SCHWING
Lucas

Final project – Optical flow
PDE for image processing and computer vision

I. Problem Overview

This final project focuses on the **optical flow problem** in image processing. Optical flow is used to detect motion between two frames in a video. This motion is caused by the movement of different objects within the frame, resulting in changes in pixel color and intensity. While the optical flow analyzed in the course was based only on pixel intensity (gray images), the optical flow in this report will be applied to RGB images, therefore it will also consider changes in pixel color to detect motion between two images. A simple example (Figure 1) illustrates what optical flow does: when there is movement between two frames, the optical flow provides an estimate of the velocities between these two frames.

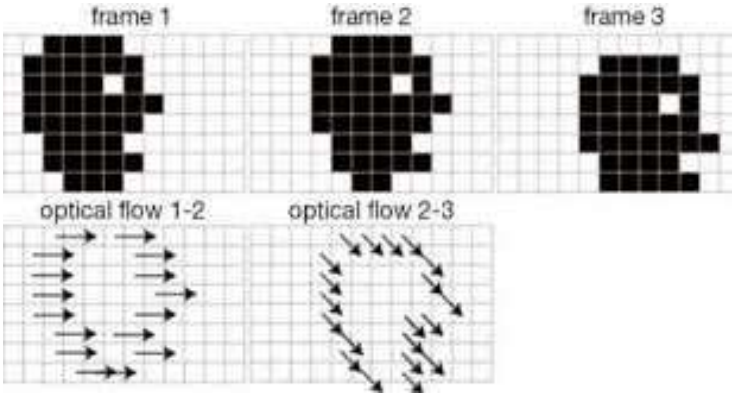

Figure 1: Simple example of optical flow[1]

II. Formulating and Implementing the PDE

a) Translating the Problem into Mathematical Language

To formulate this problem into a mathematical problem, we should start by using the optical flow equation: $I_x(x,y,t) * u(x,y,\tau) + I_y(x,y,t) * v(x,y,\tau) + I_t(x,y,t) = 0$ with I_x, I_y and I_t the derivative of $I(x,y,t)$ the intensity of the pixels (from the dataset) with respect to x, y and t and $u(x,y,\tau)$ and $v(x,y,\tau)$ are the unknown velocities in the direction x and y respectively.

This equation will be the fidelity term of the energy function that will make sure that the estimated motion is consistent with the changes in the image. In order to reduce the

[1] E. Raad, "Synthetic datasets for optical flow in computer vision," 28 May 2022. [Online]. Available: https://www.aufaitai.com/data/synthetic-datasets-optical-flow/. [Accessed 2024].

potential noise and to have a u and v function smoother a regularization term is necessary. Here, we will penalize the gradient of u and v.

Finally, it gives the following energy function:

$$E(u,v) = \int_{\Omega} \lambda * \left\| I_x * u + I_y * v + I_t \right\|^2 + (1-\lambda) * \|\nabla u\| + (1-\lambda) * \|\nabla v\|$$

The norm is used here to handle the multi-channel nature of RGB images (I_x is a matrix composed of R_x, G_x and B_x). With the square on the fidelity term, we should get something similar to what we have in class, however we are not using the square for penalizing the gradient of u and v. This allows for a more abrupt change in flow at object edges, leading to a quicker transition to zero flow for pixels beyond the moving object's edge.

b) Deriving the PDE from the Mathematical Model

Let's derive the PDE for u and v from the energy function.
Firstly, from the energy function, we get the following Lagrangian:

$$L(u,v,u_x,u_y,x,y) = \lambda * \left\| I_x * u + I_y * v + I_t \right\|^2 + (1-\lambda) * \|\nabla u\| + (1-\lambda) * \|\nabla v\|$$

With

$$\left\| I_x * u + I_y * v + I_t \right\| = \sqrt{(R_x * u + R_y * v + R_t)^2 + (G_x * u + G_y * v + G_t)^2 + (B_x * u + B_y * v + B_t)^2}$$

And

$$\|\nabla u\| = \sqrt{{u_x}^2 + {u_y}^2}$$

$$\|\nabla v\| = \sqrt{{v_x}^2 + {v_y}^2}$$

Let's derive first the PDE for u:
When v is fixed, we get the following Euler-Lagrange equation for u:

$$L_u - \frac{\partial}{\partial x} L_{u_x} - \frac{\partial}{\partial y} L_{u_y} = 0 = \nabla_u E$$

Therefore, we get $u_\tau = -\nabla_u E = -L_u + \frac{\partial}{\partial x} L_{u_x} + \frac{\partial}{\partial y} L_{u_y}$

