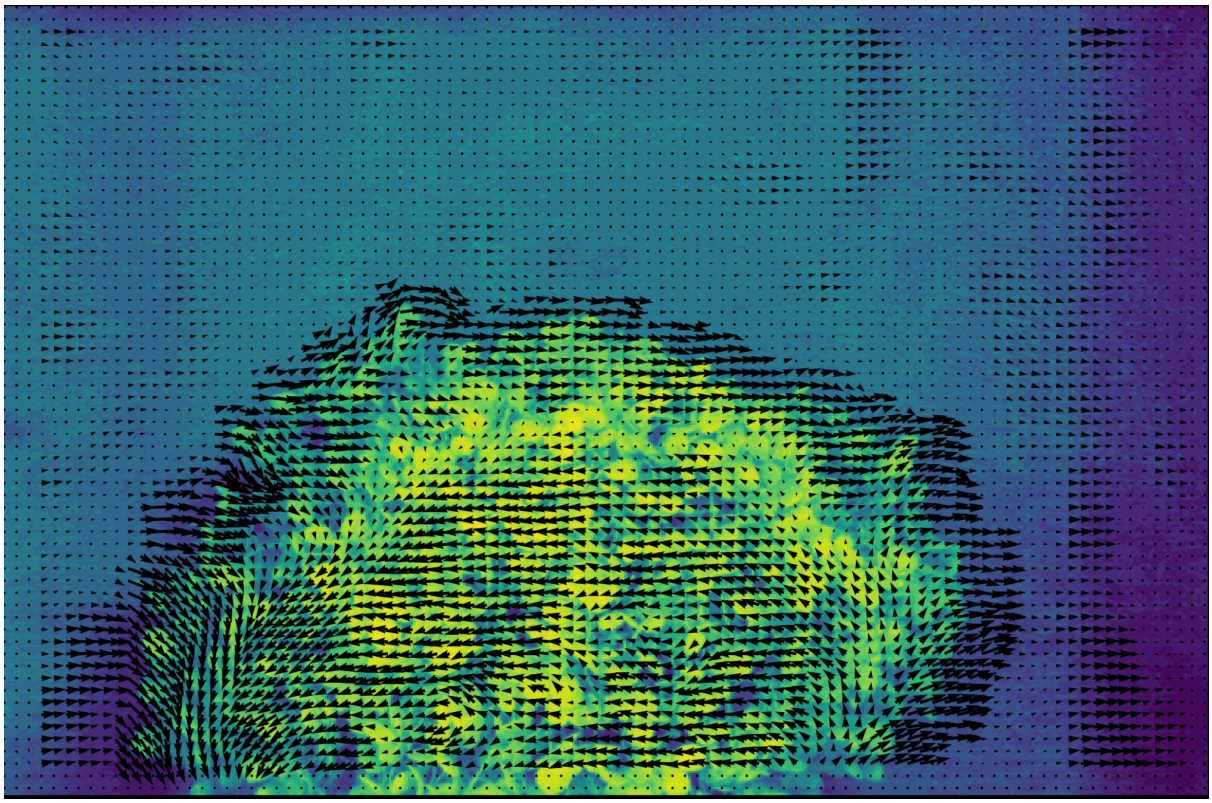
REPORT - TP VISION
OPTICAL FLOW

IMA

Patel Shubhamkumar



README :
- You can find the files **horn.py, lucas.py, lucas_gauss.py, gradhorn.py** in the **python** directory.
- The test code for all datsets is in the jupyter notebook named :
Optical_Flow_PATEL.ipynb, among with the output images resulting from the code.

GRADHORN

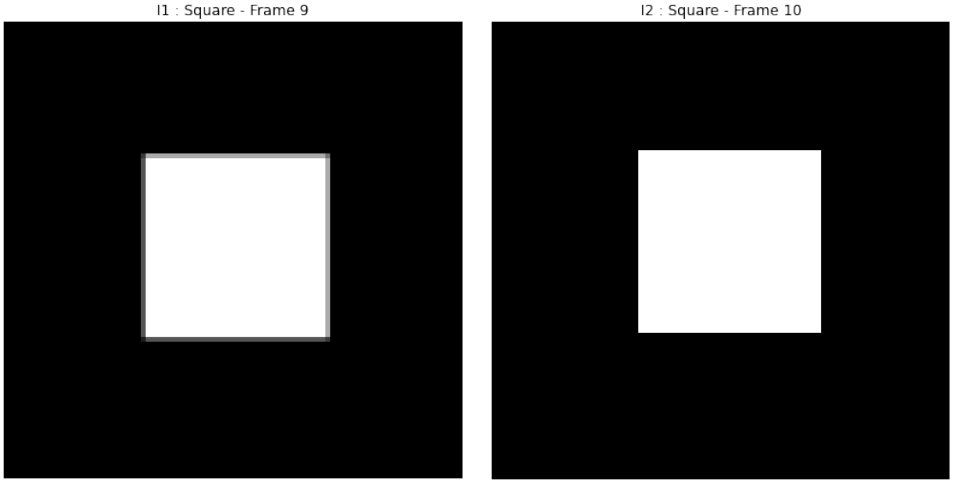
Horn and Schunck proposed the following filter:

$$
\begin{aligned}
I_x(i,j) &\simeq \frac{1}{4}[I_1(i,j+1) - I_1(i,j) + I_1(i+1,j+1) - I_1(i+1,j) \\
&\quad + I_2(i,j+1) - I_2(i,j) + I_2(i+1,j+1) - I_2(i+1,j)] \\
I_y(i,j) &\simeq \frac{1}{4}[I_1(i+1,j) - I_1(i,j) + I_1(i+1,j+1) - I_1(i,j+1) \\
&\quad + I_2(i+1,j) - I_2(i,j) + I_2(i+1,j+1) - I_2(i,j+1)] \\
I_t(i,j) &\simeq \frac{1}{4}[I_2(i,j) - I_1(i,j) + I_2(i+1,j) - I_1(i+1,j) \\
&\quad + I_2(i,j+1) - I_1(i,j+1) + I_2(i+1,j+1) - I_1(i+1,j+1)]
\end{aligned}
$$

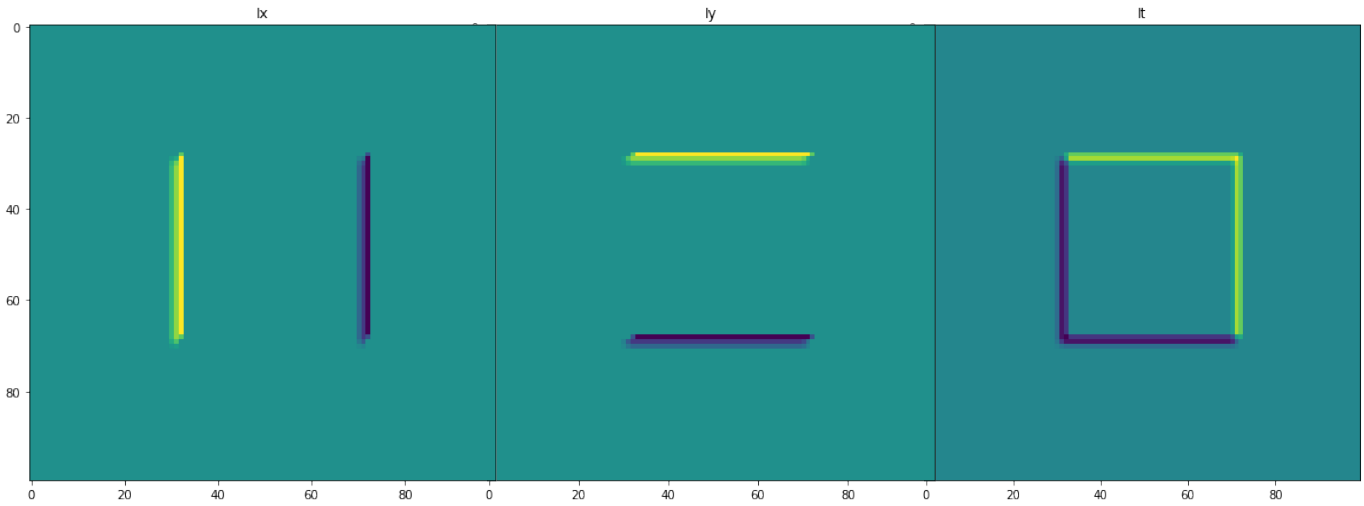
This Filter can be implemented using kernels and a few 2d convolution operations between these kernels and the **I1** and **I2** images. We get :

KernelX = [[1, -1], [1, -1]] / 4 **KernelY** = [[1, 1], [-1, -1]] / 4 **KernelT** = [[1, 1], [1, 1]] / 4	**Ix** = convolve2d(**I1** + **I2**, **kX**) **Iy** = convolve2d(**I1** + **I2**, **kY**) **It** = convolve2d(**I2** - **I1**, **kT**)

For the following I1 and I2 images :



