Tadhg Riordan
12309240
CS4053: Computer Vision
Assignment 3 - Abandoned/Removed Object Detection Report

Assignment specification

"Develop a program (in C++ using OpenCV) to locate if, when and where objects are abandoned and/or removed within a video taken by a static camera. This will be evaluated in two stages:

- **Detection of changes to the static background due to object abandonment or removal.**
- **Distinguishing between object abandonment and object removal events.**

You must also evaluate the performance of your system."

High Level Description

I combined a number of popular Computer Vision methods to complete this assignment. For the actual detection of the object movement in the scene, I used Running Average with Background Subtraction. Having produced a difference matrix from the results of these methods, I used a Binary Threshold and Connected Components to identify and store the objects individually.

Running Average/Background Subtraction

I began by loading my desired video and capturing some useful information from it. I needed to capture the video length, the Frames Per Second, and the height and width of each frame. The built-in OpenCV function used to capture video length was giving me incorrect and inconsistent results, and so I found a working function which I have referenced in the comments section of the function itself. The rest of the information could be captured using built in openCV functions. I then began set-up for the Running Average/Background Subtraction part of my solution.

The variable 'running_avg_frames' is set, which is number of frames to take for the initial background and for the running average. I have set this to 100, and when referring to 100 from here on I will have used this variable.

The way in which this is to be done can be seen in the while loop, which runs once for each frame in the video.

- The video loads the next frame into 'frame', where it is converted to grayscale.
- the "running_avg" matrix is initialised. this matrix will store the average frame for the last 100 points.
- The three data structures declared and initialised earlier are put to use. The 3D vector frames_points_lists can be understood more clearly as a 2 dimensional array of lists of points (see fig. 1). frames_points_total and frames_points_average are both 2 dimensional arrays.
- Each point in the frame is accessed. the frames_points_list pushes the point found at position (row,column) to the list of points at position (row,column). the frames_points_total adds the point found at position(row,column) to the running total of points it keeps at position (row,column).

