

Computer Vision: *Motion Estimation, Predicting the next frame of a video and Counting the number of moving vehicles in a video*

Requirements:

Task 1

Motion estimation. Block matching is a commonly used motion estimation approach. Block matching compares each block of a frame with blocks in a past or future frame in order to find the corresponding block (i.e. the block minimising an error measure, such as the Mean Square Error or the Mean Absolute Error). The search is usually restricted to a window centred on the position of the target block in the current frame. The search window defines an upper limit on how fast objects can move between frames (maximum displacement), if they are to be coded effectively.

Given two consecutive frames of a video sequence (I_t and I_{t+1}), implement a full-search block-matching algorithm that uses different block dimensions. The output of the algorithm will be (i) the motion field and (ii) the image P_{t+1} , the prediction of I_{t+1} .

- Display the motion vectors as a set of arrows starting from the centres of the blocks and superimposed on I_{t+1} . Use a block size of 16x16 pixels and a search window of 20x20 pixels.
- Based on the motion vectors you have estimated, display the image P_{t+1} , predicted version of I_{t+1} , using a block size of 16x16 pixels and a search window of 20x20 pixels.
- Use a 16x16 search window and block sizes of 4x4, 8x8 and 16x16 pixels. Display the results (you can zoom on a portion of the image) and comment the differences between the results.
- Use an 8x8 block size and search windows of 8x8, 16x16, and 32x32 pixels. Display the results (you can zoom on a portion of the image) and comment the differences between the results.
- Plot the execution time of your software (i) against the variation of the block size and (ii) against the variation of the search window. Comment the results.

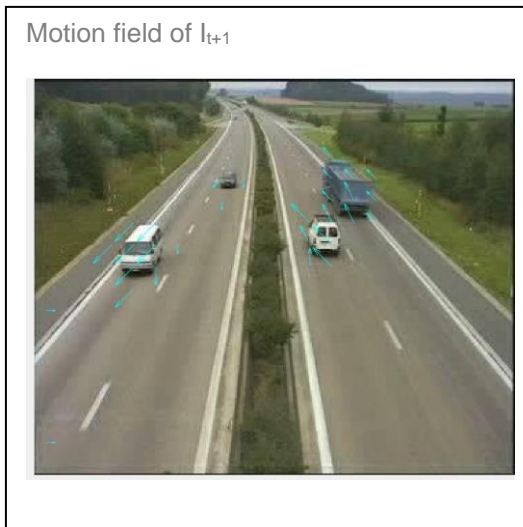
Task 2

Objects. Moving objects captured by fixed cameras are of great importance in several computer vision applications.

- Write a function that performs pixel-by-pixel frame differencing using the first frame of an image sequence as reference frame. Apply a classification threshold and save the results to disk.
- Repeat the exercise using the previous frame as reference frame (use frame I_{t-1} as reference frame for frame I_t , for each t). Comment the results.
- Write a function that generates a reference frame (background) for the sequence using for example frame differencing and a weighted temporal averaging algorithm.
- Write a function that automatically counts the number of moving objects in each frame of a sequence. Generate a bar plot that visualizes the number of objects for each frame of the sequence.

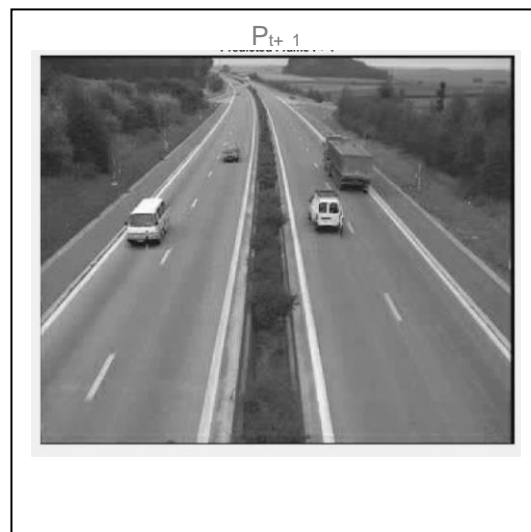
Report:

Task 1 - (a)







