SALIENT OBJECT DETECTION IN A FRAME

SUPERPIXEL BASED SPATIOTEMPORAL SALIENT OBJECT DETECTION

Independent Study Report

ECEN 5840-906

Team Members

Prof. Sam Siewert

Matthew Vis

Ramnarayan Krishnamurthy

Surjith Bhagavath Singh

Shivasankar G

Akshay Singh

Swaminath Badrinath

Written By:

Shivasankar Gunasekaran

Index

Introduction

The objective of this Independent Study was to be able to identify the most salient or interesting object in a frame. There are multiple methods that have been proposed to identify the salient object in an image. But since this research group is aimed at using a live video feed as the input, I focused on salient object detection in a video stream. While researching on this topic, I found the paper on Superpixel Based Spatiotemporal Saliency Detection interesting and decided to try to implement the algorithm defined in this paper.

I conducted a performance analysis on different sections of the code. I also performed an analysis on different algorithms that can be used to implement various section of the code to identify the best algorithm that is suitable for our research. I have also implemented the temporal salient object detection in this paper.

The main objective of this was to aid Prof. Sam Siewert in his research on developing a Software Defined Multi-Spectral Imager.

Modifications to the Initial Proposal

In the initial proposal I had, stated that I would implement salient object detection algorithm on a GPU. This was modified during the course of the independent study as per the needs of the research group. So I have been trying to identify the best suited salient object detection algorithm for the research group. So I conducted all the performance analysis on my laptop, which has a Native Ubuntu environment.

Superpixel Based Spatiotemporal Saliency Detection

The idea of this paper is to identify the object with temporal saliency and spatial saliency and then combine the two using merging techniques to get the final salient object. Rather than just using frame level object detection, this paper suggests the use of Superpixel for more accurate salient object detection. A Superpixel is a cluster of pixels which are clustered together on the basis of brightness and intensity. Hence an image can be segmented in many groups of pixels so that it would be easier and meaningful to apply various algorithms to get desired output. An example is shown below:

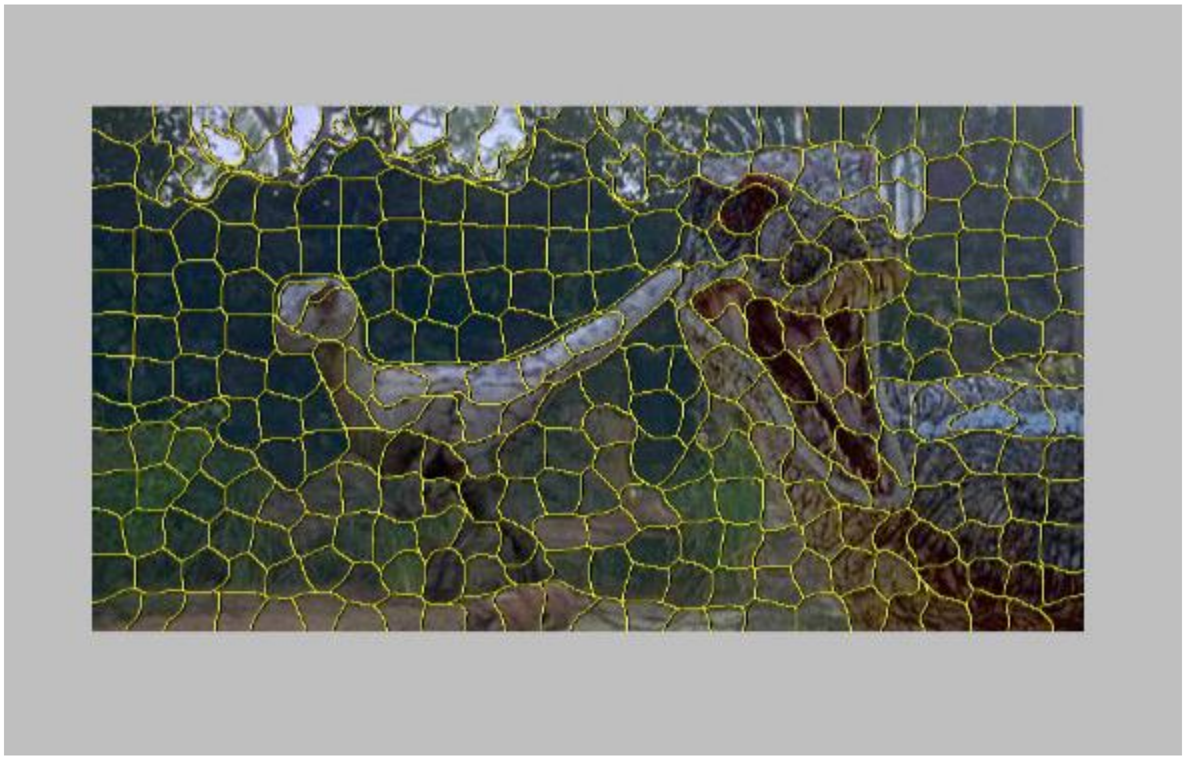


Fig 1: Image Segmented into 300 Superpixels

As it can be seen, superpixels gives us a more useful information of the different parts of the image unlike each and every single pixel.