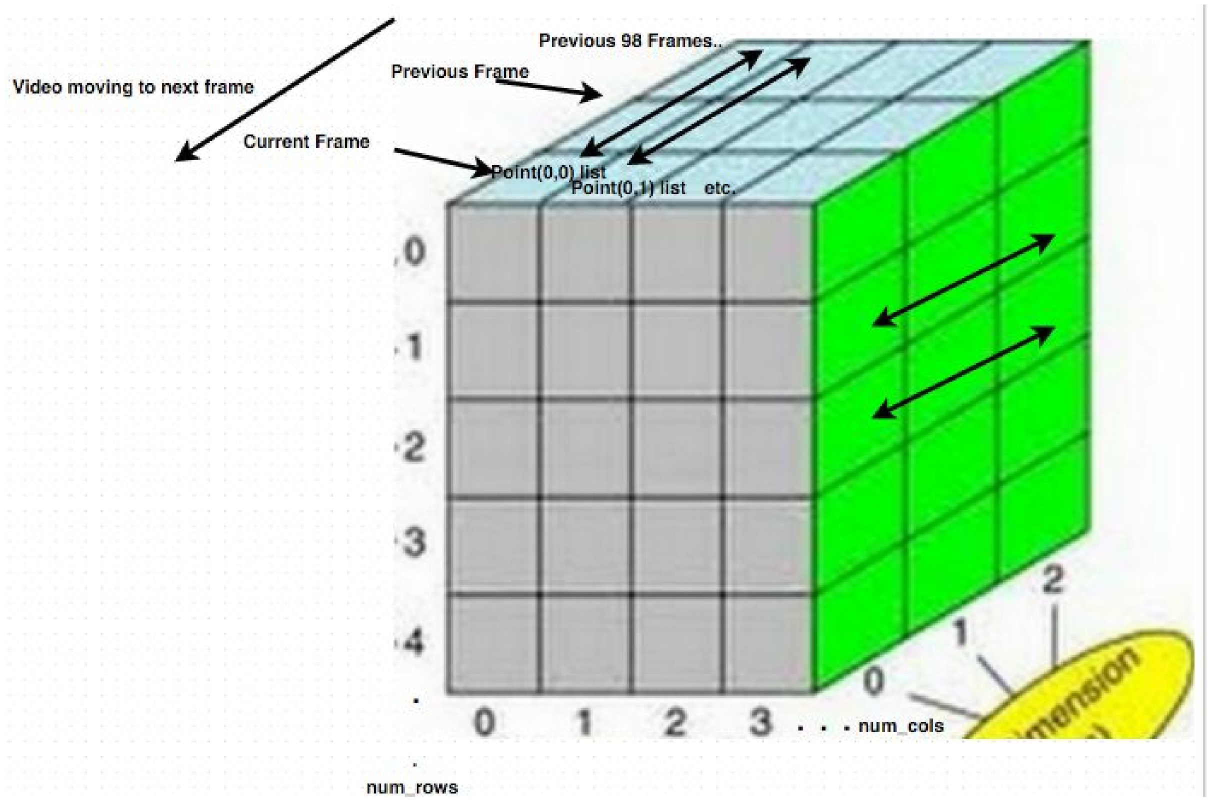

fig.1: visualisation of frames_points_list

- while the frame count is less than the number of frames to be used for the background and running average, the average is not calculated. When it becomes greater, the running total subtracts its oldest point (accessed via list), and the list erases this value from itself. this allows the average array to calculate the average point for each position in the video over the last 100 frames.
- the running_avg array at each position is then set with the next average point for that position.
- the first if statement following the while loop is executed when the background is not yet acquired, and is just used to put some background loading text on one of the output screens.
- the second if statement block is executed just once, directly after the background is acquired. It is used to set up the background; the background_diff at this point is simply the background.
- From here until the end of the video, differences between the background and the running average are stored in the abandoned matrix. This matrix holds the objects that have entered the scene. The thinking behind this is that the longer an object has been left abandoned in the scene, when compared with the background, the more intense the appearance of the object in the abandoned matrix will be.

Binary Threshold/Connected Components

After the abandoned matrix is acquired, I had to find a way to separate potential objects and their positions. For this, I decided to use contours, and I used the findContours function to get them. This function works better with a binary image, and so I binary thresholded the abandoned matrix. The value I set for the threshold was 50, high enough to filter most of the noise on the abandoned matrix.

At this point I had all of the contours for the current frame stored in the 'contours' list. Each contour draws a line around each identified object. I now had to come up with a way of identifying contours from frame to frame to see which ones have been abandoned or removed. I thought the easiest way to do this would be to create a struct for a contour. Each contour object would be identified by its centroid (most central point), which would act as a sort of 'primary key'. To calculate this point I used OpenCV's Moments. The other elements of the struct are a boolean to identify whether or not the contour was found during this frame, booleans to tell whether the contour is removed or abandoned, and integer values of how many frames the contour has been removed or abandoned for.

I call the function "set_found_this_frame_false()" which simply sets each contours' “foundThisFrame” boolean to false before this frames processing of the contours list. Each new contour found will be compared against the current list; if it finds itself in the current list (by searching for the same centroid within a margin of 1 pixel for each point), it updates it with any new data and increments “framesAbandonedFor”. if it doesn't find itself, it adds a new contour to the list, provided it doesn't have a null value for the centroid, which can happen in special cases, eg. the contour being just a single point.

Once this process is completed, the program checks the list of contours to see if any weren't present for the frame. to do this, it calls remove_old_contours. This function first checks for the FoundThisFrame value. if it is false, it then checks to see if the contour was present for the amount of time it would have to have been present for to be an abandoned object. If this is true, we know that the contour is a removed object, as it was both not found in the previous frame but was in the video for a suitable amount of time. I printed "object removed" for the FPS frames following an object removal found. after this point, the contour is removed for good.

Finally, we display all the abandoned contours by calling display_abandoned_contours. This function checks each contour to see whether it has been present for the last 4 seconds (4 by the FPS). if so, it displays "object abandoned" on-screen at the object's centroid.