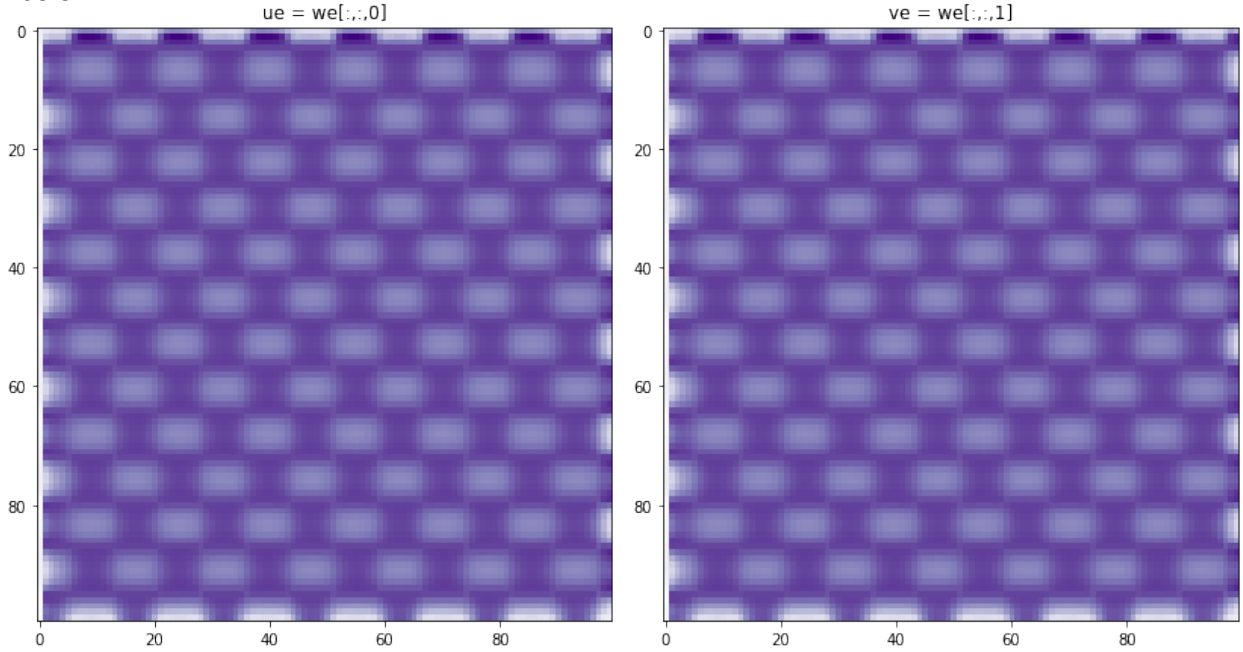
We get the following results for **Ix**, **Iy**, **It**



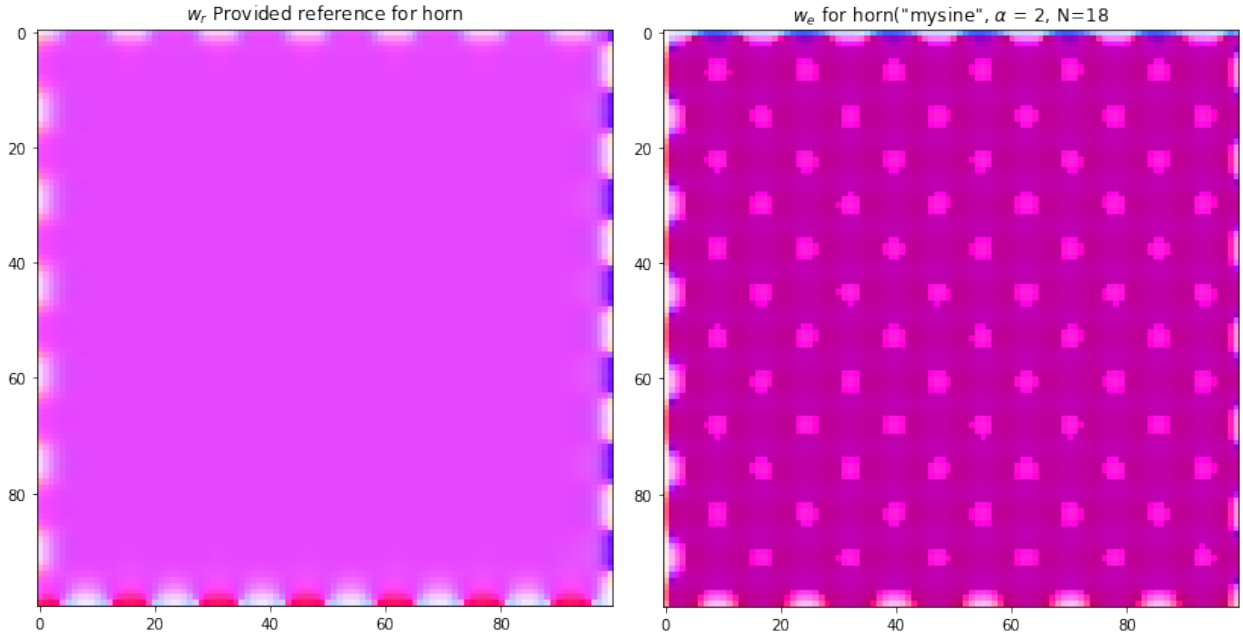
- The results we get for **Ix** is the spatial gradient of the pair of images (**I1, I2**) along the X axis.
- The results we get for **Iy** is the spatial gradient of the pair of images (**I1, I2**) along the Y axis.
- The results we get for **It** is the spatio-temporal gradient of the pair of images (**I1, I2**) along both X and Y axis as well as T axis with the delta of time between the two frames.

Horn-Schunck Method

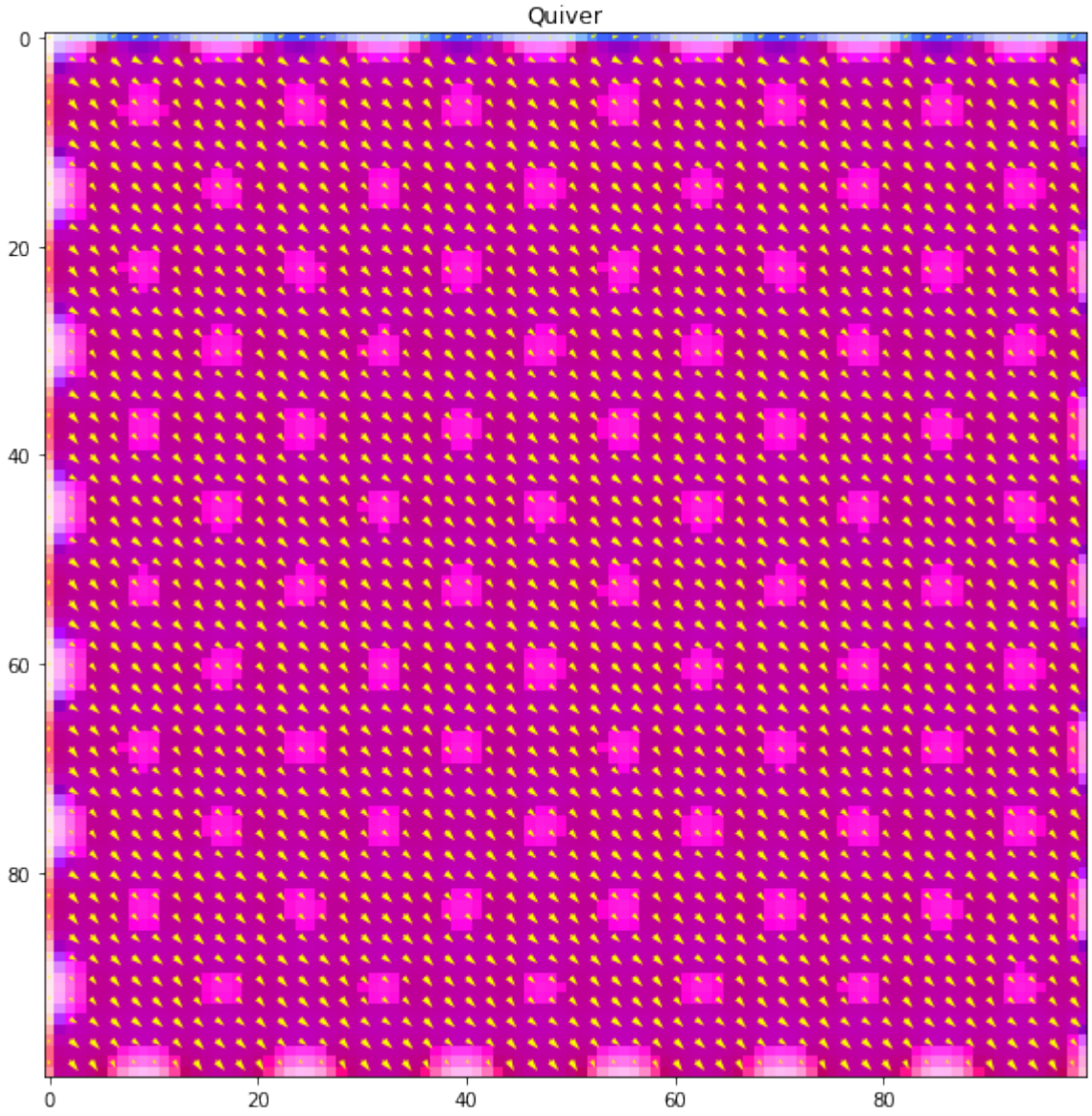
Applying an unoptimised Horn-Schunck Method with alpha = 0.1 and N = 18 on the **mysine** dataset, we get a velocity map **We,** whose individual components **u** and **v** are displayed below :



Using the **ComputeColor** method we get the following :



We can see that the unoptimized Horn method converged quite well towards the reference velocity map but there are still some artifacts remaining.



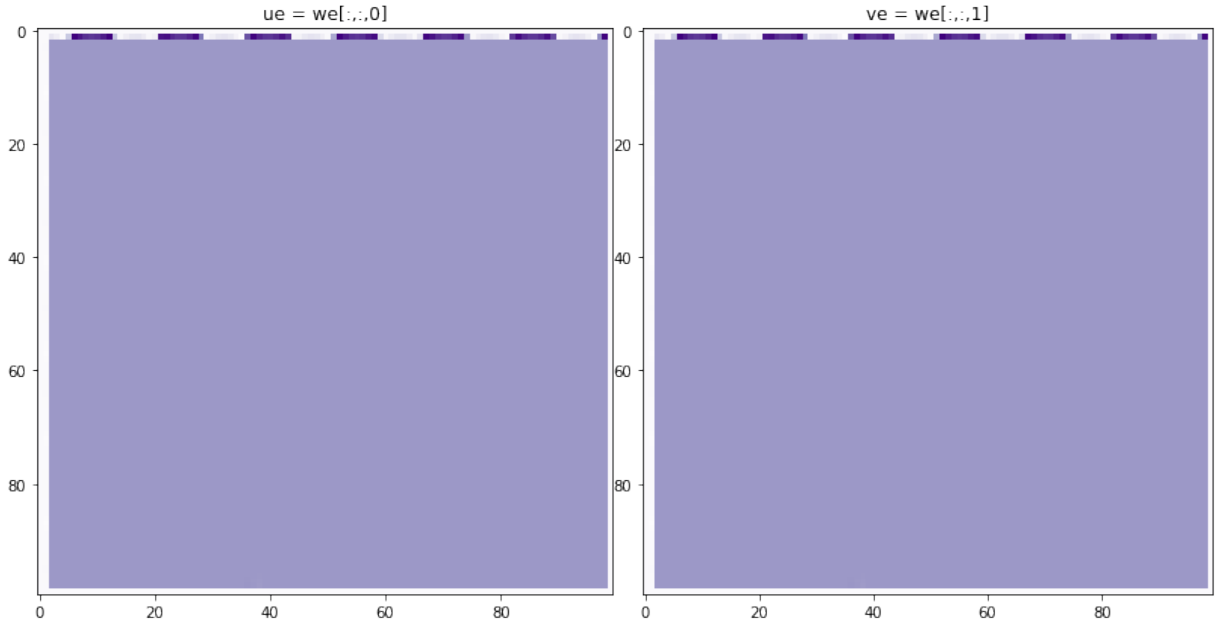
In the Quiver plot above we can see how the artifacts are disturbing the flow represented by the yellow vectors.

