ECE6560 Final Project
Stereo Disparity Framework generalized for 2D Displacement

Name: Rahul Rustagi
GTID: 904024521

Spring 2025

Contents

Chapter 1

Problem Statement

Innovation in robotics has been quite a trend in recent years. One of the very active fields today in robotics is autonomous driving. Methods like Optical Flow Field calculation allow for the estimation of velocity field between subsequent image feeds. Integrating this over consecutive "camera movements" or indirectly vehicular motion will allow us to estimate its motion through image processing solely. However, due to its monocular nature, Optical Flow lacks depth information much required to project vectors into 3D world. To resolve this, we use stereo cameras that provide us with depth information.

This work addresses a limitation in traditional stereo disparity estimation, which typically assumes purely horizontal camera translation (e.g., baseline movement between two cameras). Instead, I propose solving the stereo disparity problem for generalized 2D camera motion (horizontal and vertical translations). This expands the framework to align more closely with optical flow techniques, effectively creating a unified approach for 2D Stereo Disparity.

But why bother about this at all!? The answer is again depth richness. Since stereo disparity works for static scenes captured using "shifted" cameras - information about depth can be retrieved using triangulation useful for scenarios like scene reconstruction, Object segmentation, spatial layout analysis, and autonomous driving! Optical flow on the other hand estimates the 2D motion field and therefore has no depth information of the 3D world. This limits its applicability to object tracking and motion estimation of moving scenes.

Chapter 2

Mathematical Concept

In this section, I will present the mathematical formulation. This would typically start by writing down the generalized stereo disparity equation for 2D and computing the energy functional.

I define the variables in use below. This would typically help in understanding how the energy functional and the laplacian is formed and what the different terms in it mean.

Key changes in this work are:

1. The camera moves in a static environment and I use the stereo framework to treat right image as I_2 and left image I_1. In the example picture below it can be inferred that the camera has moved to the right and a bit below

$$I_1 = I(x, y)$$

$$I_2 = I'(x, y) = I(x + a(x, y), y + b(x, y))$$

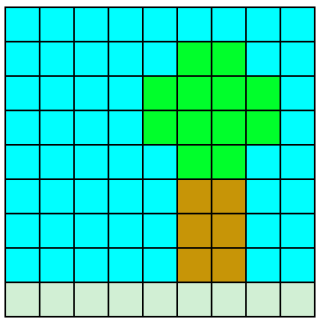

Figure 2.1: Stereo Left Image I_1

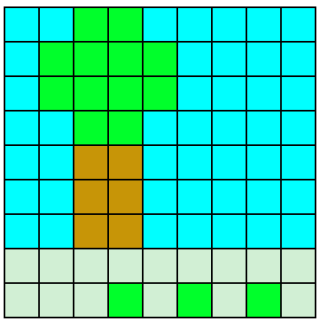

Figure 2.2: Stereo Right Image I_2

Let's build the energy functional E from first principles, carefully motivating each term and connecting it to the problem requirements. The task is to formulate a Lagrangian L such that it captures the data richness term and also ensures smoothness in spatial dimension.

2. Fundamental Energy Structure

$$E = \iint_{\Omega} L(a, b, \nabla a, \nabla b)\, dxdy \tag{2.1}$$

where the Lagrangian L contains:

- **Data fidelity**: Corresponding pixels in $I_1(x, y)$ and $I_2(x, y)$ should have similar intensities.
- **Regularization**: Disparity fields $a(x, y)$ and $b(x, y)$ should vary smoothly for continuity beyond boundaries

3. Data Fidelity Development

The data fidelity term is formulated using stereo disparity framework where I have considered displacement field, $a(x, y)$ and $b(x, y)$ rather than $u(x, y)$ and $v(x, y)$ velocity fields.

$$L_{data} = [I_2(x + a(x, y), y + b(x, y)) - I_1(x, y)]^2 \tag{2.2}$$

4. Regularization Design

Traditional smoothness term:

$$L_{\text{reg}} = \lambda(|\nabla a|^2 + |\nabla b|^2) \tag{2.3}$$

Note that λ is constant here and therefore requires tuning.

5. Complete Lagrangian

Combining both components:

$$L = \underbrace{(I_2(x + a, y + b) - I_1(x, y))^2}_{\text{Data fidelity}} + \underbrace{\lambda(|\nabla a|^2 + |\nabla b|^2)}_{\text{Regularization}} \tag{2.4}$$

6. Final Energy Functional

$$\boxed{E(a, b) = \iint_{\Omega} \left[(I_2(x + a, y + b) - I_1(x, y))^2 + \lambda(a_x^2 + a_y^2 + b_x^2 + b_y^2)\right] dxdy} \tag{2.5}$$

Chapter 3

PDE Formulation

In this section, I would convert the above mathematical problem into a Partial Differential Equation to solve it.

