FAST OBJECT SEGMENTATION IN UNCONSTRAINED VIDEO

WRITTEN BY

YAIR ROTEM & RAN DVIRI

VERSION 1.0

Table of Contents

Table of Figures

Introduction

IBM wants to create a near real-time application that detects moving objects from an airborne platform.

We started by looking in the literature to find papers on that specific subject.

There were three candidates at the end of the time period that was set. With the help of Dror Porat and Udi Barzelay we managed to choose the best one that applies to the needs of the project. The selected paper was “Fast object segmentation in unconstrained video”, by Anestis Papazoglou and Vittorio Ferrari from the University of Edinburgh.

The main reason for choosing that paper was the fact that it has good results for different kind of video inputs. As a bonus an implemented Matlab code that can help us understand the algorithm even better.

Unfortunately for us we didn’t have the full knowledge on a lot of the main topics that were used in the paper and we had to learn all about them. The main topics of the paper are:

- Optical Flow
- Super Pixels
- Integral Intersection algorithm
- GMM

In addition we implement and run the algorithm in OpenCV environment that we learned from scratch.

In the end we implemented almost all the algorithm. The implementation of the 3.1 section of the algorithm is full. The implementation of the 3.2 section is half full, we lack the parts of the calculation of the appearance model and the location model.

We both learned a lot from the project, about topics related to computer vision, working with video and the difficulties in image processing.

In this report you can find the main algorithms of the paper and the related functions we used to implement them using OpenCV, Optical Flow, SLIC algorithm for super pixels and “integral Intersections” algorithm.

