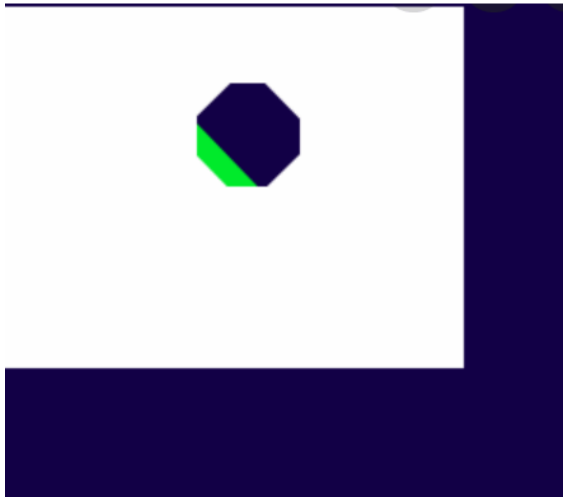
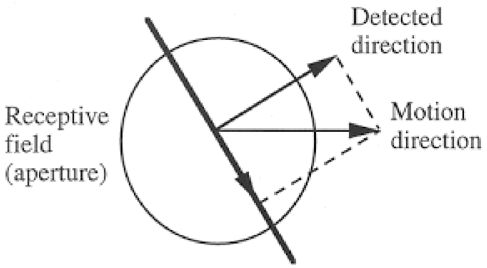
BAYESIAN INFERENCE OF MODELS ON EVENT-BASED OPTICAL FLOW TO SOLVE APERTURE PROBLEM

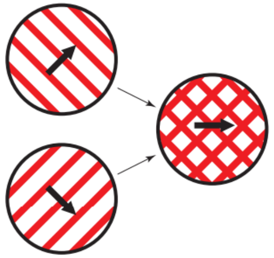
Weijie Qi, Florian Jäger

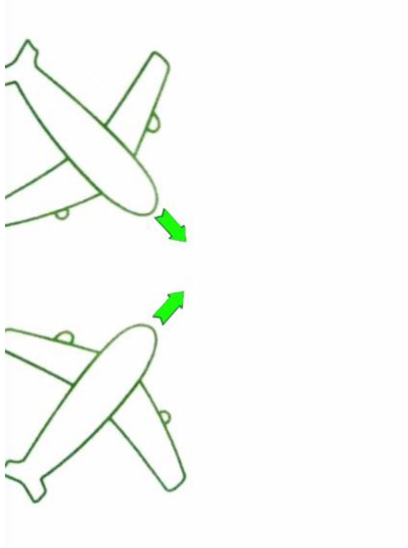
Our research question:

Two types of aperture problems:







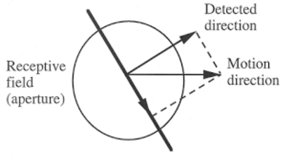


One object

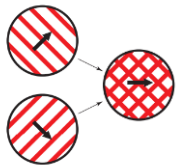
Two objects

Our research question:

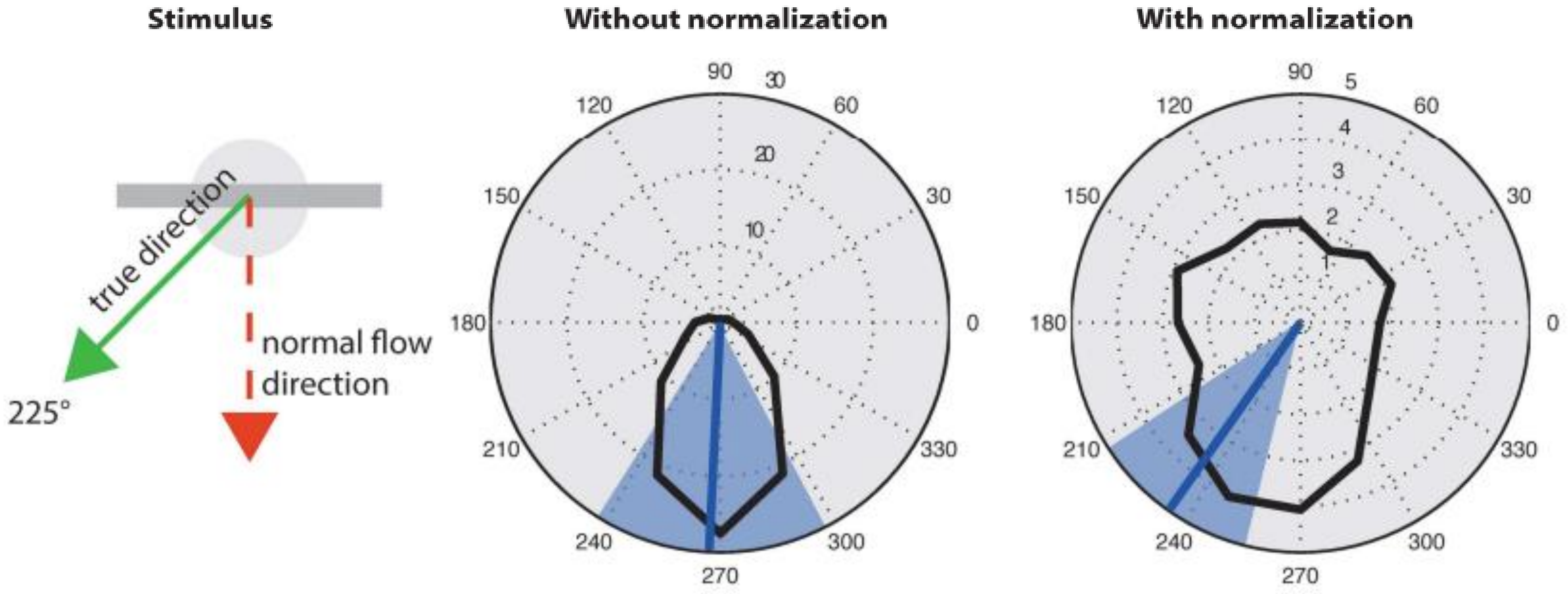
Paper “*On Event-based Optical Flow Detection*” has a solution to solve the first aperture problem.



One object



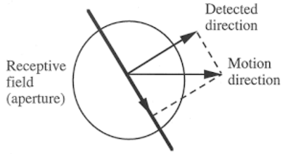
Two objects



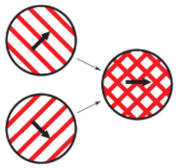
On Event-based Optical Flow Detection, Figure 9

Our research question:

Paper “*On Event-based Optical Flow Detection*” has a solution to solve the first aperture problem.



One object



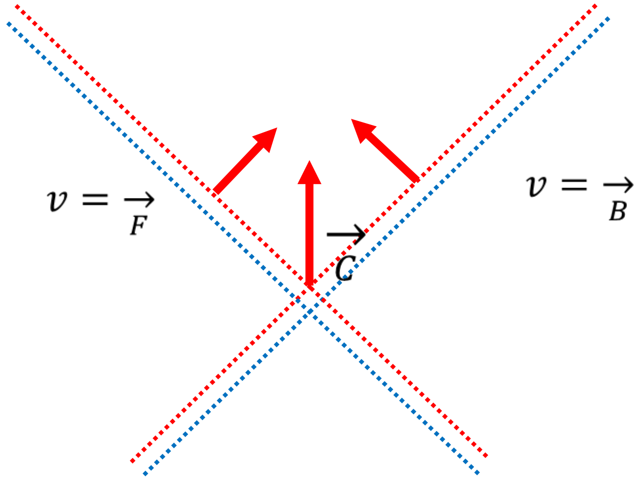
Two objects

When there are two moving objects,

particularly when they have overlapping part in the space,

how to solve the second aperture problem?