Superpixel Segmentation:

An image can be segmented into Superpixels using various methods. In this paper they suggest to use Simple Linear Iterative Clustering [SLIC] method. I found a website which did a study on the different Superpixel segmentation methods.

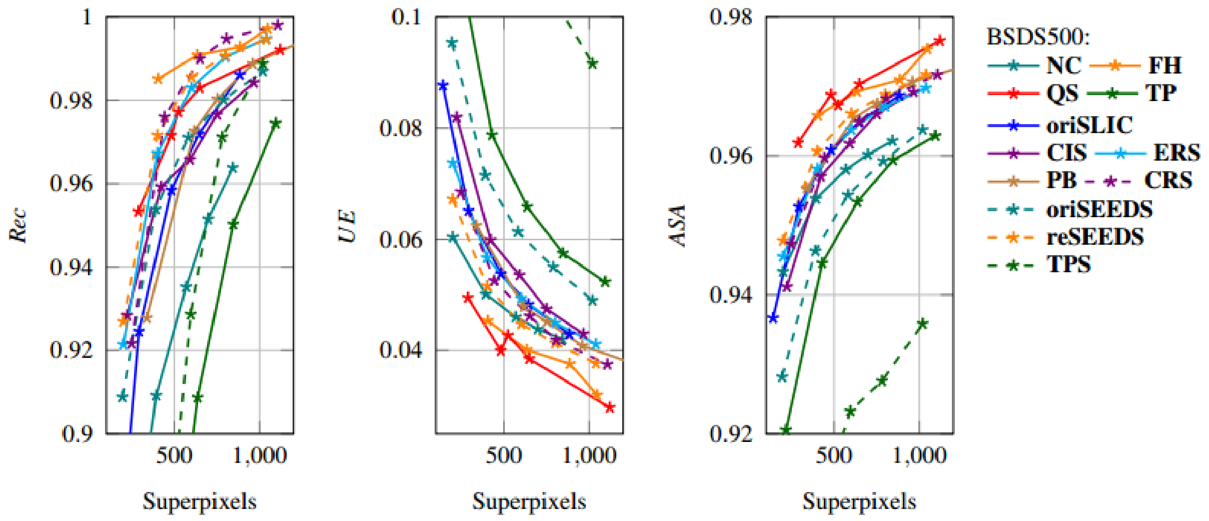


Fig 2: Comparison of several superpixel algorithms; Rec denotes the Boundary Recall, UE denotes the Undersegmentation Error and ASA denotes the Achievable Segmentation Accuracy

After reading this, I found that SEEDS – Superpixels Extracted via Energy Driven Sampling algorithm and SLIC algorithms are have better performance [time and space complexity] and widely used. So I did my own study on this, by comparing the execution time of each of the algorithms.

The following are the images obtained after Superpixel segmentation:

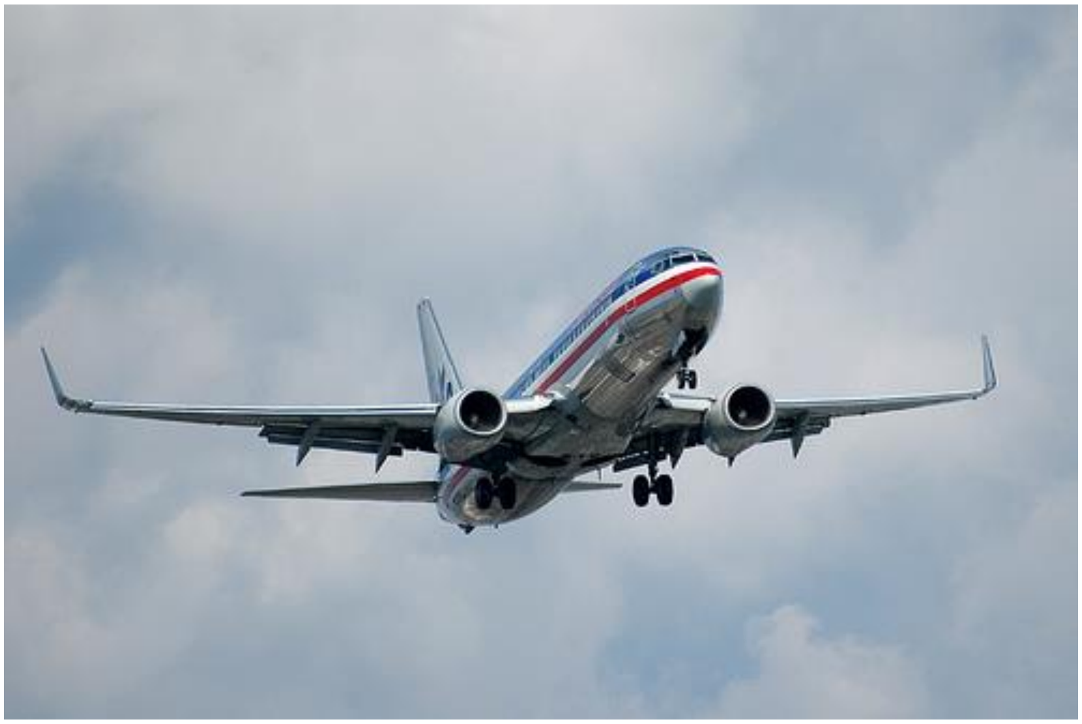


Fig 3 Original Image of Aircraft

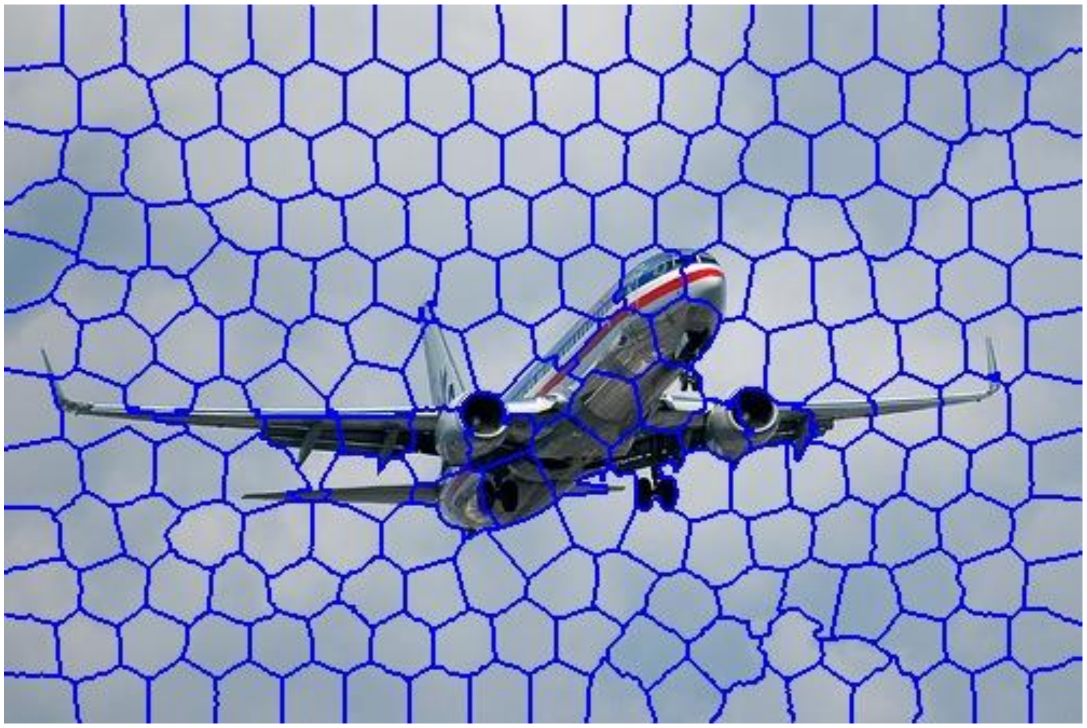


Fig 4 Image after SLIC Segmentation

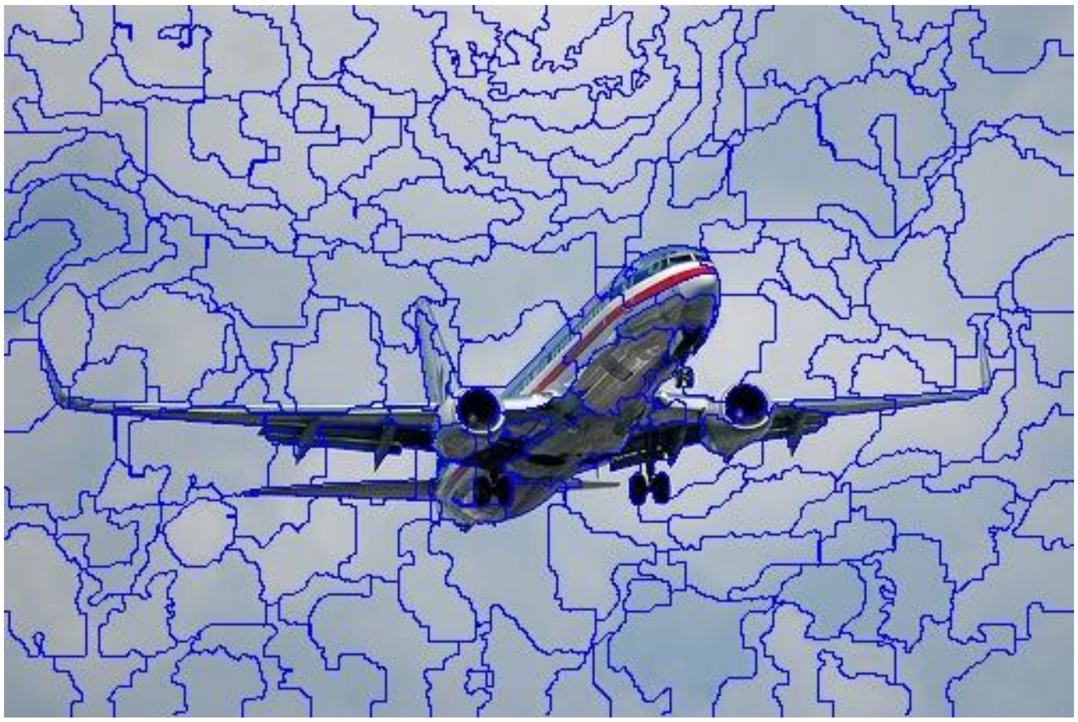


Fig 5 SEEDS Segmentation

I obtained the following results with the number of superpixels used to segment the image as the variable of this small analysis.

Number of Superpixels Segmented	SLIC Segmentation (in ms)	SEEDS Segmentation (in ms)
100	700	300
200	780	500
300	860	680
500	900	1200

From this table it can be understood that execution time of SEEDS Segmentation algorithms increases as Number of Superpixels increases. But the SLIC Segmentation algorithm, takes almost the same time even as the number of superpixels increases. The image used was a 320x240 resolution image.

To segment images into Superpixels using SLIC, I used two files, a "slic.cpp" file and "slic.h" file. Hence to generate superpixels only one function needed to be called, which is described below:

First an object of class SLIC is created. Using this object we generate superpixels. This object has 3 public member functions as described below.

cv::Mat GenerateSuperpixels(cv::Mat& img, uint numSuperpixels) : This function is used to generate superpixels.
img : input image or frame.
numSuperpixels : the number of superpixels the input image should be segmented into.

int* GetLabel() : This function returns an array, which contains the superpixel number to which each pixel belongs to.

cv::Mat GetImgWithContours(cv::Scalar color) : This function draws the superpixels on the image to show how the image is segmented.
color : the color of the contour to show the superpixels.