Comments

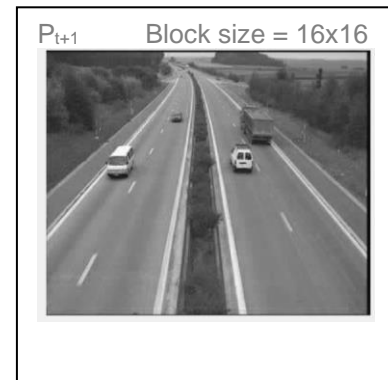
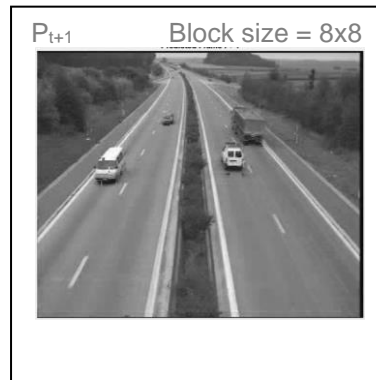
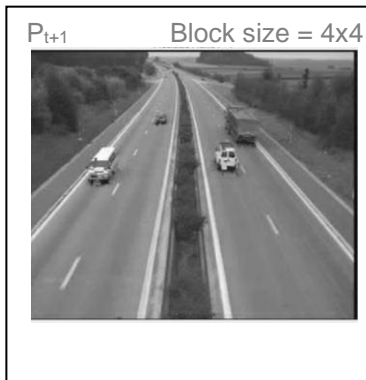
Task 1 - a) Block matching is implemented in the `ICV_Motion_Estimation.m` file by taking each block in the frame and comparing it to potential blocks in the next frame inside the specified window size. The Mean Absolute Error (MAE) is calculated to find the most similar block in the next frame. The differences in the horizontal and vertical directions are stored as the motion vectors (dy, dx) and are used along with the “quiver” function to superimpose arrows to represent the displacement of each block.

Task 1 - (b)**Comments**

Task 1 - b) When using a frame to predict the next frame (see `ICV_Predict_Frame.m` file) I started by using the matrix of frame I_t as the base for the prediction of P_{t+1} instead of starting with a matrix filled with zeros and then moved each block in this frame according to the motion vectors calculated previously. By doing this I managed to obtain a more cleaner and more accurate image as it avoided slight fractures in the image which I faced previously.

However, the predicted frame still has some flaws when compared to the actual frame (as shown below). We can see some blurred lines on the vans in the predicted image which are not present in the actual image. In addition when calculating the MAE between the predicted image and the actual image I found it to be 5801271.

	<i>From Frame I_{t+1}</i>	<i>From Predicted Frame P_{t+1}</i>
Van on the right		
Van on the left		

Task 1 - (c)**Comments:**

When using a 16x16 search window with a 4x4 block size I found the predicted image to be least visually pleasing due to the fractures in the image and calculated the MAE to be 4222736 between the predicted and actual images. The 8x8 block size produced a slightly more visually pleasing image with a lower MAE of 2154390. The 16x16 block size produced the worst prediction with an MAE of 11751290 and is the exact same image as frame I_{t+1} .

Differences between the results are shown below:

	P_{t+1} Block size = 4x4	P_{t+1} Block size = 8x8	P_{t+1} Block size = 16x16
Van on the left			
Van on the right			

We can see from these 2 vans from the images that as the block size decreases the more fractured and unclear the objects in the predicted image are. The output produced by the 8x8 block size scored the lowest MAE out of the 3 and seems to be the most similar to the actual image I_{t+1} .

Task 1 - (d)**Comments:**

When using a 8x8 block size I found the 32x32 window size produced an MAE of 2442290 and the 16x16 window size produced an MAE of 2154390. Both images seemed equally appealing visually with a similar amount of blurred lines in the moving objects in the frames. The 16x16 window size produced an identical result to the previous question as it used the same block size (8x8) and window size.

The 8x8 window size produced an MAE of 11751290. This result is identical to that from the previous question which uses a window size of 16x16 with a block size of 16x16. This produced the worst prediction and is the exact same image as frame I_t as it gives an MAE of 0 between frame I_t and P_{t+1} .