With

$$L_u = \lambda * 2 * \left[(R_x * u + R_y * v + R_t) * R_x + (G_x * u + G_y * v + G_t) * G_x + (B_x * u + B_y * v + B_t) * B_x \right]$$

$$L_{u_x} = (1-\lambda) * \frac{u_x}{\sqrt{{u_x}^2 + {u_y}^2}}$$

$$L_{u_y} = (1-\lambda) * \frac{u_y}{\sqrt{{u_x}^2 + {u_y}^2}}$$

So, we have $\frac{\partial}{\partial x} L_{u_x} = (1-\lambda) * (\frac{u_{xx}}{\sqrt{{u_x}^2+{u_y}^2}} - \frac{u_x*(u_x u_{xx}+u_y u_{yx})}{\left(\sqrt{{u_x}^2+{u_y}^2}\right)^3})$

This gives:

$$\frac{\partial}{\partial x} L_{u_x} = (1-\lambda) * \frac{u_{xx}{u_y}^2 - u_x u_y u_{yx}}{\left(\sqrt{{u_x}^2+{u_y}^2}\right)^3}$$

By symmetry we have:

$$\frac{\partial}{\partial y} L_{u_y} = (1-\lambda) * \frac{u_{yy}{u_x}^2 - u_y u_x u_{yx}}{\left(\sqrt{{u_x}^2+{u_y}^2}\right)^3}$$

Putting everything together we get the following gradient descent for u:

$$u_\tau = -2 * \lambda * \left[\left(R_x * u + R_y * v + R_t\right) * R_x + \left(G_x * u + G_y * v + G_t\right) * G_x + \left(B_x * u + B_y * v + B_t\right) * B_x\right] + (1-\lambda) * \frac{{u_y}^2 u_{xx} - 2u_x u_y u_{xy} + {u_x}^2 u_{yy}}{\left(\sqrt{{u_x}^2+{u_y}^2}\right)^3}$$

To avoid dividing by 0 when ${u_x}^2 + {u_y}^2 = 0$, we will add a small epsilon to the denominator therefore after computing again the different derivatives we get the following PDE:

$$u_\tau = -2 * \lambda * \left[\left(R_x * u + R_y * v + R_t\right) * R_x + \left(G_x * u + G_y * v + G_t\right) * G_x + \left(B_x * u + B_y * v + B_t\right) * B_x\right] + (1-\lambda) * \frac{{u_y}^2 u_{xx} - 2u_x u_y u_{xy} + {u_x}^2 u_{yy} + \varepsilon * \Delta u}{\left(\sqrt{{u_x}^2+{u_y}^2+\varepsilon}\right)^3}$$

By symmetry, we get for v when u is fixed the following gradient descent:

$$v_\tau = -2 * \lambda * \left[\left(R_x * u + R_y * v + R_t\right) * R_y + \left(G_x * u + G_y * v + G_t\right) * G_y + \left(B_x * u + B_y * v + B_t\right) * B_y\right] + (1-\lambda) * \frac{{v_y}^2 v_{xx} - 2v_x v_y v_{xy} + {v_x}^2 v_{yy} + \varepsilon * \Delta v}{\left(\sqrt{{v_x}^2+{v_y}^2+\varepsilon}\right)^3}$$

c) Implementing the PDE

We need to discretize the following term: I_x, I_y, I_t, and u_x, u_y, u_{xy}, u_{xx}, u_{yy}.

For each derivative a central difference with delta equal to 1 is used for the best precision, and for the first element (index 0) a forward difference is used since there is no element before, and for the last element a backward difference is used.

Now, for the u_τ and v_τ a forward difference will be used, therefore in the code, we will compute at each iteration the new u and v with:

$$u(x, y, \tau + \Delta\tau) = u(x, y, \tau) + \Delta\tau * u_\tau$$

$$v(x, y, \tau + \Delta\tau) = v(x, y, \tau) + \Delta\tau * v_\tau$$

To know how to fix the value of $\Delta\tau$ we need to find the CFL condition. First the homogeneous equation of u_τ gives:

$$u_\tau = -2 * \lambda * \left({R_x}^2 u + {G_x}^2 u + {B_x}^2 u\right) + (1-\lambda) * \frac{{u_y}^2 u_{xx} - 2u_x u_y u_{xy} + {u_x}^2 u_{yy}}{\left(\sqrt{{u_x}^2 + {u_y}^2}\right)^3}$$