1. Energy Functional

NOTE: I have skipped the $\frac{1}{2}$ factor from both of the terms here and there is no $(1-\lambda)$ coefficient for the data fidelity term.

$$E(a,b) = \iint_{\Omega} \left[\underbrace{\left(I_2(x+a, y+b) - I_1(x,y)\right)^2}_{\text{Data fidelity}} + \underbrace{\lambda \left(a_x^2 + a_y^2 + b_x^2 + b_y^2\right)}_{\text{Regularization}} \right] dxdy \tag{3.1}$$

2. Euler-Lagrange Equations

$$\frac{\partial L}{\partial a} - \frac{\partial}{\partial x}\frac{\partial L}{\partial a_x} - \frac{\partial}{\partial y}\frac{\partial L}{\partial a_y} = 0 \tag{3.2}$$

$$\frac{\partial L}{\partial b} - \frac{\partial}{\partial x}\frac{\partial L}{\partial b_x} - \frac{\partial}{\partial y}\frac{\partial L}{\partial b_y} = 0 \tag{3.3}$$

3. Writing Equations for $a(x,y)$ evolution

Let $\vec{d}$ denote $\begin{bmatrix} a \\ b \end{bmatrix}$. Therefore, I can write $I_2(x+a, y+b) = I_2 \cdot \vec{d}$ since writing I_2 means evaluating the image at (x,y) which would be incorrect since I would want to compare the image I_2 convolved with $\vec{d}$ with I_1.

$$\begin{aligned} \frac{\partial L}{\partial a} &= 2\left(I_2(x+a, y+b) - I_1(x,y)\right) \frac{\partial (I_2 \cdot \vec{d})}{\partial x} \frac{\partial (x+a)}{\partial a} \\ &= 2\left(I_2(x+a, y+b) - I_1(x,y)\right) \frac{\partial (I_2 \cdot \vec{d})}{\partial x} \end{aligned}$$

$$\frac{\partial L}{\partial a_x} = 0 + 2\lambda a_x = 2\lambda a_x \quad \frac{\partial L}{\partial a_y} = 0 + 2\lambda a_y = 2\lambda a_y$$
$$\frac{d}{dx}\left(\frac{\partial L}{\partial a_x}\right) = 2\lambda a_{xx} \quad \frac{d}{dy}\left(\frac{\partial L}{\partial a_y}\right) = 2\lambda a_{yy}$$

3. Assembling Terms for u

$$\frac{\partial L}{\partial a} - \frac{d}{dx}\left(\frac{\partial L}{\partial a_x}\right) - \frac{d}{dy}\left(\frac{\partial L}{\partial a_y}\right) = 0$$
$$\Rightarrow 2\left(I_2(x+a, y+b) - I_1(x,y)\right)\frac{\partial(I_2 \cdot \vec{d})}{\partial x} - 2\left[\lambda a_{xx} + \lambda a_{yy}\right] = 0$$
$$\Rightarrow \cancel{2}\ (I_2 \cdot \vec{d} - I_1)\frac{\partial(I_2 \cdot \vec{d})}{\partial x} = \cancel{2}\left[\lambda a_{xx} + \lambda a_{yy}\right]$$
$$\Rightarrow \boxed{(\lambda \Delta a) = \ (I_2 \cdot \vec{d} - I_1)\frac{\partial(I_2 \cdot \vec{d})}{\partial x}}$$

4. Euler-Lagrange Equation for v

$$\frac{\partial L}{\partial b} = 2\left(I_2(x+a, y+b) - I_1(x,y)\right)\frac{\partial(I_2 \cdot \vec{d})}{\partial y}\frac{\partial(y+b)}{\partial b}$$
$$= 2\left(I_2(x+a, y+b) - I_1(x,y)\right)\frac{\partial(I_2 \cdot \vec{d})}{\partial y}$$
$$\frac{\partial L}{\partial b_x} = 0 + 2\lambda b_x = 2\lambda b_x \qquad \Rightarrow \frac{d}{dx}\left(\frac{\partial L}{\partial b_x}\right) = 2\lambda b_{xx}$$
$$\frac{\partial L}{\partial b_y} = 0 + 2\lambda b_y = 2\lambda b_y \qquad \Rightarrow \frac{d}{dy}\left(\frac{\partial L}{\partial b_y}\right) = 2\lambda b_{yy}$$

5. Gradient Descent PDE

The two equations given below are to be simultaneously evolved which could be done by writing them in a matrix form and ensuring that eigenvalues of the amplification matrix formed has magnitude ≤ 1

$$\frac{\partial a}{\partial t} = -\nabla E(a,b) = (\lambda \Delta a) - ((I_2 \cdot \vec{d}) - I_1)\frac{\partial I_2}{\partial x} \tag{3.4}$$

$$\frac{\partial b}{\partial t} = -\nabla E(a,b) = (\lambda \Delta a) - ((I_2 \cdot \vec{d}) - I_1)\frac{\partial I_2}{\partial y} \tag{3.5}$$

Instead of forming a system of equations and analyzing the eigenvalues, I am doing a workaround where I pick a Δt such that it satisfies the CFL conditions for both of the independently evolving PDE equations.

