# Edge Flow

Gruffydd Morris, Plamen Angelov
Data Science Group, School of Computing and Communications
Lancaster University
Lancaster, LA1 4WA, UK
g.morris2@lancaster.ac.uk
p.angelov@lancaster.ac.uk

***Abstract*—Herein is a new data driven method to novelty detection and object definition in dynamic video streams that indiscriminately detects both static and moving objects in the scene. Using a modified version of recursive density estimation to reliably detect texture edges and a Sobel filtering process to extract gradient edges, detection of object textures can be done accurately and in real-time. In this paper we demonstrate the capabilities of the algorithm in video scenarios, and show that each object texture in the scene is reliably detected. We are able to show clearly the capability of the algorithm to be robust in occlusion scenarios; working in real-time, and defining clear objects where other techniques would attribute such small detections to noise.**

## I. Introduction

The world of computer vision and camera surveillance is growing at an alarming rate. The number of mobile video cameras on earth is set to outnumber the human population by the end of 2014. Analysing the video streams from the cameras has become a hotbed of research, with many approaches already existing for the static camera platform [1]. A video stream taken from a moving camera is much harder to analyse due to the lack of reference points, and the inability to directly compare consecutive frames (the scene has moved, therefore the content of the scene frame to frame is always changing). Currently, for dynamic / moving cameras there are two prominent solutions, motion estimation and optical flow. I wish you the best of success.

mds
January 11, 2007

### A. Motion Estimation

This is a near real-time approach and attempts to simplify the problem of no static reference points by warping and overlapping two or more consecutive frames together [2, 3]. This has the effect of creating an area of static imagery, much like the static / fixed camera platforms. Traditional background subtraction can then be applied to the stream and moving or dynamic objects can be extracted from the scene. The main drawbacks of this approach are that the warping and overlapping of frames is not perfect and therefore artificial noise is introduced into the scene, reducing the robustness of the overall technique. Also, as the precision of warping and overlapping is improved, the computational processing required increases making the approach less attractive. On a Windows based PC , the frames processing time is between 200 and 600 ms per frame, depending on the precision of the warping used.

### B. Optical Flow

Optical flow is a non-real-time approach that calculates the vectors of motion of each pixel between consecutive frames [4, 5]. The flow vectors indicate the relative motion of each novelty in the scene. The technique is good at detecting moving objects in the scene, although the processing time between each frame is between 2 - 3 seconds per frame on the same hardware mentioned earlier. The main reason for the large processing time is that it needs to calculate the optical flow of each individual pixel, and there are over 300,000 in a 640x480 frame. Furthermore, if the motion or change between scenes is sufficiently large, the optical flow accuracy is reduced. An approach to optical flow using stereo cameras to mitigate this problem has been developed, and this can facilitate a much more accurate detection of moving objects in a scene albeit at a cost of processing due to the calculation of disparity between cameras that is required.

Given the drawbacks of the above techniques, and that optimizing them can only go so far  brute force is the only answer to make both approaches truly real-time. Therefore a step change is required; a new way of thinking about the problem is needed. The basis of the proposed method originates from the way humans look at a scene and interpret their surroundings[6]. The detection of an object by people is done by contrasting textures. Looking out of a window you will see a myriad of various objects in the scene ranging from trees, to fields to buildings and cars. The human brain has evolved to detect sudden motion in a scene, and prioritizes such objects. This is part of the fight or flight instinct. In our modern world, many objects of interest and importance are static and unchanging for lengthy periods of time. An analyst looking at a video stream with a mix of the moving and static objects would find it difficult to focus solely on the important static object. The method presented here isolates and highlights all object textures within a scene, static or dynamic, and extracts key information about each individual objects. Later, the detected objects can be filtered depending on the type, size, motion and other characteristics of the object the analyst is looking for.