Algorithm Block diagrams

Main algorithm

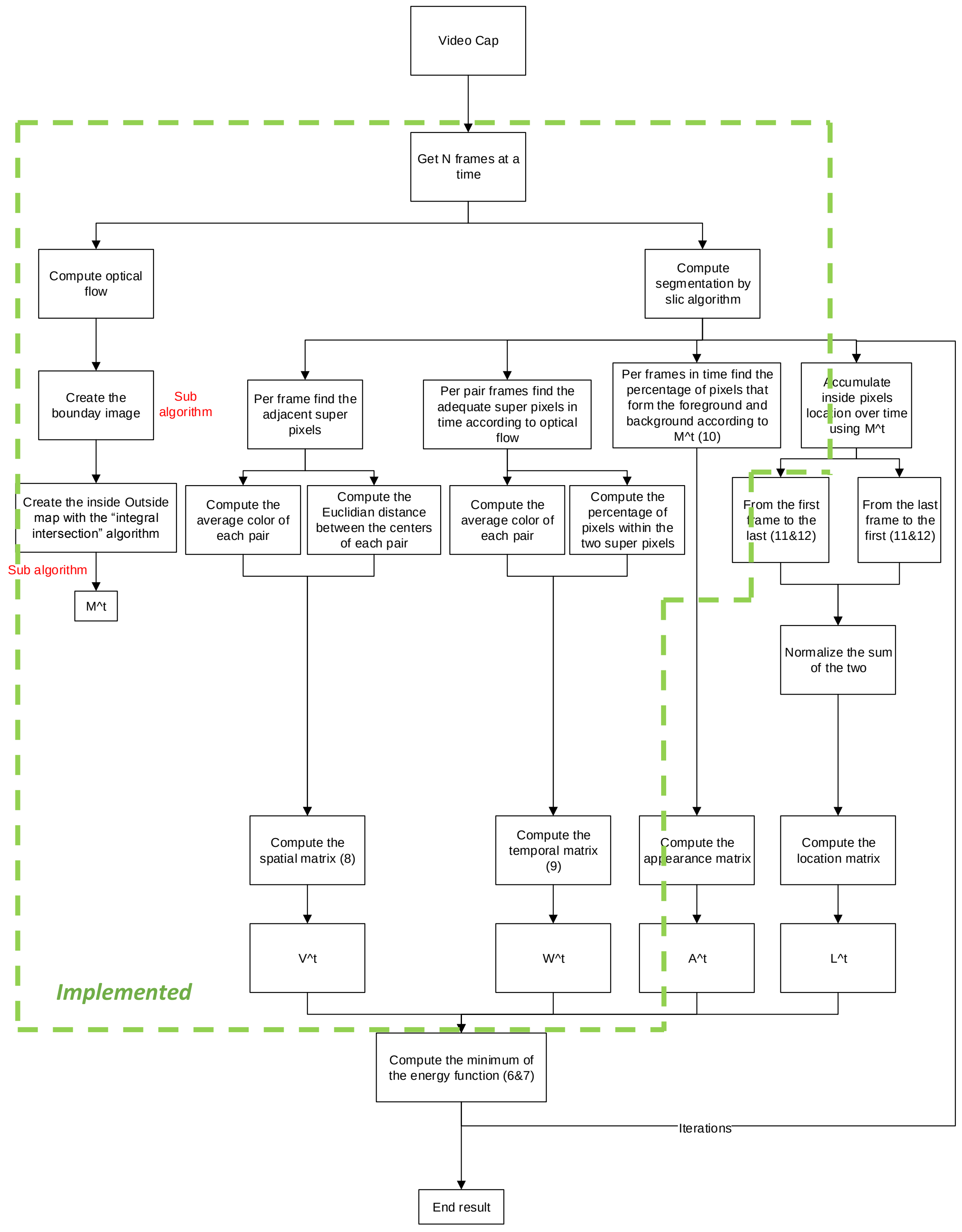

Video Cap
Get N frames at a time
Compute optical flow
Compute segmentation by slic algorithm
Create the bounday image
Sub algorithm
Create the inside Outside map with the "integral intersection" algorithm
Sub algorithm
M^t
Per frame find the adjacent super pixels
Per pair frames find the adequate super pixels in time according to optical flow
Per frames in time find the percentage of pixels that form the foreground and background according to M^t (10)
Accumulate inside pixels location over time using M^t
Compute the average color of each pair
Compute the Euclidian distance between the centers of each pair
Compute the average color of each pair
Compute the percentage of pixels within the two super pixels
From the first frame to the last (11&12)
From the last frame to the first (11&12)
Normalize the sum of the two
Compute the spatial matrix (8)
Compute the temporal matrix (9)
Compute the appearance matrix
Compute the location matrix
V^t
W^t
A^t
L^t
Implemented
Compute the minimum of the energy function (6&7)
Iterations
End result