Superpixel segmentation is done by first converting the image to LAB color space. LAB color space is where the image is divided into three channels where each of them are L: Lightness, A and B : are color opponent dimensions. Once the image is converted to LAB space, then kmeans segmentation is done on the entire image. It picks the maximum value of the color distance as a compact factor M and maximum pixel distance as grid step size S from each cluster.

As the paper suggests I have taken the number of superpixels as 200. On applying superpixel segmentation, I was able to obtain 194 superpixels.

Temporal Saliency:

Optical Flow:

To detect the object using temporal saliency we compare two adjacent frames. First the image is segmented into superpixels. Then the motion vector field for each pixel is determined by using optical flow estimation methods. There are two famous optical flow estimation methods, one is Lucas Kanade algorithm and the other is Farneback algorithm. I tried both the algorithms and did a small analysis on the performance of each of the algorithms.

	Farneback	Lucas Kanade
Time for execution	35 ms	25 ms

In both the algorithms, two adjacent frames are taken and then flow of each pixel is determined. The farneback algorithm gives the result in a desired fashion, i.e. as a vector, so this method was chosen. The output of the farneback algorithm is the motion vector field of each pixel in Cartesian coordinate system. OpenCV provides a function for calculation of the optical flow using Farneback algorithm. The function call is explained below:

void **calcOpticalFlowFarneback**(InputArray **prev**, InputArray **next**, InputOutputArray **flow**, double **pyr_scale**, int **levels**, int **winsize**, int **iterations**, int **poly_n**, double **poly_sigma**, int **flags**)

prev : is the previous frame captured, which is to be compared with the new frame to find the motion vector field of each pixel.

next : current frame captured which is used to compare with the previous frame.

