Jerry Chang
Problem Set 10

6.815: Motion Estimation Implementation

Implementation

I implemented an algorithm for detecting and visualizing motion in an image sequence. Given an input of a sequence of images, it outputs a series of motion vectors which are mapped onto the images to display motion. For my implementation I divided each image into 16x16 pixel blocks, used an exhaustive block matching method to determine the block with the minimal square difference within a 7px search window. Then I mapped a motion vector corresponding to the motion detected between the two frames. The test image sequence I used was from 101 Dalmations. The main challenge in implementing this was to break down the algorithm into smaller functions that would generate the desired result without making excessive copies of the images.

Test Cases

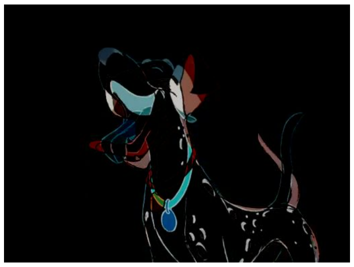
Test cases are written in a10_main.cpp and test each of the functions that make up motion estimation.

Test Images:

Frame 1

Frame 2

1) testConstDiff() : tests the constDiff function which outputs an image of the difference between two image frames.



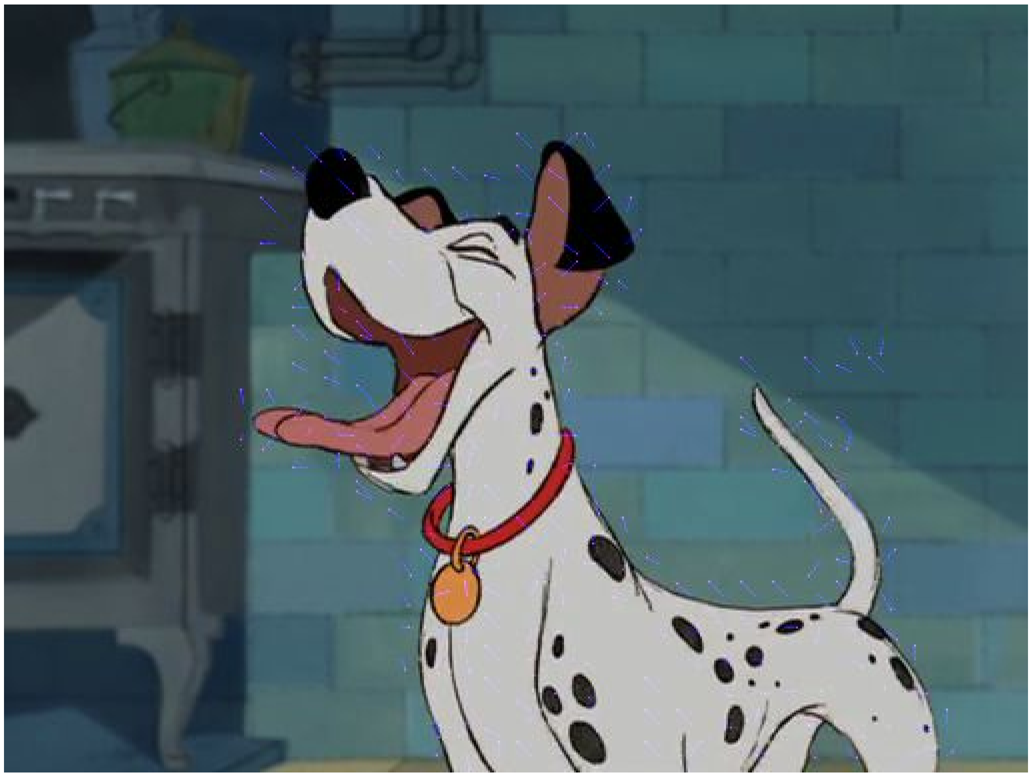
2) testCompareImages(): tests the compareImages function which outputs the total square difference between pixels in two images

3) testCreateBlock(): tests the creatBlock function which outputs a 16 x 16 px block with top left corner at location x and y.

