# Motion Estimation and Visualization

**By William Walker**

**Introduction**

This report outlines an algorithm used to track the general motion of on-screen artefacts in a video. The core technique employed is the macro-block matching algorithm. This algorithm is applied to a sequence of frames extracted from a video, allowing us to analyse and visualise the motion of objects within the frames.

## Algorithm Description

The algorithm starts with the extraction of individual frames from the video source. Each frame is considered as a unique image, and the algorithm tracks motion between these frames. To accomplish this, the following steps are taken:

**Frame Segmentation**

Each frame is divided into smaller blocks of size X by X. The specific value of X can be adjusted to control the granularity of motion tracking. This segmentation reduces the image's complexity and size.

**Motion Vector Calculation**

Within each segmented frame, the algorithm aims to find a block that closely resembles its counterpart in the next frame. This is achieved using the Square Root Sum of Squared Differences (SSD) approach. By comparing the segmented blocks in the current and subsequent frames, we calculate the displacement (motion vector) between these blocks.

**Vector Space Representation**

The calculated motion vectors are stored in a vectorSpace canvas. Each vector is associated with its initial x and y position within the current frame. This canvas captures the spatial distribution of motion within the frame.

**Thresholding**

To eliminate unwanted vectors caused by the background or other artefacts, a thresholding mechanism is applied. Segments primarily containing blue (background) are skipped. This selective analysis ensures that only the desired on-screen artefacts' motion is captured.

**Arrow Generation**
Initially, the algorithm faces challenges in generating clear and concise arrows that effectively communicate the motion of the primary on-screen artefact. Varying the block size and search width parameters resulted in different issues. A small block size with a large search width obscured the primary artefact with excessive arrows, while a large block size reduced arrow quantity and size, leading to unclear motion representation.

**Abstraction Layers**
To address these issues, additional layers of abstraction were introduced. Using a small block size and a large search width, a significant number of vectors were generated. While this quantity of arrows could obscure the primary artefact, it provided an accurate representation of motion. A further layer of segmentation was applied to these vectors, averaging their motion and caching their path. The centre of these vector segments was denoted as the coordinate constructing the overall path. By performing this process over multiple frames, a more concise vector was created that effectively communicates motion without obscuring the primary artefact.

You can view the progression of my motion tracking in the PrevAttempts folder.

**Hyperparameter Descriptions**
**Frame Segmentation (blockSize):**
Divide frames into smaller X by X blocks. Adjust blockSize to control motion tracking detail. Smaller blockSize values capture finer motion details; larger values give smoother motion representation.

**Motion Vector Calculation (width):**
In segmented frames, find similar blocks using the Square Root Sum of Squared Differences (SSD) method. The width hyperparameter sets the search range for motion vectors. A wider width detects long-range motion but may introduce noise with excessive values.

**Arrow Generation (jump):**
To convey on-screen motion effectively, use the jump hyperparameter for vector averaging. It determines the step size when creating search rectangles over segmented images. Smaller jump values yield finer search rectangles, capturing fine motion details for accurate vector averaging. Larger jump values reduce search rectangles for computational efficiency but may miss small-scale motion.