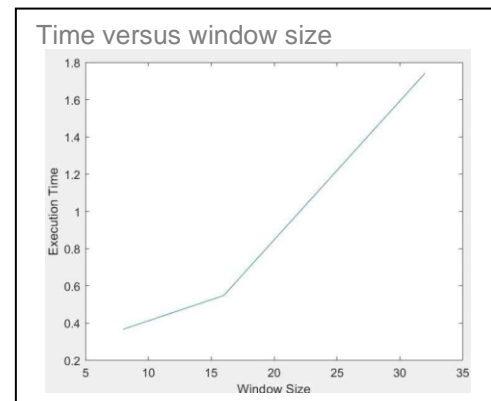
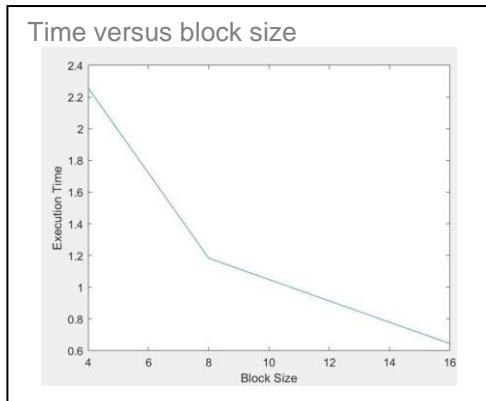
Differences between the results are shown below:

	P_{t+1} Window size = 8x8	P_{t+1} Window size = 16x16	P_{t+1} Window size = 32x32
Van on the left			
Van on the right			

Comparing the 'van on the left' for window sizes 16x16 and 32x32 we can see it is similar but with some differences. Both images contain blurred lines but the 32x32 window size output has a large darker square shaped blur in the front centre of the van whereas the 16x16 window size output does not.

The 'van on the right' however is strikingly similar for the 16x16 and 32x32 window size outputs with similar blurred patterns below and on the right side of the van.

Both the vans for the 8x8 window size outputs are identical to the ones in the image I_{t+1} and therefore don't contain any extra fractures or blurred lines.

Task 1 - (e)**Plot graphs:****Comments:**

The graph showing the execution time against the variation of the block size displayed that the larger the block size the shorter the execution time. Whereas the graph showing the execution time against the variation of the window size displayed that the larger the window size the greater the execution time.

The results make sense as the closer the block size is to the window size the lower the number of blocks to be compared to in the next frame meaning less computation takes place and therefore a shorter execution time. Similarly when the window size is larger and the block size is small the software must compare more blocks in the next frame to each block in the current frame meaning more computation is done and therefore a longer execution time takes place.

Results Collected using the 'tic' 'toc' functions in MatLab:**Window Size 16:**

Blocksize,4x4:
2.257681seconds.

Blocksize,8x8:
1.182999 seconds.

Blocksize, 16x16:
0.644236 seconds.

Block Size 8:

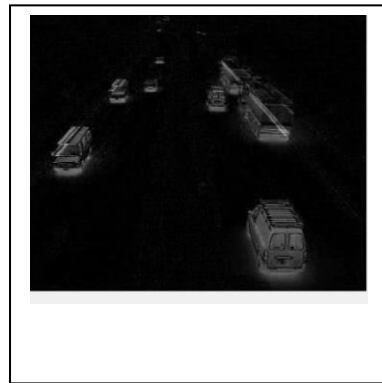
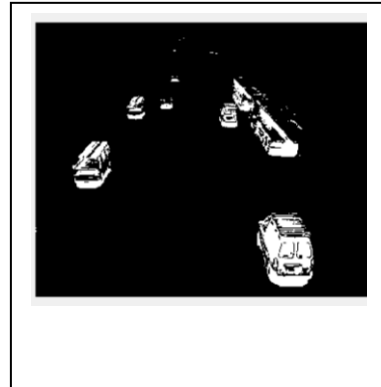
Window size,8x8:
0.367121 seconds.