Sample Images

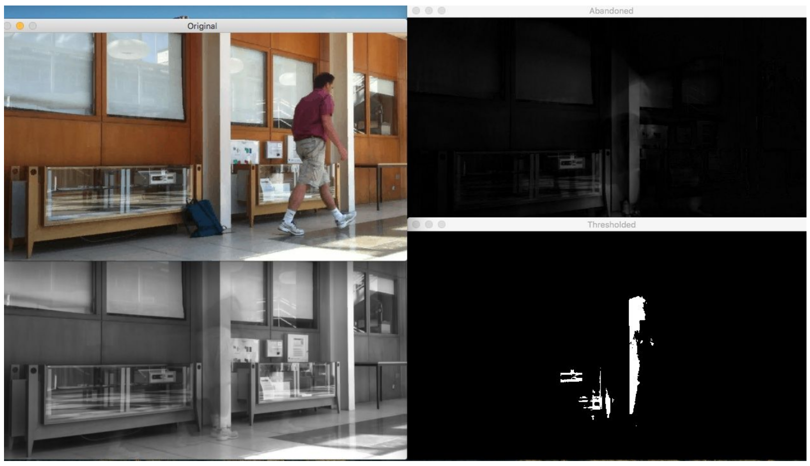

fig. 2

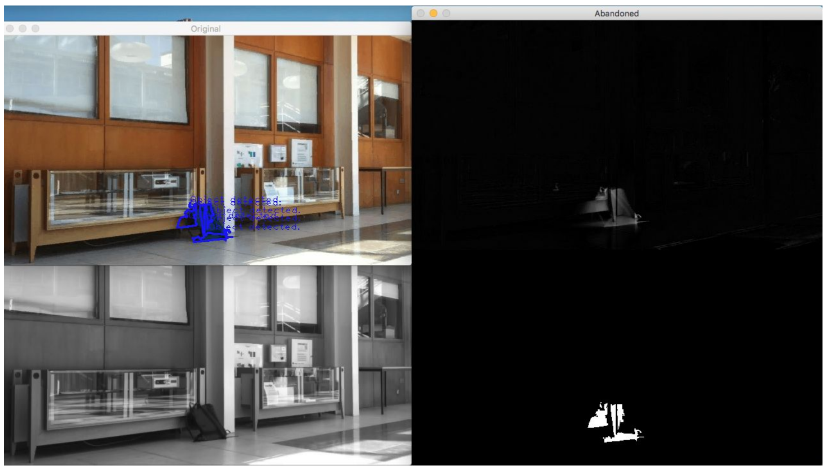

fig. 3: Object Abandonment Detection.

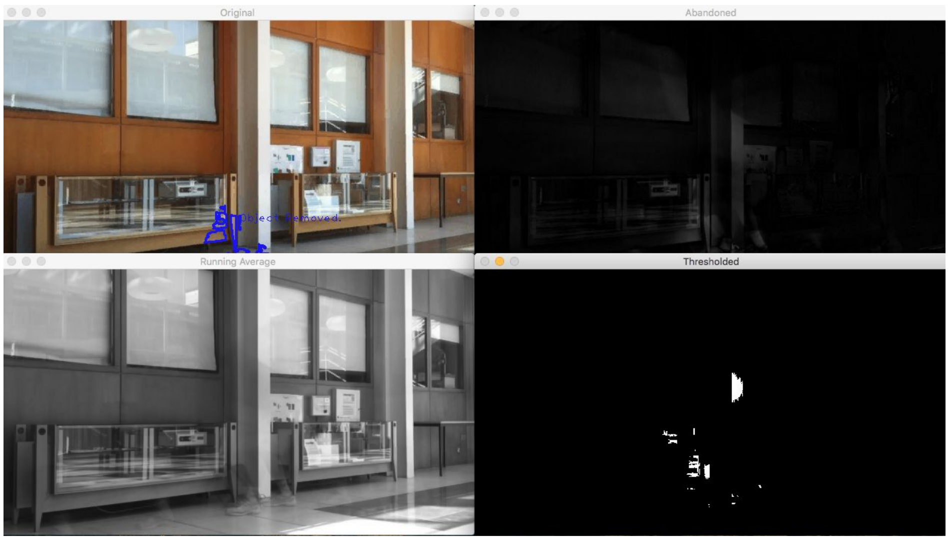

fig.4: Object Removal Detection.

