# Computational Photography and Capture
# Project 2: Timelapse Photography

Jaspreet Singh Dhanjan
University College London

Wednesday 22nd April 2020

## Contents

# 1 Introduction

This is the report for the final submission in Computational Photography and Capture. This project will look into timelapse technologies and developing a user-friendly implementation. This project will follow some of the ideas described in **Computational Timelapse Video** by Eric P. Bennett and Leonard McMillan.

The remainder of this section will outline the goals for this particular project. The following sections will discuss the implementation and give a final overall conclusion to this project.

For any referenced demo videos that are highlighted in bold, please look in the **examples** folder in the submission.

## 1.1 Goals

**Priorities:**

- **Use a web interface** - this project should be deployed as a web application. The front-facing HTML should provide an easy user-interface to generate a timelapse. The backend system should receive the files, compute the timelapse and serve a .mp4 of the timelapse to the client.
- **Sampling** - this project should allow for uniform sampling and non-uniform sampling.
- **Motion tails** - each sampling configuration should have the option for motion tails, as described in the paper as the virtual shutter.
- **Hyperlapse mode** - have a 'hyperlapse' mode where the user can select a relevant search term from source of information (such as Instagram, Google Images, etc). Use a REST API to download the frames and generate this hyperlapse.

**Nice to have:**

- **Stabilisation** - as images are collected for a timelapse, the camera may slightly move. To diminish the effects of this, a SIFT feature detector should detect the stationary objects in the scene and warp the frame accordingly if it has moved. This is not mentioned in the paper explicitly, but it would be a convenience for the user.
- **Should be deployed on the web** - once complete, the project should be deployed on Amazon Web Services and be available to use for everyone.
- **Helpful UI** - if files are uploaded in the incorrect format or other errors occur, then the user interface should explain this to the client. If there is enough time, a UI framework such as React could be used.

## 1.2 Requirements

Executing such a web application would not be possible in MATLAB. Therefore, this application will use Python. The following libraries will be used:

- Flask for the web server
- Numpy for mathematical representations
- OpenCV for image manipulations

## 1.3 Data Sets

Two demo image sequences were downloaded under a creative commons license from openfootage.net. The last footage was a timelapse sequence downloaded from YouTube without the authors permission and spliced into frames.

- Footage A: `https://www.openfootage.net/paris-timelapse-streets-at-night/` [Accessed: 11/04/2020]
- Footage B: `https://www.openfootage.net/timelapse-sunset-salzburg/` [Accessed: 11/04/2020]
- Footage C: `https://www.youtube.com/watch?v=dpSlyNBYdnQ` [Accessed: 16/04/2020]

Please remember all three videos as they will be used as examples in this report and in the submission.

For data sets that have been acquired using Bing's search API for hyperlapses, please refer to the standard out console when the server is running to find an individual link to each image.

# 2 Implementation

## 2.1 Web Server

Creating the web server and frontend was the first task to complete. It would establish an easy framework of knowing what to do next. The front-facing interface included a form, where the client would specify their timelapse/hyperlapse files and configurations. These files were uploaded to the server and processed on locally. The user would then be redirected to watch results of the processing.

## 2.2 Timelapse Processing

The basic idea of timelapse video is very simple. If the playback rate of the frames his higher than the capture rate, then this is a timelapse video.

### 2.2.1 Uniform Sampling

Thus, uniform sampling was simple to implement. Each frame was outputted to disk at a standard rate.

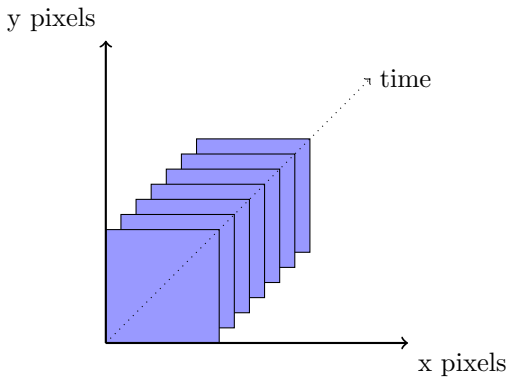


Figure 1: Visualisation of uniform sampling of frames

### 2.2.2 Non-uniform sampling

Non-uniform sampling extracts the key differences from the frames to produce a less choppy timelapse sequence. This is achieved (partially) by looking at the cost between frames, also referred to as the min-error metric:

$$\Delta(i,j) = \sum_{x}\sum_{y}\left[\sum_{k=i}^{j}((a_{ij}^{xy} + b_{ij}^{xy}k) - s_k^{xy})^2\right] \tag{1}$$