Chapter 4

Discretization and Implementation

In this section, I talk about how the PDE formulated above is discretized. I have used central differences for spatial derivatives and the nonlinear term R_x is incorporated later on to account as a shift in Δx

1. Discretized PDE for $a(x, y, t)$

The continuous PDE for horizontal disparity:

$$\frac{\partial a}{\partial t} = \lambda \left(\frac{\partial^2 a}{\partial x^2} + \frac{\partial^2 a}{\partial y^2} \right) - \underbrace{(I_2 - I_1)\frac{\partial I_2}{\partial x}}_{R_x}$$

Discretized using central differences in space and forward Euler in time. This is because it provides a stable and more average and unbiased value for space derivatives which are more meaningful if dealing with image grids:

$$\frac{a(x, y, t + \Delta t) - a(x, y, t)}{\Delta t} = \lambda \Bigg(\frac{a(x + \Delta x, y, t) - 2a(x, y, t) + a(x - \Delta x, y, t)}{(\Delta x)^2} + \frac{a(x, y + \Delta y, t) - 2a(x, y, t) + a(x, y - \Delta y, t)}{(\Delta y)^2} \Bigg)$$

2. Von Neumann Stability Analysis

Assume solution form:

$$a(x, y, t) = \xi_a(t) e^{i(\omega_x x + \omega_y y)}$$

Substitute into (1) and simplify:

$$\frac{\xi_a(t + \Delta t) - \xi_a(t)}{\Delta t} e^{i(\omega_x x + \omega_y y)} = \lambda \xi_a(t) \left(\frac{e^{i\omega_x \Delta x} - 2 + e^{-i\omega_x \Delta x}}{(\Delta x)^2} + \frac{e^{i\omega_y \Delta y} - 2 + e^{-i\omega_y \Delta y}}{(\Delta y)^2} \right) e^{i(\omega_x x + \omega_y y)}$$

$$\frac{\xi_a(t + \Delta t)}{\xi_a(t)} = 1 + \Delta t \left[\frac{2\lambda}{(\Delta x)^2} (\cos(\omega_x \Delta x) - 1) + \frac{2\lambda}{(\Delta y)^2} (\cos(\omega_y \Delta y) - 1) \right]$$

3. Worst-Case Stability

At highest frequencies ($\omega_x \Delta x = \pi$, $\omega_y \Delta y = \pi$):

$$\alpha(\omega) = 1 - \Delta t \left(\frac{4\lambda}{(\Delta x)^2} + \frac{4\lambda}{(\Delta y)^2} \right)$$

CFL Condition:
Stability requires $|\alpha(\omega)| \leq 1$:

$$0 \leq \Delta t \leq \frac{2}{\frac{4\lambda}{(\Delta x)^2} + \frac{4\lambda}{(\Delta y)^2}}$$

4. Modified CFL Conditions

Now accounting for the nonlinear R_x term along with the CFL condition, we get, For horizontal (a) and vertical (b) disparities:

$$\Delta t \leq \min \left(\frac{2}{\frac{4\lambda}{(\Delta x)^2} + \frac{4\lambda}{(\Delta y)^2} +}, \frac{(\Delta x) + (\Delta y)}{\max \left(\max |(I_2 - I_1) \frac{\partial I_2}{\partial x}|, \max |(I_2 - I_1) \frac{\partial I_2}{\partial y}| \right)} \right)$$

5. Implementation (Isotropic Grid)

Taking $\Delta x = \Delta y = h$:

$$\Delta t \leq \min \left(\frac{2}{\frac{8\lambda}{h^2}}, \frac{(\Delta x) + (\Delta y)}{\max \left(\max |(I_2 - I_1) \frac{\partial I_2}{\partial x}|, \max |(I_2 - I_1) \frac{\partial I_2}{\partial y}| \right)} \right)$$

Assuming pixel-level displacement: Taking $h = 1$:

$$\Delta t \leq \min \left(\frac{2}{8\lambda}, \frac{2}{\max \left(\max |(I_2 - I_1) \frac{\partial I_2}{\partial x}|, \max |(I_2 - I_1) \frac{\partial I_2}{\partial y}| \right)} \right)$$

Chapter 5

Experiment

This section will be divided into below 5 parts:

- Evaluation Criteria - In this subsection, I would introduce the different test case images I would be testing the PDE against.
- Evolution of E - This part tests the how the energy functional evolves with number of timesteps, and different values of λ. I will vary and play around with different values of λ to understand the tradeoff between data fidelity and regularization term.
- Convergence Plots - In this experiment, I aim to find the coorelation between hyper-parameters of the PDE and see how it affects convergence. Further, I will compare the displacement flow field with the ground truth.
- Quantitative Analysis - Here I will here talk about the quantitative metrics. I would have typically shown quantitative values of predicted vs ground truth shifts in Convergence plots section.
- Experiment with real-world images - Here, I will test with images taken through a stereo-camera setup to see results for a real-time setup.