Since we are using a central difference for most of the u, we will compute the CFL condition with the central difference for the derivatives. We get:

$$u_x(x,y,\tau) = \frac{u(x+\Delta x, y, \tau) - u(x-\Delta x, y, \tau)}{2\Delta x}$$

$$u_{xx}(x,y,\tau) = \frac{u(x+\Delta x, y, \tau) - 2u(x,y) + u(x-\Delta x, y, \tau)}{\Delta x^2}$$

$$u_{xy}(x,y,\tau) = \frac{u(x+\Delta x, y+\Delta y, \tau) - u(x+\Delta x, y-\Delta y, \tau) - u(x-\Delta x, y+\Delta y, \tau) + u(x-\Delta x, y-\Delta y, \tau)}{4\Delta x \Delta y}$$

By symmetry we get the same equation for u_y and u_{yy} but with y instead of x.
By applying $DFT\ (x,y) \to w$, we get:

$$\int_{(x,y)\to w} u_x = \frac{1}{\Delta x}\cos(w\Delta x) * U(w,\tau)$$

$$\int_{(x,y)\to w} u_{xx} = \frac{1}{2\Delta x^2}(\cos(w\Delta x) - 1) * U(w,\tau)$$

$$\int_{(x,y)\to w} u_{xy} = \frac{1}{\Delta x \Delta y} 2\cos(w\Delta x \Delta y) * U(w,\tau)$$

In the code, we are using $\Delta x = \Delta y = 1$, hence, we get:

$$\int_{(x,y)\to w} u_\tau = U(w,\tau) * [-2 * \lambda * \left({R_x}^2 + {G_x}^2 + {B_x}^2\right) + (1-\lambda) * \frac{-4 * \cos^2(w)}{\left(\sqrt{2 * \cos^2(w)}\right)^3}]$$

Finally, we get:

$$U(w,\tau+\Delta\tau) = U(w,\tau)[1 + \Delta\tau * (-2 * \lambda * \left({R_x}^2 + {G_x}^2 + {B_x}^2\right) + (1-\lambda) * \frac{-4 * \cos^2(w)}{\left(\sqrt{2 * \cos^2(w)}\right)^3})]$$

This gives the amplification factor:

$$\alpha(w) = 1 + \Delta\tau * (-2 * \lambda * \left({R_x}^2 + {G_x}^2 + {B_x}^2\right) + (1-\lambda) * \frac{-4 * \cos^2(w)}{\left(\sqrt{2 * \cos^2(w)}\right)^3})$$

For stability we must have $|\alpha(w)| \leq 1$, hence:

$$-2 \leq \Delta\tau * \left(-2 * \lambda * \left({R_x}^2 + {G_x}^2 + {B_x}^2 \right) + (1-\lambda) * \frac{-4 * \cos^2(w)}{\left(\sqrt{2 * \cos^2(w)} \right)^3} \right) \leq 0$$

Since $\Delta\tau$, λ and $\cos^2(w)$ are always positive (and $\lambda < 1$) the right inequation is always true, we only need:

$$2 \geq \Delta\tau * \left(2 * \lambda * \left({R_x}^2 + {G_x}^2 + {B_x}^2 \right) + (1-\lambda) * \frac{4 * \cos^2(w)}{\left(\sqrt{2 * \cos^2(w)} \right)^3} \right)$$

In the worst case $\cos^2(w) = 1$, therefore we have:

$$2 \geq \Delta\tau * \left(2 * \lambda * \left({R_x}^2 + {G_x}^2 + {B_x}^2 \right) + (1-\lambda) * \frac{2}{\sqrt{2}} \right)$$

In the worst case R_x, G_x and B_x are equal to 255 (white pixel) and this is much larger than $\frac{2}{\sqrt{2}}$, therefore in the worst case $\lambda = 1$ and we finally get the CFL condition:

$$\boxed{\Delta\tau \leq \frac{2}{2 * (255^2 + 255^2 + 255^2)} = 5{,}13 * 10^{-6}}$$

Note that we will get the same result by symmetry for v except it would be ${R_y}^2 + {G_y}^2 + {B_y}^2$ but in the worst case they are also equal to 255.

III. Results

The dataset used is made for optical flow, and is composed of frames of a 3D animated short film[2].

Firstly, some preprocessing is made on the image dataset. The images are rescaled in order to reduce the number of pixels and to earn some computational time, also a plot of the mean of u and v against the iterations (tau) is computed in order to look at the evolution of these vectors. Finally, the arrows that represent u and v are multiplied by 2 in order to see them better.