Furthermore, the paper described a simple method of finding the similarity (or dissimilarity) between frames, this is referred to as the min-change error metric:

$$\delta(i,j) = \sum_x \sum_y \left[ (s_j^{xy} - s_i^{xy})^2 \right] \tag{2}$$

Essentially, this is the sum-of-squared differences between intensities of each frame $i$ and $j$. If the minimum change error metric between frames is small, then it should use less samples than a frame that is significantly different from its prior. The following visualisation describes this:

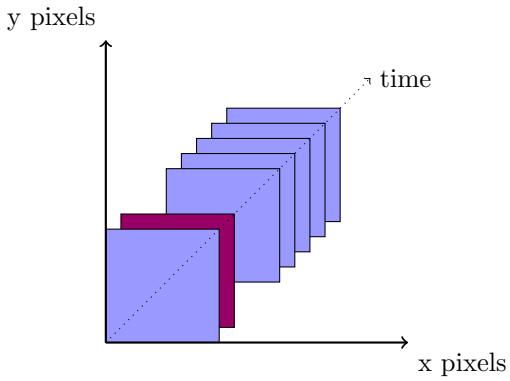


Figure 2: Visualisation of non-uniform sampling of frames. The maroon frame ($j$) has a significantly higher min-change error metric than the first frame ($i$), which means a longer amount of sampling for that frame sequence

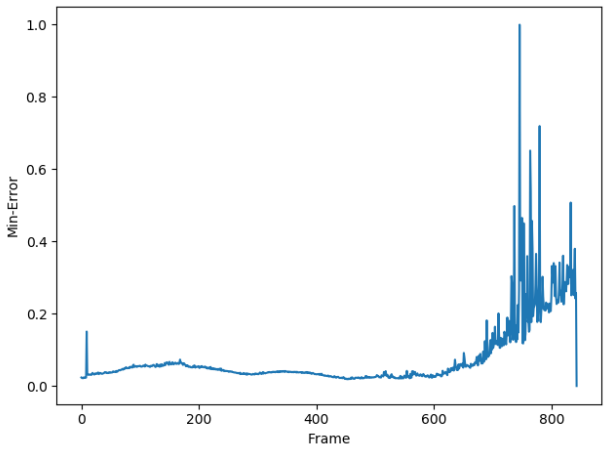


Figure 3: The minimum change error $\delta$ for the sequence Video B

The goals was to skip over the sequences of the video that had similar amounts of error, but take more samples when error was more irregular. In Figure 4, frame difference was fairly regular until around frame 650, where each frame was very different from the last.

This submission includes a demonstration of non-uniform sampling for Video B. Please refer to **examples/demo-non-uniform-vidB.mp4** and compare this against to **examples/demo-uniform-vidB.mp4**. Alternatively, use your own footage by running the web server.

### 2.2.3 Motion Trails

The paper discusses a method of adding motion trails to the timelapse video through a "Virtual Shutter". In classical photography, the shutter speed or exposure time is the length of time the camera sensor is exposed to light. Increasing the amount of exposure time will create an effect referred to as a motion trail.

Motion trails cannot be reproduced in timelapse photography. Thus, the Virtual Shutter aims to achieve this instead by compositing "multiple frames together to simulate dense stroboscopic motion".

(a) Original frame (no motion trail)

(b) Frame with Virtual Shutter

Figure 4: The images describe the before and after frames of applying the Virtual Shutter on Footage C.