Our solution:



The detected velocity:

Vector sum of two velocities based on parallelogram law

$$\vec{C} = \alpha \vec{F} + \boldsymbol{\beta} \vec{B}$$

Our solution:

Inspired by the paper *Bayesian Estimation of Layers from Multiple Images*, we put an alpha matte towards all pixels and model each pixel as a combined effect from the object 1 and object 2

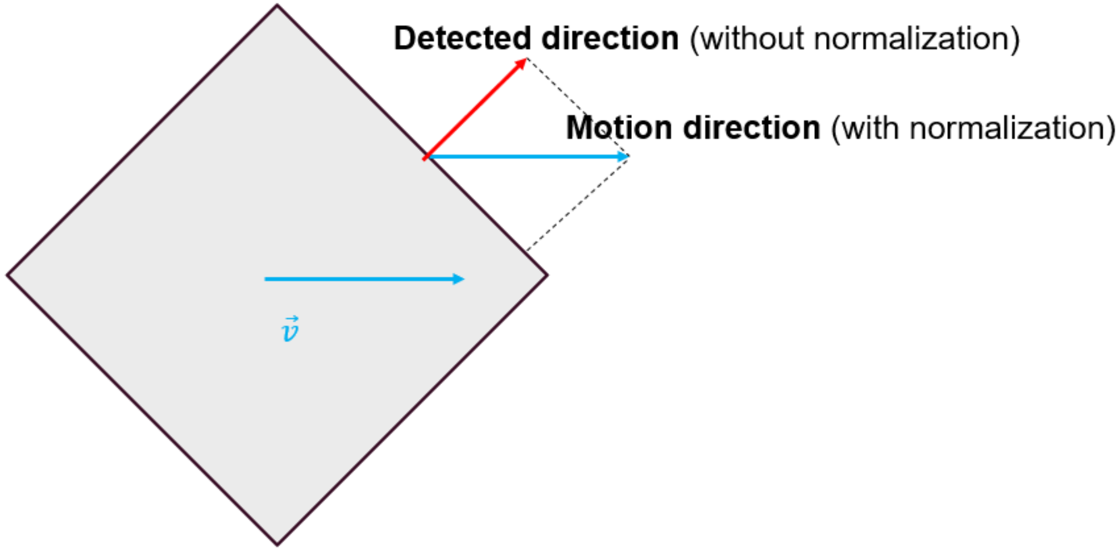
Input: Cx, Cy, Fx, Fy, Bx, By

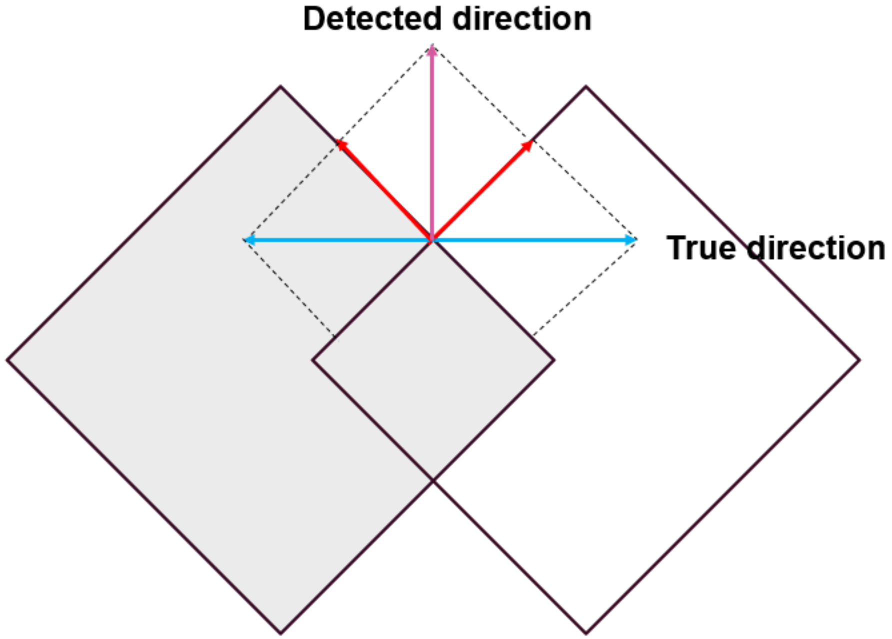
Output: α, β, fx, fy, bx, by

Alpha matte: a concept from digital image processing

Our solution:

Why we still need the unnormalized optical flow for the first step:





With normalization

Without normalization

Still need the unnormalized velocities to sum up to the detected velocity.

Our solution:

Overall step:

1、get the unnormalized optical flow

2、motion segmentation into two objects

3、build a Bayesian framework

4、use MAP estimate to recover the two optical flows

5、normalization?

Implementation:

The dataset we use:

(a) APS Data (b) DVS Data

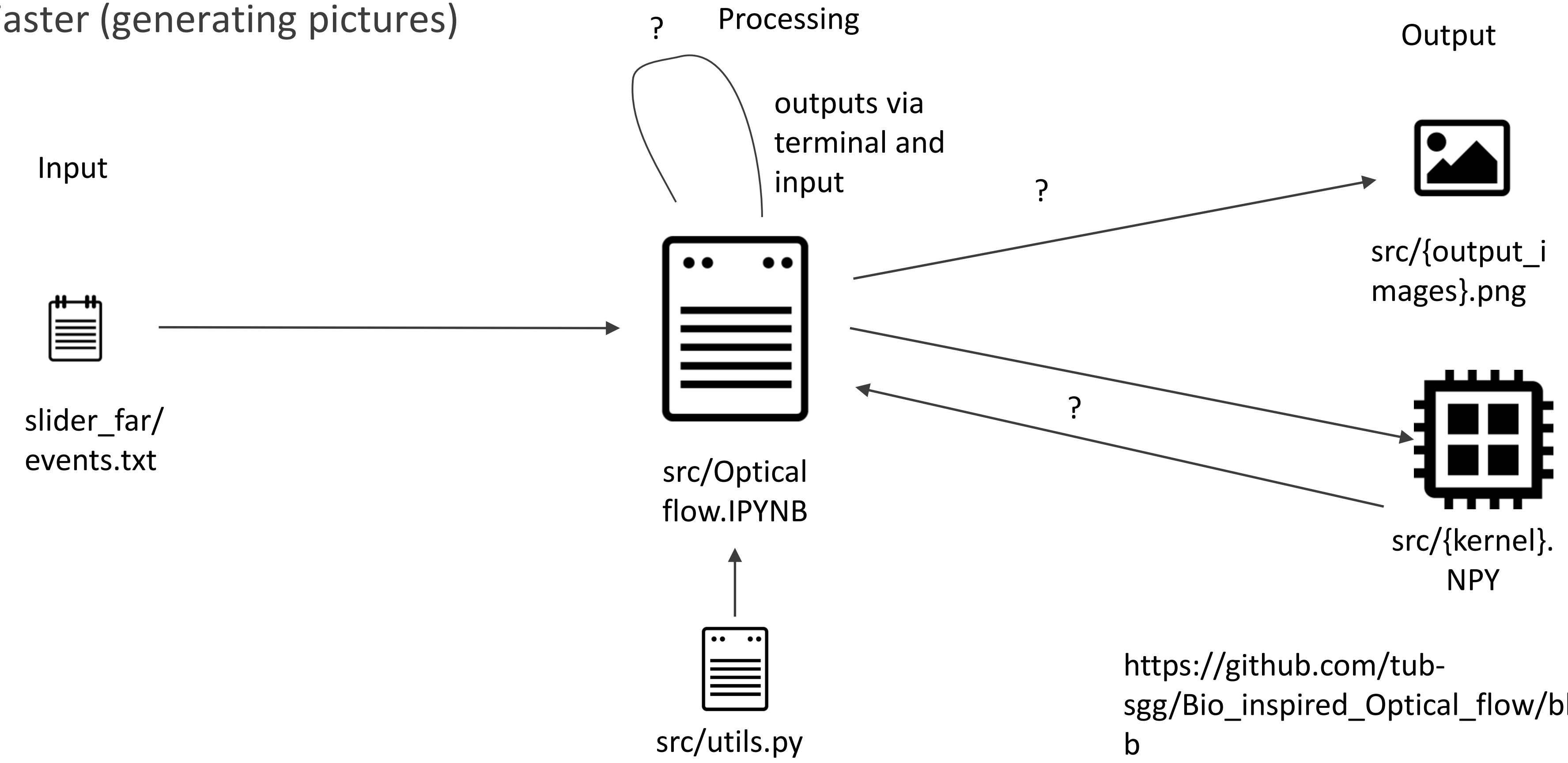
DVSMOTION20 copyrighted by PI Keigo Hirakawa from University of Dayton

Computing optical flow

Code restructuring

Why?