To start we can verify that the CFL condition computed just before is right, with $\Delta\tau = 5 * 10^{-5}$ and $\lambda = 0.8$, we get for u and v:

[2] MPI-SINTEL DATA SET - Copyright (c) 2012 Daniel Butler, Jonas Wulff, Garrett Stanley, Michael Black, Max-Planck Institute for Intelligent Systems, Tuebingen

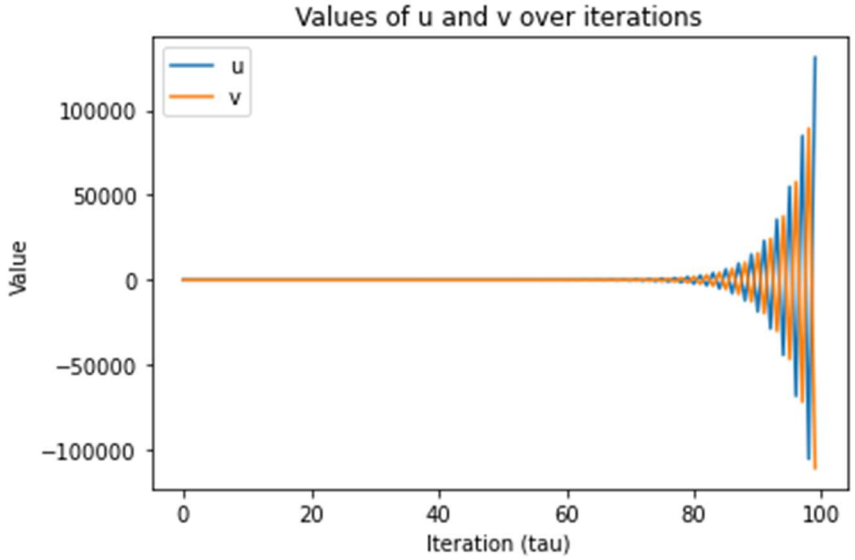

Figure 2: Mean of u and v vs iteration(tau) for a $\Delta\tau$ unstable

As expected, u and v are unstable with this $\Delta\tau$.

Now, with a $\Delta\tau = 5 * 10^{-6}$ and still $\lambda = 0.8$, we get:

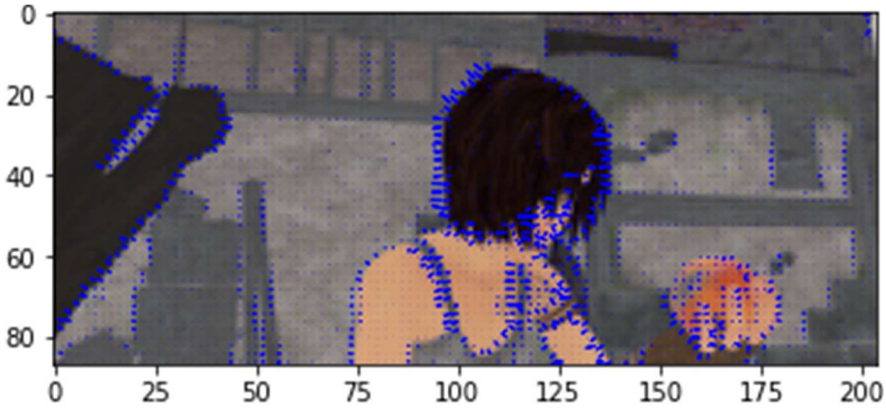

Figure 3: Image with optical flow for $\Delta\tau = 5 * 10^{-6}$, $\lambda = 0.8$ and 100 iterations

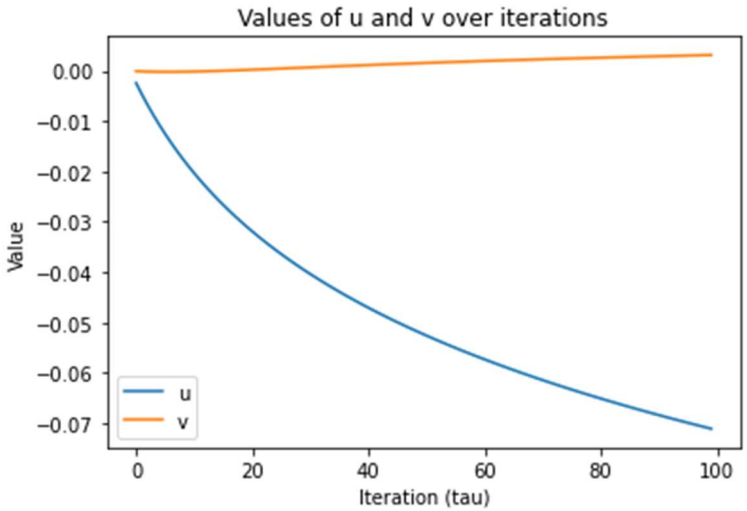

Figure 4: Evolution of u and v during the iterations

The image show that most of the movement and the arrow are coming from the edge of object and are going to the left. When comparing with the frame just after we can see that indeed the movement is going from the right to the left.

Now, we can look at the impact of lambda on u and v.