1. Evaluation Criteria

Below are the images on which I test out the PDE result.

The aim is to evaluate test the PDE formulation on the below images and see how it converges. I have tested it on the below images with known shifts to understand how each hyperparameter λ, T depends on each other.

Case 1: Simple Binary Image with small (5px, 3px) shift

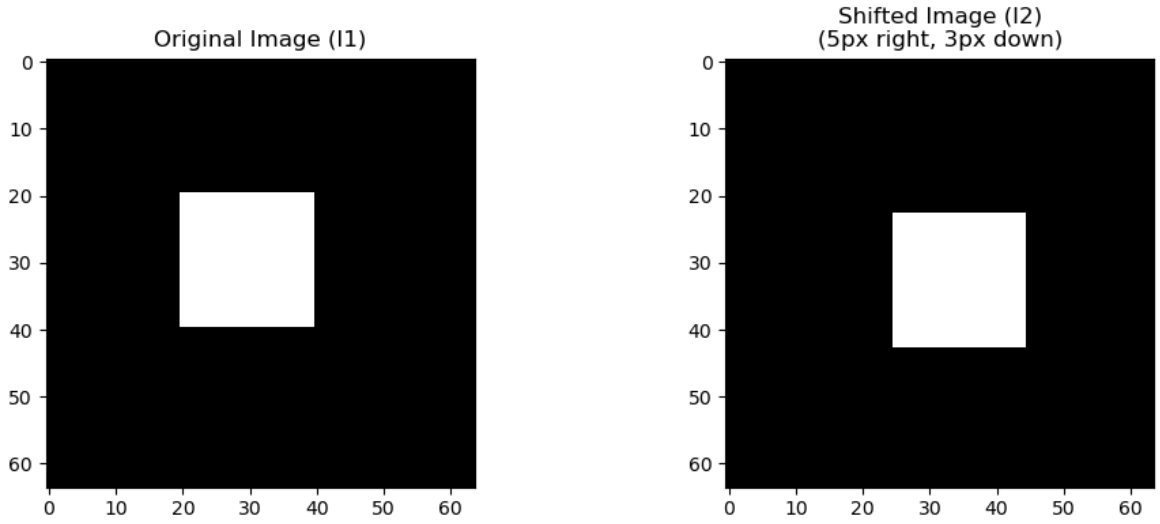

Figure 5.1: Binary Image with less than 10px shift

Case 2: Simple Binary Image with big (15px, 10px) shift

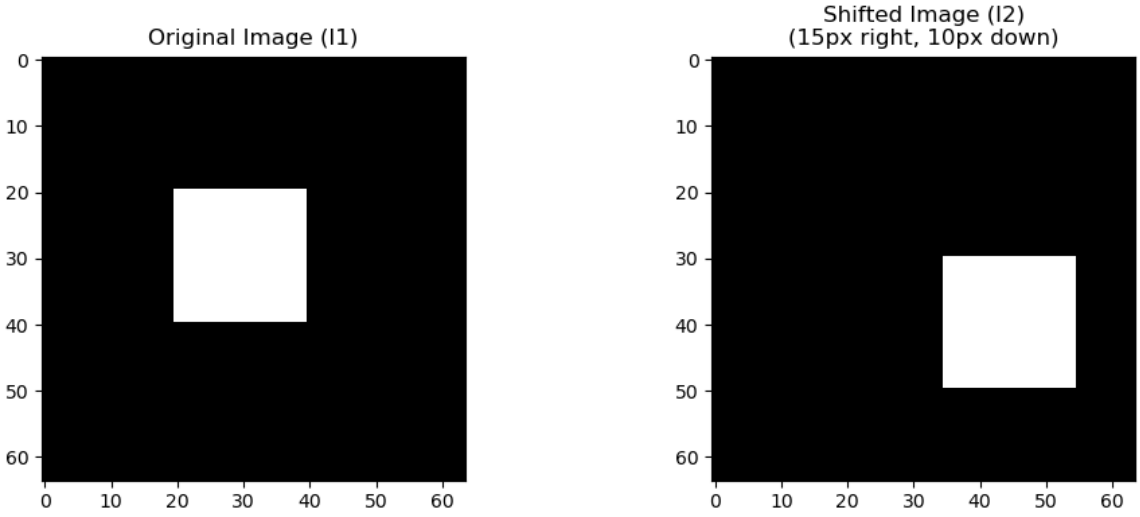

Figure 5.2: Binary Image with more than 10px shift

Case 3.1: Complex Binary Image with big horizontal (15px, 0px) shift