- Easier to understand, change parameters, execute via terminal, (problems with executing parts of the code)
- Faster (generating pictures)

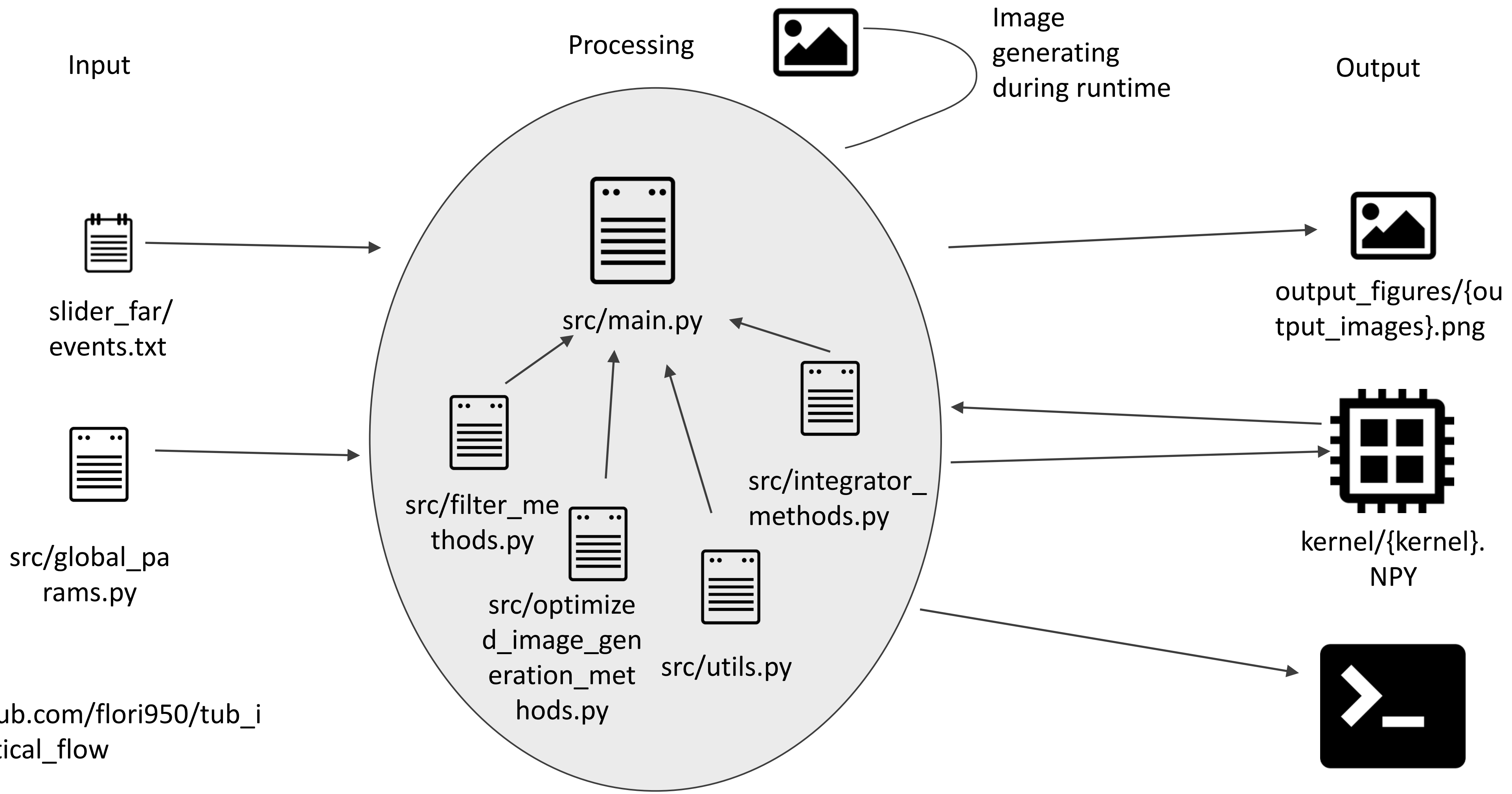


https://github.com/tub-sgg/Bio_inspired_Optical_flow/blob/master/src/optical_flow.ipynb

Code restructuring

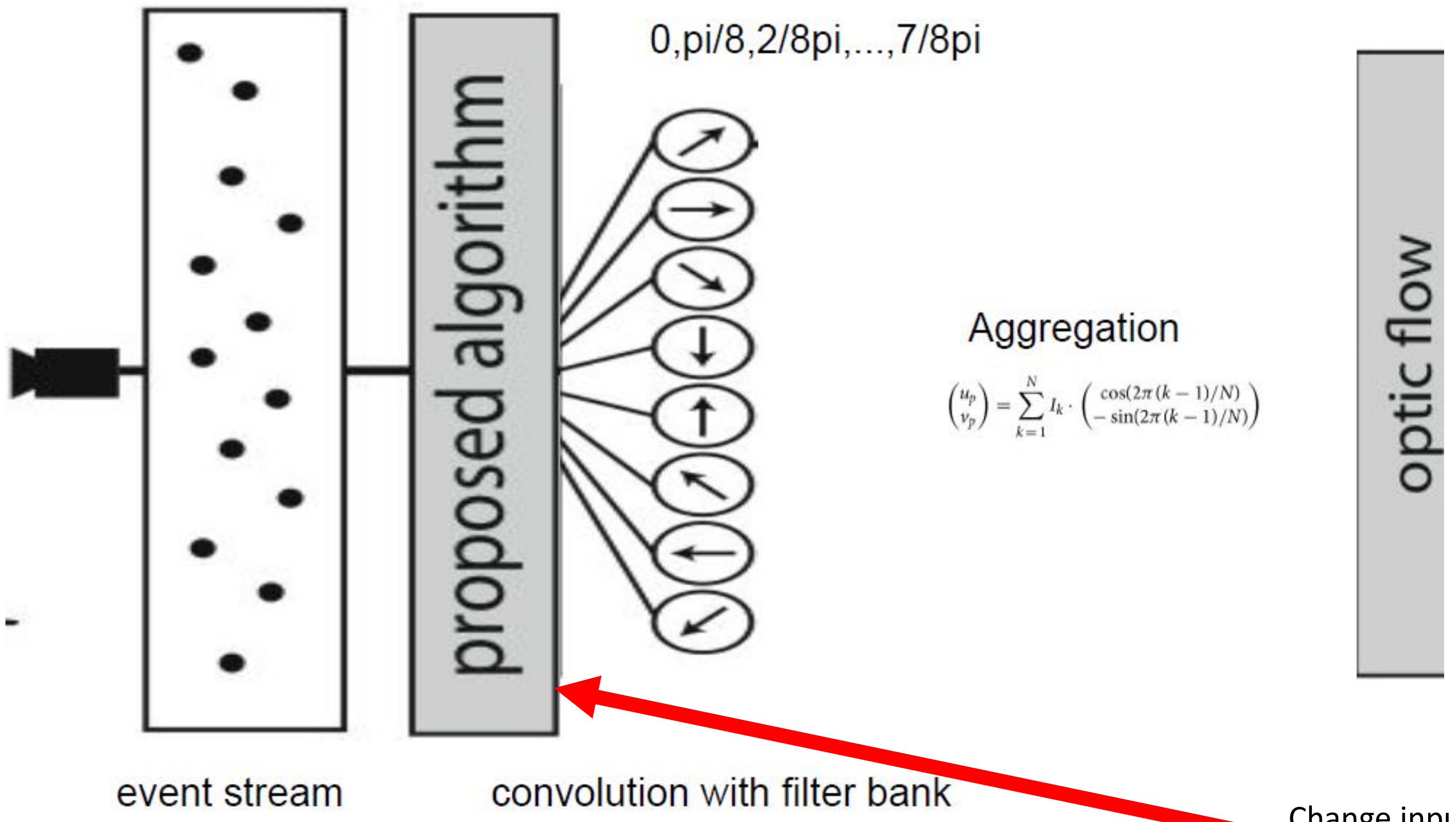
- Output folder wasn’t included
- Unused methods couldn’t be recognized
- Undocumented methods and parameters
- Print-outputs in the terminal are useless
- Kernel files in the src folder
- ...