With all the disturbance combined we get an angular error **AP = 0.219,** and the Euclidean error **EPE = 0.232** and a mean squared error **MSE = 0.563** .

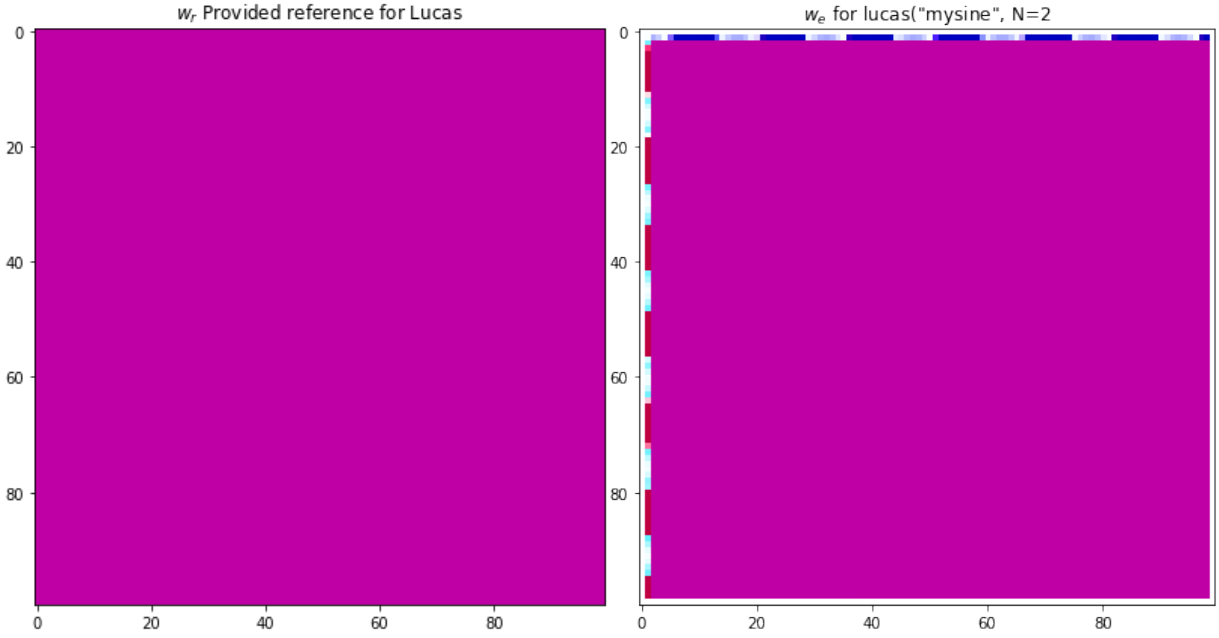
We will see how after the optimization process we are able to get a better convergence towards the reference values.

Lucas-Kanade Method

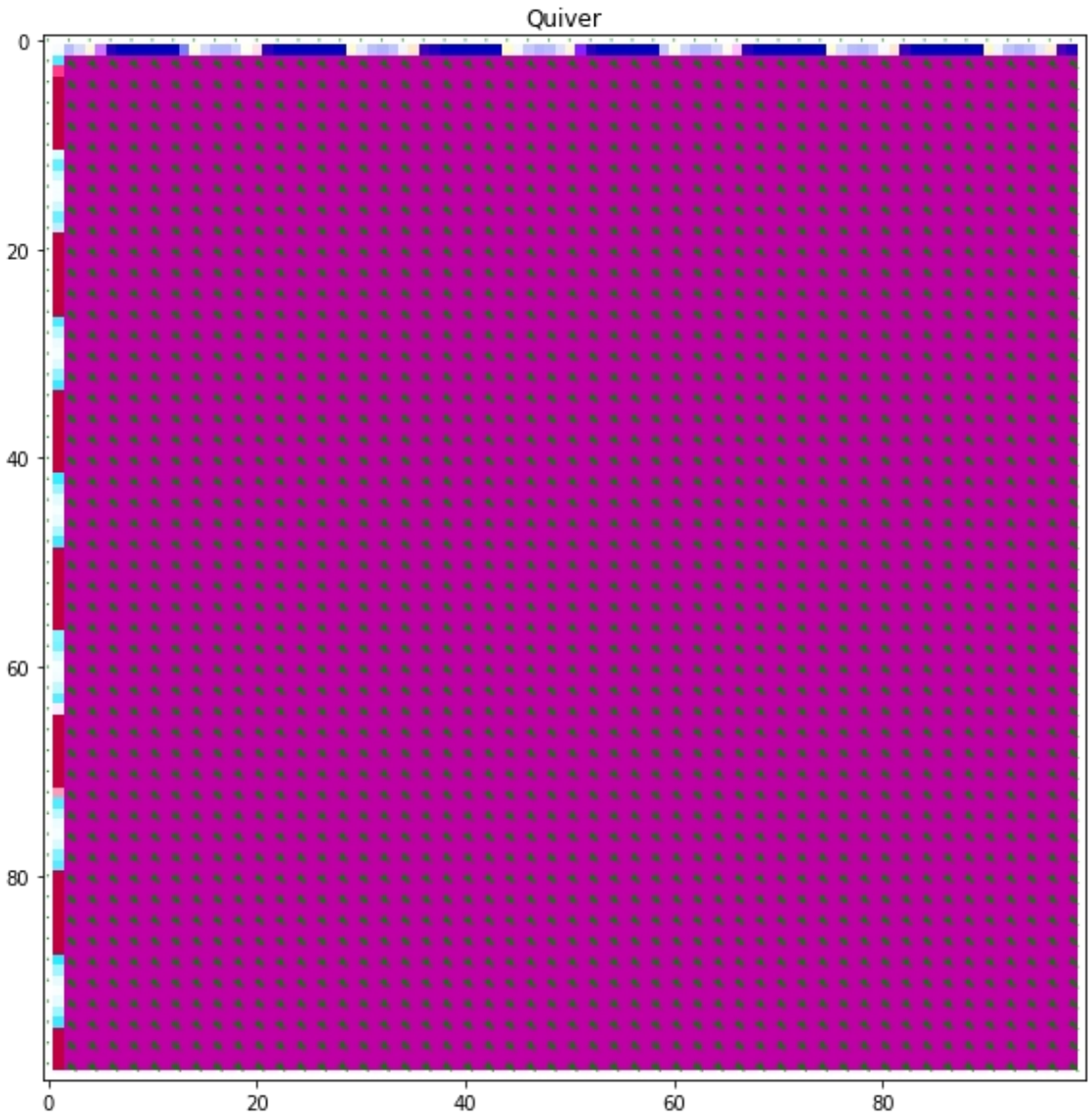
Applying an unoptimised Lucas-Kanade Method with a window of size N = 2 on the **mysine** dataset, we get a velocity map **We,** whose individual components **u** and **v** are displayed below :



Using the **ComputeColor** method we get the following :



We can see that the Lucas-Kanade method converged quite well towards the reference velocity, There are only a few artifcats around the image due to the way we handle the borders with the image.



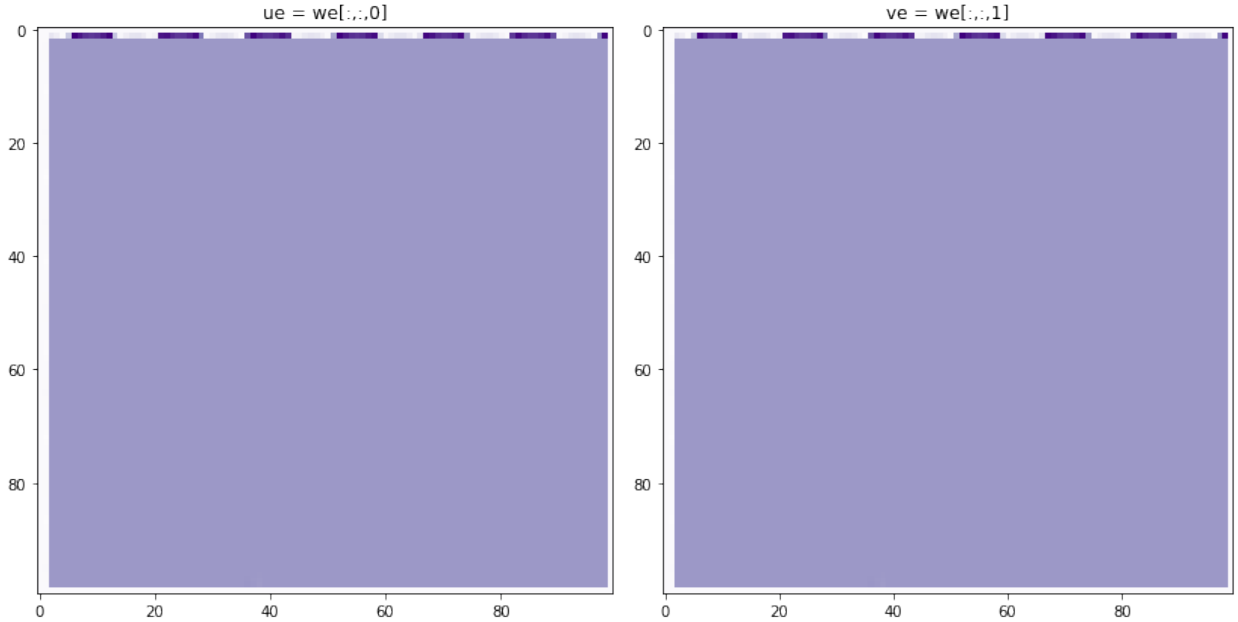
In the Quiver plot above we can see the homogeneity in the flow represented by the yellow vectors, apart from the disturbance it create around the borders.