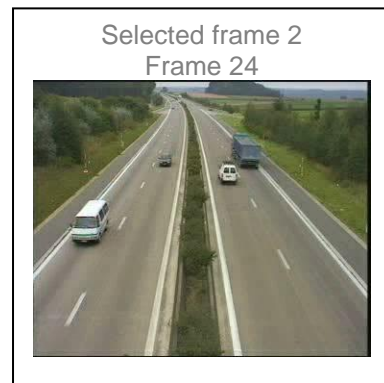
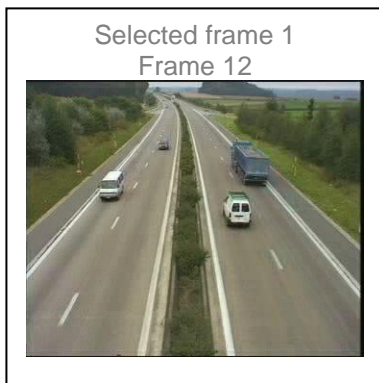
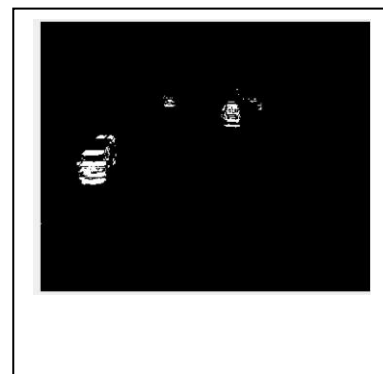
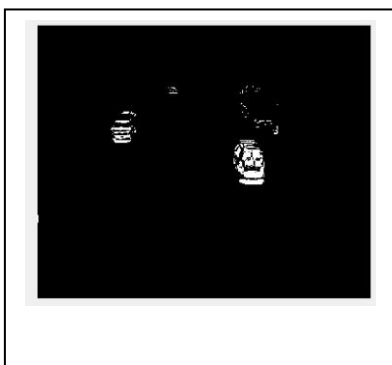
Window size,16x16:
0.548471 seconds.

Window size, 32x32:
1.742362 seconds

Task 2 - (a)**Original frames:**

Reference frame

Selected frame 1
Frame 12Selected frame 2
Frame 24**Frame differencing:****Threshold results:****Comments: (See page 9)**

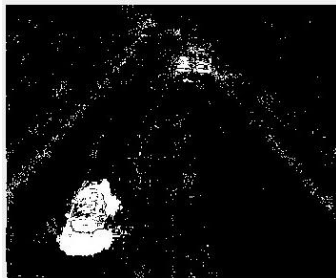
Task 2 - (b)**Original frame:****Frame differencing:****Threshold results:**

Comments

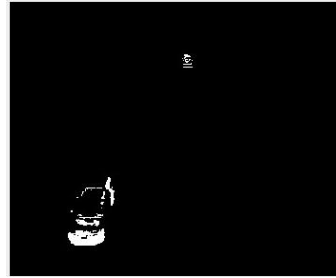
Task 2 – (a) Frame differencing was done by getting the absolute difference between the reference frame and the selected frame which eliminates the static background of the video and highlights the moving objects between the frames. I implemented a threshold whereby any pixel value less than the threshold was assigned the value 0(black) and anything above the threshold was assigned the value 255 (white). This was done to get a black and white binary image.

When using a threshold value of 50 I found that when compared to the frame differencing images, moving objects were highlighted better and could be seen more clearly. When using the first frame in the image sequence as the reference image we can see the outline of the vehicles from the reference image in our difference image as well as the outlines of the vehicles from the current frame.

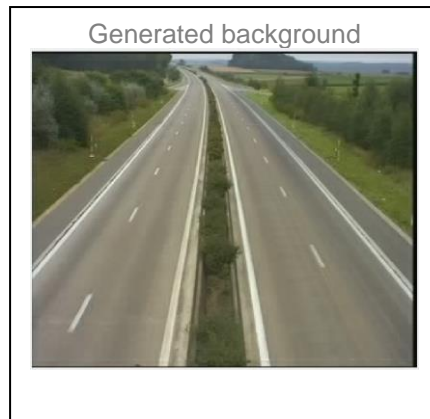
Task 2 – (b) By using frame I_{t-1} as the reference image I found that the frame difference images were fainter and harder to see in Task 2a. In addition even when using the same threshold value of 50 I found the threshold results were harder to see than when compared to the results from Task 2a. By lowering the threshold value I managed to get a closer result to Task 2a where the vehicles were clearer however I found that this also introduced noise in the image whereby there would be stray white pixels throughout some frames. The higher the threshold value the less visible the vehicles were but the lower the threshold value the more prone it is to noise, so it is vital to get a good balance. (See below).