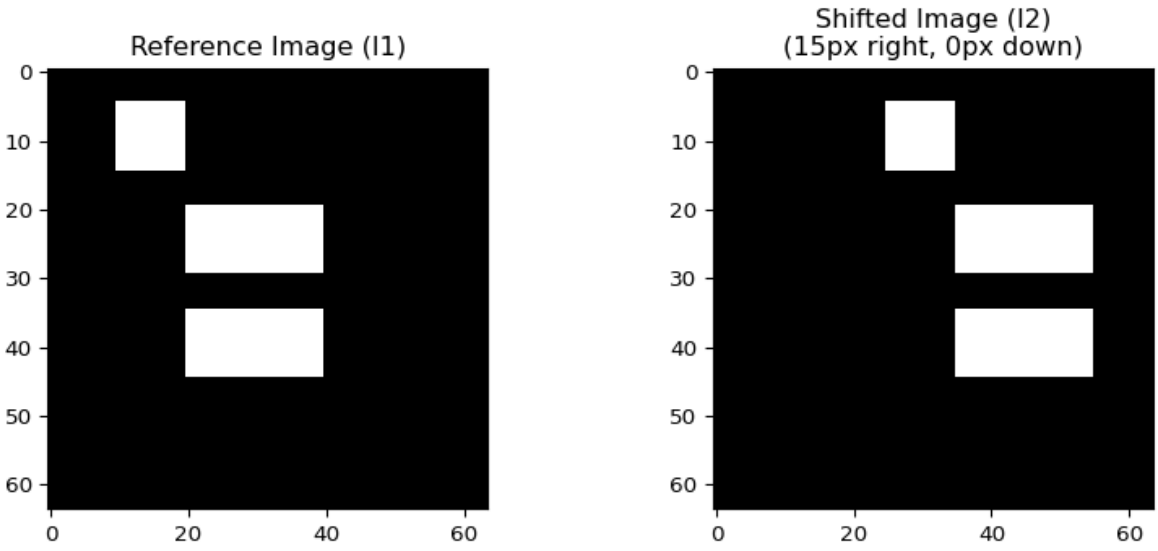

Figure 5.3: Complex image with non-overlapping horizontal shift

Case 3.2/3.3/3.4: Complex Binary Image with big vertical (0px, 10px) shift

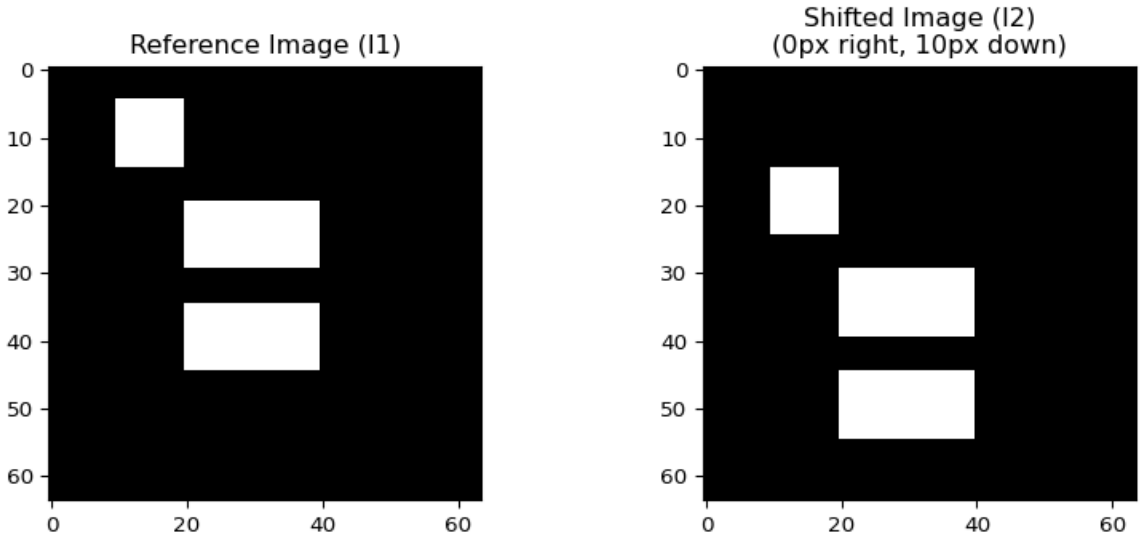

Figure 5.4: Complex image with overlapping vertical shift

2. Evolution of E

I would present and analyze the graphs of the evolution of the energy functional to find optimal values of λ, T for which equation has reaches a minima.

Case 1: 5px, 3px shift

The regularization parameter used is 0.5. One can see that $\lambda = 0.5$ allows the PDE to converge at near 20,000 timestep and not earlier.