With all the disturbance combined, we get an angular error **AP = 0.247,** and the Euclidean error **EPE = 0.084** and a mean squared error **MSE = 0.060**.

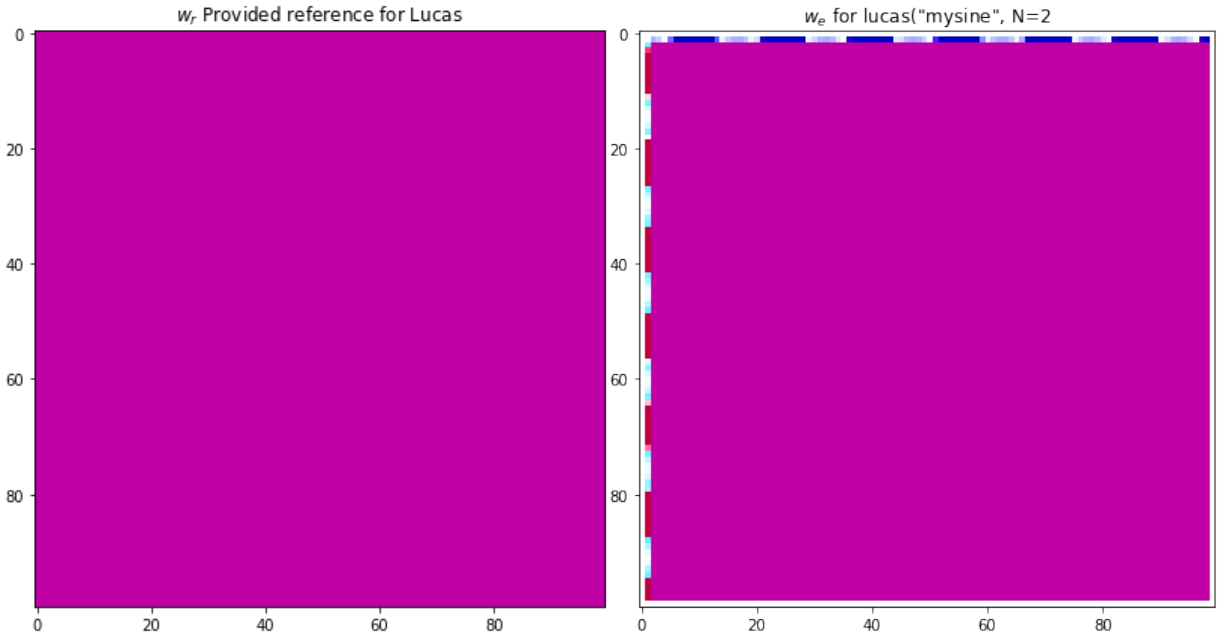
Lucas-Kanade with Gaussian Method

While computing the A matrix, we are computing it for the neighborhood around a pixel **p** in each of **Ix, Iy, It** images**.** It is during this step that we need to apply a gaussian weighting around the neighborhood for the pixel p using a gaussian kernel of size roughly equal to the window W center on p and of parameter sigma.

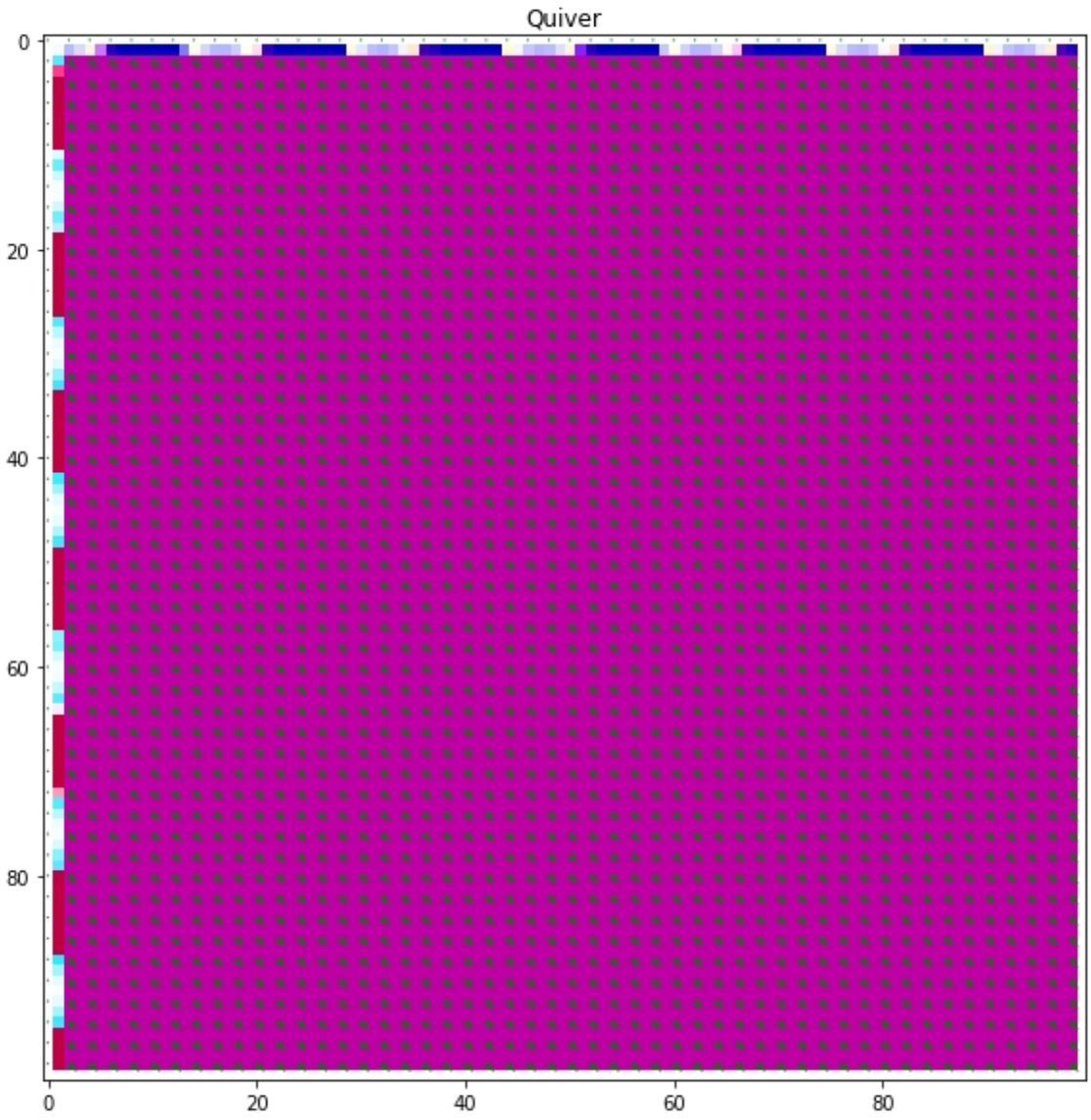
After applying it we get the following velocity map (**we**) components **u** and **v**.



Using the **ComputeColor** method we get the following :



We can see that the Lucas-Kanade method converged quite well towards the reference velocity, There are only a few artifcats around the image due to the way we handle the borders with the image.



In the Quiver plot above we can see the homogeneity in the flow represented by the yellow vectors, apart from the disturbance it create around the borders.

With all the disturbance combined, we get an angular error **AP = 0.247,** and the Euclidean error **EPE = 0.084** and a mean squared error **MSE = 0.060**.

One disadvantage of this method is that it takes far longer to compute due to the pseudo inverse operation we need to apply.

Error :

We want to compute the error between the following

- the ***Estimated Velocity Map*** **: We**
- the ***Reference Velocity Map*** **: Wr (=Ground Truth)**

To compute the error between these Velocity maps we can use the following errors :

Angular Error (in space-time) :

$$\theta(i,j) = \arccos\left(\frac{1 + w_r(i,j).w_e(i,j)}{\sqrt{1 + \|w_r(i,j)\|^2}\sqrt{1 + \|w_e(i,j)\|^2}}\right)$$

This represents the angle between the two extended vectors : (1, ur, vr) and (1, ur, vr)

Euclidean distance or EPE:

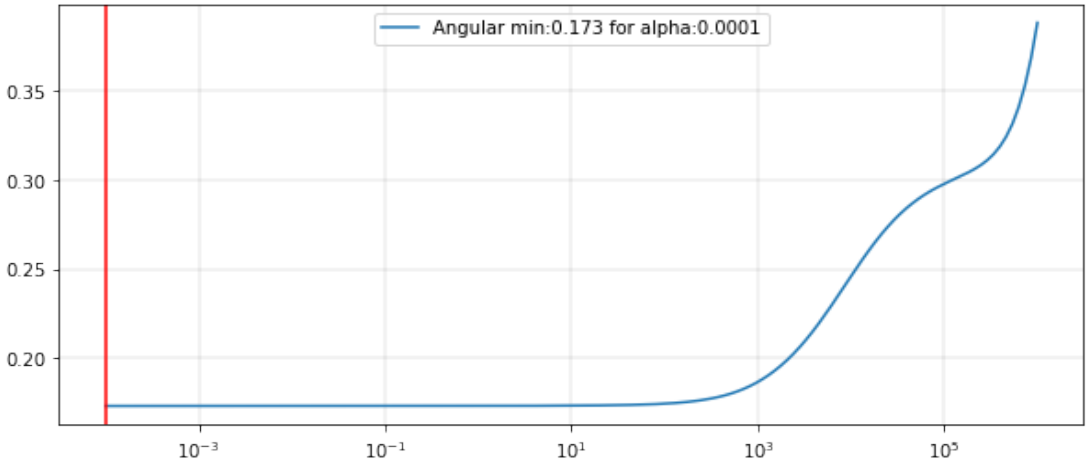
$$EPE = \sqrt{(u_e - u_r)^2 + (v_e - v_r)^2}\text{ , where } w_e = (u_e, v_e) \text{ and } w_r = (u_r, v_r)$$