flow : this contains the motion vector field of each pixel

pyr_scale : parameter, specifying the image scale (<1) to build pyramids for each image; pyr_scale=0.5 means a classical pyramid, where each next layer is twice smaller than the previous one

levels : number of pyramid layers including the initial image; levels=1 means that no extra layers are created and only the original images are used

winsize : averaging window size; larger values increase the algorithm robustness to image noise and give more chances for fast motion detection, but yield more blurred motion field

iterations : number of iterations the algorithm does at each pyramid level

poly_n : size of the pixel neighborhood used to find polynomial expansion in each pixel; larger values mean that the image will be approximated with smoother surfaces, yielding more robust algorithm and more blurred motion field, typically poly_n =5 or 7

poly_sigma : standard deviation of the Gaussian that is used to smooth derivatives used as a basis for the polynomial expansion; for poly_n=5, you can set poly_sigma=1.1, for poly_n=7, a good value would be poly_sigma=1.5

flags : this parameter can be either
OPTFLOW_USE_INITIAL_FLOW uses the input flow as an initial flow approximation
OPTFLOW_FARNEBACK_GAUSSIAN uses the Gaussian winsizeXwinsize filter instead of a box filter of the same size for optical flow estimation; usually, this option gives z more accurate flow than with a box filter, at the

cost of lower speed; normally, winsize for a Gaussian window should be set to a larger value to achieve the same level of robustness

Histogram Calculation:

I applied this optical flow estimation on my incoming frames from the live video feed. I was able to obtain the output flow vector. The next step is to build a histogram with 10 bins. The flow matrix gives the output in Cartesian coordinate system. I converted this into polar coordinates so that they can be divided into bins based on the magnitude and angle. The following function can be used to convert the matrix from Cartesian to polar coordinate system.

```
void cartToPolar(InputArray x, InputArray y, OutputArray magnitude,
OutputArray angle, bool angleInDegrees=false)
```

I developed my own code for histogram binning which takes the flow vector as the input and divides it into 10 bins. To divide into equal bins, the maximum value possible is needed. I found out that the maximum value of the magnitude of the motion vector field doesn't exceed 100. So I selected 100 as my maximum value.

Once the image is divided into bins it is to be normalized such that sum of all the motion histogram bins is one. Then the quantized motion vector field is to be determined. Quantized motion vector field is obtained by finding the mean of all the values of the motion vectors that are present in a bin. Hence each frame would contain 'n' quantized motion vector field where 'n' is the number of bins.

Motion Distinctiveness:

Until now I calculated the values that correspond to frame level. Next step is to develop the same for superpixel level. The pixels in each superpixel can be identified as suggested earlier [using GetLabel function]. Each superpixel is contains a cluster of individual pixels. These pixels are the divided into bins. So each superpixel is divided into 10 bins. Then the motion distinctiveness is calculated as the formula below:

$$\mathbf{S_{MD}}(\mathrm{sp}_t^i) = \sum_{j=1}^{q_M} \left[\mathbf{MH}_t^i(j) \sum_{k=1}^{q_M} \|\mathbf{qmv}_t(j) - \mathbf{qmv}_t(k)\|_2 \cdot \mathbf{MH}_t^0(k) \right].$$

where :

MH^i_t (j) represents the number of elements in the jth bin of the ith superpixel of the frame taken at time t.

MH^0_t (k) represents the number of elements in the kth bin of the frame level histogram at time t.

qmv_t (j) represents the value of the quantized motion vector of the jth bin at time t.

So the norm of the two quantized motion vectors is identified and then each of them is multiplied by the number of occurrences of the bin. In simple words, the motion distinctiveness is calculated as the sum of distances between the two different quantized motion vectors weighted by their occurrence probability. The frame level motion significance is defined as:

$$\mathrm{MS}_t = \sum_{i=1}^{n_t} \mathbf{S_{MD}}(\mathrm{sp}_t^i) \cdot |\mathrm{sp}_t^i|$$

where $|sp^i_t|$ is the number of pixels in the ith superpixel.

I obtained the following output after applying frame level Motion distinctiveness.