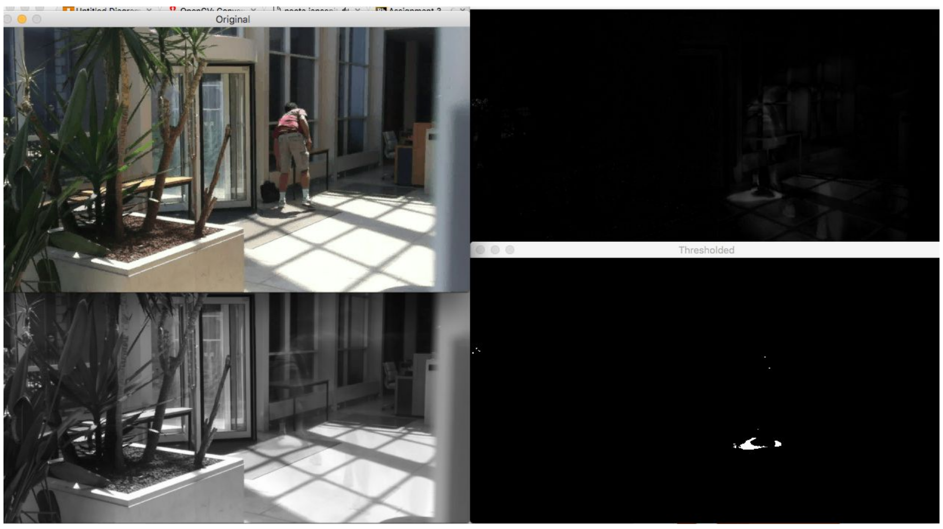

fig. 5

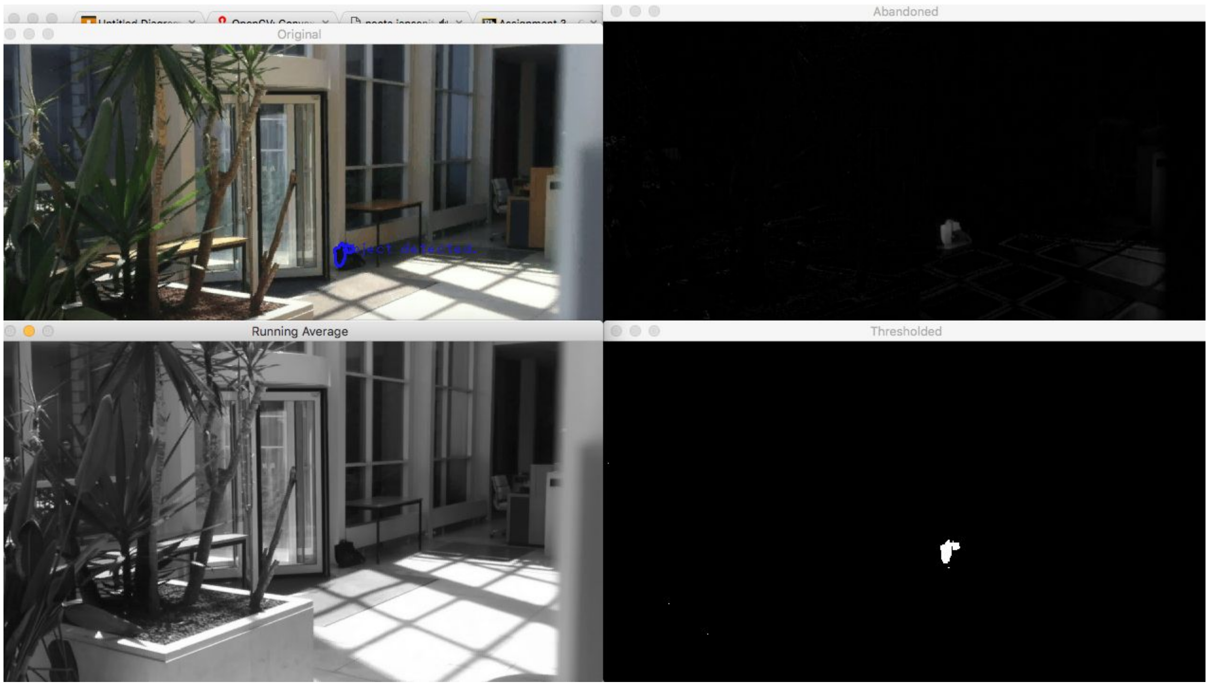

