   Various Region of Interest selection algorithms should be implemented and tested, although their effect on accuracy cannot be measured yet, they should be tested for correctness.
4. **21/12/2019 - 04/01/2020**
   Several different point tracking algorithms should be selected and included in the program. At present, this is likely to include Lucas-Kanade optical flow (a sparse technique) and dense optical flow.
5. **04/01/2020 - 18/01/2020**
   Implement signal processing pipeline to allow for cleaning of RGB signal and extracting heart rate. This will complete the analysis software and I expect tweaking of the pipeline to occur at this stage based on any accuracy issues.
6. **18/01/2020 - 01/02/2020**
   Write the progress report and begin writing an Android application to allow for users to measure their own heart rate, should also allow for overlaying heart rate on the image.
7. **01/02/2020 - 15/02/2020**
   Write serialisation code to allow for communication between Android application and the analysis program. With the completion of this, the core project should be finished.
8. **15/02/2020 - 29/02/2020**
   Begin drafting the introduction chapter of the dissertation as well as beginning work on the code required to evaluate the project. This will include code for running experiments and measuring accuracy.
9. **29/02/2020 - 14/03/2020**
   Continue writing the introduction chapter and begin work on the preparation chapter, as well as beginning work on the extensions.

10. **14/03/2020 - 28/03/2020**
    Work on the extensions should be concluded and the implementation chapter finished. Feedback on the previous chapters should be sought out and appropriate modifications made.

11. **28/03/2020 - 11/04/2020**
    Finish writing the dissertation and seek final supervisor comments.

12. **11/04/2020 - 25/04/2020**
    Incorporate suggestions and hand in the final version.

Resource Declaration

I will use my own laptop, a Lenovo 510s with a 2.5 GHz Intel Core i5 CPU and 8GB of RAM. I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure. All my code for the project and the dissertation itself will be pushed to a Git repository that will be hosted on GitHub. I will push to this regularly so that, in the case of an issue with my laptop, I will be able to resume work promptly. I own an Android phone (Google Pixel 3A XL) which will be used as part of the development process. I have also been provided on loan with a smartwatch by my supervisor, the Fossil Q smartwatch which has Wear OS installed, will be used for any evaluation requiring a smartwatch comparison.

References

[1] N. J. Verkruysse W, Svaasand LO, "Remote plethysmographic imaging using ambient light," Jul 2009.

[2] K. M. van der Kooij and M. Naber, "An open-source remote heart rate imaging method with practical apparatus and algorithms," *Behavior Research Methods*, May 2019.

[3] F. Peng, Z. Zhang, X. Gou, H. Liu, and W. Wang, "Motion artifact removal from photoplethysmographic signals by combining temporally constrained independent component analysis and adaptive filter," *BioMedical Engineering OnLine*, vol. 13, no. 1, p. 50, 2014.

[4] W. Chen and D. McDuff, "Deepphys: Video-based physiological measurement using convolutional attention networks," in *Computer Vision – ECCV 2018* (V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, eds.), (Cham), pp. 356–373, Springer International Publishing, 2018.

[5] T.-M. Li, M. Gharbi, A. Adams, F. Durand, and J. Ragan-Kelley, "Differentiable programming for image processing and deep learning in halide," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, p. 139, 2018.