Threshold value = 15

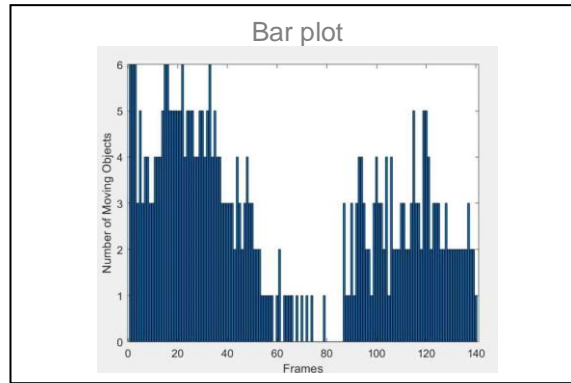
Shows a lot of noise in the image

Threshold value = 50

Shows no noise but displayed moving objects are less visible

Task 2 - (c)**Comments:**

To generate this background image I calculated the sum of all the frames in the video and divided it by the number of frames in the video to get the weighted temporal average. The result generated is a highly accurate depiction of the background region from the dataset video. This can be used for frame differencing by deducting it from any given frame in the video to highlight any moving objects in the video. The generated background contains an averaged image of all the static components in the image sequence.

Task 2 - (d)**Comments:**

To be able to count the number of moving objects in each frame I first took the difference between each frame from the background image calculated from Task 2c. (The file `ICV_Get_Segments.m` was made to obtain the segmented objects in each frame by using a series of methodologies I created).

First each difference frame is converted to a black and white binary image based on a threshold value. I came across difficulties as some frames had a lot of stray white pixels (noise) which I had to remove. I made a function called `ICV_Remove_White_Noise.m` which takes each white pixel and checks its surrounding 8 pixels and if more than 4 of the surrounding pixels are black then it is converted to black. Similarly I found black holes in the segmented objects in my image so I made a function called `ICV_Remove_Black_Noise.m` which checks the surrounding 8 pixels and if more than 4 of its neighbours are white it is changed to white.

Even after removing noise and cleaning up the image a bit I still found that some objects were disconnected, whereby 1 object would be split into multiple segments. To overcome this I came up with a solution which involves connecting the disconnected pieces of an object by creating the function `ICV_Blockify.m`. This function simply takes as parameters the difference image and a magnitude. It then finds each white pixel bordering one or more black pixels and creates a layer of white pixels around it in a block like manner. The 'blockifyMagnitude' variable decides how thick this layer will be. If the 'blockifyMagnitude' variable is too small then the object could remain disconnected but if the 'blockifyMagnitude' variable is too high then it may overlap other objects so it was vital to find an optimal value. After hours of experimenting I found optimal values for the 'blockifyMagnitude' variable which compliments the applied threshold value.

Now that my disconnected objects were connected using my 'blockify' method I decided to implement a connected components algorithm to label each component at each frame of the image sequence so I could count the number of moving objects per frame. The connected components algorithm I implemented goes through each black and white difference frame, finds the first white pixel and labels it '1'. It then gets each white pixel and checks the surrounding pixels to see if any of its neighbours are labelled. If a neighbour is labelled then the current pixel is classified with the same label. If the current pixel does not have neighbours which are labelled then it is detected as a new object and is given a new label (e.g '2'). By running this algorithm at each frame in the video, my software was able to make reasonably accurate predictions for the number of moving objects in each frame which are shown on the bar plot above.

I found when using the 'blockify' function with an optimal magnitude, that some moving objects far in the distance were detected better as the function highlighted them by enlarging the small white blocks in the distance. (See below for outputs from the algorithm)