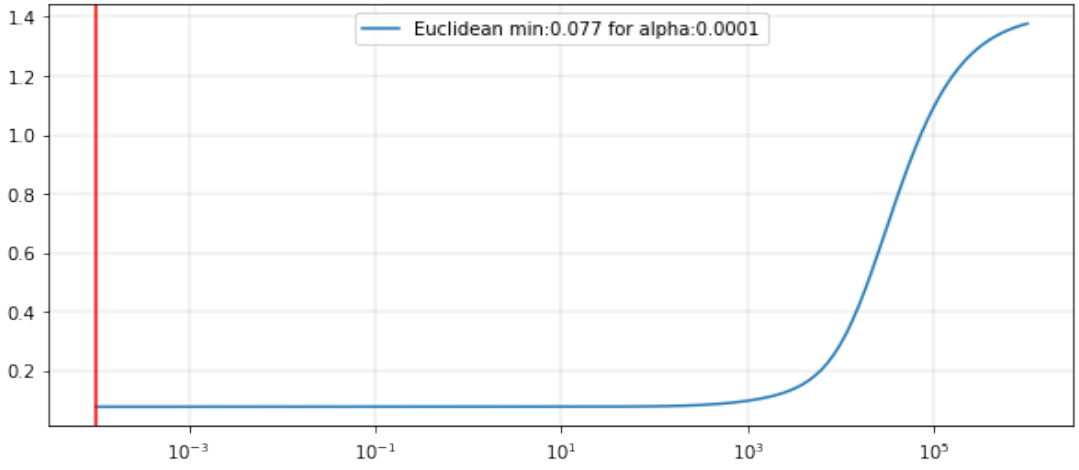
This simply represents the distance between two vectors.

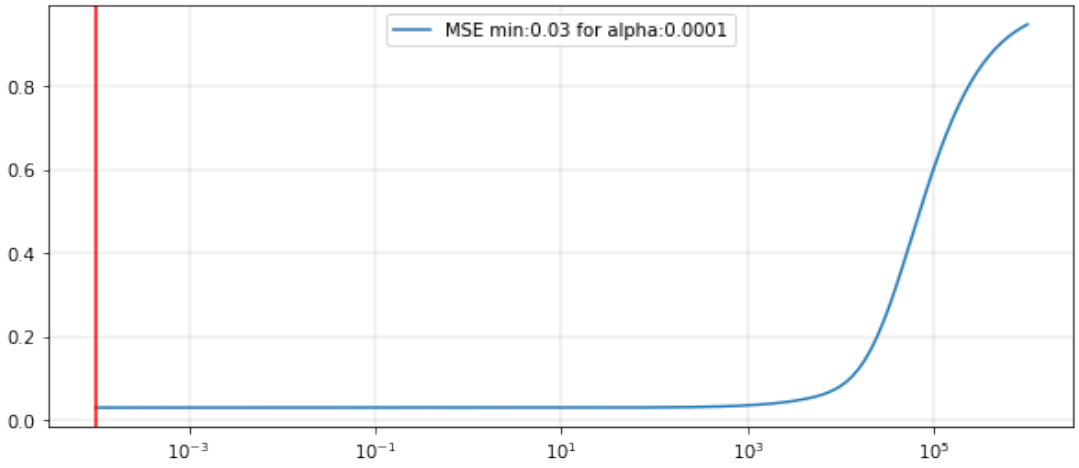
Notes :

- Angular Error is very sensitive to small estimation errors caused by small displacements
- Whereas EPE hardly discriminates between close vectors.

Optimizing Horn

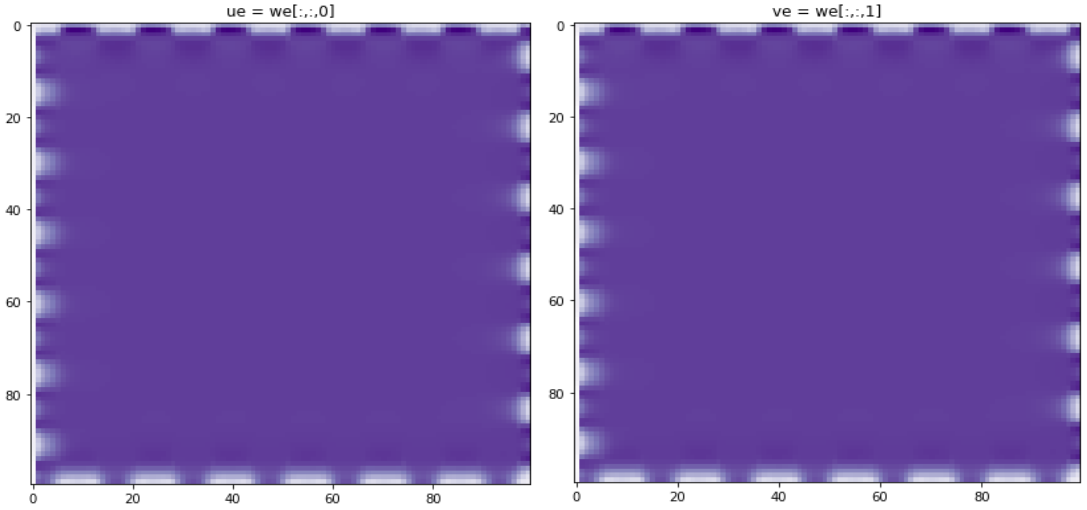


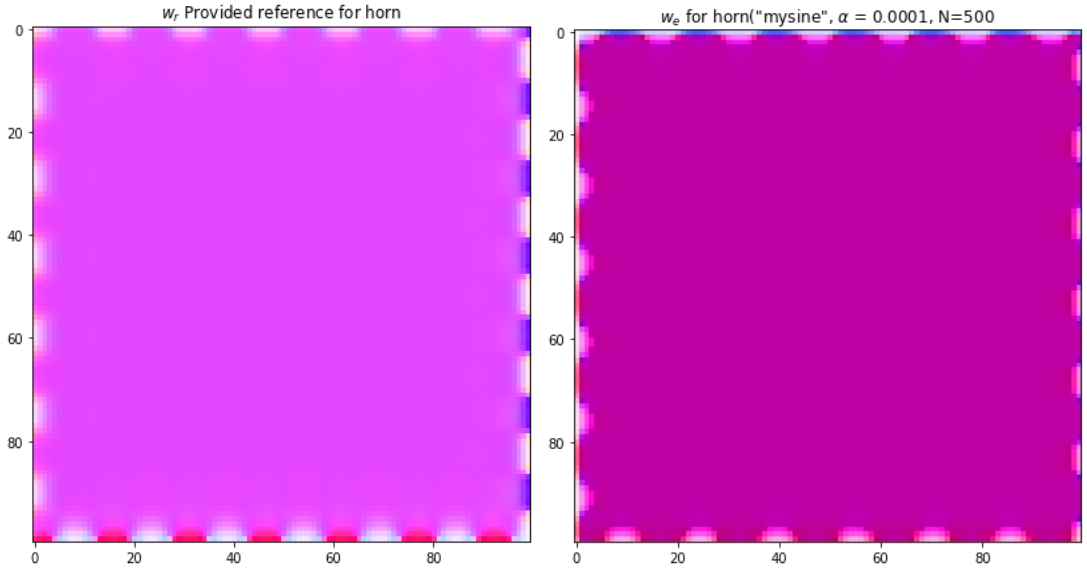


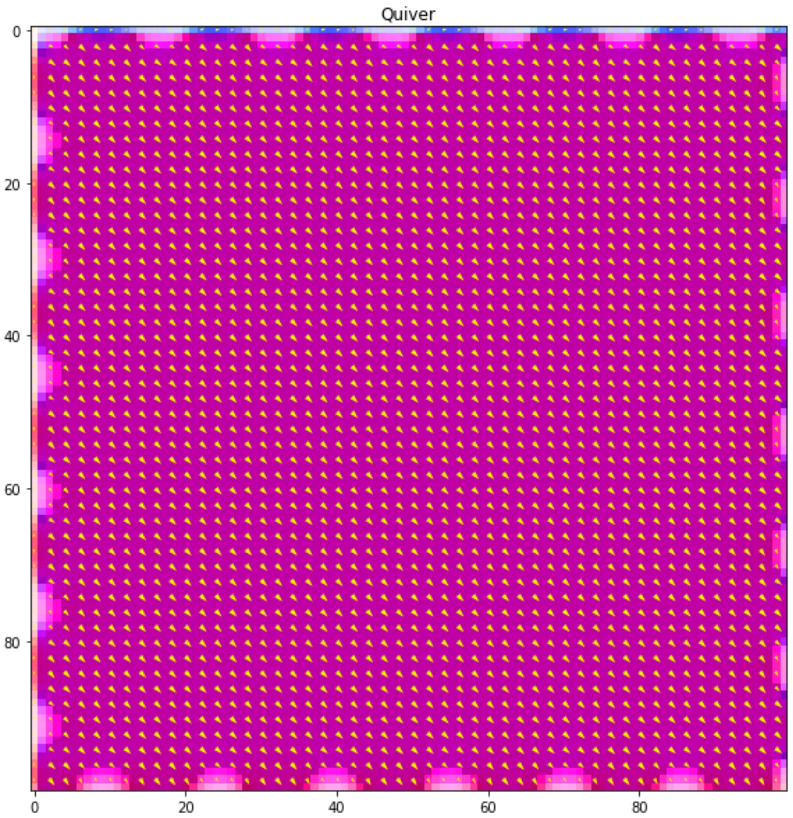


We get an optimal error rates for alpha value around 0.0001 and N >= 500.
With these optimized Horn method, we get an angular error **AP = 0.173,** and the Euclidean error **EPE = 0.077** and a mean squared error **MSE = 0.029**.