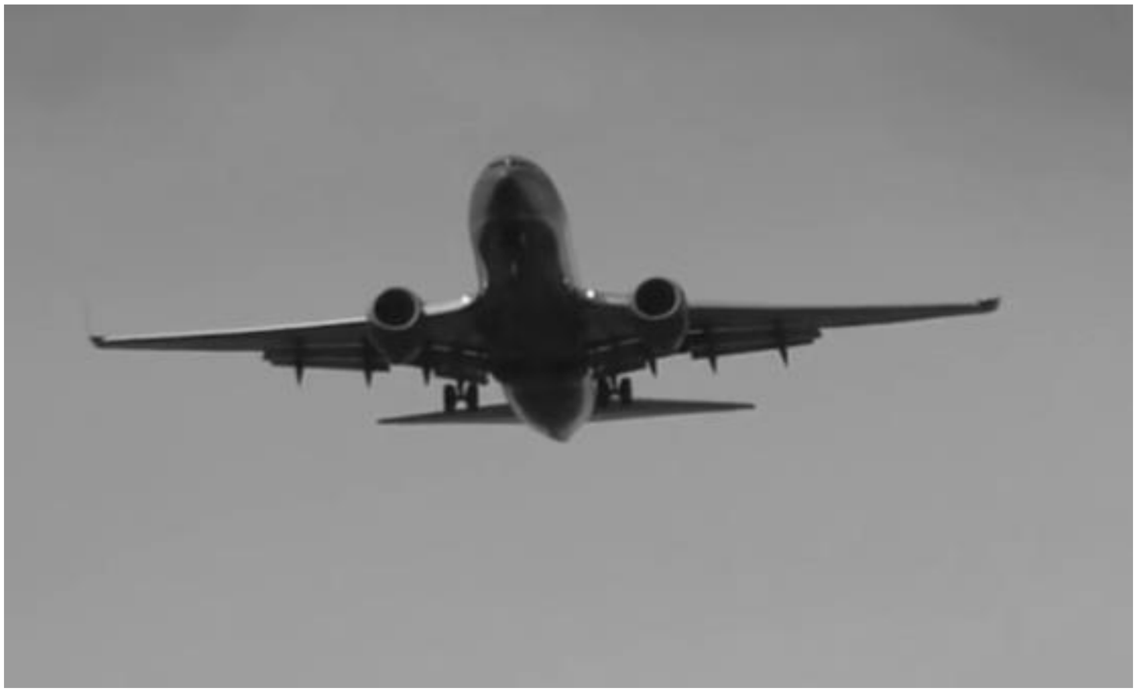


Fig 6 Input image

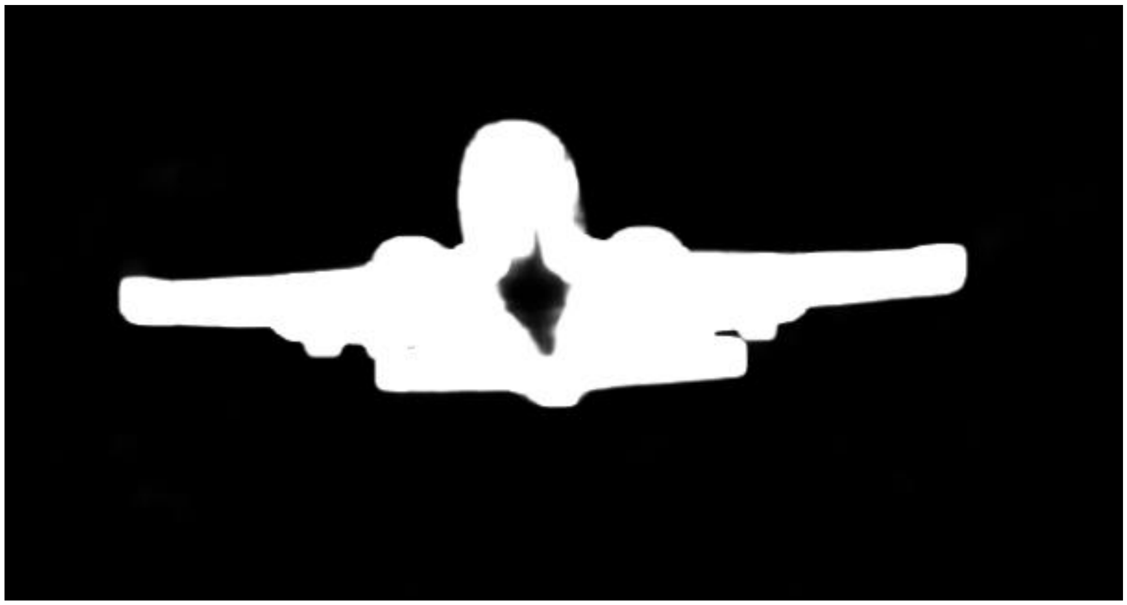


Fig 7 Frame Level Motion Distinctiveness

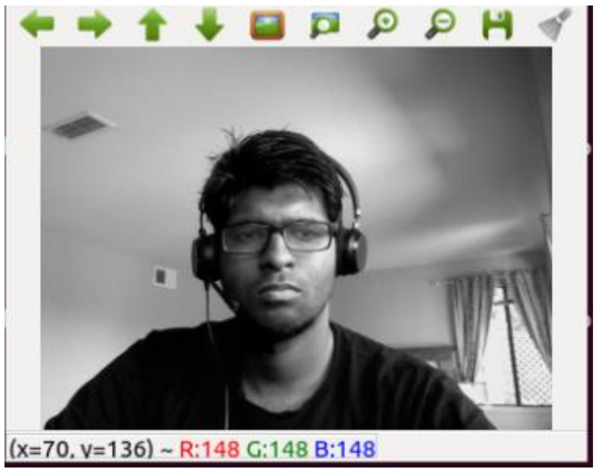


Fig 8 Input Image

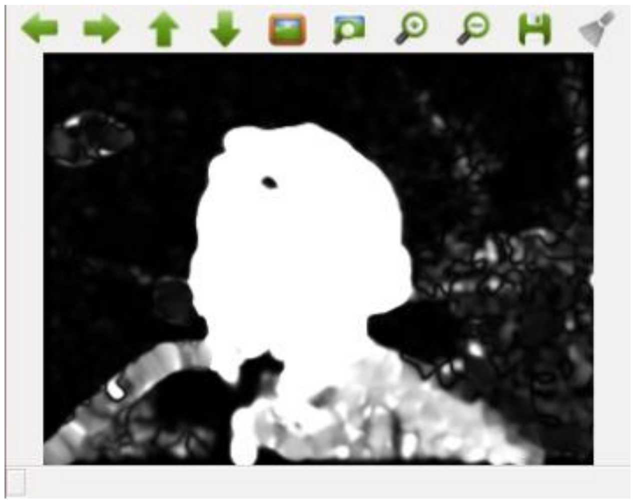
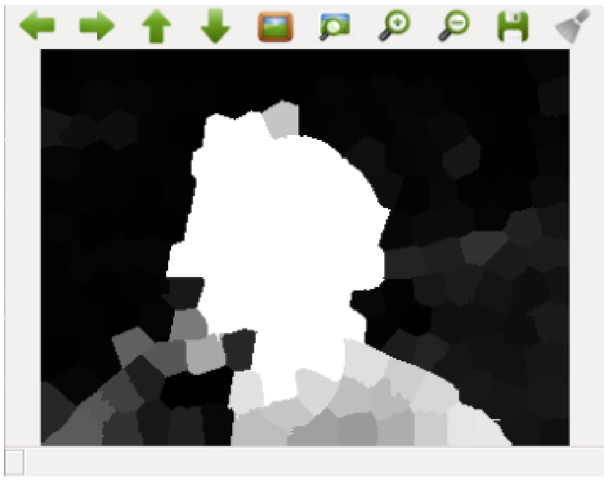


Fig 9 Frame Level



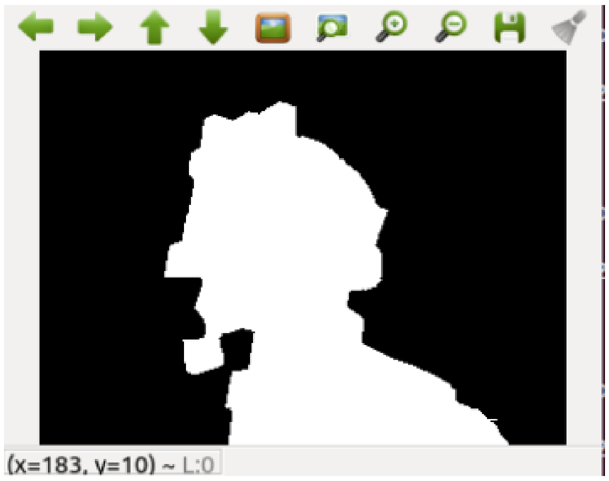


Fig 10 SuperPixel Level

Fig 11 Superpixel level, after applying thresholding

Steps to Reproduce the Output:

1) Make sure that the slic.cpp and slic.h files are present to perform Superpixel segmentation.
2) Create an object of class SLIC.
3) For determining the optical flow, two adjacent frames are needed. So after reading the first frame, store it into a variable and read the next frame.
4) Segment the frame into superpixels using the SLIC object and calling its member function **GenerateSuperpixels**
5) Once there are two frames, the optical flow estimation can be applied using the previous frame and current frame. This is done by invoking the **calcOpticalFlowFarneback** function.
6) Next step is to calculate the histogram of the image. So call the function **frameHist** with the flow output as a parameter to the function
    a. The flow matrix would be in Cartesian coordinate system. Split this into two separate matrices by using the **split** function provided by OpenCV.
    b. Use the two separated matrices [x axis and y axis values] as input to the **cartToPolar** function to convert to Polar coordinates.
    c. After converting to polar coordinates, the image can be divided into bins. The number of bins is hard coded to be 10 [numBins = 10] because the paper suggests 10 bins.
    d. After dividing the image into different bins, then the quantized motion vector field is determined for each bin. To find the mean easily, the polar coordinates is converted back to Cartesian coordinates using **polarToCart** and the mean is determined.