fig. 6: Object Abandonment Detection.

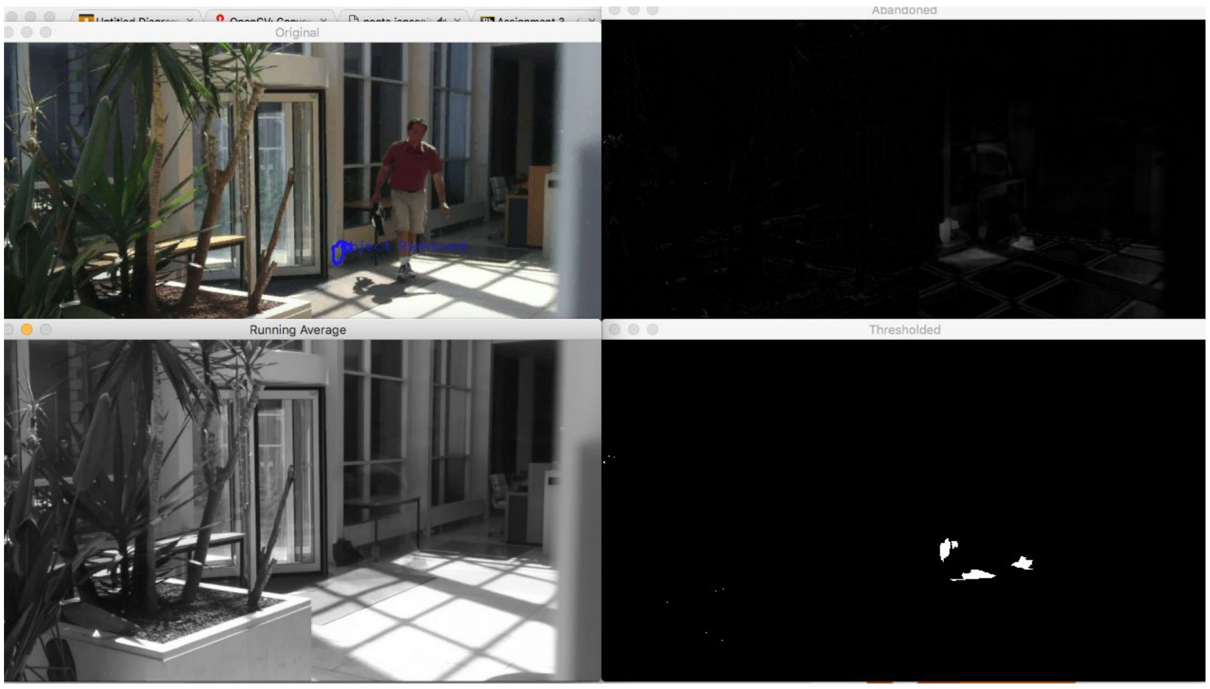

fig. 7: Object Removal Detection.