## II. Proposed Method

The method set out here is revolutionary and novel in its approach to the moving camera problem. The novelty that is presented here is that it detects all objects in a scene. All existing techniques assume that foreground objects of interest must be moving or changing in some way and can only detect such objects. This method allows both moving and static (unchanging) objects to be detected. This is a significant step forward, paving the way for detections of small minor objects as well as the large moving parts of a scene. Also, the method doesnt make any prior assumptions about the scene, and is a wholly data driven approach. The latter statement is critical, what other techniques dismiss as noise or unimportant, this technique extracts and highlights it as an object texture. The point being, information is lost if something is dismissed at the detection phase which could be filtered at a later stage. Key objects or people can easily disappear into the background if the detection algorithm dismisses small or noise-like novelties early on. This can later be filtered out based on the object parameters the analyst is looking for (type, size, motion, texture etc. of the detected object).

Figure 1 illustrates the components of the proposed method:

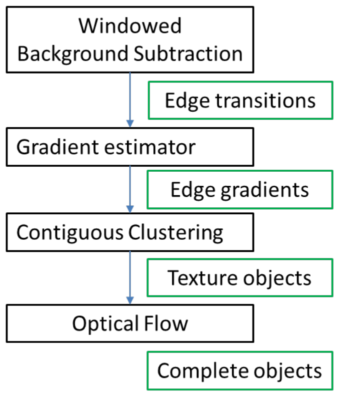


Fig. 1: Edge flow components, and in green, the output at each component stage

*1) Windowed Background Subtraction:* The first component of edge flow utilizes a novel approach to existing background subtraction algorithms by using a non-thresholded, windowed approach that yields grayscale edge definitions between n frames. In this case, we define window as n consecutive frames that are used in generating the background subtraction output. The background subtraction algorithm used in this work is a modified implementation of Recursive Density Estimation (RDE) [7]. The original equation for density calculation in this work can be seen in equation 1, with the recursive update formulas in 2 and 3. Finally the thresholding method which is usually used with RDE is shown in equations 4 and 5. The equations are obtained from [8].

(1)

(2)

(3)

(4)

(5)

In the Edge Flow implementation of RDE, the threshold calculation is removed from the algorithm. The purpose of this is that we now see density change detections across the entire scene, not just when the density reaches a threshold. The motivation behind this is that the density of a particular pixel changes over time depending on the movement of a texture object between consecutive frames. The leading edge of the texture object that is moving will yield a sharp density change  because the edge indicates a contrast change, and as this moves over a different texture object the distance in colour space between the moving texture and the texture object being overlapped is significant. In subsequent frames, the leading edge will have a sharp density change, because it is always at the point of significant change; the pixels where the leading edge was in the previous frame will have a reduced density change  but still important. This reduced density is due to the body of the moving object texture now being in-place of the leading edge. Thus over 3 frames the mean colour of the pixel is 1/3 original, 2/3 moving object texture. Over four or five frames there is a gradual density change up to the leading edge of the object texture (current frame), and then a sharp drop to the original texture not being overlapped. If the densities were allowed to build up over an infinite number of frames, eventually the entire scene will average out to a small range of densities  and given long enough to a single density across the scene. This is where the requirement for a windowed approach is needed, whereby the contribution to the density from n frames ago is removed. This ensures that edges within the scene are consistently represented with a gradient of densities  and a limited averaging of the densities (in a very cluttered scene with many object textures, it is , unavoidable to have some density overlap). The modifications to the standard RDE has the effect of this is that each edge that is moving

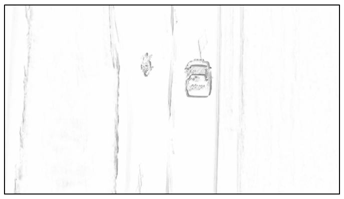


Fig. 2: RDE applied to moving camera to define object texture edges

relative to the camera platform will be detected. Thus in a moving camera scenario  all edges in the scene will be detected (apart from those which are precisely in synchronous movement with the camera). To avoid any edges being missed (ones that are synchronous with the camera movement), the camera should move in an asynchronous manner. In summary the two modifications to the original implementation of RDE:

1) 1. Removal of the threshold applied to densities. This yields the densities across the entire scene as opposed to just high density detections. This is important in order to show edge gradients, and thus the edge flow over consecutive frames.
2) 2. Windowed recursive density estimation over n frames. Equation 5 and 6 show the updated versions of the recursive functions (2 and 3) to accommodate the window function.