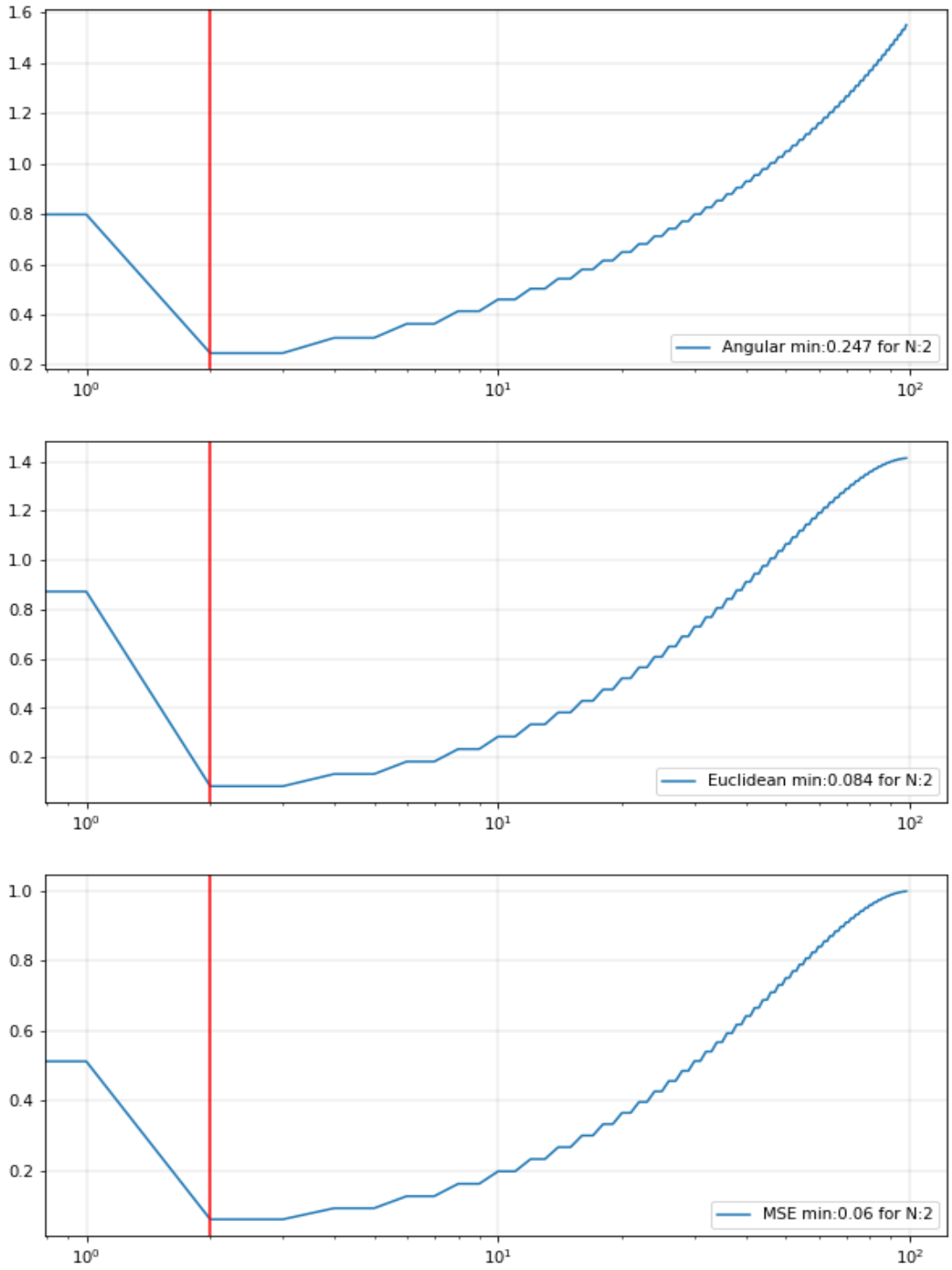
The results below show the application of these optimized Horn on mysine :