Code restructuring



https://github.com/flori950/tub_inspired_optical_flow

Changing the filterbank



slider_far/events.txt (original)
slider_far/output.txt (Moving Hand)

Change input-parameters
Add new filter(s)

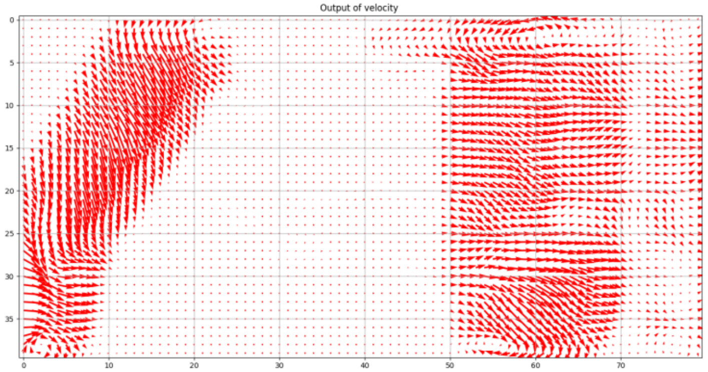
Changing input-parameters

src/global_params.py

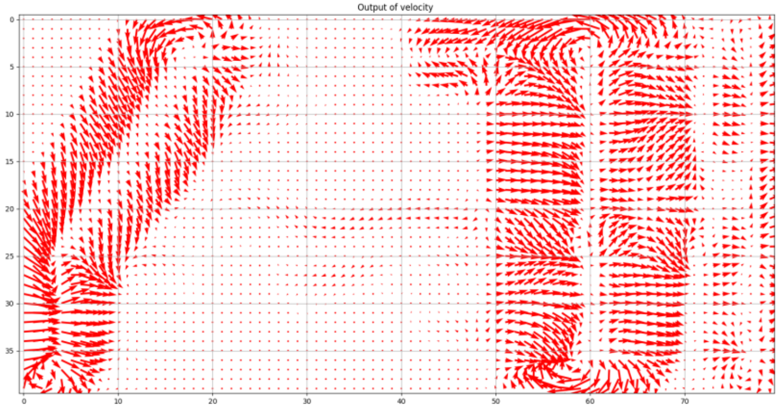
- `scale_factor = 0.1`
- `sigma = 3`

Unchanged:

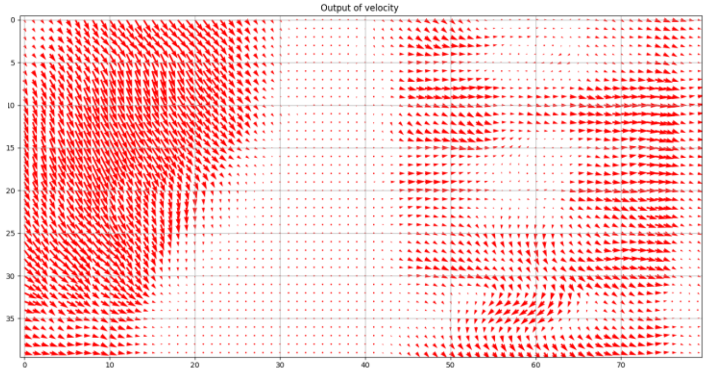
- `band_height = 40`
- `band_width = 80`
- `offset_height = 20`
- `offset_width = 70`



original generated



scale_factor = 1



sigma = 25

Adding a new filter

src/main.py

Include the methods

src/filter_methods.py

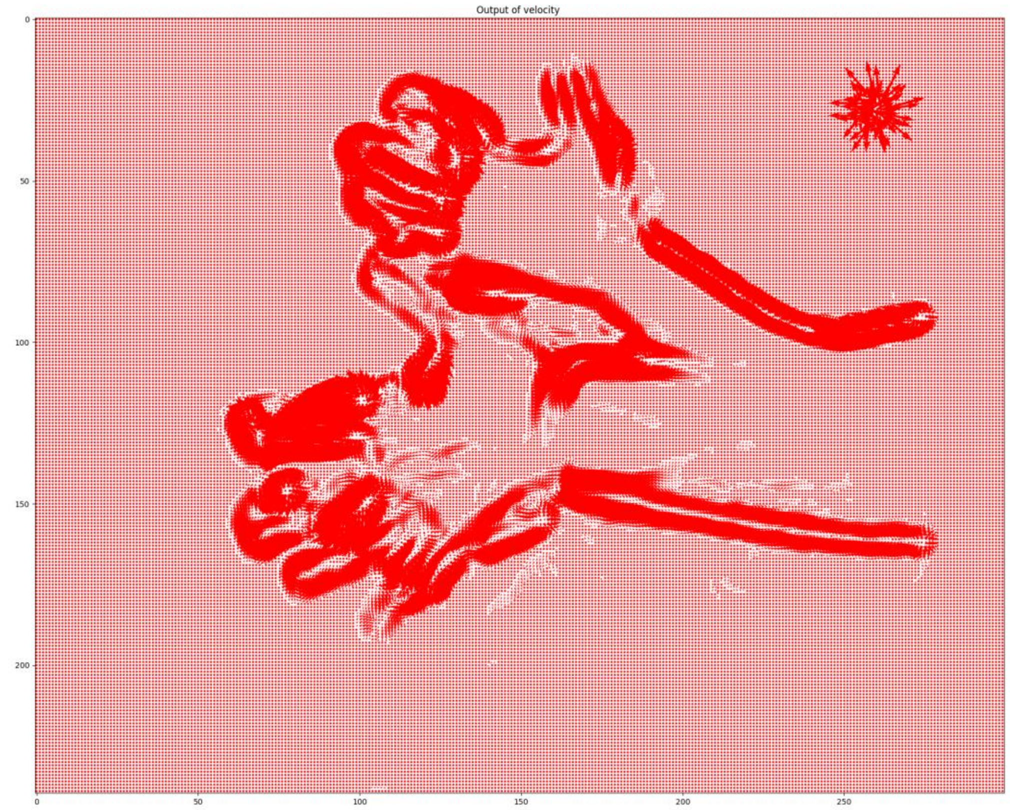
Adding new Gabor Filter, temporal Filter
Adjust filter_bi, filter_mono