(6)

(7)

Figure 2 illustrates the effect of applying Windowed RDE with no threshold to a moving camera scenario.

*2) Gradient Estimator:* The second component of Edge Flow is to isolate the magnitudes of the gradients of each edge detected in the previous stage. The direction and relative speed can be associated with the edge gradients  a sharper gradient indicates a higher relative speed, with the sharper gradient being the leading edge of the object texture. To establish the gradients of each edge, a Sobel operator is applied to the resultant edges in both the x and y plane. Figure 3 shows the different Sobel operators applied to the frames. The size of the Sobel operator (illustrated is a 3x3) determines the area of surrounding pixels that influence the gradient value of the current pixel. A small Sobel operator only looks at the immediate surrounding area of a pixel, thus the output is more sensitive to large local changes in pixel value. A large Sobel operator (say 7x7), on the other hand, smooths the influence of large local changes and takes into account the wider area

| | | |
|---|---|---|
| -1 | 0 | +1 |
| -2 | 0 | +2 |
| -1 | 0 | +1 |

x filter

| | | |
|---|---|---|
| +1 | +2 | +1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

y filter

Fig. 3: X and Y Sobel filters

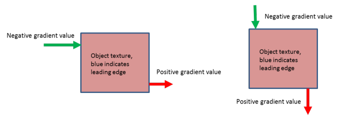


Fig. 4: Left, gradient values assigned for and x-plane Sobel filter. Right, gradient values assigned based on a y-plane Sobel filter

around the pixel. Given that we are interested in detecting the large local changes a small Sobel operator (3x3 or 5x5) is appropriate for the edge gradient detection.

The Sobel filters are applied separately to the frame in question, and each filter yields a separate output. The x filter applied to the frame applies a gradient value to the pixel based on changes in the x-plane; changes in pixel value going horizontally across the image. If there is a positive gradient, this indicates a gradient going from low values to high values, and similarly a negative value indicates a gradient going from high values to low values. In the case of this application, because non-edges of object textures (background) are defined as white and edges of object detections are a gray value between white and black, positive gradients indicate transition from a leading edge of an object texture and negative gradient values indicate transition towards a leading edge of an object texture. The y filter does the same except it applies the gradient value based on changes in the y-plane; changes in the pixel value going vertically across the image. Figure 4 illustrates how the gradient values are assigned.

The Sobel filters, in both cases, are applied from top-left to bottom right. Thus the first encountered edge of a texture object will always be negative, and the final edge of the texture object will be positive. The absolute value of the Sobel filter output indicates the magnitude of the gradient at the pixel  a large value (positive or negative) indicates a large change  which typically indicates the leading edge of a texture object. A smaller value typically indicates the historical path of the leading edge of the texture object. The approach is novel through combining the first two components, and the processing speed surpasses any available algorithm for novelty detection in a moving camera scene. The processing of a 640x480 video frame is done at a real-time speed of 40 frames per second (25ms per frame) on a Windows based PC1.

*3) Contiguous Clustering:* At this stage we have two outputs from our initial frame input  the x-plane and y-plane Sobel filter gradients for each of the edges in the scene. In order to revert back to a single output, and extract texture objects

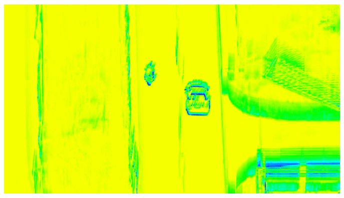


Fig. 5: Result of a Sobel filter in the y-plane applied to RDE image. The gradient values have been coloured for visual effect blue indicates large gradient changes, whilst green is a smaller gradient value. Yellow indicates areas of no edge gradients (and therefore no edges  the texture of an object).