This submission includes a demonstration of the above sequence titled **examples/vidC-motion-trails.mp4**. Please watch this to gain an understanding of how this works.

## 2.3 Hyperlapse Photography using the Bing Image Search API

Hyperlapse photography is a popular type of timelapse photography that uses motion. One of the many problems encountered in the project was finding suitable frames for the timelapse.

One idea was to utilise the massive amount of data already taken and put on the internet. Using Bing's image search API over REST, locations could be searched and the subsequent images could be downloaded and put into a timelapse. Obviously, without any semantic knowledge of these images this would be difficult and not entirely accurate. However, it was worth a try.

After following the Microsoft documentation found here: `https://docs.microsoft.com/en-us/azure/cognitive-services/bing-image-search/quickstarts/python`, it was fairly quick to download up to a hundred small images into RAM.

Figure 5: First sixteen search results for "Empire State Building"

The resulting images were striking similar. These images were then pushed to the Timelapse Processor to generate a hyperlapse. This video is included in the submission as **examples/hyperlapse-empire-state.mp4**.

This was a good start. But more work was required to integrate the images better. Resolutions and colour accuracy varied heavily throughout the images that Bing found.

# 3 User Guide to Web Application

**Note: Flask is not a production server. Please turn off browser caching before using the web application. If you cannot play the video in the browser then please find the video under: /static/output/output.mp4**

## 3.1 Generating a timelapse from an existing data set

Complete the following steps:

1. Run the web application.
2. Under 'Generating your Timelapse' click 'Choose Files' and select timelapse frames.
3. Next, select your option of Uniform Sampling, Uniform Sampling with Motion Trails, Non-uniform Sampling and Non-uniform Sampling with Motion Trails.
4. Click 'Make timelapse!'

A new page will open called 'Here you go...'. Play the timelapse from there or right-click to download.

## 3.2 Generating a hyperlapse from a Bing search term

Complete the following steps:

1. Run the web application.
2. Under 'Generating your Hyperlapse', type in your search term
3. Next, type in the number of images from Bing you would like to include in the hyperlapse.
4. Click 'Make hyperlapse!'

A new page will open called 'Here you go...'. Play the hyperlapse from there or right-click to download.

Please visit the Appendix of this report for further visual aid.

# 4 Evaluation

This project was particularly fun because of the freedom to experiment and play with different technologies. Motion trails implemented as a Virtual Shutter had great results and looks particularly great on timelapses with dark backgrounds and light foregrounds.

Python and OpenCV was a great tool to use for this application. OpenCV provides a wealth of built-in tools for image processing. Furthermore, Python allowed for this project to use interaction methods such as a web interface. It was a language that was very easy to get working from the start. For example, calling REST APIs and loading images into memory for processing was very simple and worked out of the box

It took some time to understand the full extent of OpenCV. For example, HTML5 only supports H.264 encoded videos and it was tricky to understand how this worked with OpenCV's VideoWriter interface. However, this encoding format could be specified as: `cv2.VideoWriter_fourcc('H', '2', '6', '4')`. It was a big but worth-while step up from MATLAB.

Each 'Priority' goal that was defined at the start of the project was completed and tested. However, only one of the 'Nice to have' goals were complete which was creating a Helpful UI. This was due to running out of time. One feature that would have made a big impact on the timelapse feature would have been image stabilisation as most timelapses do suffer from camera movement.

As defined earlier, implementing a web frame framework such as React or Angular would have been nice, but it would not have contributed to a better end-product as the real part of this project is the final timelapse or hyperlapse videos.

Another possible improvement to make was to utilise the method of non-uniform sampling to order the frames based on their sum-of-squared differences in the hyperlapse feature. This would have improved the transitions between frames, as opposed to using random frame orders as what as submitted. This was not considered during the project at any time but soon after completion, it was realised that this should have been implemented.

The paper describes that the sampling interval of a non-uniform timelapse to be user-defined. However, this was left static in the final project. This means that some timelapse footage that is uploaded could not look as good as it could have been if it was set custom.