Figure 1 main algorithm

Boundary algorithm

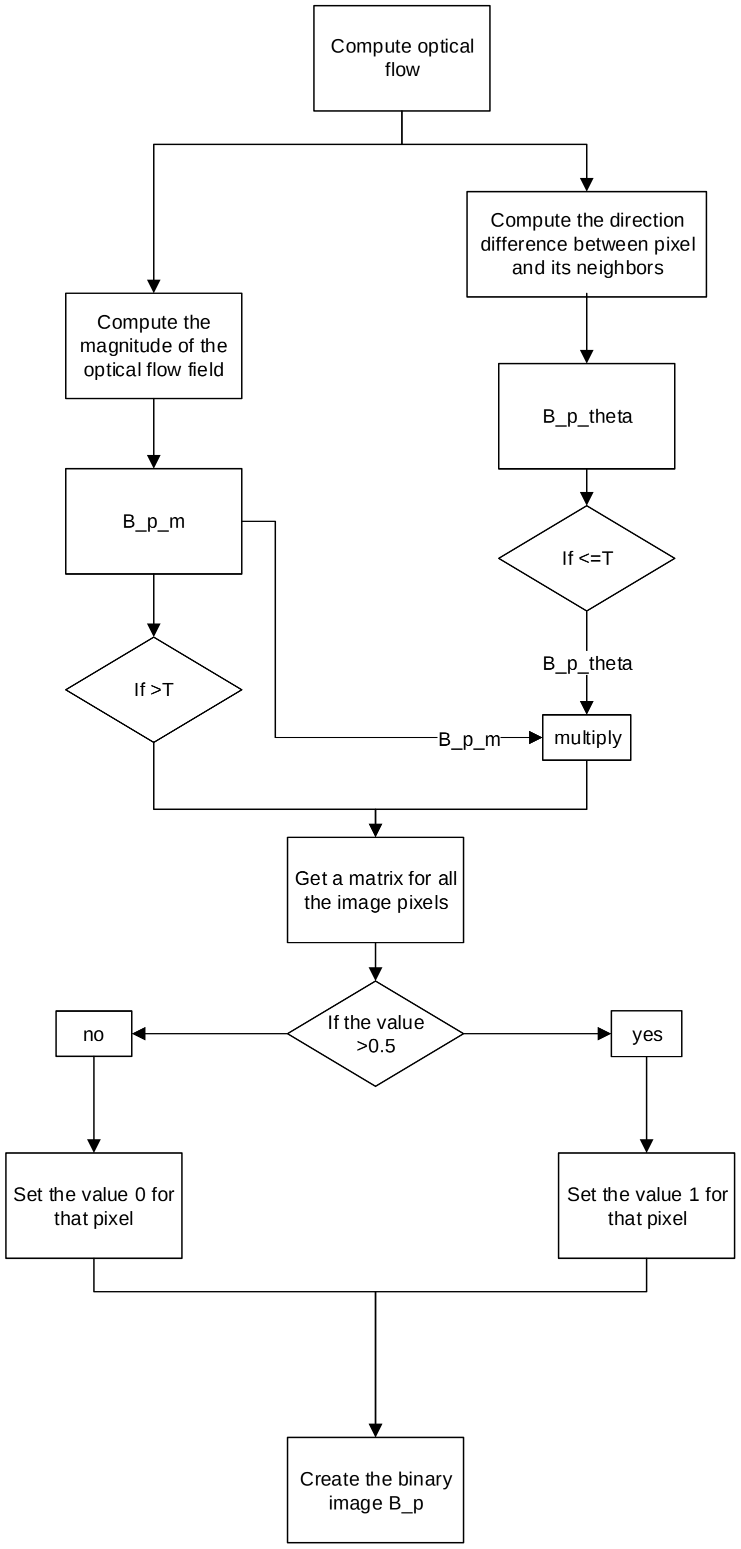
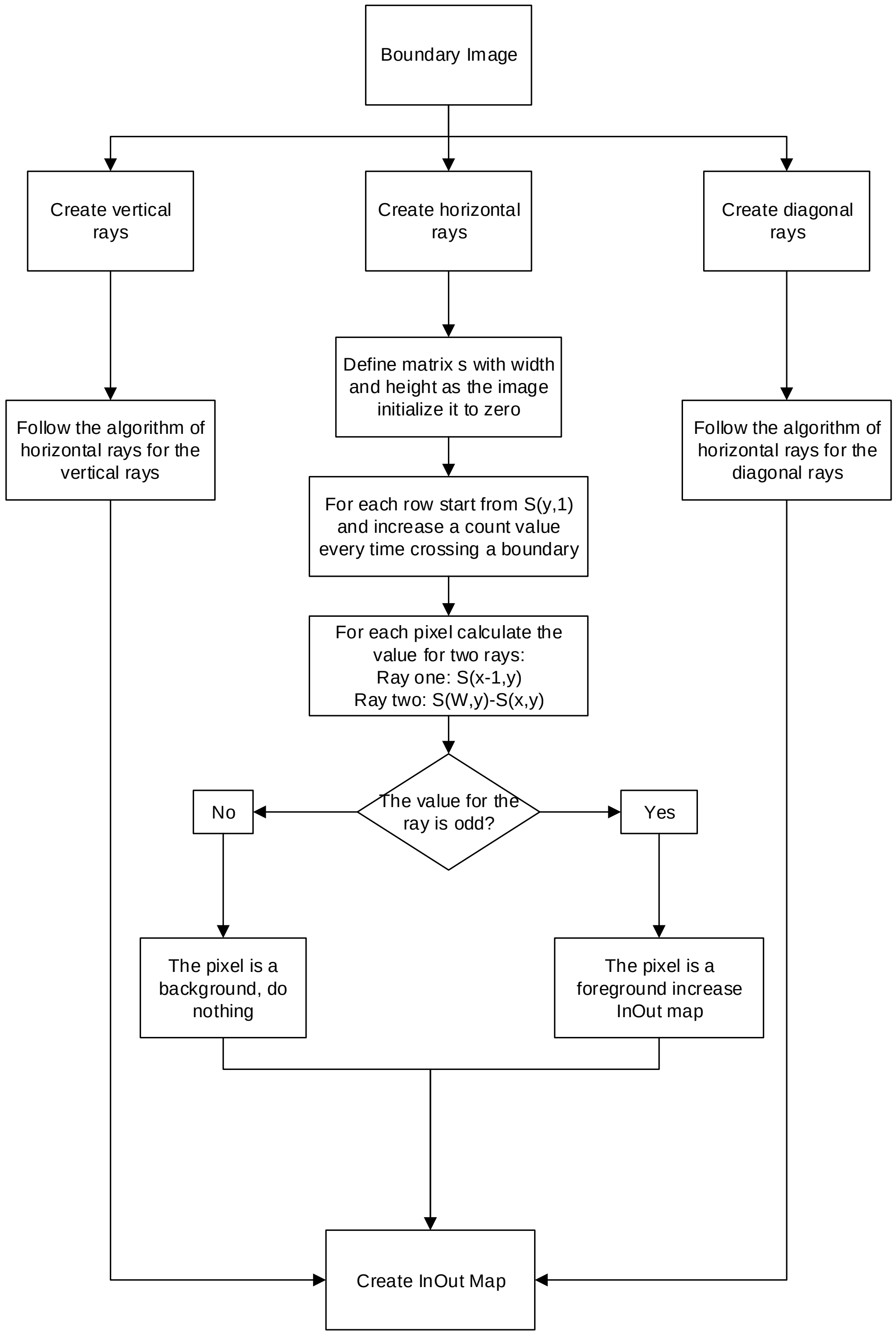