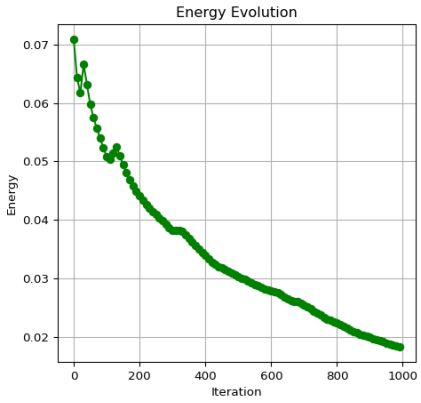

Figure 5.5: Energy function with 1000 iterations

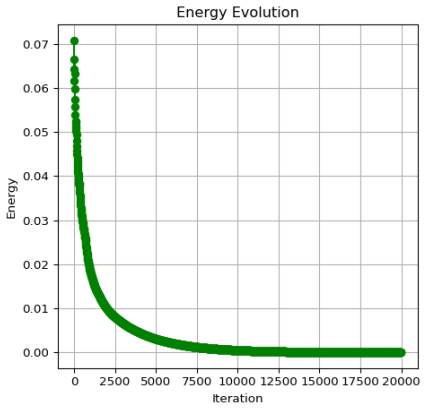

Figure 5.6: Energy function with 20,000 iterations

Case 2: 15px, 10px shift

The regularization parameter used is 0.5. One can see that $\lambda = 0.5$ allows the PDE to hold the minima solution even at 30,000 timesteps which means it has converged.

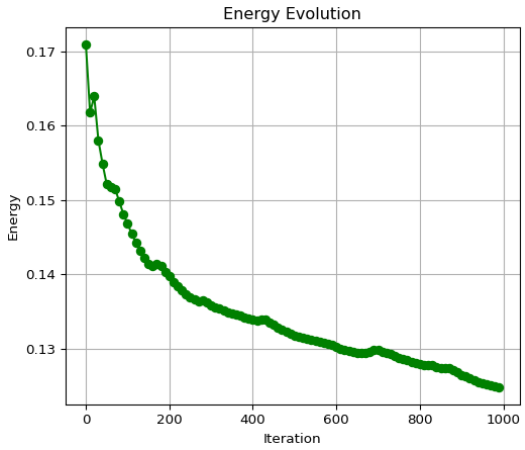

Figure 5.7: Energy function with 1000 iterations

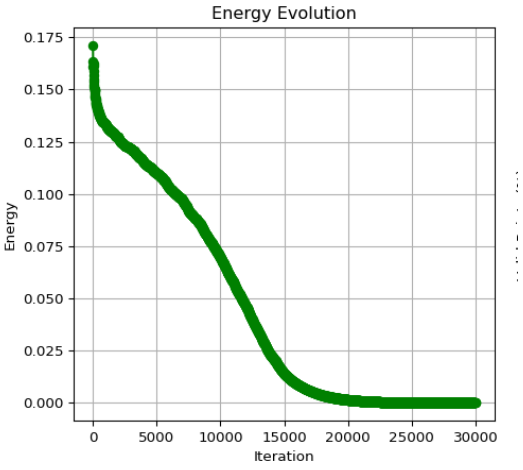

Figure 5.8: Energy function with 30,000 iterations

Case 3.1: 15px, 0px shift

The regularization parameter used is 0.5. One can see that $\lambda = 0.5$ allows the PDE to attain the minima solution starting 10000 timstep. This maybe due to the data richness of the image as we see later.

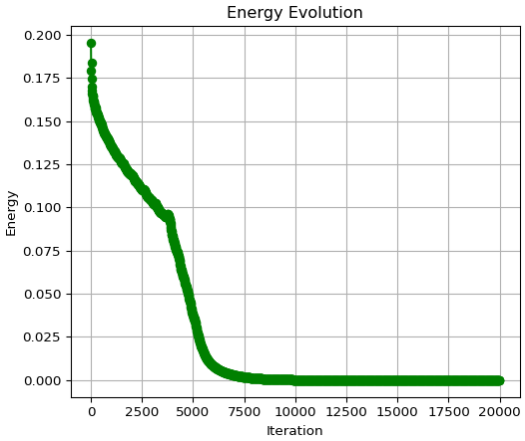

Figure 5.9: Energy function with 20,000 iterations, $\lambda = 0.5$

Case 3.2: 0px, 10px shift

In this plot, it seems that convergence is reached quite early which is suspicious and is further examined in the next subsection.

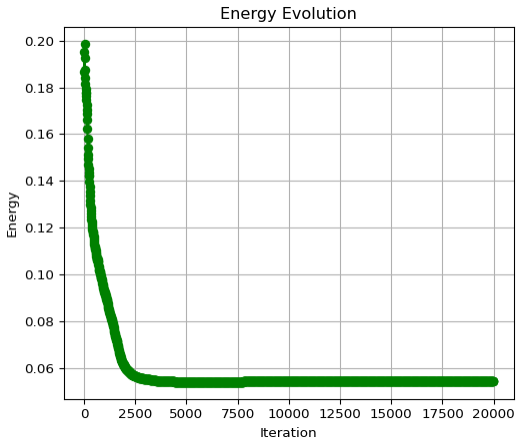

Figure 5.10: Energy function with 20,000 iterations, $\lambda = 0.5$

Therefore, we decide that the optimal values of the parameters λ and T are 0.5 and 20,000 respectively. However upon closely analysing the displacement vector field we realise that this energy formulation is probably stuck in a local minima and limits in its accuracy to correctly estimate few complex images as shown later.