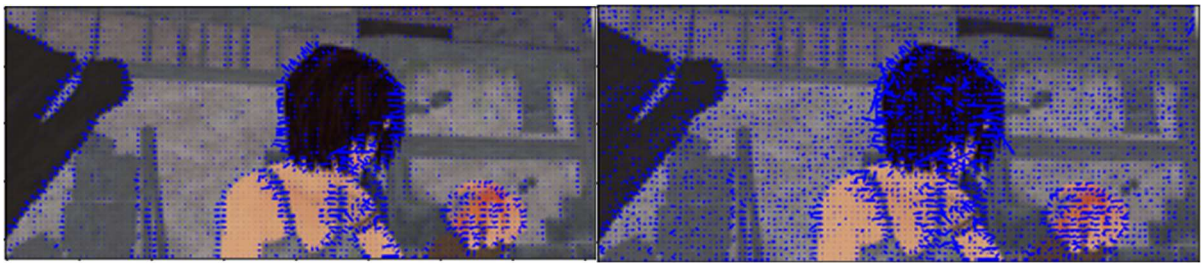

Figure 5: Comparison of frames with λ=0.1(left) and λ=0.9(right) and 5000 iterations

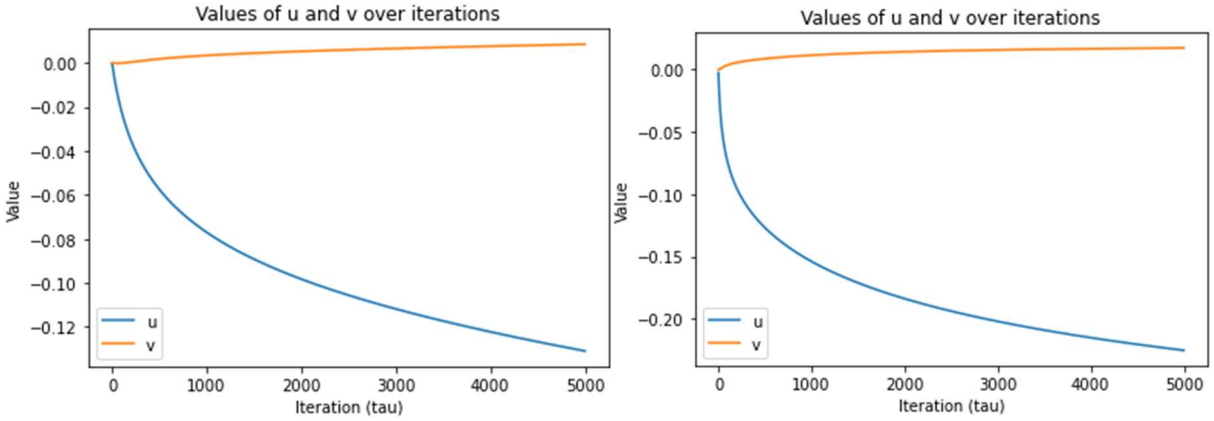

Figure 6: Comparison of u and v with λ=0.1(left) and λ=0.9(right)

As expected, when λ is small we are penalizing much more u and v, so there are evolving slower but smoother. The most noticeable difference can be saw on the images, the penalize term is acting like a filter, therefore the result when penalizing a lot is much cleaner. Note that trying $\lambda = 0$ does not make sense since with $\lambda = 0$ we will not take into consideration the optical flow.

In order to try to get something that converge fully, we can increase the number of iterations like we did just before, but we can also use as a starting value for u and v their previous value from the frame just before. Generally, when an object is moving, the movement is lasting during several frames. Moreover, since we just want to see when u and v are converging fully, we want them to converge faster so we will use a bigger λ.

With $\Delta\tau = 5 * 10^{-6}$, $\lambda = 0.8$, and 2000 iterations we get for the first frame:

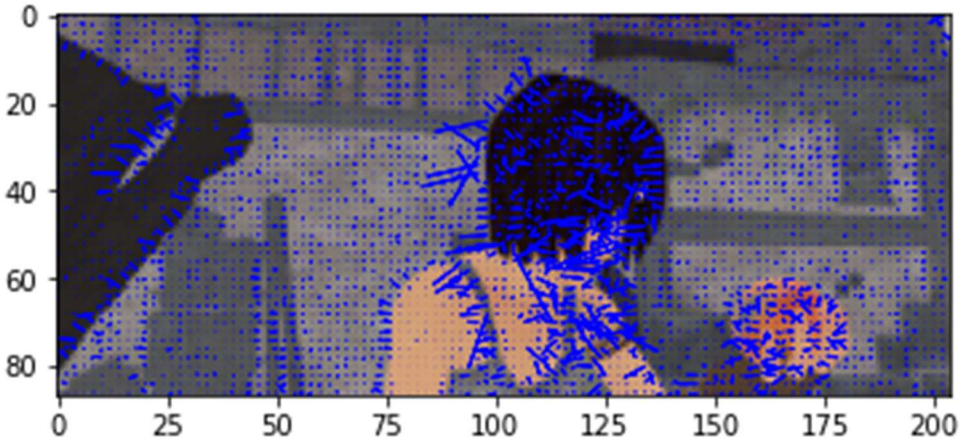

*Figure 7: Images/frame 1 for Δτ=5*10^(-6), λ=0.8 and 2000 iterations*

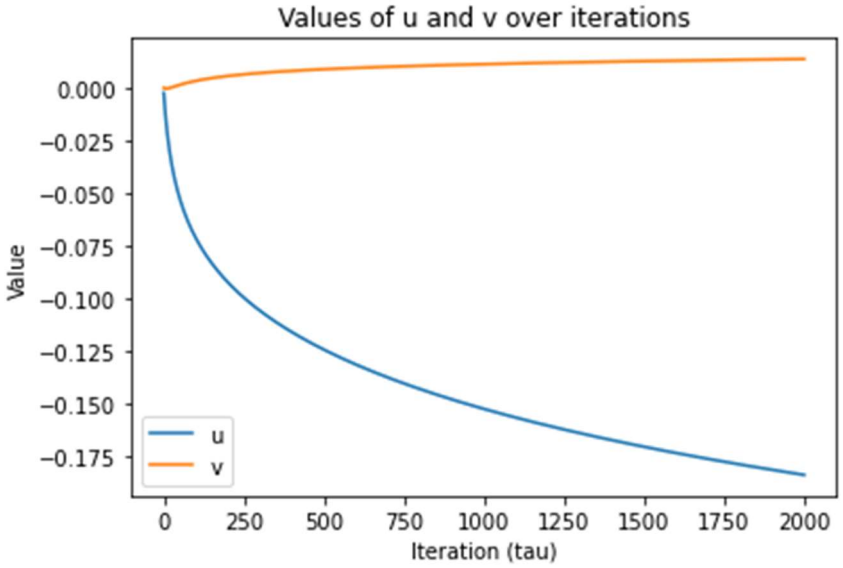

*Figure 8: U and v for frame 1 with Δτ=5*10^(-6) and λ=0.8*

U and v are still not finish to converge, so let's look at the second frame:

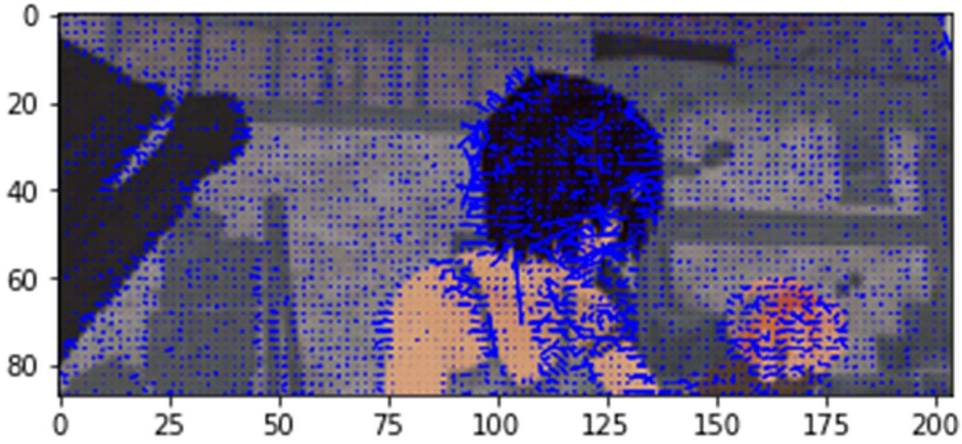

*Figure 9: Images/frame 2 for Δτ=5*10^(-6), λ=0.8 and 2000 iterations*

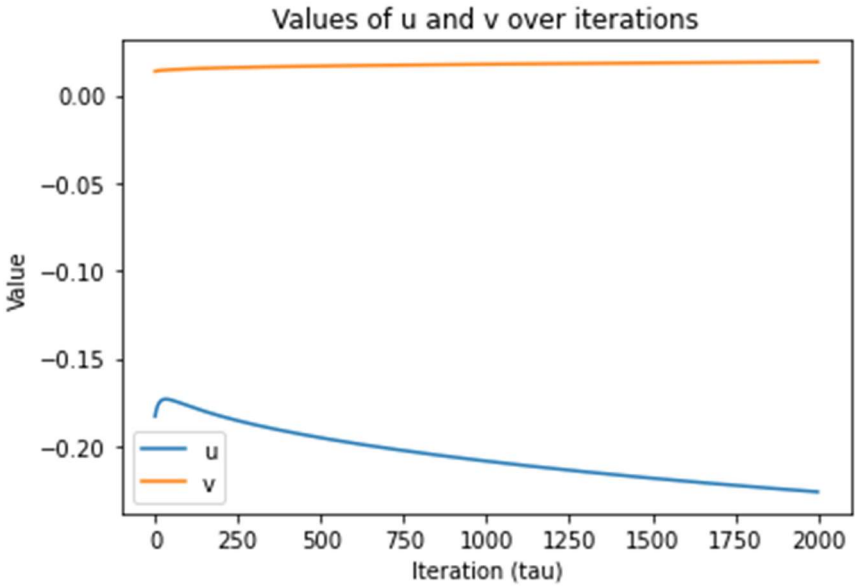

*Figure 10: U and v for frame 2 with Δτ=5*10^(-6) and λ=0.8*

U and v are converging well, we will stop to this frame since they seem to almost achieve their final value.

Now, another comparison that can be made is to look at the difference between this PDE and the one with a penalize term equal to the gradient squared. We use λ=0.1 here so the penalize term have more impact on the results.

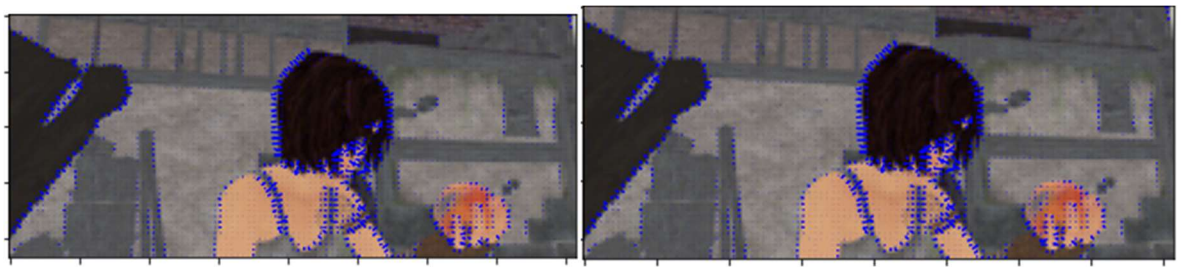

Figure 11: Image comparison between penalize term not squared (left) and squared (right)

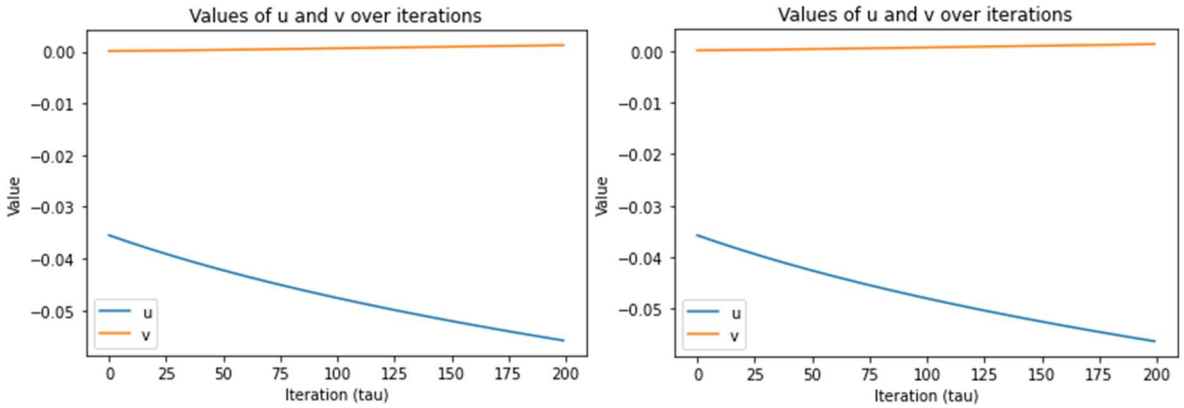

Figure 12: U and v comparison between penalize term not squared (left) and squared (right)

We can see that even with λ=0.1 there is close to no changes between the two PDE.

For the rest of the project, we will use λ=0.3, since it seems to be the best tradeoff between the convergence of u and v and having something that stay smooth.

As a last comparison, we will compare the PDE for RGB image that we did in this project with the one in the course for gray images.

