# 3. Video Processing I

(a) Example foreground frame (b) Example background frame (c) Example result frame

Figure 3.1: An illustration of background replacement.

## 3.1 Video Background Replacement

This task is to access and modify each frame of a video 'monkey.mov' and replace the blue background with 'star_trails.mov'.

1. Download video extension for Processing through plug-in manager
2. Play the background video
3. Save each frame as 'PATHTOBACKGROUND/*i*.png', where *i* is the frame index.
4. Play the foreground video. For each non-blue pixel per frame, draw it onto a background frame which is sequentially loaded
5. Save the resulted frame
6. After all frames are generated, merge the saved frames into a QuickTime video using Processing Movie Maker (Menu/Tools-Movie-Maker).

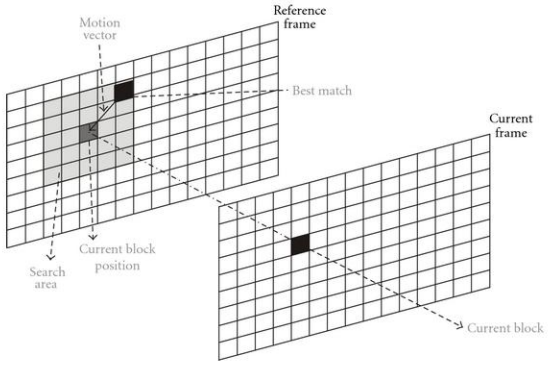
R Video I/O APIs of Processing: https://www.processing.org/reference/libraries/video/

R The *MovieEvent()* and *draw()* functions are not called at the same rate. So it would be nice to do your frame processing all within *MovieEvent()*. Or you can save all the frames at first and load them sequentially as image files in *draw()*.

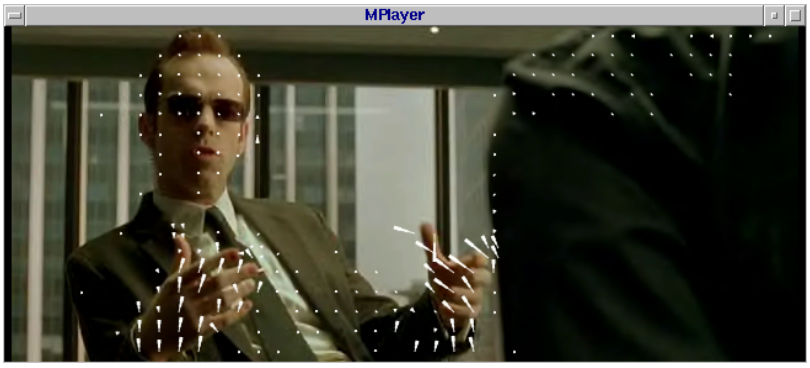
R An example of removing background: `http://learningprocessing.com/examples/chp16/example-16-12-BackgroundRemove`

R saveFrame API: `https://processing.org/reference/saveFrame_.html`

## 3.2 Motion estimation with Macroblock matching (Lab Submission 1)




(a) Illustration of the block matching




(b) Example extracted motion vectors. Please note that in your lab submission, you do not need to draw arrows. Highlight the pixels with high magnitude would be sufficient. Do not forget to do step 5.

Figure 3.2: The illustration of block matching algorithm and the extracted optical flows.

This task is to perform the motion estimation with macroblock matching. The basic premise of motion estimation is that in most cases, consecutive video frames will be similar except for changes induced by objects moving within the frames. The basic idea of motion estimation is to define grids

of block regions on two adjacent frames and find the displacement vector (a 2D Cartesian vector in 2D videos) between the matched blocks. To describe the meta-algorithm step by step:

1. Iterate the video frames $F_i$ of size $f_x \times f_y$; Define a grid block size $K \times K$, $K$ is preferred to be odd to make it easier to determine the central coordinate of each grid block. Each frame $F_i$ results in $f_x f_y / K^2$ grid blocks overall.
2. For each grid block $B_i$ at $(x, y)$ in frame $F_i$, search for the grid block $B'_{i+1}$ at $(x', y')$ in $F_{i+1}$ with the minimum sum squared distance (SSD) between $B_i$ and $B'_{i+1}$. SSD can be computed as

$$SSD(B_i, B'_{i+1}) = \sqrt{\sum_{bx} \sum_{by} \sum_{bc} (B_i(bx, by, bc) - B_{i+1}(bx, by, bc))^2} \tag{3.1}$$

   The displacement from $B_i$ to $B'_{i+1}$ can be represented as $(x' - x, y' - y)$, a 2-D vector. To speed up, you can search the neighbour blocks only within a certain radius.
3. Save the displacement vectors of frame $F_i$ in a 3D matrix of size $f_x/K \times f_y/K \times 2$ or two 2D matrices with size $f_x/K \times f_y/K$ each.
4. Visualise the displacement fields with a 2D image with the same size as frame $F_i$. The same magnitude value is assigned to all the pixels within the region covered by a single grid block. Alternatively you can draw arrows to represent the extracted displacement fields.
5. Visualise the boundary of the object with large displacement. Hint : the dilation or erosion might be helpful.
6. Repeat step 1-5 for all frames

The sources of this task are supposed to be submitted via eLearning and counts for 5% of your total assessment. Do not submit a video file. You are allowed to use any video to demonstrate your code. You are required to provide a working live demo of your work to your tutor (Lab 6 Week 7).

Drawing arrow is not required. If you really want to draw arrows, the following code can be your starting point. void arrowdraw(int x1, int y1, int x2, int y2) { line(x1, y1, x2, y2); pushMatrix(); translate(x2, y2); float a = atan2(x1-x2, y2-y1); rotate(a); line(0, 0, -10, -10); line(0, 0, 10, -10); popMatrix(); }

For more details of in-depth understanding of the video estimation algorithm, you may refer to `http://web.stanford.edu/class/ee398a/handouts/lectures/EE398a_MotionEstimation_2012.pdf`

For a faster solution of estimating optical flow with Lucas-Kanade method please see
`http://research.ijcaonline.org/volume61/number10/pxc3884611.pdf`
and a Matlab solution at
`http://www.cs.ucf.edu/~gvaca/REU2013/p4_opticalFlow.pdf`
Please note for the lab submission you are not required to use the Lucas-Kanade method.