# COMP 50CP Project: Parallel Video Content Matching

Matthew Yaspan, Isabelle Sennett, Mitchell Katz, Alex King

**Design Story**

For our final project, we will be implementing a *video content matching* program; effectively a search-by-video search engine. This is an interesting and open problem in computing, and it has very practical applications in areas such as copyright protection.

To match the content of video, there are several possible approaches. Many videos include video and audio, so some sort of programmatic matching of video frames and/or audio needs to take place. We have chosen to focus on video content matching rather than audio content matching. In order to detect similarity between two videos, the content needs to be mathematically represented, and two representations need to be evaluated for closeness. One such way of matching video content is with motion vectors, which are a way of detecting position of objects and their movement during a video. This is a powerful mechanism that leads to versatile matching, but it is complex and beyond the scope of what we feel we can accomplish in this project's timeframe.

Instead, we have chosen a more straightforward approach: match videos by matching a sequence of frames as images. If we can define similarity between two still images, we can average it over a sequence of images and make a prediction as to whether two videos share the same content or not. This pushes our solution to rely on one of several different image similarity algorithms. We want to strike a balance between designing the algorithm ourselves and achieving comparison accuracy that is as good as possible.

We have a few ideas for image comparison modules, and we plan on implementing increasingly complex ones as time allows. The first and most naïve module will be a root-mean-square error comparison much like the *ppmdiff* program from COMP 40. This is a brute force mathematical similarity between two images of the same size. It works well for detecting subtle changes due to compression, but can easily fail if slight transformations are applied to the image. A second, more resilient algorithm will be a histogram-of-quadrants approach, where subsections of an image are tallied for color distribution, and each quadrant is compared for similarity. A third, further resilient algorithm would focus on more attributes of an image, such as edge and point detection. This type of comparison becomes more complicated. We will not demonstrate the use of any predefined image comparison libraries unless we have already designed a few of our own.

We will store a local database of videos that a producer will scan and analyze for scannable subsections of a proper size, which will become jobs. Multiple worker threads will process these jobs, searching small subsequences of the video database, checking for similar frames and returning an overall score of similarity, which will then be "interpreted" by our program, eventually returning a score and information to the user.

We plan on using Python, with a wrapper to use FFMPEG for video frame decoding. We will either use the Python Image Library for manipulating frame buffers as PNG files, or we will edit

them directly as NumPy arrays. We will use the multiprocessing module for true parallel processing speed.

**Class and Object Diagrams**

Provided alongside this document.

**Timeline and Work Division**

We have a basic image comparison module working now, and we know how to extract video frames as arrays with FFMPEG. We have divided the work between us roughly by module. At the moment, Alex is working on image comparison modules, Mitchell is working on the segment comparison "consumer" code, Matt is working on the job creation "producer" code, and Isabelle is working on the GUI.

Timeline:

- 11/9: Initial Design due, lock in Python libraries and frameworks that will be used
- 11/16: Have minimum viable product completed, have GUI designed (maybe not implemented)
- 11/23: Refined Design due, have working prototype completed with GUI. After this point, we will investigate various optimizations and feature improvements, specified below.
- 12/7: Be prepared to present our work

**Possible Improvements**

1) Optimizations: heuristics for quitting a sequence search early, matching on only every nth frame, etc.
2) Other stronger comparison modules, or checking against various video filters (flipping, cropping, windowboxing, speeding up)
3) Investigating the use of an internet video database