3. Convergence Plots

Case 1: 5px, 3px shift

- **Iterations:** 1000 — **Result :** a(x,y) at center = 2.76px and b(x,y) at center = 1.85px
- **Regularization λ:** 0.5. Note that coefficient of data-fidelity term is 1 and regularization term is λ

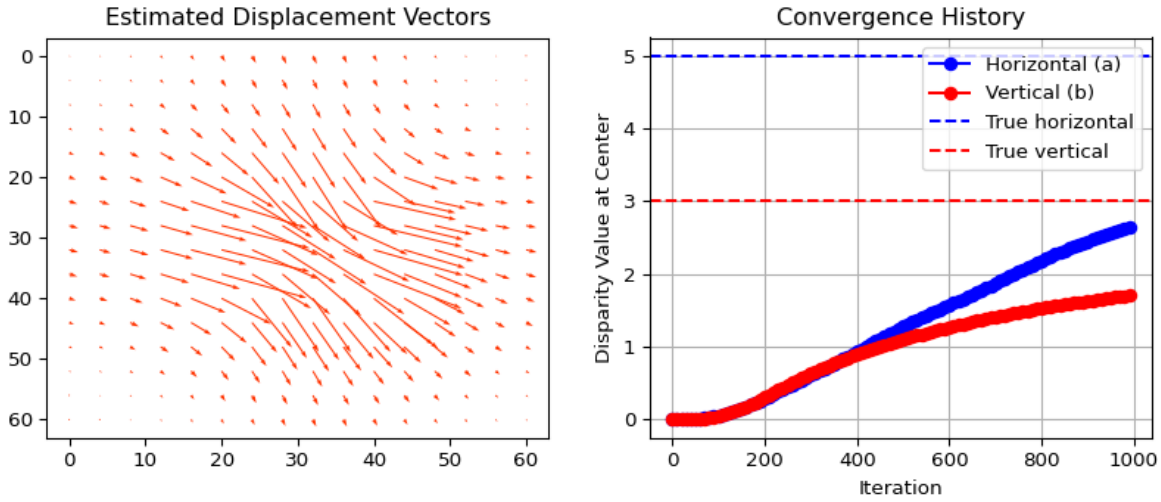

Figure 5.11: Convergence with 1000 iterations

- **Iterations:** 20,000 — **Result :** a(x,y) at center = 4.96px and b(x,y) at center = 2.97px

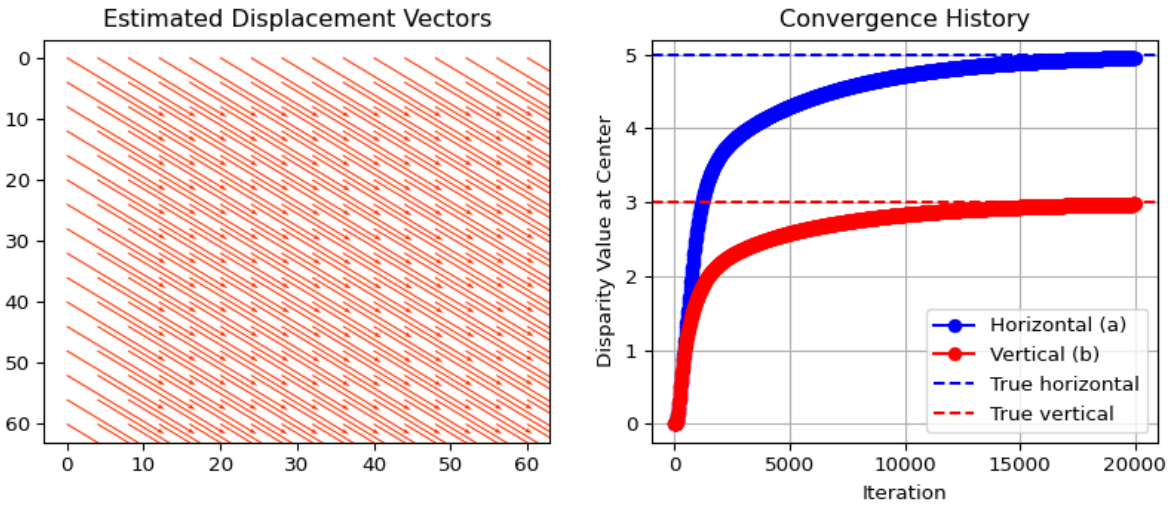

Figure 5.12: Convergence with 20,000 iterations

Here, we learn that even for a small shift, 1000 iterations fall short to let the PDE converge. When number of iterations were increased to 10,000 the PDE converges correctly for the image-pair with the regularization.