Compute optical flow
Compute the direction difference between pixel and its neighbors
Compute the magnitude of the optical flow field
B_p_theta
B_p_m
If <=T
If >T
B_p_theta
B_p_m
multiply
Get a matrix for all the image pixels
If the value >0.5
no
yes
Set the value 0 for that pixel
Set the value 1 for that pixel
Create the binary image B_p

Figure 2 boundary image algorithm

Integral Intersection algorithm



Boundary Image
Create vertical rays
Create horizontal rays
Create diagonal rays
Define matrix s with width and height as the image initialize it to zero
Follow the algorithm of horizontal rays for the vertical rays
Follow the algorithm of horizontal rays for the diagonal rays
For each row start from S(y,1) and increase a count value every time crossing a boundary
For each pixel calculate the value for two rays: Ray one: S(x-1,y) Ray two: S(W,y)-S(x,y)
The value for the ray is odd?
No
Yes
The pixel is a background, do nothing
The pixel is a foreground increase InOut map
Create InOut Map

Figure 3 integral intersections block diagram

Checking the algorithm

We manage to check the first part of the algorithm, section 3.1. The InOut map can be seen in the following figures.

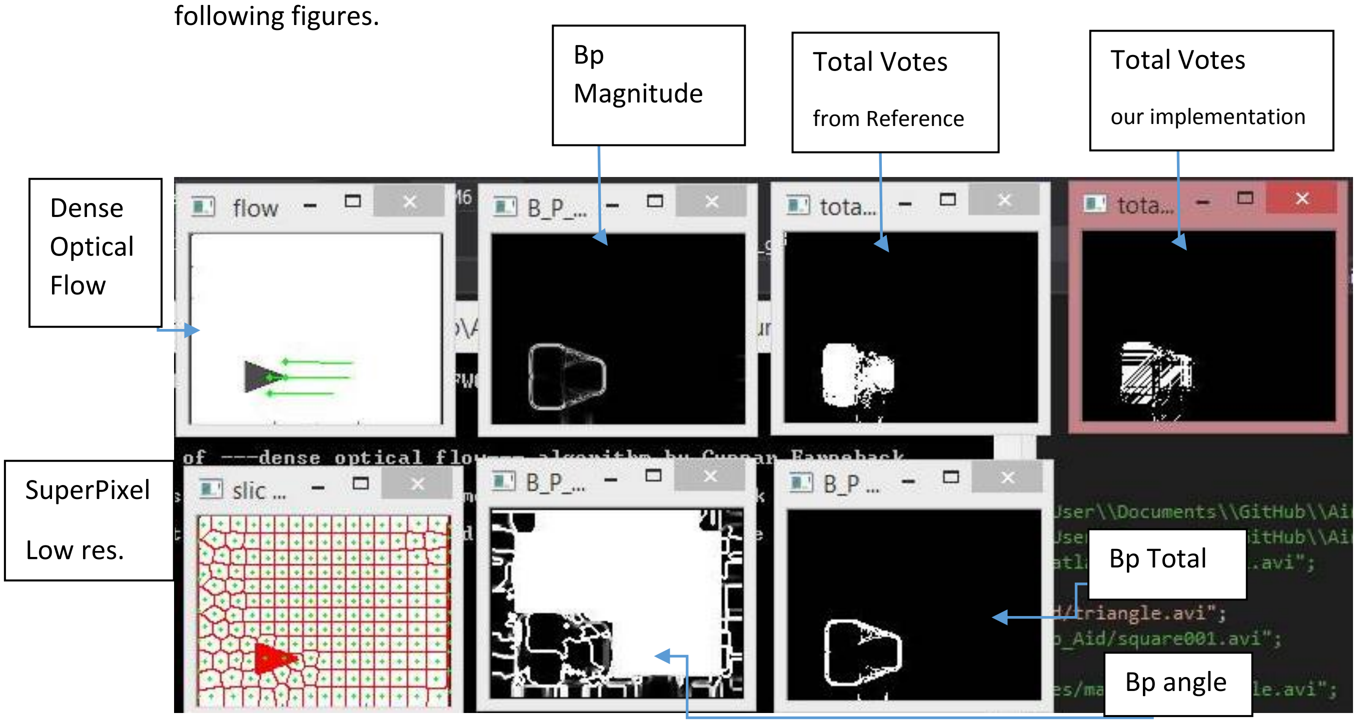


Figure 4 triangle object check