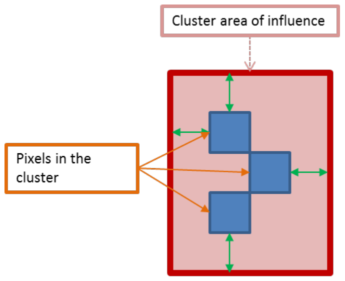


Fig. 6: Pixels in a cluster, and the clusters area of influence.

from the scene, a third component of Edge Flow needs to be applied. This stage applies a custom form of clustering, dubbed Contiguous Clustering; we group each contiguous edge defined by a gradient together. This form of clustering is robust and combined with the first two components is resistant to occlusion. Should an object texture be occluded partially by another object texture, they will remain separate clusters unless the object is completely occluded. Further, once the occluding object has moved on, the cluster will reform the same as before defining the texture object. The approach is made faster by not needing to process every pixel within the frame  only those that have a gradient assigned to them (the body of a texture object wont have a gradient  its not an edge). The process for this clustering method is as follows:

1) Working from the top left of each image (x-plane and y-plane Sobel images) find the first pixel with a non-zero gradient value.
2) Create a new cluster on this pixel, and give the cluster an area of influence of 1 pixel either side of the pixel. The area of influence is the region the cluster considers contiguous for new candidate pixels. The area of influence also applies to the other Sobel image i.e. even if in the pixel is not a detection in the other Sobel image, assess the surrounding area in the same way  this will permit the combination of both images into one set of clusters on the original frame image.
3) Assess the surrounding pixels within the area of influence of the pixel (except for image edges) for pixels (or clusters where this pixel overlaps an existing cluster area of influence) within the same gradient range. The gradient range is a pre-defined parameter on initialization (currently not autonomously defined).
    a) If this pixel is within the gradient range of a cluster, and it is within its area of influence, add to a cluster.
    b) Otherwise, if not the first pixel being assessed, create a new cluster, and if a neighbouring pixel is within the gradient range, and is contiguous (within the area of influence), add to the newly created cluster.
    c) If there are no neighbouring pixels within its gradient range, dont remove the cluster, and leave as a singleton. It is either a very small texture object (a mole hill in a field for example), or it will be absorbed by another cluster as its area of influence expands.
4) Adjust the area of influence of the clusters that were affected by 3. See Figure 6 for an illustration.
    a) The minimum x and y influence is the lowest pixel coordinates that is a member of the cluster, minus one in both directions.
    b) The maximum x and y influence is the highest pixel coordinates that is a member of the cluster, plus one in both directions.
    c) Update the mean gradient value of the cluster  this will be used for assessing the proximity of new pixels to the gradient value of each cluster.
5) Flag any pixels that were assigned to a cluster to avoid re-clustering these pixels.
6) Find next pixel with a non-zero gradient and repeat steps 3 to 5 until all non-zero gradient pixels have been assigned  from both images. It is not important which Sobel image is processed first (as there will be crossovers from the images anyway).

Currently the approach is parametric, requiring a magnitude range which defines the similarity of candidate pixel gradients required to be clustered together. There is scope to autonomously define a gradient magnitude range in future improvement works. Through this method each similar and proximate edge are clustered together, resulting in a contiguous object being defined for each different texture (object with edges); an object is defined as an area of similar texture, not as an isolated object in the truest sense. For example, a car may be defined as 3 separate objects in edge flow  the bonnet which is of a particular texture, the roof which is a different texture, and the boot which is the same texture as the bonnet but separated by the roof. The main novelties of this approach are it works well with partially occluded object textures and keeps them separate until completely occluded, it rediscovers the object textures post-occlusion, static and moving object textures are clearly separable, and the processing speed combined with the first two components of edge flow remains real time; between 25  40 frames per second depending on the number of object textures discovered in a scene. This is significantly faster than other routines, and still permits

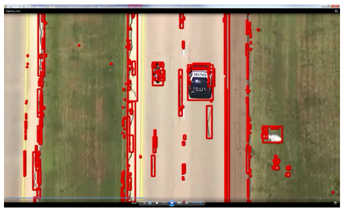
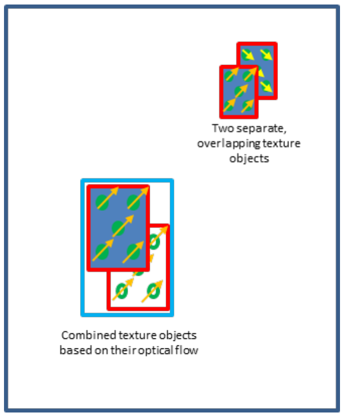