src/global_params.py

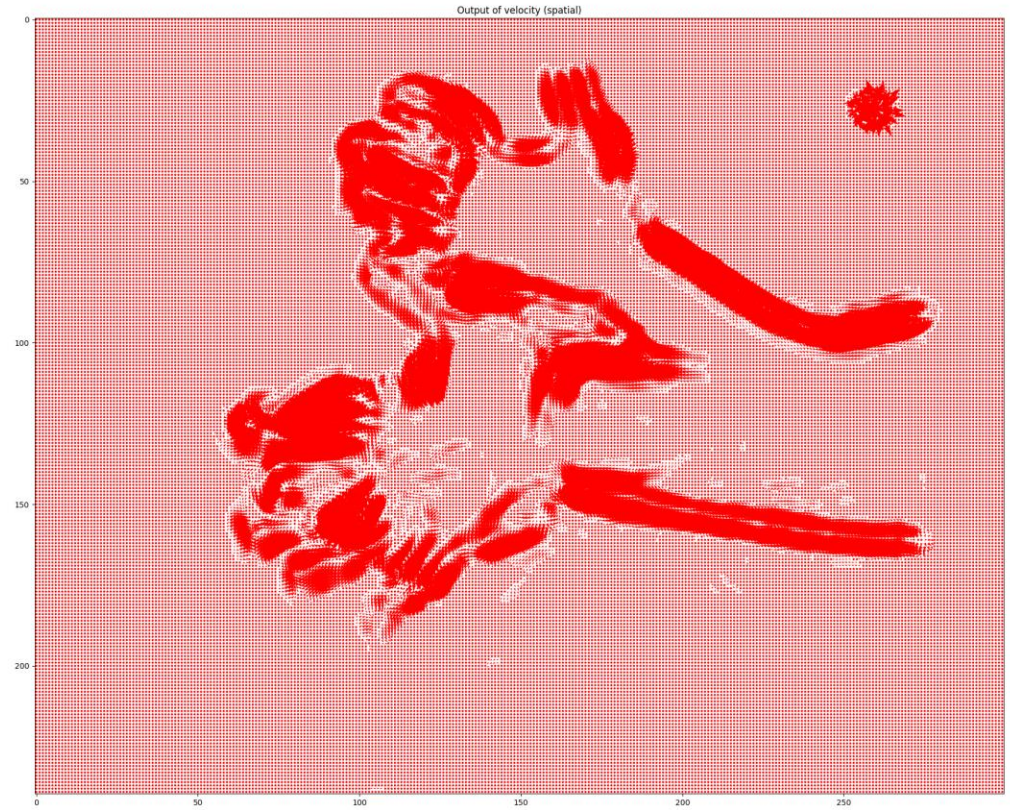
adding specific filter parameters

```python
###############################################

def spatial_gabor_filter_even(x, y, sigma, theta, f0x, f0y):
    x_hat = np.cos(theta) * x + np.sin(theta) * y
    y_hat = -np.sin(theta) * x + np.cos(theta) * y

    gabor_first = np.exp(-1 * ((x_hat - f0x)**2 + (y_hat - f0y)**2) * (2 * math.pi**2) / sigma**2)
    gabor_second = np.cos(params.xi0 * (f0x * x_hat + f0y * y_hat))
    return (params.xi0 / sigma**2) * gabor_first * gabor_second



def spatial_gabor_filter_odd(x, y, sigma, theta, f0x, f0y):
    x_hat = np.cos(theta) * x + np.sin(theta) * y
    y_hat = -np.sin(theta) * x + np.cos(theta) * y

    gabor_first = np.exp(-1 * ((x_hat - f0x)**2 + (y_hat - f0y)**2) * (2 * math.pi**2 / sigma**2))
    gabor_second = np.sin(params.xi0 * (f0x * x_hat + f0y * y_hat))
    return (params.xi0 / sigma**2) * gabor_first * gabor_second


###############################################

def filter_bi_spacial(t_spatial):
    return -1 * params.scale_bi1 * temporal_filter(t_spatial, params.bi1_mean(), params.bi1_sigma())+ params.scale_bi2 * temporal_filter(t_spatial, params.bi2_mean(), params.bi2_sigma()

def filter_mono_spacial(t_spatial):
    return temporal_filter(t_spatial, params.mono_mean(), params.mono_sigma())

###############################################

def temporal_filter(t, mu, sigma):
    temp_filter = np.exp(-(t - mu)**2 / (2 * sigma**2))
    return temp_filter
```

Adding a filter



Original generated

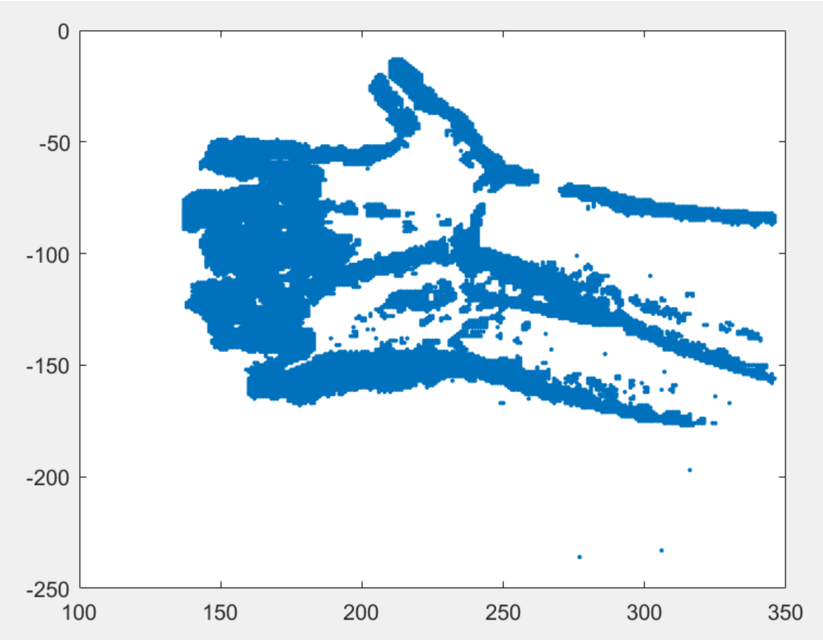


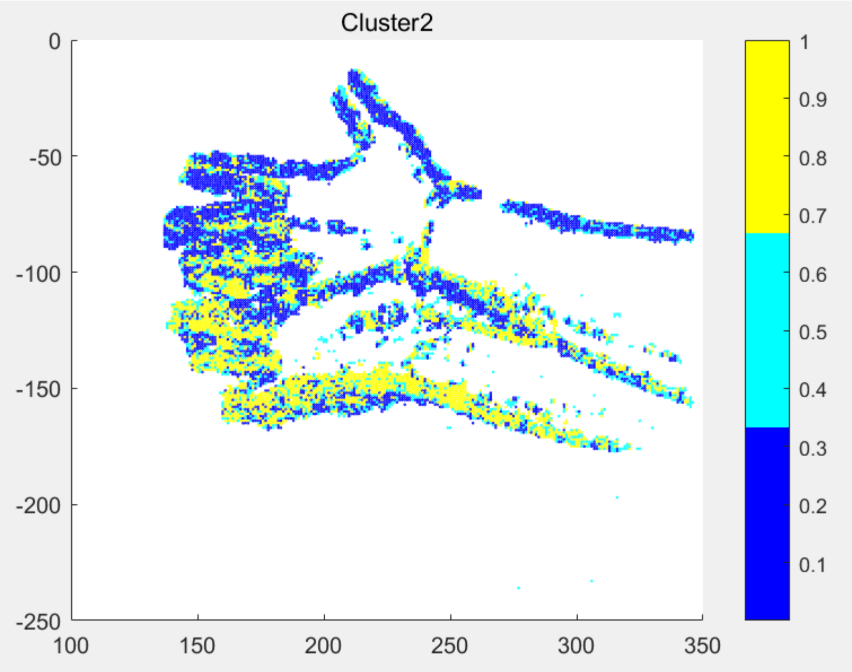
Spatial generated

Implementation:

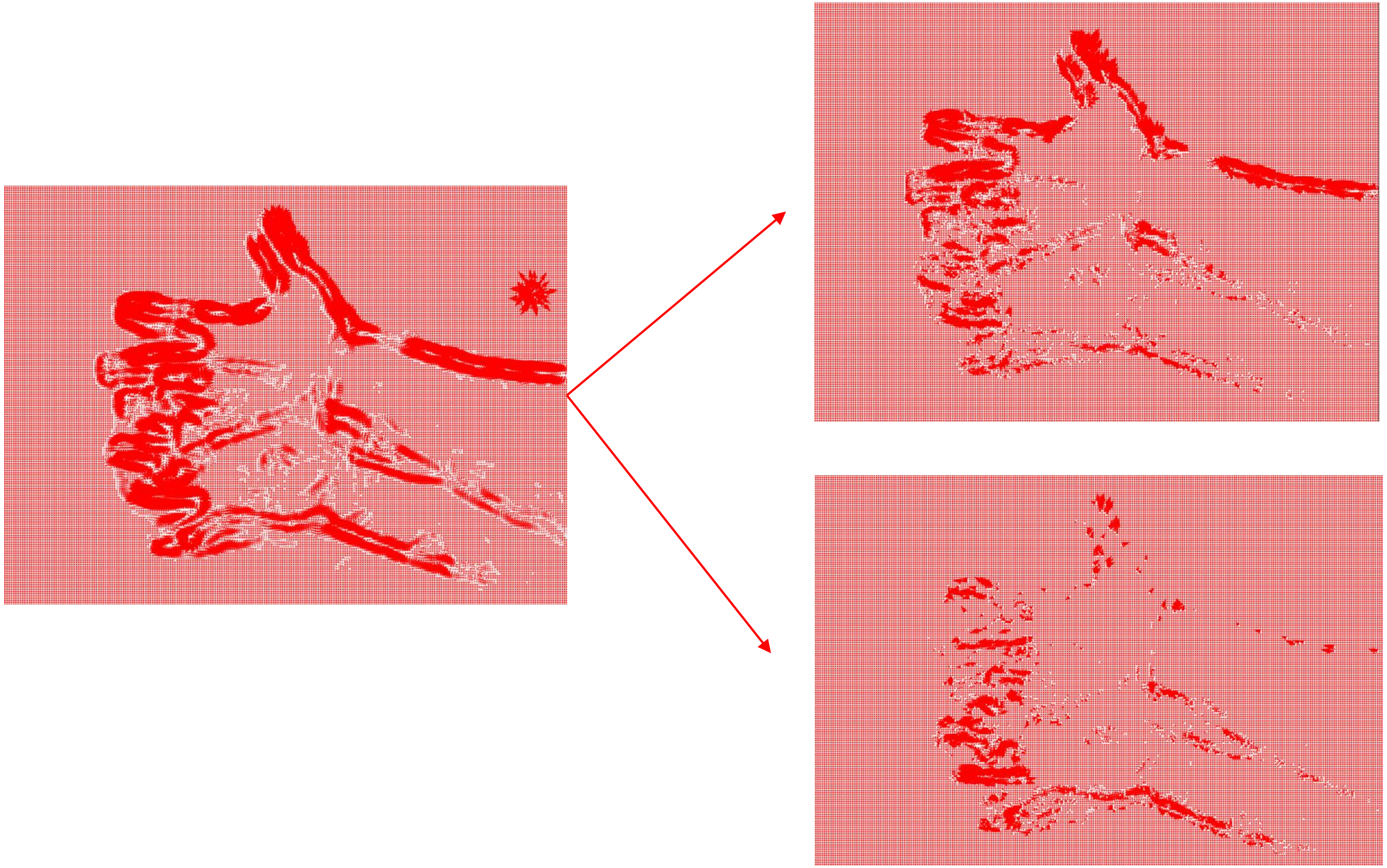
We referenced the code available on GitHub which is the MATLAB code for paper "Event-Based Motion Segmentation by Motion Compensation" https://github.com/remindof/EV-MotionSeg

We choose a subset from the dataset when the two waving hands have overlapping parts in space.

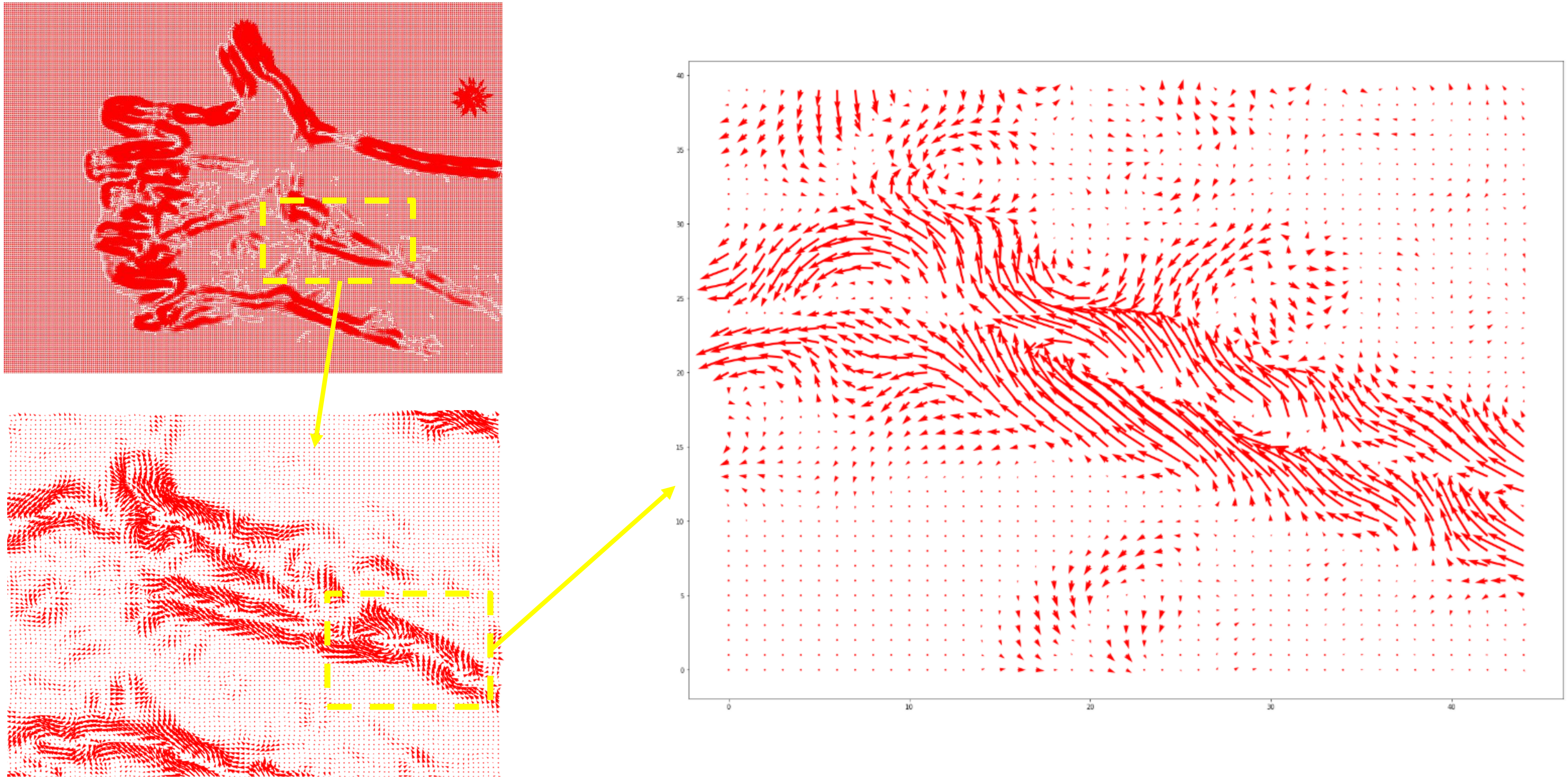




Implementation:



Implementation:



Implementation:

Bayesian framework

To minimize: $$L(\alpha, F) = \sum_{i=1}^{N} \iint_{\mathbf{x} \in \mathbb{R}^2} \left\| C(\mathbf{x}) - \left(\alpha(\mathbf{x}) \cdot F(\mathbf{x}) + (1 - \alpha(\mathbf{x})) \cdot B^i(\mathbf{x})\right) \right\|^2 \mathrm{d}\mathbf{x}$$

Cost function: $$E(\alpha, F) = L(\alpha, F) + R_\alpha(\alpha) + R_F(F)$$

Implementation:

Bayesian framework

From paper: “*Bayesian Estimation of Layers from Multiple Images*”

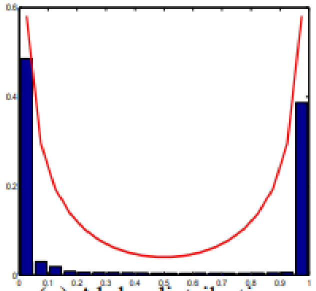
Prior 1: Bounded reconstruction

$$0 \leq \alpha \leq 1$$

$$0 \leq F \leq 1$$

Prior 2: Measured α distribute

$$E_{\mathrm{d}} = \frac{1}{n} \sum_i (\alpha F + (1-\alpha)B - C_i)^2 + \frac{\rho}{\beta(\eta,\tau)} \alpha^{\eta-1}(1-\alpha)^{\tau-1}$$



(c) Alpha distribution

Prior 3: Spatial consistency

$$W_{\mathbf{p}} = \exp\left(-\frac{\mathbf{p}^\top \mathbf{G} \mathbf{p}}{\mathbf{p}^\top \mathbf{p}}\right)$$

$$\sum_{\mathbf{x}} \sum_{\|\mathbf{p}\|<=1} (W_{\mathbf{p}}(\alpha(\mathbf{x}+\mathbf{p}) - \alpha(\mathbf{x})))^2$$