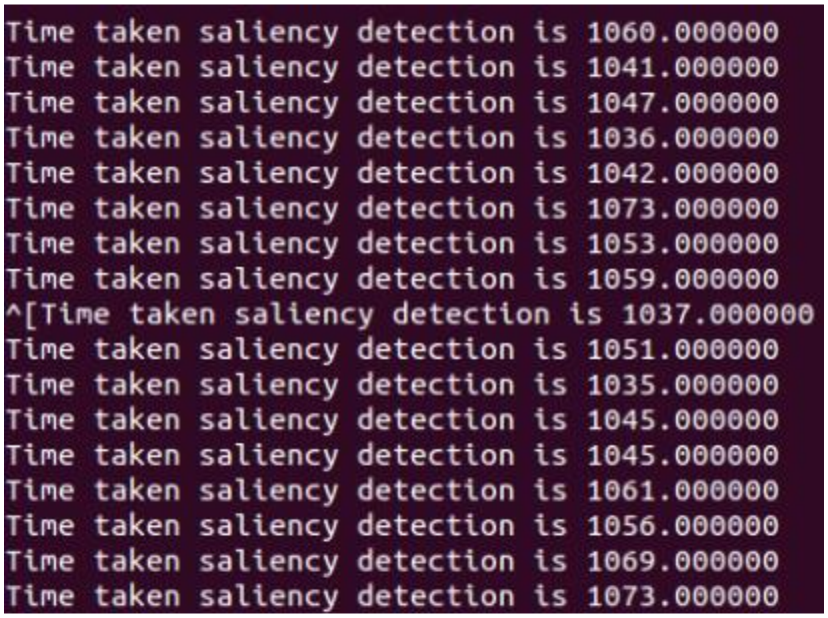
   e. The quantized motion vector field for each axis is determined and stored in the vector variables qmvX and qmvY.

7) Next the superpixel level histogram is to be calculated. This can be done by calling the function **superpixelHist** with the flow matrix as the input parameter.
   a. Similar to frame level histogram calculation the input flow matrix is first split into two matrices for each axis.
   b. This is then converted to polar coordinates
   c. The function GetLabels is invoked to get the label of each pixel.
   d. First the elements of each super pixel are stored in a vector to be able to access them easily. This vector is named '**sp**'.
   e. After identifying all the superpixels, the histogram for each of the superpixel is calculated. This divided each superpixel into 10 bins.
   f. Since the value in each bin is not needed, only the count of superpixels in each bin is noted and stored in the variable countInSpMagBins.
   g. Using these variables and the formula used to calculate the motion distinctiveness, the motion distinctiveness for each superpixel is calculated and finally the Motion Significance is calculated.
   h. The median of each of the superpixel group was calculated, and the whole of the superpixel was assigned the median value. And the output was checked to see whether the salient object was detected. The output is as shown earlier. [The median was chosen because the mean of all the values didn't represent the superpixel group accurately]

Significance of results and Suggestions for Next Steps

The next steps from this point would be to try to optimize the temporal saliency portion of code, by modifying the formula so that it isn't that compute intensive.

The time taken for execution of this code is:



```
Time taken saliency detection is 1060.000000
Time taken saliency detection is 1041.000000
Time taken saliency detection is 1047.000000
Time taken saliency detection is 1036.000000
Time taken saliency detection is 1042.000000
Time taken saliency detection is 1073.000000
Time taken saliency detection is 1053.000000
Time taken saliency detection is 1059.000000
^[Time taken saliency detection is 1037.000000
Time taken saliency detection is 1051.000000
Time taken saliency detection is 1035.000000
Time taken saliency detection is 1045.000000
Time taken saliency detection is 1045.000000
Time taken saliency detection is 1061.000000
Time taken saliency detection is 1056.000000
Time taken saliency detection is 1069.000000
Time taken saliency detection is 1073.000000
```

The time indicated is in milliseconds. So on an average the time taken is 1.1 secs.

Also the spatial saliency can be computed for image. I have started developing the code for calculating the Color Histogram values of each color. But due to time constraints couldn't complete implementing it. I believe if the spatial saliency is also implemented and its performance is compared with temporal saliency, then either one could be chosen which gives a fairly reliable output and takes less computation time. This could help improve the overall performance of the system.

Conclusion

The idea to use superpixels to identify salient objects is effective in a video stream. The paper suggests doing both temporal saliency detection and spatial saliency detection. If the temporal saliency code can be optimized and the spatial saliency performance is evaluated, then the better performing saliency detection can be used rather than using both and merging them together in the end. I have attached all the codes and relevant documents in the zip file.

References

1) http://docs.opencv.org/2.4/
2) http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6748868&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel7%2F76%2F4358651%2F06748868.pdf%3Farnumber%3D6748868
3) http://davidstutz.de/superpixel-algorithms-overview-comparison/