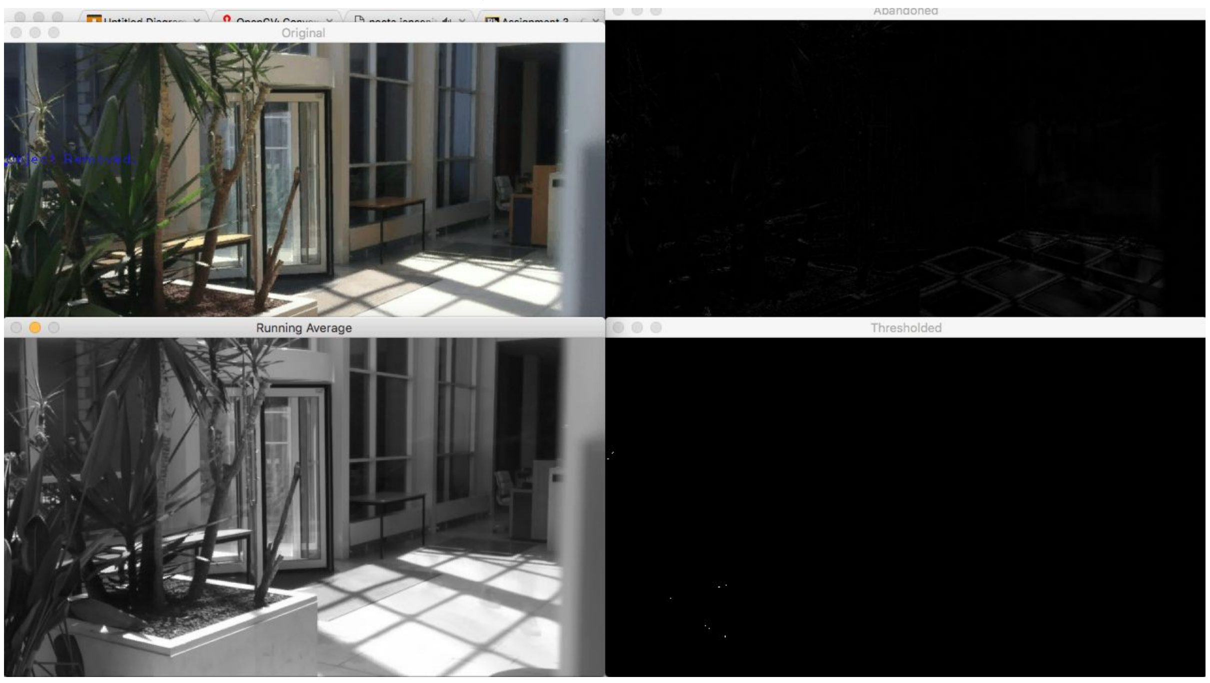

fig.8: False Positive Detection

Metrics

“For the two videos determine the number of False Positive events (i.e. the number of times an event which is not in the ground truth is detected) and the number of False Negative events (i.e. the number of times an event is not detected). From these (and the True Positives), determine the Precision and Recall.”

True Positives: 2
False Positives: 3
True Negatives: 2
False Negatives: 0
Precision = TP/TP+FP = 2/2+3 = .4.

Recall = TP/TP+FN = 2/2+0 = 1.

“For the four events determine the time (in seconds) it takes for the event to be detected (if it is detected). This is NOT the processing time - it is merely the number of frames divided by the frame rate.”

Video 1: Event Abandonment - Frame 511 - 511/25 = 20.44 seconds
Event Removal - Frame 626 - 626/25 = 25.04 seconds.

Video 2: Event Abandonment - Frame 340 - 340/25 = 13.6 seconds
Event Removal - Frame 579 - 579/25 = 23.16 seconds.

Observations

- When I tested my solution on the other downloadable videos provided, I realised that my solution was not setup to handle objects being part of the background initially and then leaving, ie. object removal happening first. I think this could be solved by using something like OpenCV’s **Inpaint.** This function interpolates points from the neighbourhood of a removed object in the scene and fills them in. In my case, I could perform this on an object detection, and if the points are similar to the original background, we know an object has been placed in the scene (abandonment), whereas if the background is different we know an object has been removed from the scene.
- In the second video, several contours are detected for the one object, instead of a single uniform contour. I didn’t have enough time, put my solution to this would have been to use a convex hull, check if the centroids of the contour are within a reasonable margin from each other, and if they are, draw a bounding box around all of these contours.
- The performance of my solution is relatively slow. I think this could be improved if I took a smaller section of the frame and only operated on that; although that would still leave out parts of the frame where objects could potentially be placed.