Implementation:

Bayesian framework

Prior 1: Bounded reconstruction

$$0 \leq \alpha \leq 1$$
$$0 \leq F \leq 1$$

$$0 \leq \alpha \leq 1$$
$$0 \leq \beta \leq 1$$

```
from scipy.optimize import minimize
```

```
res = minimize(fun, x0, method='SLSQP',constraints=cons)
```

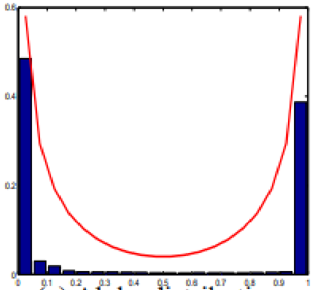
```
cons = ({'type': 'ineq', 'fun': lambda x: x[0:2*w] - e },\
        {'type': 'ineq', 'fun': lambda x: -x[0:2*w] + 1},\
```

Implementation:

Bayesian framework

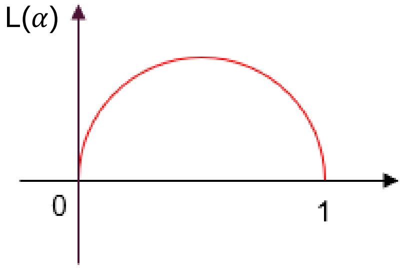
Prior 2: Measured α distribute

$$E_{\mathrm{d}} = \frac{1}{n}\sum_i (\alpha F + (1-\alpha)B - C_i)^2 + \frac{\rho}{\beta(\eta,\tau)}\alpha^{\eta-1}(1-\alpha)^{\tau-1}$$



(c) Alpha distribution

Integrate into loss function



Implementation:

Bayesian framework

Prior 3: Spatial (in)consistency

On α:

$$W_{\mathbf{p}} = \exp\left(-\frac{\mathbf{p}^\top \mathbf{G} \mathbf{p}}{\mathbf{p}^\top \mathbf{p}}\right)$$

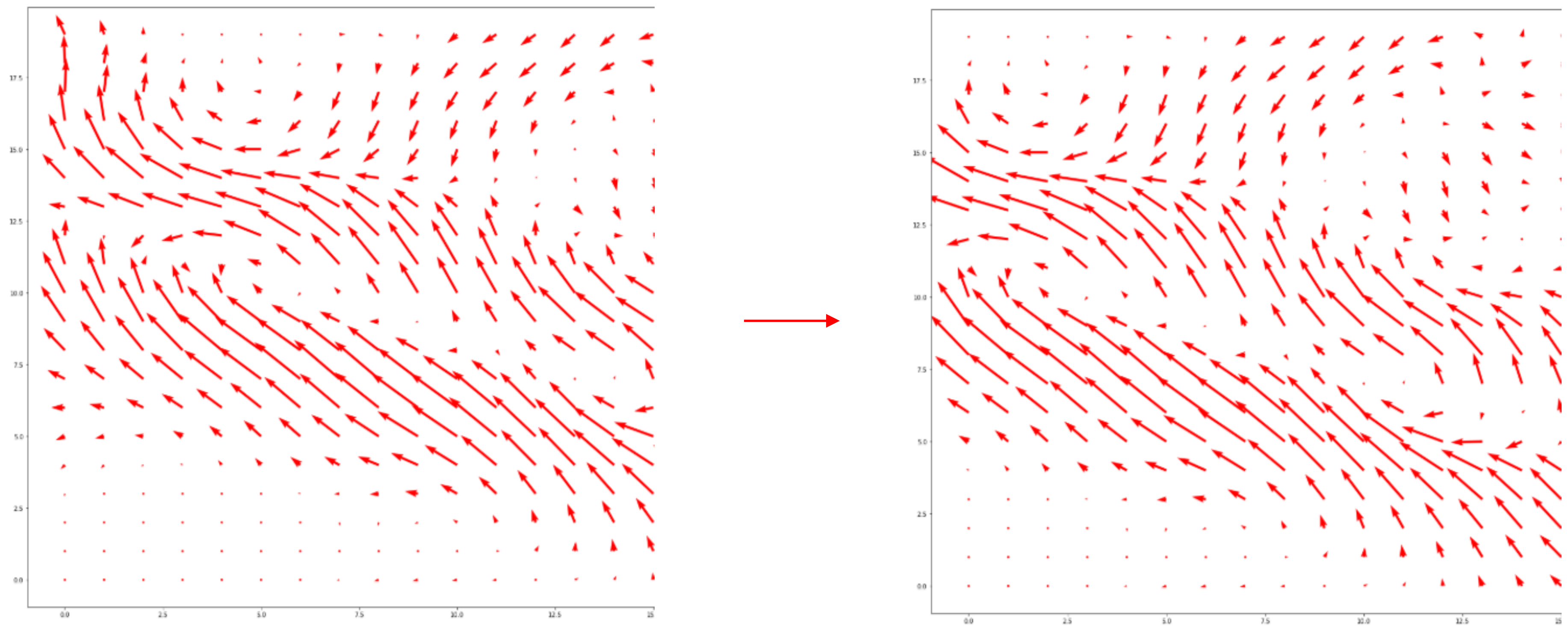
$$\sum_{\mathbf{x}} \sum_{\|\mathbf{p}\|<=1} \left(W_{\mathbf{p}}(\alpha(\mathbf{x}+\mathbf{p}) - \alpha(\mathbf{x}))\right)^2$$

On B and F:

median filter

```
def MedianFilter(src, dst, k = 3, padding = None):

    imarray = np.array(Image.open(src))
    height, width = imarray.shape
```

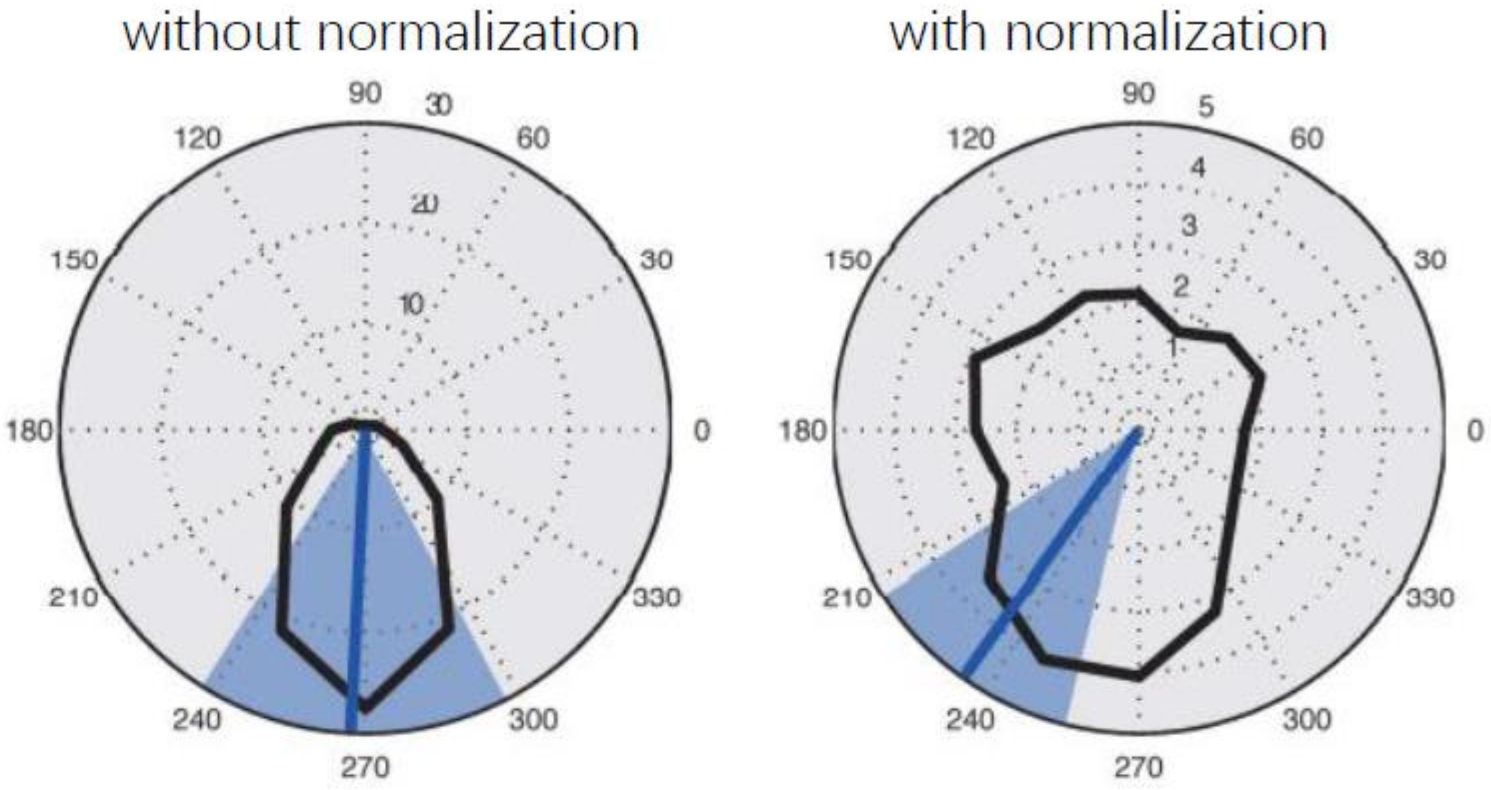
Implementation:



Normalization

Idea:

getting the global direction instead of a local one



Normalization

src/main.py

Include the methods

src/filter_methods.py

Adding Normalization

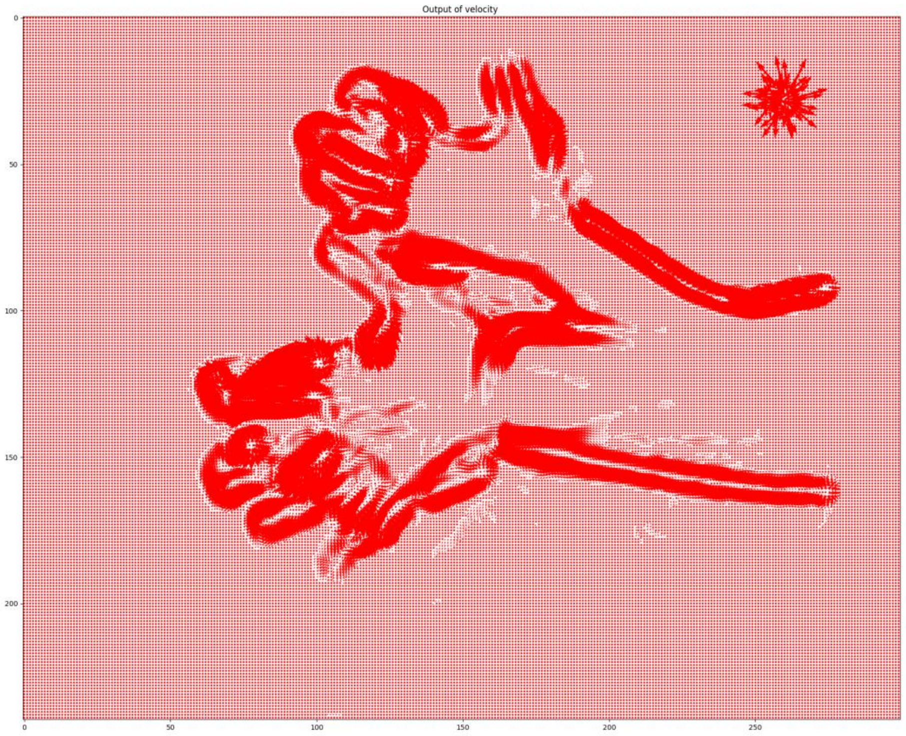
src/global_params.py

adding specific filter parameters

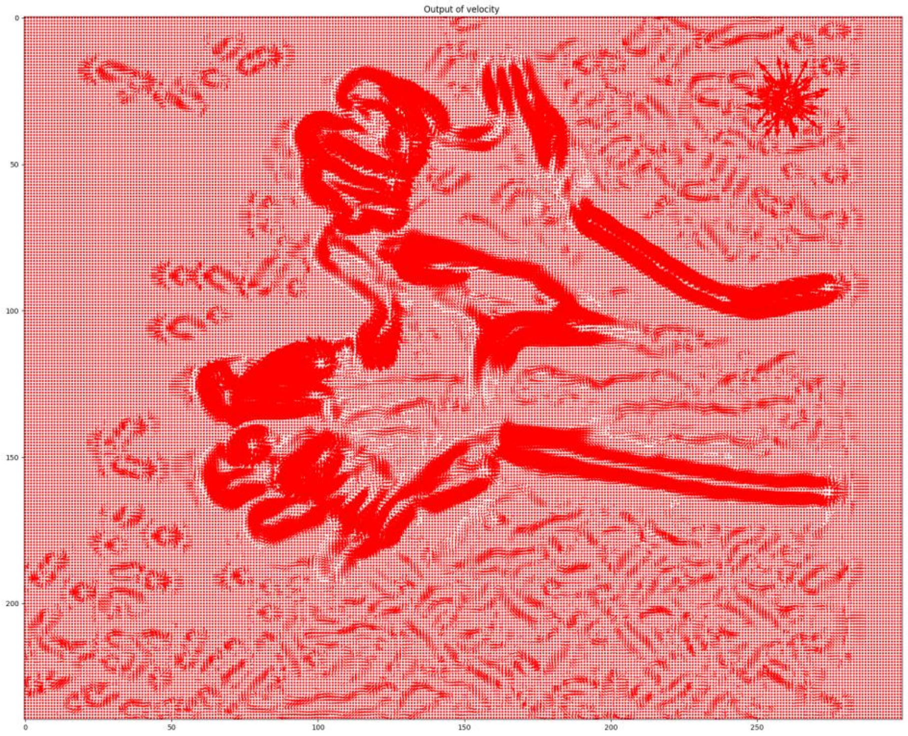
```python
def normalize(u, v):
    beta_response = 1
    alpha_p = 0.1
    alpha_q = 0.002
    sigma_response = 3.6

    center = math.ceil(sigma_response * 3)
    size = center * 2 + 1

    filter_gaussian = cv2.getGaussianKernel(size, sigma_response)

    def relu(x):
        return x * (x > 0)

    uv_response = np.sqrt(u**2 + v**2)

    relu_response = relu(uv_response)
    gaussian_response = cv2.filter2D(relu_response, -1, filter_gaussian)
    normalized_response = beta_response * uv_response / (alpha_p + uv_response + relu(gaussian_response / alpha_q))

    ratio = normalized_response / uv_response
    u_normalized = ratio * u
    v_normalized = ratio * v

    return u_normalized, v_normalized
```

```python
#Normalize

print('Normalize')
u = np.zeros((params.band_height, params.band_width), dtype=np.float32)
v = np.zeros((params.band_height, params.band_width), dtype=np.float32)
N = 8
with Timer("Aggregate..."):
    for k in range(len(filters)):
        u = u+np.cos(np.pi*2*k/N)*out_xy[k]
        v = v+(-1)*np.sin(np.pi*2*k/N)*out_xy[k]

u_normalized, v_normalized = normalize(u, v)
plt.quiver(X, Y, u_normalized, v_normalized, color='r')
plt.savefig("../output_figures/plt_save_whole_image_normalized.png")
```

Normalization



Original generated



Normalized generated

Discussion:

Normalization Code Problems

- Produce just more motions in the field
- Takes 10 minutes longer to generate than the original one

Discussion:

Problems of Optical Flow Code

- Not all implemented Filters are used in the running code
- Some code snipes in Main are still unused
- Incompatible packages for generating 3D-images
- Not documented well
- The functionality of some code snipes in main.py are still unkown because of missed documentation
- Output images mentioned in the notebook couldn’t be generated by the code

Discussion:

What we have achieved:
recover the two velocities belong to two objects intuitively.

Our limit:
Do not have the ground truth value so that we cannot quantify our effects and errors.
Depends a lot on the result of motion segmentation.