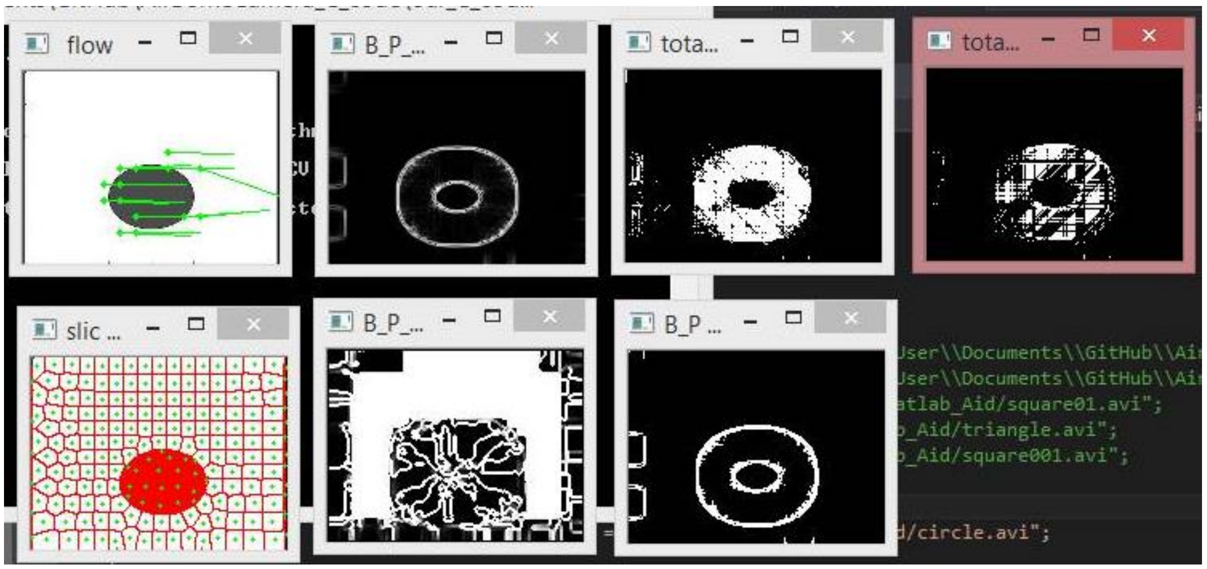


Figure 5 ellipse object check

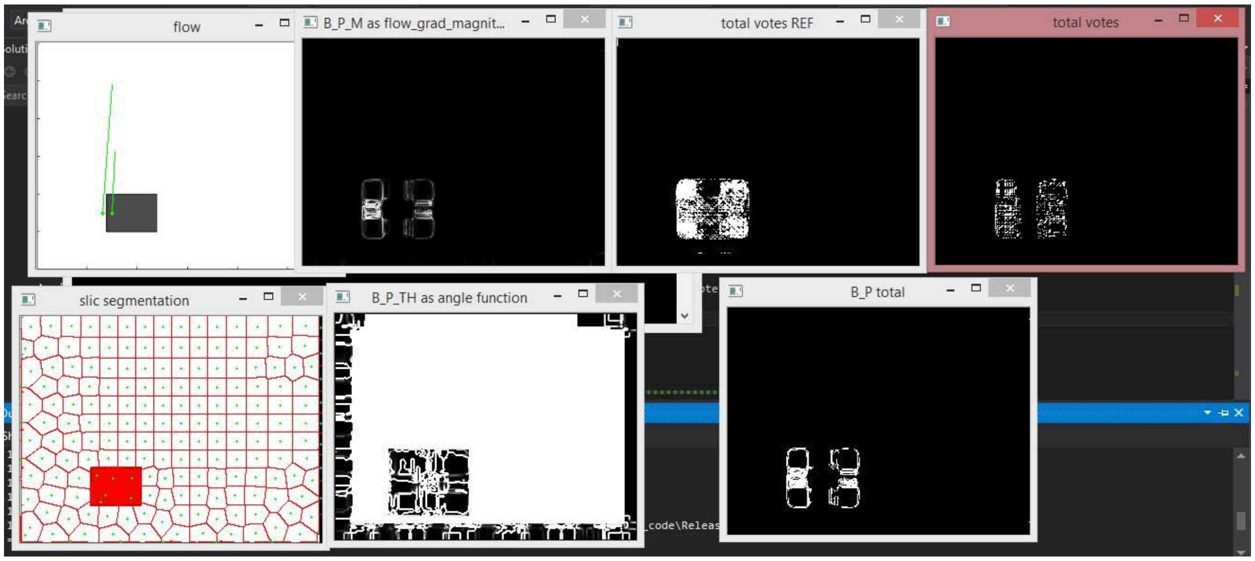


Figure 6 square object check

Conclusions

The algorithm as implemented is very sensitive to the resolution of the video. Once we used different resolution, the thresholds needed to be fit manually.

We had to condition the thresholds with some variables. It is not that flexible. The authors of the article seam to ‘leave it for the reader’ to better take care for that point.

In a runtime perspective:

- We would recommend to use GPU algorithms. It is becoming very popular and available to use this methods. Yet not all hardware is fit, and if we have no knowledge on the target so we cannot trust it. It is probably does available when doing offline processing, and testing.
- When using all these algorithms, each one has some initialization, and iterations procedure for convergence. In our application we processed each step on the way for 1 frame (SuperPixel segmentation) or 2 (Optical-Flow, Votes calculation etc.) – it is very recommended, in a final consumer application, to consider initializing each next step by the results of the previous frame processing. It can skip guessing for initializations, and reduce the required iterations for the new added frame.