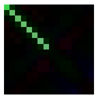
4) testSearchWindow(): tests the searchWindow function which takes a block with top left corner at location x and y and compares it to a seven pixel search window surrounding the block returning the minimum difference block.

block1 block2

5) testEstMotion(): tests the overall motion estimation function which takes two image and outputs the second image with motion vectors mapped on corresponding to motion within the frame. Direction of motion is marked in red.



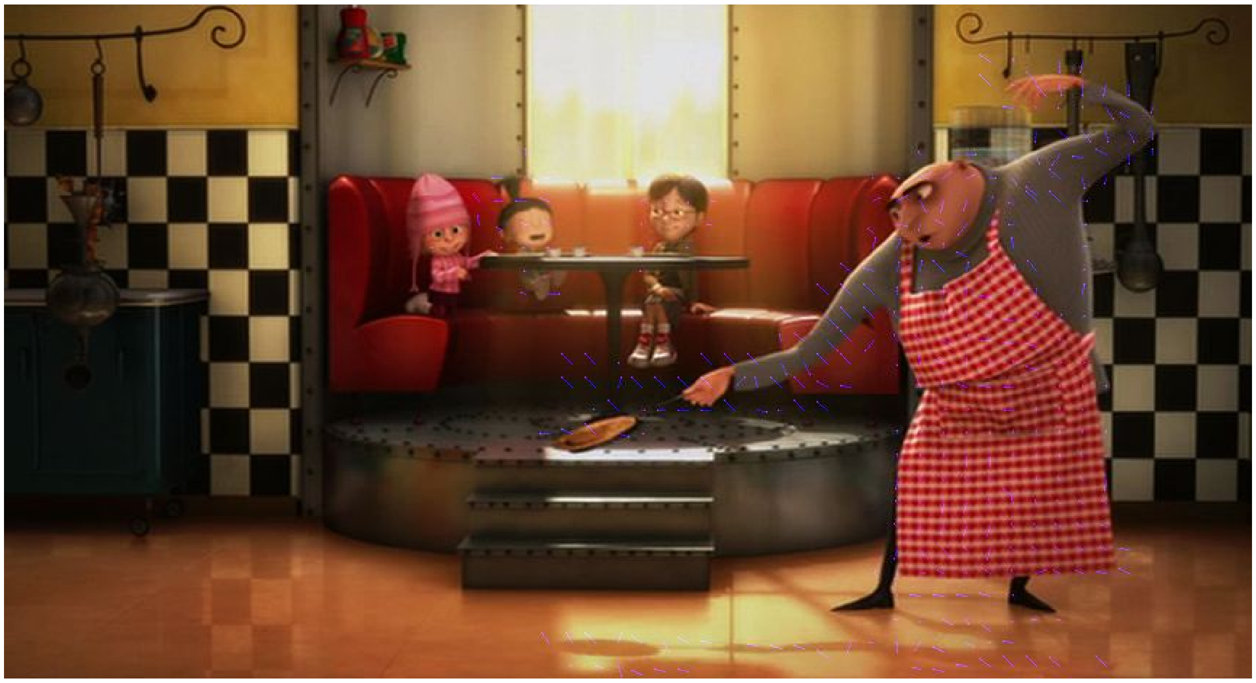
6) testVisVector: tests the visVector function which uses the Bresenham line algorithm to draw a line between two points in a block



Motion Estimation on Despicable Me Images

Input

Output



<u>Background</u>

Q: What is motion estimation?
A: The conventional application of motion estimation is for reducing temporal redundancy in video encoding. However, it can also be used for estimating the motion of any object. The key idea is to use a motion vector to predict the current frame from a reference frame.

Q: What are the advantages of block-matching algorithms?
A: The common video compression standards H.261-H.264, MPEG-1-MPEG-4 all use Fixed Size Block Motion Estimation (FSBME) due to the algorithm's simplicity as well as the fact that no image segmentation information needs to be transmitted.

Q: How does Fixed Size Block Motion Estimation work?