Fig. 7: Clustered contiguous gradients

some head room for additional processing. An example of the occlusion discrimination capabilities; if a car drives over a crack in the road (both of which have been clustered and identified) the clusters will remain entirely separate unless the car completely occludes the road crack. Once the crack appears the other side of the car it is immediately re-discovered and clustered as a separate object texture. Figure 7 shows the outputted clusters from a car and bike video scene, clearly showing static and moving object textures being detected with excellent discrimination between object textures

*4) Optical Flow:* The component that is optical flow is an additional process to extract further information about the texture objects that were detected. This allows further detail about the object that they belong to. The optical flow algorithm is not applied to the entire image because this is computationally resource heavy and defeats the purpose of Edge Flow. Five pixels are selected within each cluster, one from the centre and four from the edges of the cluster. Optical flow [4] is then applied to these individual points which yields a flow vector for each of the individual points within a texture object. By taking the mean flow vector from each texture object we can determine the similarity of movement between texture objects. If the flow vectors are similar, and the texture objects overlap in x-y proximity in both frames (two frames required for optical flow), the texture objects can be considered are actually belonging to the same object. As shown in Figure 8, the concept holds true in the case of texture object occlusion as discussed in the clustering component. The case of the car and the crack in the road  the car texture objects will have a different motion vector to the crack in the road, so despite visually occluding the crack somewhat, they can be considered as wholly separate objects and will not merge as the same object (unless one is completely occluded).

Optical flow is not essential to the detection technique, the texture objects are detected prior to the application of optical flow. It is used as a method to define contiguous texture objects and form the real object. Optical flow is made tractable compared to the sole use of it in video processing by limiting the points tracked to five per texture object, usually resulting in one or two thousand points in total. This is significantly reduced from over 300,000 pixels required if optical flow was applied directly to the source image. The novelty of this component is that a motion vector can be extracted from each object within a scene in real-time. Furthermore, objects which are moving in separate directions can be clearly seperated despite any occlusion in the scene. The motion vector is a


Two separate, overlapping texture objects
Combined texture objects based on their optical flow
Fig. 8: Two separate scenarios for texture objects with optical flow calculated for each of the 5 pixels within them.

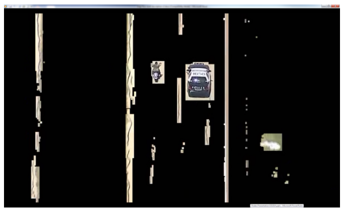


Fig. 9: Result of optical flow applied to clusters

representation of the relative velocity of an object compared to the camera platform; later, given the platform velocity, this can be used to determine the absolute velocity of all the objects within a scene. With the inclusion of optical flow in the method, the average processing time remains around 20 frames per second (50ms per frame) for a 640x480 video stream. As with the clustering technique the processing time changes slightly dependent on how many objects are detected.

## III. Experimental Results

A series of experiments were conducted on the proposed algorithm to test its performance. The tests are performed in this paper are split into sections due to the novel capabilities of the proposed approach; the objective is to demonstrate the capabilities with reference to other techniques. The following performance characteristics are tested in individual experiments:

- Dynamic and static object detection in a moving scene
- Occlusion Resistance
- Frame processing rate
- False detection tolerance and filtration

The experiments use a selection of videos to demonstrate the capabilities across different scenarios  testing differing camera motions, object motions, and object dimensions. The video descriptions and parameters:

1) Video 1: Helicopter video sequence featuring a police chase with two moving objects, a car and motorcycle. The scene is simple open terrain with roads, verges, a bridge and occasional road side objects. The purpose is

to represent one of the simplest scenarios with a low density of objects to clearly show important detections. Resolution: 640 x 360. Pixel count: 230,400 Frame rate: 29 frames per second
2) Video 2: Dashboard mounted camera which has multiple moving objects that are occasionally occluded. Resolution: 848 x 480. Pixel count: 407,040 Frame rate: 30 frame per second
3) Video 3: High density video of a drone launch with multiple textures and moving objects. Resolution: 1920 x 1020. Pixel count: 2,073,600 Frame rate: 30 frames per second

The test setup uses the Windows based PC1 and the algorithm is run as a release version from Microsoft Visual Studio 2010. Set parameters of edge flow were used for the testing of experiments A and B. Experiment C is such that the parameters are adjusted to provide different filtration parameters.