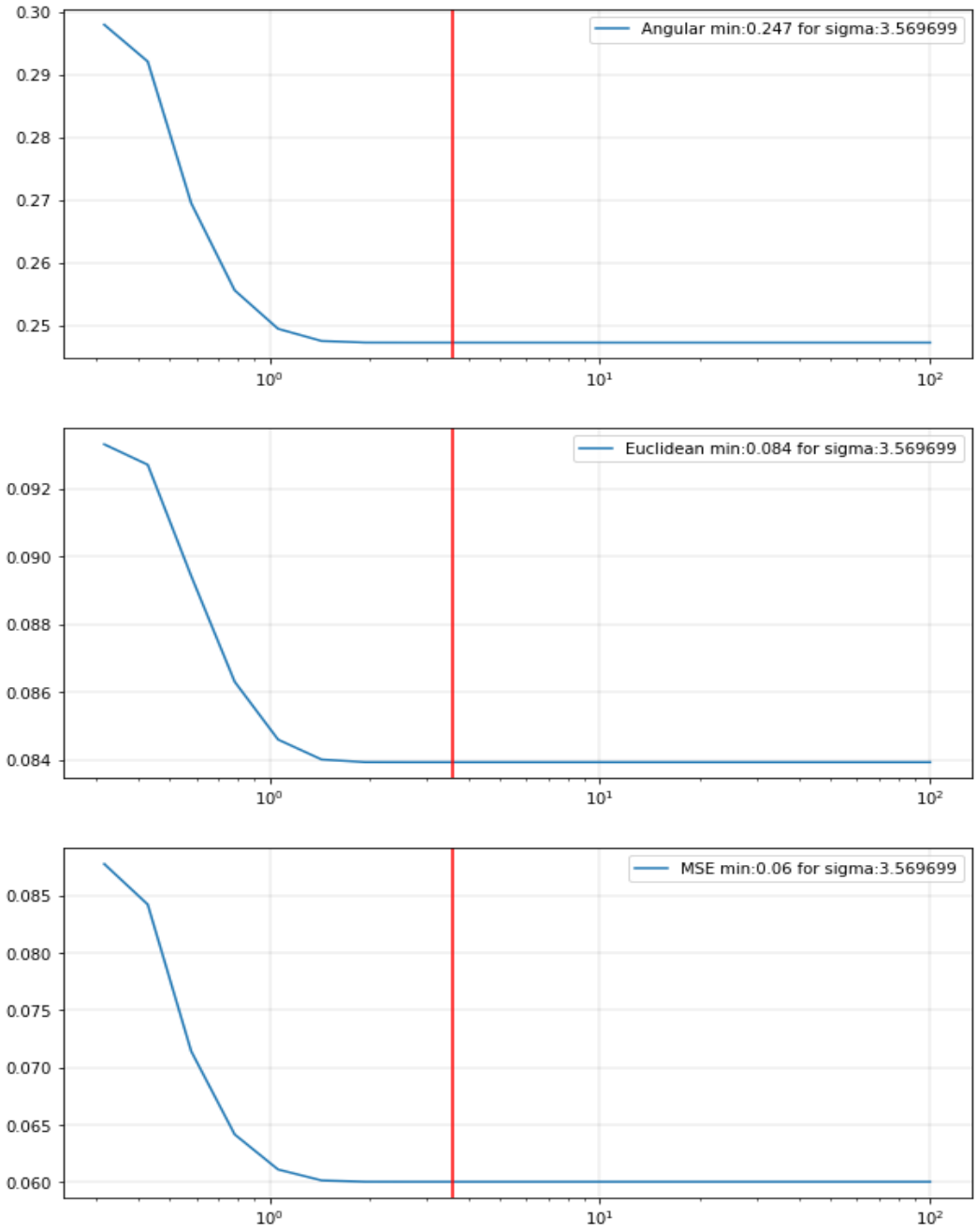


Optimizing Lucas



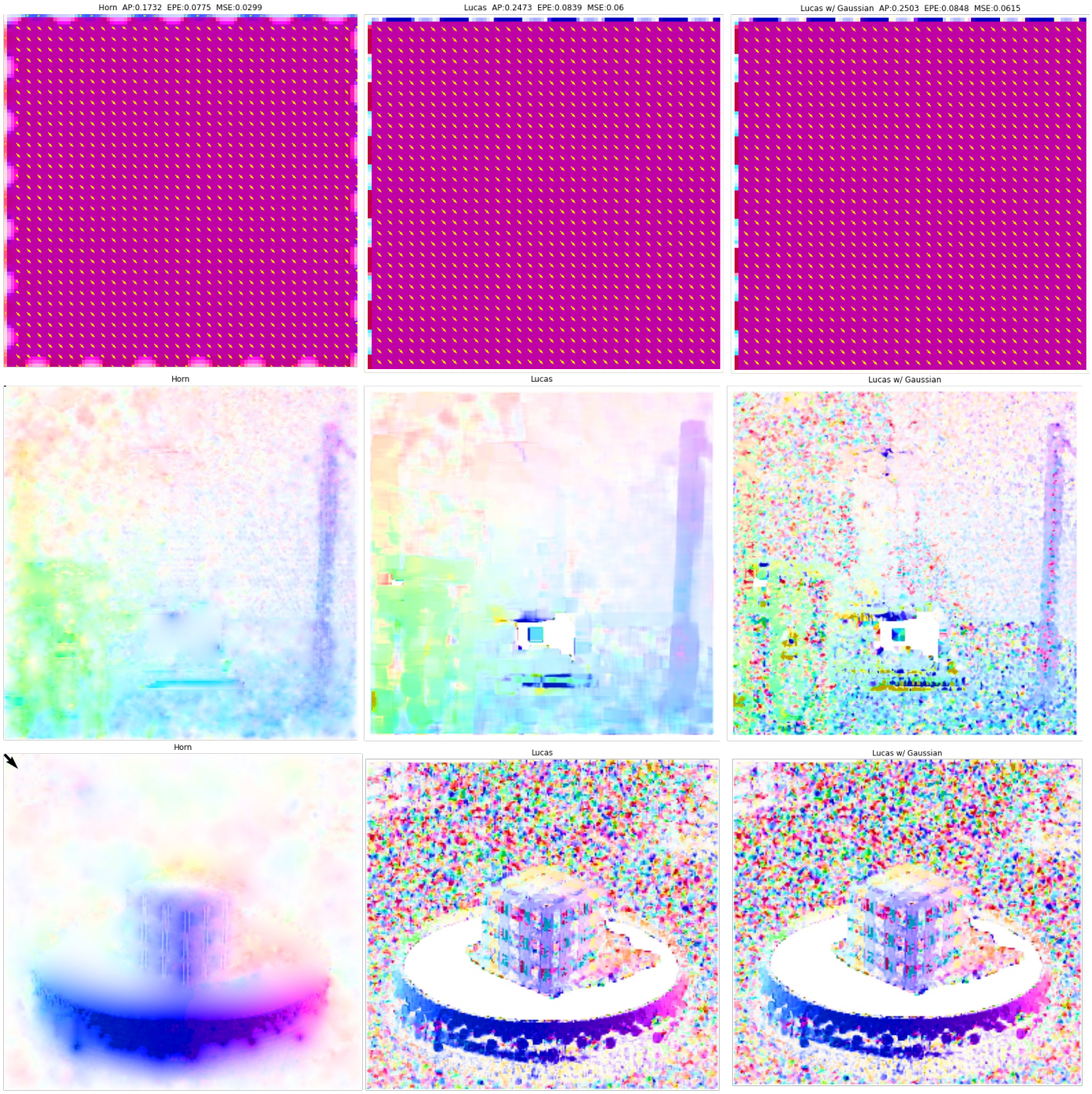
Unfortunately we didn't see any improvement as the value we choose for the window size was exactly the same, But the results of the optimization confirms that choice of N = 2 as the optimal values for the mysine dataset.
With these optimized Lucas method, we get an angular error **AP = 0.247,** and the Euclidean error **EPE = 0.084** and a mean squared error **MSE = 0.060**.

Optimizing Lucas w/ Gaussian



We didn't see a huge improvement by using the gaussian weighting in the lucas method applied on the mysine dataset but we alteast get the best sigma value to choose to get similar results as the best lucas method without gaussian weighting.
With these optimized Lucas with gaussian method, we get an angular error **AP = 0.247,** and the Euclidean error **EPE = 0.084** and a mean squared error **MSE = 0.060** which are the same as the non gaussian method atleast for the case of mysine dataset.

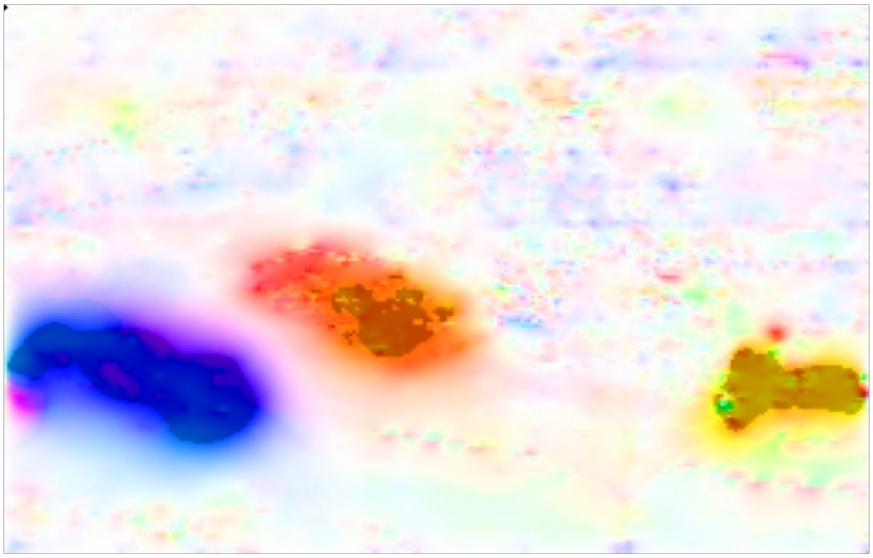
Comparison : **Horn** VS **Lucas** (VS **Lucas w/ Gaussian**) on different datasets



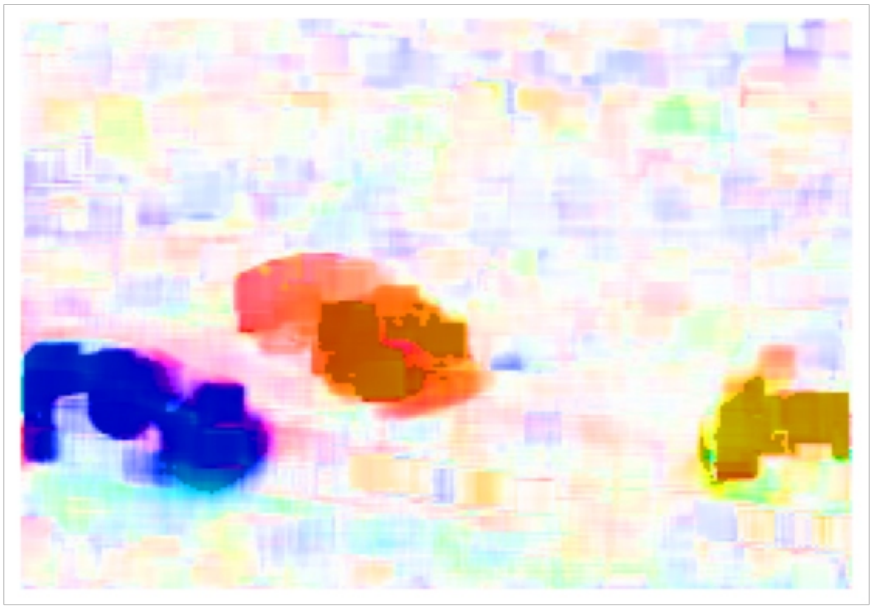
We can see that overall Horn is gives us pretty smooth results when we optimize the parameter alpha and N either empircally by trying out diffrent values or by using a ground truth to compare errors (AP, EPE, MSE).
Overall Lucas also works but we get a lot of errors in areas where the moving window takes all the noise and yield bad results in the velocity map. We are able to reduce some of it by applying a gaussian filtering in the neighborhood of the window W of the lucas method, but we are not able to get as good results as with Horn method.