Overall, I am quite proud of the project and the results I have managed to gain. I have gained a good understanding of how modern computational timelapse photography works. Slightly more time was required to fully flesh out hyperlapse photography through Bing search. More polishing and enhancements could have been made, but the application is still very much user-ready and would be considered a deliverable in the real world.

# 5 Appendix

## 5.1 Web Server Interface

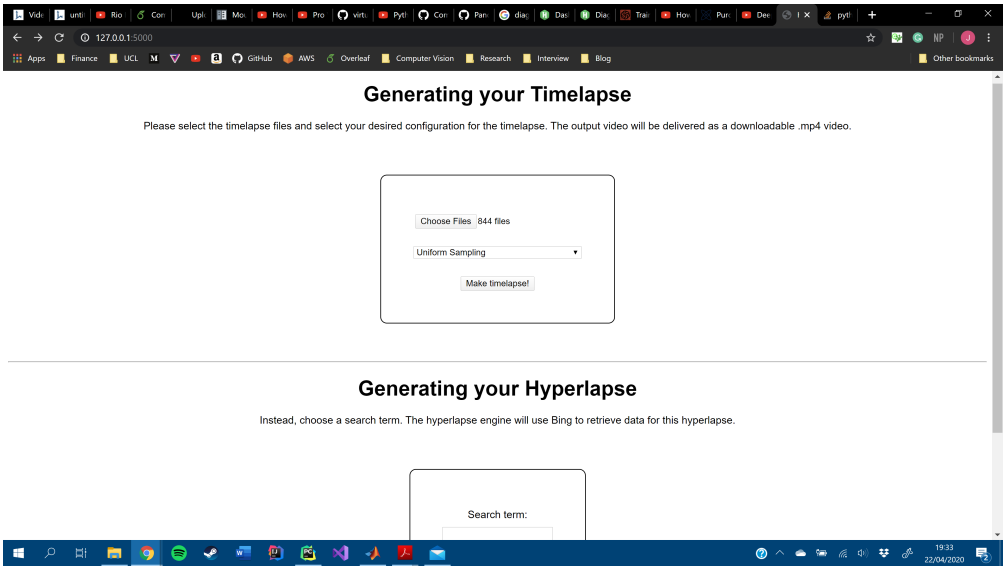


Figure 6: Homepage

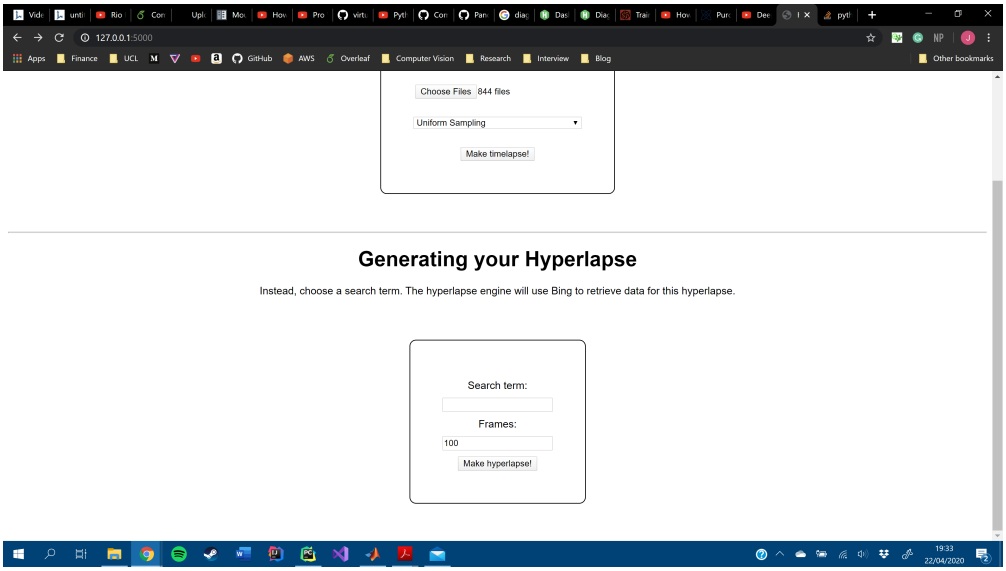


Figure 7: Homepage

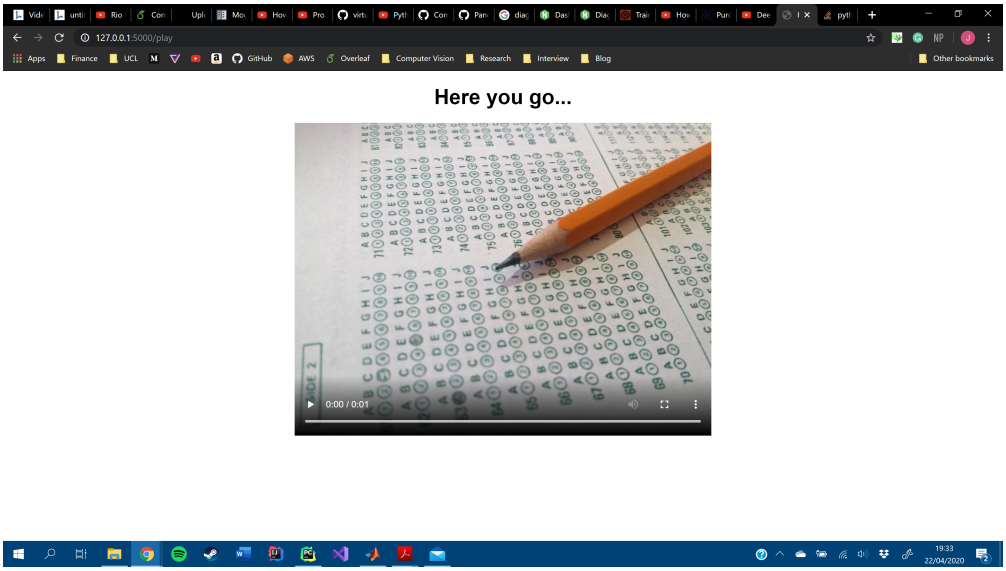


Figure 8: Preview page

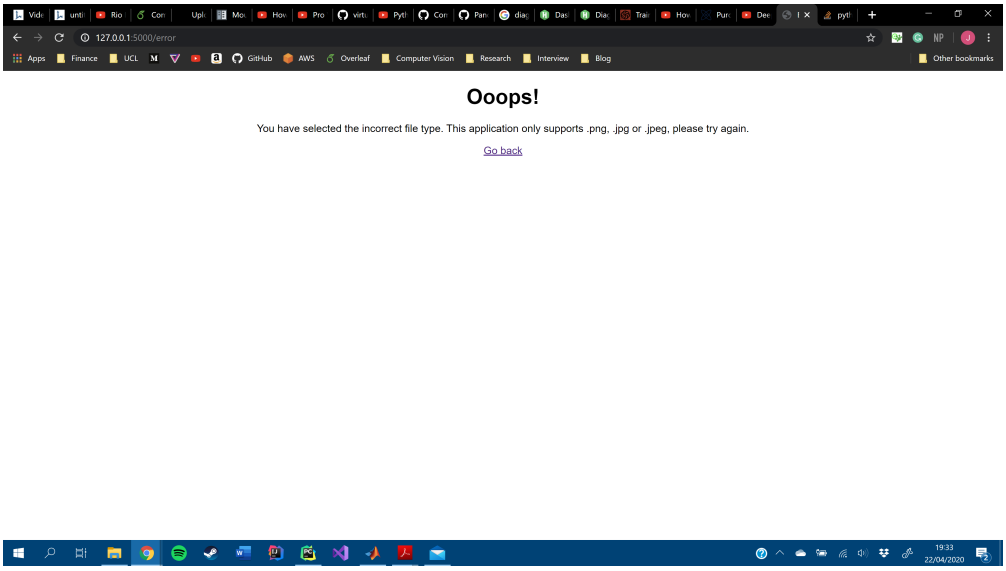


Figure 9: Error page