- Gradient range that is considered too minor for an edge (filtering minor oscillations in the texture gradient): +/- 20.
- Gradient range required to match a contiguous edge: +/- 20.
- RDE frame window size: 3.

### A. *Video 1 - Helicopter, translational motion*

This experiment explores the capability to detect a variety of objects in a simple, uncluttered scene, both static and dynamic. The objective is to contrast the results of edge flow on the same scene as motion estimation, and explore the differences. The scene used in this experiment is shown in Figure 10. This scene is specifically used because not only is there two moving objects there are the occasional stationary object on the verge (white objects), some road markings, and some damage to said road.

In Figure 11, the motion estimation result clearly identifies two objects  the motorbike and car which are the two moving objects within this scene. Edge flow however detects a significant number more objects, namely the static white objects on the right of the road, the road defects, road markings and the road verge. The empirical results for this run are shown in

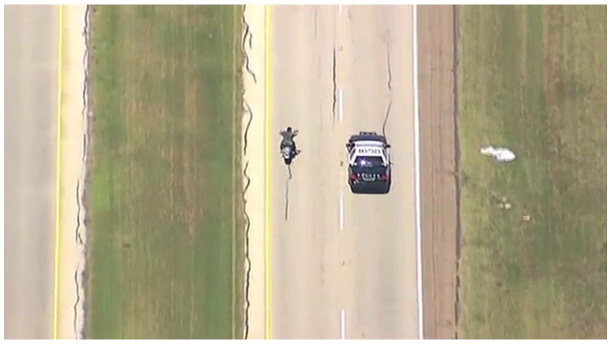


Fig. 10: Scene from video 1 to demonstrate performance of edge flow

## IV. Conclusion

The conclusion goes here.

## Acknowledgment

The authors would like to thank...

## References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

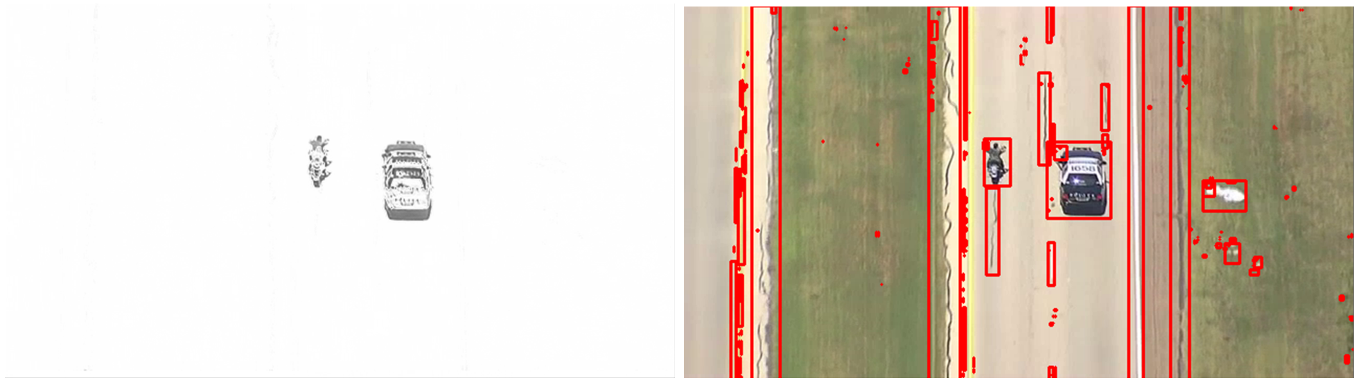


Fig. 11: Motion estimation result from scene (left) Edge flow result from scene (right). Red boxes are included on edge flow to highlight detections more clearly