List of functions

`int main(int argc, char** argv)`

The main function of the project. The function that runs all parts of the algorithm.

`void copy_vectors_to_array(int w, int h,  Slic *slic_obj, unsigned int * out_array)`

Copies the data created by the Slic algorithm from its structure to a regular array that is more comfortable to work with.
The function receives:

1) Height and width of the frame
2) Slic object containing data for S.P. as centers position, color data etc.

The function returns:

1) The data from the slic object in an array

`void copy_floatMat_to_shortArray(int w, int h,  Mat & flow , short * out_array)`

Copies the data created by the flow algorithm from its structure to a regular array that is more comfortable to work with.
The function receives:

1) Height and width of the frame
2) Flow structure containing data of flow magnitude.

The function returns:

2) The flow data in a regular array

`void calc_motion_boundaries(const Mat &flow, Mat &totalVotes)`

Calculates the InOut matrix per frame by using the optical flow to create the motion boundaries for magnitude and angle and counting the votes each pixel from 8 directions according to the the integral intersection algorithm.

`void calc_bpm(Mat& flow_grad_mag, Mat& result)`

Estimating the boundaries motion by computing the magnitude gradient for each pixel in frame. $b_p^m = 1 - exp(-\lambda^m ||\nabla fp||)$
The function receives:

1) Flow data

The function returns:

1) The result of the calculation of b_p^m for the frame

`void calc_bpTheta( Mat& alpha_input,Mat& flowX ,Mat& flowY ,Mat& out_bpTheta )`

Estimating the boundaries motion by computing the difference of angle between each pixel and its neighbors in frame. $b_p^\theta = 1 - exp(-\lambda^\theta \max_{\{q\in N\}}(\delta\theta_{p,q}^2))$

The function receives:

1) Array of flow angles
2) Flow data:
    a. Magnitude in X
    b. Magnitude in Y

The function returns:

2) The result of the calculation of b_p^θ for the frame

```
void calc_bp_total(Mat& bpm, Mat& bpTheta, Mat& bp_Out)
```

Calculating the motion boundaries matrix.

$$b_p = \begin{cases} b_p^m & , \text{if } b_p^m > T \\ b_p^m \cdot b_p^\theta & , \text{if } b_p^m \leq T \end{cases}$$

The function receives:

1) b_p^m
2) b_p^θ

The function returns:

3) The result of the calculation of b_p for the frame

```
void calc_votes_1(Mat& bp, Mat& out1)
```

Calculates the votes from the horizontal rays and vertical rays.
The function receives the binary matrix of the motion boundaries and returns the updated InVotes matrix.

```
void calc_votes_2(Mat& bp, Mat& out3)
```

Calculates the votes from the diagonal rays.
The function receives the binary matrix of the motion boundaries and returns the updated InVotes matrix.

```
void calc_total_8_votes(Mat& out1, Mat& out2, Mat& out3, Mat& out4, Mat& totalVotes)
```

Sums the votes from the 8 different kind of rays and creates the InOut Matrix by calculating for each pixel if the number of votes is bigger than 5.
The function receives 4 votes matrixes: Horizontal, vertical, and two diagonal, and returns the InOut matrix.

```
void calc_Spatial_distances_and_Weights(unsigned int *sources, unsigned int *targets,
                                         unsigned int vectors_length , vec2dd centers ,
                                         double *alpha2,double *color_dist, double
                                         *center_dist , double *Vij , double *sum_Out)
```

This function calculates the weight of the spatial matrix in the energy equation.

