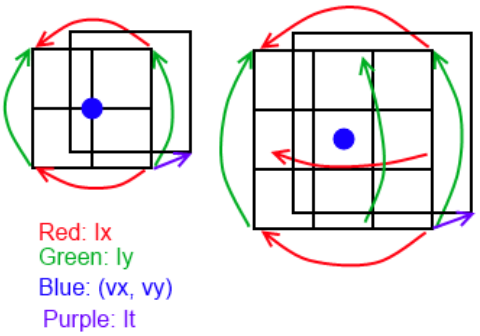
COMS30121 Image Processing and Computer Vision

Motion Assignment Report

J. Bligh, Q. He, N. Williams

Task One:

For task one, we implemented a function called calculate_derivatives. This function takes two frames, one at time t, and the other at t+1 (takes every single frame). Each frame is converted to a grayscale image so we only have to deal with one channel. Inside the function we create 3 matrices representing I_x, I_y, and I_t. Initially we tried to work out the I's for the centre point of a 2x2 pixel grid as done in the lectures, but changed that to a 3x3 grid (as shown in the diagram) as averaging over a larger region gives better results. To display them we normalised them in our separate function called showNorms. This takes each matrix, normalises it (to ensure it's between 0 and 255), converts its type to CV_8UC1 and then shows them in a new window. Despite normalising each matrix in exactly the same way, our I_x doesn't quite show properly in the output. However since the rest of the calculations are done on the pre-normalised data the rest of the program is unaffected.



Task Two:

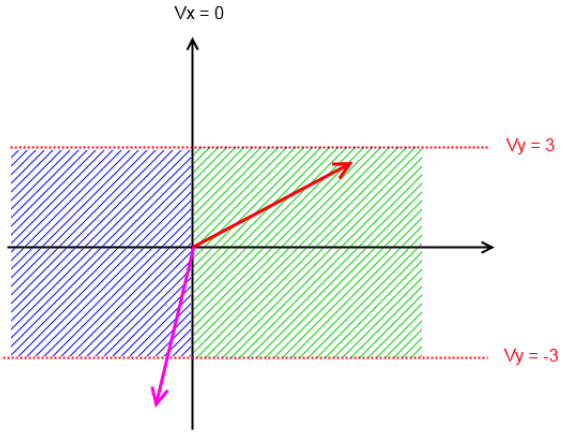
Part 1: The LKTracker function takes 4 parameters: The position where we want to compute the motion at, the current frame – just for tests to make sure nothing goes out of bounds, and a vector of matrices containing I_x, I_y, and I_t. The region size is a variable defined at the top of the code so that it can be changed easily enough. The position is passed to the function using a for loop – every 10th pixel is passed (since the program runs much slower if you pass the positions more frequently).

Part 2: Then we calculate the matrices A and b, and using the built in OpenCV functions we find V (= $A^{-1}b$). To make the vector long enough to be seen, we multiply V by 3.

Part 3: For this part we implemented a separate function that took a copy of the original video and, at each point the motion has been calculated for, we draw a line onto the copy of the original video.
When using the webcam we have shrunk the video (by a third) so that the motion is more precise; otherwise all the lines drawn look erratic and incorrect.

Task Three:

For this task we created a new copy of the original video simply to show the left and right movement on, instead of combining it with the original motion field. We stated that for each vector, it must be a certain length before it is drawn (using a lineLength function we implemented). Then for a left movement, we said that V_x must be negative (blue shaded area in diagram) and V_y must be within a certain range. For a right movement V_x must be positive (green shaded diagram).



If V_y (see the red dashed line in the diagram) sits in a particular range, then up and down movements won't be considered. In the diagram, the red arrow is within the range and so is considered a right-movement. The pink arrow, however, is outside of the range and so is not considered.

If our region size is too big for the frame, the motion field simply isn't drawn.