Comparison on Taxi Data

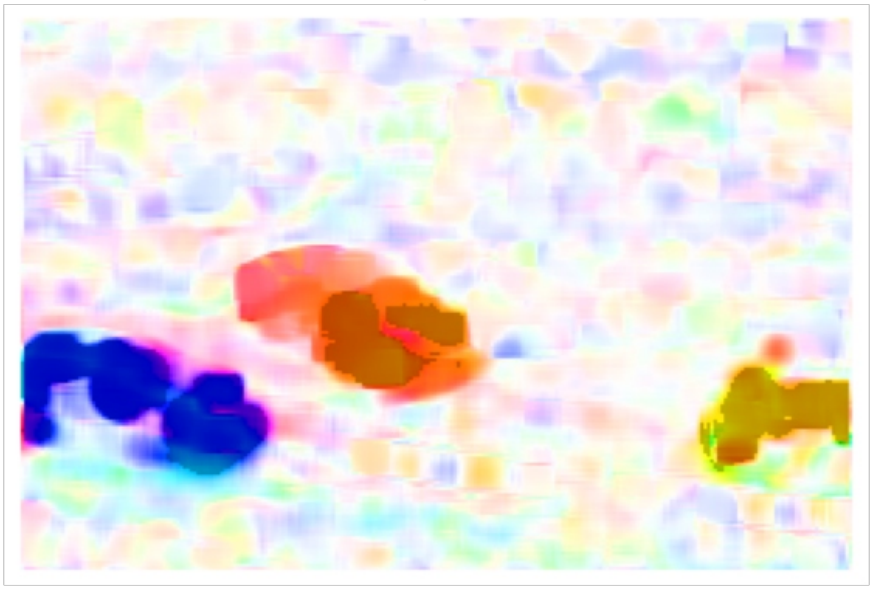
Horn



Lucas

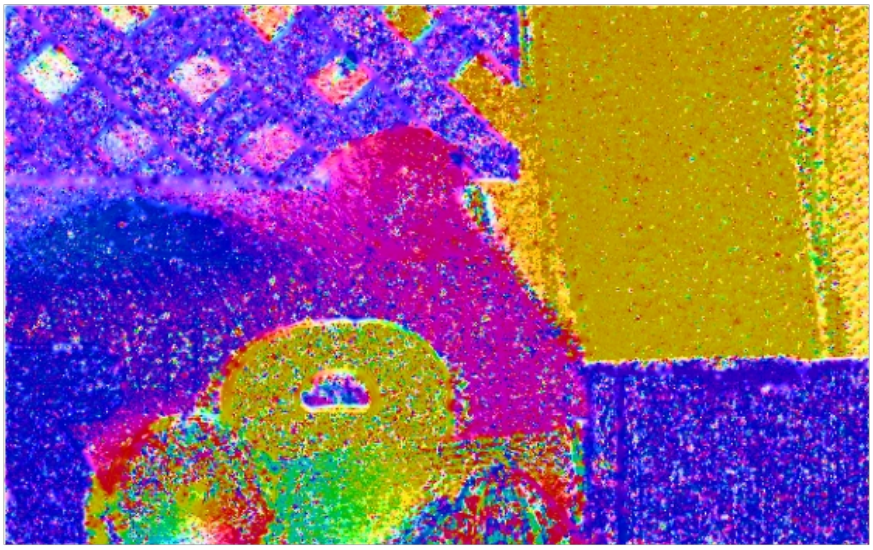


Lucas w/ Gaussian

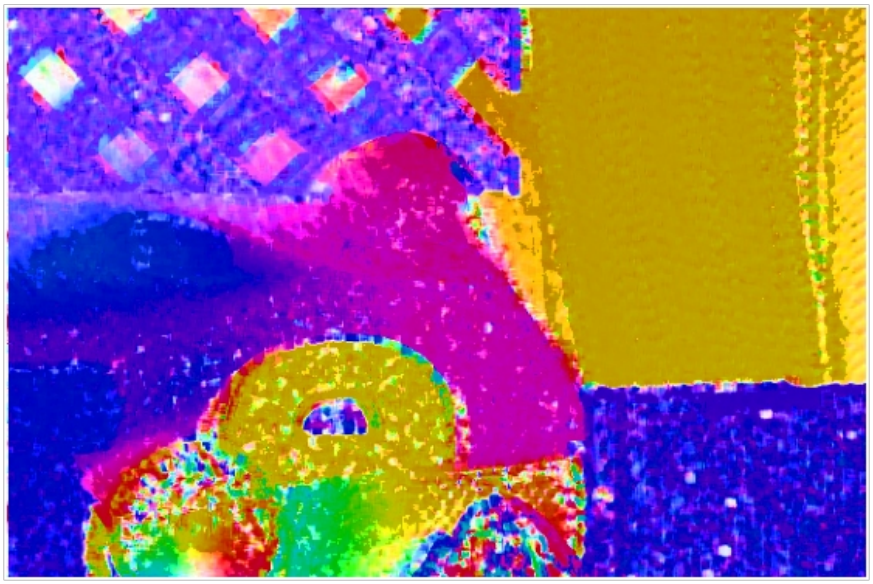


Comparison on Rubberwhale Data

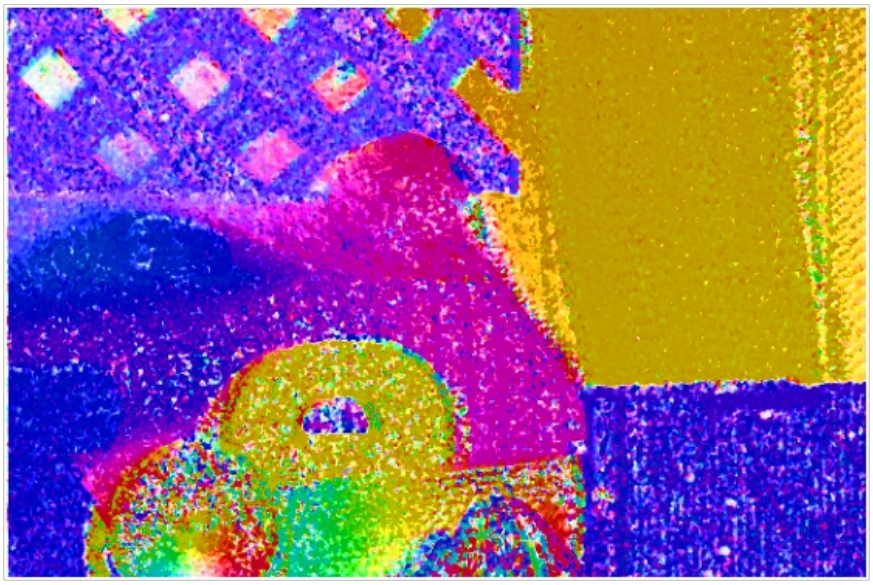
Horn AP:1.5769 EPE:37676248.3242 MSE:4.4401885953755144e+16



Lucas AP:1.5936 EPE:37676248.0257 MSE:4.4401885958807256e+16

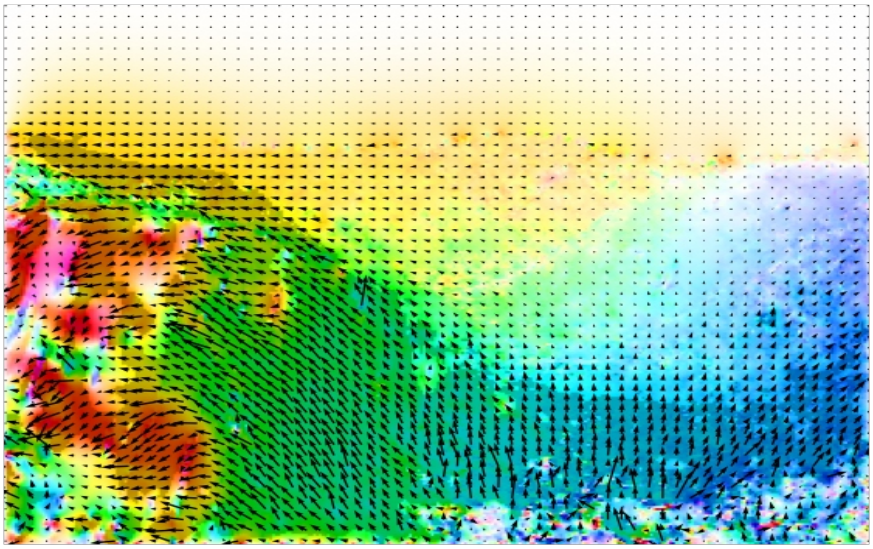


Lucas w/ Gaussian AP:1.5897 EPE:37676248.1329 MSE:4.440188595782636e+16

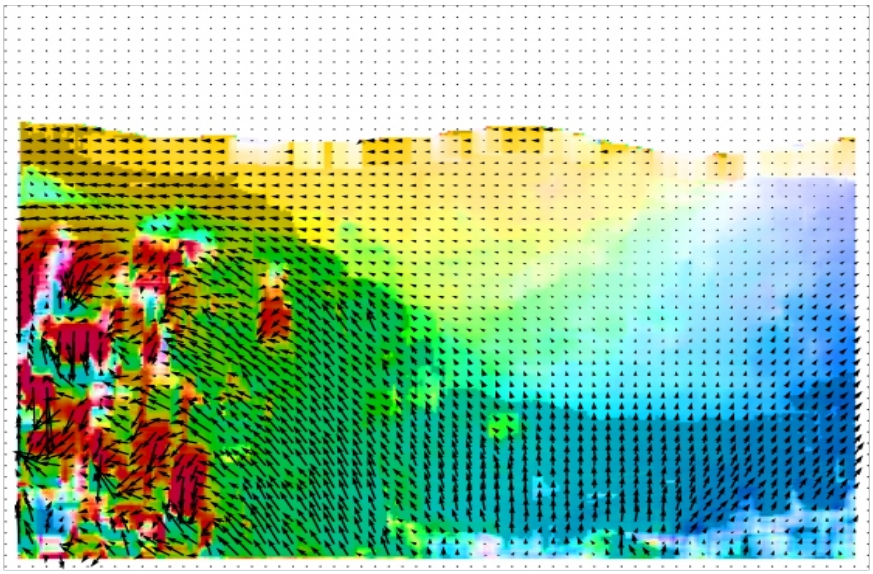


Comparison on Yosemite Data with quiver activated

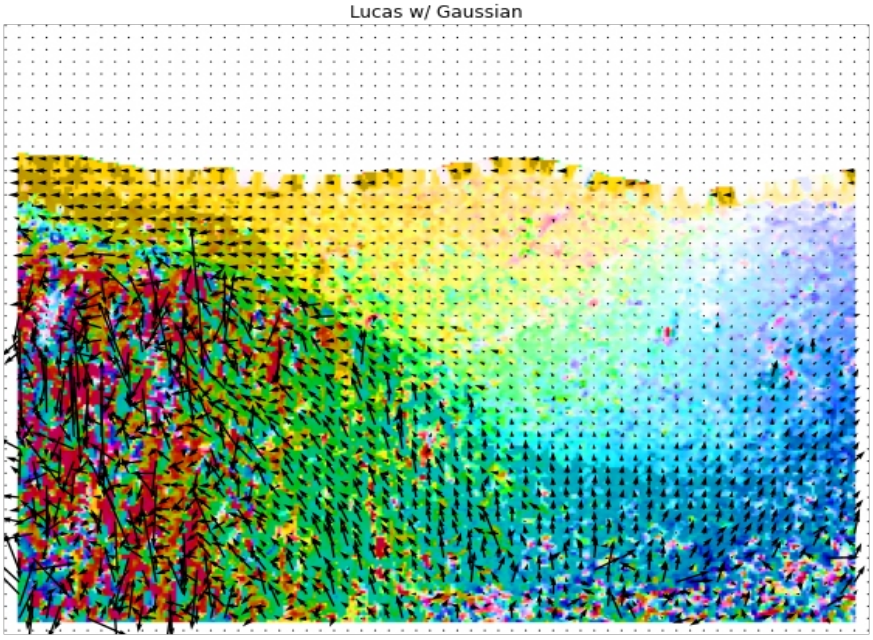
Horn



Lucas



Lucas w/ Gaussian



Optical Flow Animated (GIFs)

I tried to create some Animation using dataset for optical flow that i found online. You can find these animations on Youtube in this Playlist: *https://youtube.com/playlist?list=PL4EQh9QTnivpzw3PYcNtnb8g7W405Jgia*

Conclusion :
If we want to work on scenes that have a non homogeneous textures like trees, grass, roads or other we might need to consider the Horn method because it provides good results in thoses cases.

For scenes where we have quite homogeneous forms or objects we can consider the Lucas-Kanade method which gives quite good results. But there is a drawback to this method, which is that if we have objects of different size we first need to find a fitting window size in order to keep the information on the object of interest. For example if we have balls of radius 5px and we use a window of size 20px the optical flow of the ball will be lost in the velocity maps. We can actually see this in the results on **nasa** data above, where horn kept the velocity map quite homogeneous, we get patch/box like articfacts in the Lucas-Kanade method (indicating the use of a window of a certain size). In the **rubberwhale** and the **taxi** data we can see the improvments brought by the gaussian weighting in the lucas-Kanade method. We can see that these box like artifacts are much smoother and the edges of objects are much better preserved.

These methods allowed us to study the optical flow from only two frames without any prior. We can imagine an optimisation of these methods using a hybrid approch. Or by using a segmentation of the studied objects as a prior in order to construct an adaptive window approch (using bounding boxes around each important objects) in order to preserve the details for elements of different size when we are using the Lucas-Kanade approach.