The function receives:

1) Sources and target labels of super pixels
2) The vector length of the sources and targets
3) The adequate centers and colors of the super pixels relevant to the sources and targets
4) The weight of the spatial matrix in the calculation - Alpha

The function returns:

1) Color distance
2) Center distance
3) The spatial matrix V
4) The weight value for each super pixel in the spatial matrix

```
void calc_Temporal_distances_and_Weights( unsigned int *sources, unsigned int
                                          *targets, float *connectionRatios, unsigned int
                                          vectors_length , vec2dd centers , double
                                          *alpha3, double* color_dist , double *Wij,
                                          double *sum_Out)
```

This function calculates the weight of the temporal matrix in the energy equation.

The function receives:

1) Sources and target labels of super pixels
2) The percentage of pixels forming the connections
3) The vector length of the sources and targets
4) The adequate centers and colors of the super pixels relevant to the sources and targets
5) The weight of the spatial matrix in the calculation - Alpha

The function returns:

1) Color distance
2) Center distance
3) The temporal matrix W
4) The weight value for each super pixel in the temporal matrix

```
void calc_pairwisePotentials(Slic *segmented_slic,Slic *prev_segmented_slic, Mat &flow,
                             long long superPixels_accumulated, double *pairWise_Weight)
```

This function manage the calculation of the weights of the spatial and temporal matrixes as part of the energy equation

The function receives:

1) The super pixel data in Slic structure for one frame
2) The super pixel data in Slic structure for second frame
3) Optical flow data of those two frames
4) The total number of super pixels in both frames

The function returns:

1) The sum weight of the spatial and temporal matrixes

```
void calc_inRatios(Slic *segmented_slic, Mat &votes, float  * inRatios)
void calcSuperpixelInRatio(unsigned int *superpixels, int height, int width,const
                           unsigned short *labels, bool *inOutMap,
                                          float * output)
```

This function calculates the In-Ratio value of each super pixel by the amount of consistency to the In-Votes map.

The function receives:

1) Array of super pixels label for all pixels
2) Height and width of the frame
3) The total number of super pixels in the frame
4) Array of the input and output map (zeros and ones)

The function returns:

1) The ratio of being a forground – percentage of the adequacy between each super pixel

```
void writeMat( cv::Mat const& mat, const char* filename, const char* varName,

              bool bgr2rgb , int writeMode)
```

This function writes data to mat file for importing in Matlab.

```
void calc_unary_potentials(Slic *segmented_slic,Slic *prev_segmented_slic, Mat &flow,
                                   long long superPixels_accumulated)
```

This function calculates the appearance and location matrixes.

```
void calcSpatialConnections(unsigned int      *superpixelMap,    // the superpixels
segmantation matrix
                      unsigned int          height,              // the H of the frame
                      unsigned int          width,               // the W of the frame
                      unsigned long long    superpixels,         // total number of s.pixels
                      unsigned int *sources,                     // output of sPixels sources
                      unsigned int *targets,      // output of sPixels targets as the sources
neighbours
                      unsigned int *vectors_length)
```

This function calculates the spatial connections between super pixels in one frame. The way of doing it is to count the number of super pixels neighbors with different label and write the one with the most tangency.

The function receives:

1) Array of super pixels label for all pixels
2) Height and width of the frame
3) The total number of super pixels in the frame

The function returns:

1) Array of sources – super pixel labels
2) Array of targets – the closest super pixel to the one of the source super pixel (from the sources array)

```
void calcTemporalConnections(      short          *flow,
                                   unsigned int *superpixelMap,
                                   unsigned int  height,
                                   unsigned int  width,
                                   unsigned int *superpixelMapNext,
                                   unsigned long long superpixels,
                                   unsigned int *sources,
                                   unsigned int *targets,
                                   float          *connectionRatio,
                                   unsigned int *vectors_length)
```

This function calculates the temporal connections between super pixels in two frames. The way of doing it is to count specify the most adequate super pixel in the second frame to the one in the first according to the direction of the flow.

The function receives:

1) Flow data
2) Array of super pixels label for all pixels
3) Height and width of the frame
4) The total number of super pixels in the frame

The function returns:

1) Array of sources – super pixel labels
2) Array of targets – the closest super pixel to the one of the source super pixel (from the sources array)
3) The ratio of the connection – percentage of the adequacy between each super pixel
4) The length of the output vector