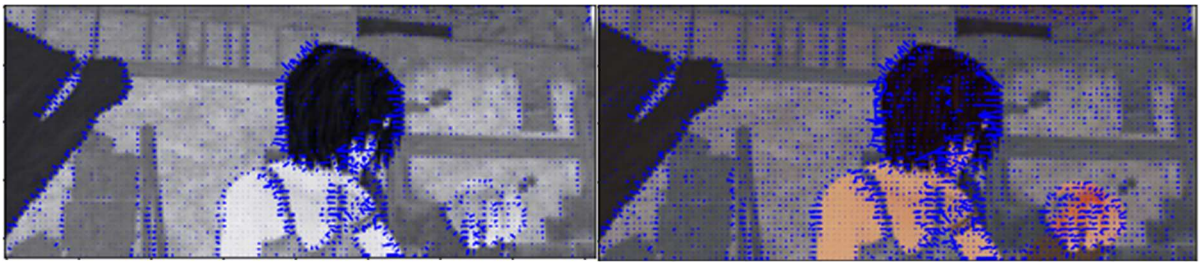

*Figure 13: Comparison of gray and RGB images for λ=0.3, Δτ=5*10^(-6) and 5000 iterations*

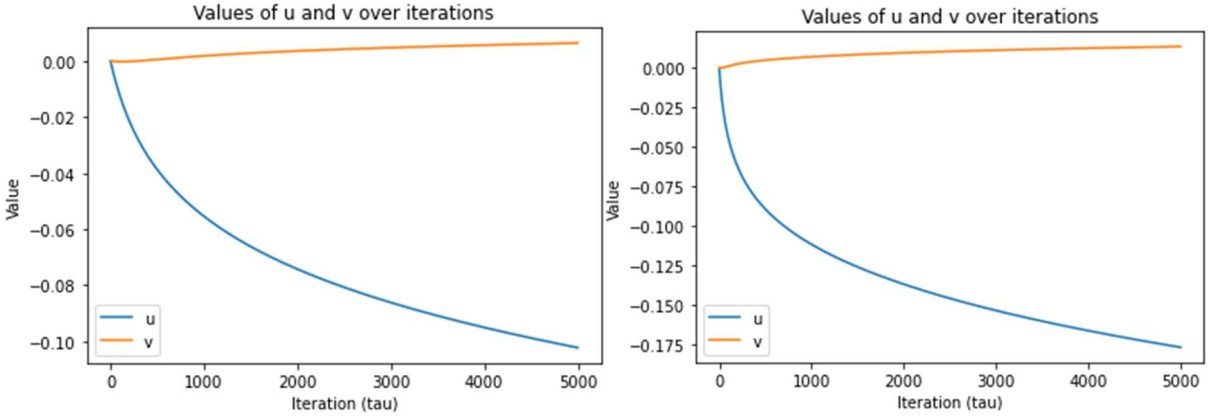

*Figure 14: U and v comparison of gray (left) and RGB (right) images for λ=0.3, Δτ=5*10^(-6)*

The plot of u and v show that the value is bigger for the RGB image, this is expected as we have three values (for each color). Also on the images, it seems that there is more data with RGB having more arrows. Overall, there are not huge differences, except for the computational time, the gray image was much faster to compute.

Using gray images can be a preprocessing to earn some computational time and achieve still good performance even if we lost a bit of information on the images.

As a last result the optical flow has been computed for every frame, therefore, there is a video in the zip file that contain the original video with the optical flow (with λ=0.3, $\Delta\tau = 5 * 10^{-6}$, 500 iterations for each frame and u and v are reset for everyframe).

IV. Conclusion

The results show that a trade-off needs to be chosen with the λ between having a faster convergence and having a result smoother and cleaner. In this project, it seems that the more iterations you do, the more you should penalize and have a smaller λ. Indeed, when having only 100 iterations, like in Figure 1, a larger λ is preferable to start to converge faster and to see noticeable changes in u and v. But when using 5000 iterations (Figure 5) having a large λ is not good as we start to have larger values of u and v.

The biggest weakness would be that when dealing with high resolution images, the PDE take a lot of computational time to be compute. Moreover, the PDE is also taking a lot of time to converge, in order to have a full convergence it is taking more than 5000 iterations. Even with a scaled image like the one used in this project, it takes at least 20 minutes to compute just for one frame. Therefore, doing preprocessing tasks on the image beforehand is really important, as this is the easiest way to save computational time.

However, the results are good, with a proper detection of the motions in the frame.

In the future, several things can be done to improve it. Firstly, trying to reduce the computational time by using some mathematical library. Also, since the derivative of the intensity will always be the same, it could be interesting to compute it once and then save it, so the algorithm doesn't need to recompute it every time we change the parameter. Finally, testing other image preprocessing techniques, especially those used in computer vision and machine learning, could also be interesting.