Block Matching Algorithms
Block based motion estimation is the most common motion estimation approach for video encoding. The supposition is objects move within a frame. The frame is divided into a matrix of blocks that are compared with adjacent neighbors within a search area of p pixels on each side where p is the search parameter. Larger motions will require a larger search parameter. Common block sizes are 16px by 16px with a search parameter of 7px. For identifying the best matching block Minimum Square Error, Minimum Absolute Difference, and Sum of Absolute Difference can be used. After computing the similarity, the algorithm will output motion vectors for each block from the reference frame to the current

Fast Methods (Low Computational Complexity)
Fast block matching tries to achieve the same Peak Signal to Noise Ratio (PSNR) as a full search with fewer computations. In essence we try to compare as few block as possible without significantly reducing PSNR.

Q: How do we compare performance of motion estimation algorithms?
A: Motion estimation approaches emphasize either reducing computational complexity, representing true motion, and reducing bit rate.

Q: What algorithms are used for Fixed Size Block Motion Estimation?
Full Search Algorithm
This is an exhaustive search that compares a block with every possible location in a search window and will provide the highest PSNR at the cost of computational complexity (N^2).

Three Step Search
Three step search uses a logarithmically decreasing step size to find the matching block. We assume that the difference in block distortion measure increases monotonically as the checking point moves away from the global minimum. Due to noise and luminance changes this assumption may not always be true. The initial step size is half the maximum distance d wherein 9 points are checked from the starting center including the center (d/2 above, d/2 below, d/2 left, d/2 right, d/2 at each corner). The closest matching point is chosen as the new center and the step size is halved if the center remains the same.

New Three Step Search
This is a fast algorithm that uses a center biased searching scheme which reduces computational costs but may miss small motions. Like three step search 9 surrounding points of a certain step size are searched in addition to 8 points neighboring the center. If the closest matching point is the center, then the search stops. If the 8 points neighboring the center is the closets match then one additional search is performed only for the 8 points neighboring that match and then the search stops.

Four Step Search
This algorithm also uses a center-biased approach by starting with a step size of d/4 which requires four steps to cover the search window of 7px.

Diamond and Hexagonal Search
Diamond Search (DS) and Hexagonal Search (HEXBS) are similar to four step search but the shape of the search pattern is changed to be a diamond or hexagon and the number of steps to the algorithm are not limited. In Diamond Search both Large Diamond Search Pattern (LDSP) and Small Diamond Search Pattern (SDSP) starting with LDSP on the first step and SDSP on the last step. Because the search pattern is not too small or large and the number of steps is not bounded, DS results in a PSNR close to Full Search without the computational complexity.

HEXBS requires fewer search points and thus performs better than DS.
Cross-diamond-hexagonal searches (CDHS) have also been shown to perform faster than DS.

Hierarchical Block Matching

Hierarchical block matching calls for performing motion estimation starting with the lowest resolution and then passing the motion vector onto the next higher resolution where the motion estimate is refined and smaller search windows are used. Reducing resolution also reduces motion speed which makes hierarchical approaches useful for high motion video.

Adaptive Motion Estimation Methods

Adaptive Motion Estimation uses the fact the general motion in a frame is coherent. Thus the motion block to the immediate left of a block is used to estimate its motion vector. This approach checks the predicted point as well as a rood pattern around the point.

Intern-Block Motion Algorithm

This approach uses the motion of adjacent reference blocks to predict a block’s motion vector.

Variable Sized Block Matching Methods

Using fixed sized blocks may fail to match motion in a sequence especially along moving edges. Smaller blocks are used to describe complex motion while larger blocks are used in stationary areas and areas undergoing uniform motion.

Optical Flow Computation Method

Optical flow uses an inverse finite element approach by calculating the gradient, Laplacian, and velocities of each pixel. This is often a pre-processing steps for other algorithms. First you sample the image intensity at a point (x,y). One method is to assume this intensity is constant and compute displacement dx and dy over dt. The Lucas-Kanade method uses least squares considerations to solve optical flow around a neighborhood.

Phase Correlation

These algorithms work in the frequency domain and are based on cyclic correlation. Using phase correlation, you can achieve sub pixel estimation.

Source:

Patel, Bhavina, R. V. Kshirsagar, and Vilas Nitnaware. "REVIEW AND COMPARATIVE STUDY OF MOTION ESTIMATION TECHNIQUES TO REDUCE COMPLEXITY IN VIDEO COMPRESSION." International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering 2.8 (2013): 3574-2584. Web. 23 Nov. 2015.