Case 2: 15px, 10px shift

- **Iterations:** 10,000
- **Result :** a(x,y) at center = 8.88px and b(x,y) at center = 6.47px
- **Regularization λ:** 0.5
  Note that coefficient of data-fidelity term is 1 and regularization term is λ

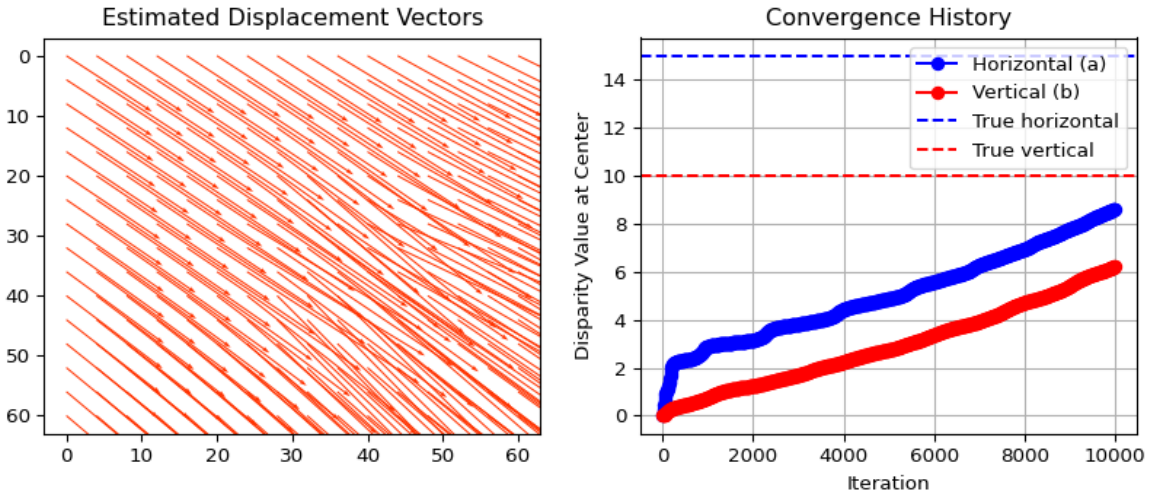

Figure 5.13: Convergence with 10,000 iterations

Here, we observe that even 10,000 iterations fall short to let the PDE converge. When number of iterations were increased to 30,000 the PDE evolves for more time for the image-pair.

In this case, the PDE does not converge since the number of timesteps fall short. Now, 10,000 timesteps should have been enough as learnt from previous scenario, however since the disparity fields is initialized to 0 and the true shifts is now increased as compared to previous values; the PDE would take more time for the diffusion term to spread the bigger shift.

This tells us that for bigger shift, we need to let the PDE evolve for more time than what is required for smaller shifts. Note that $a(x, y)$ and $b(x, y)$ are initialized to 0 and then evolved which indicates that this is an expected behavior.

- **Iterations:** 30,000
- **Result :** a(x,y) at center = 15.09px and b(x,y) at center = 9.97px

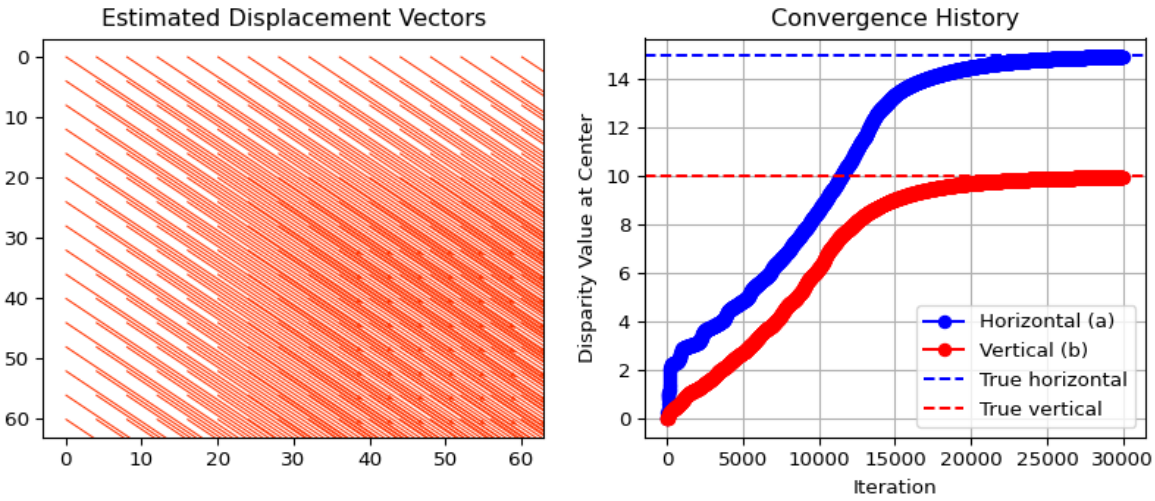

Figure 5.14: Convergence with 30,000 iterations

Case 3.1: 15px, 0px shift

- **Iterations:** 20,000
- **Result :** a(x,y) at center = 14.86px and b(x,y) at center = 0px
- **Regularization λ:** 0.5
  Note that coefficient of data-fidelity term is 1 and regularization term is λ

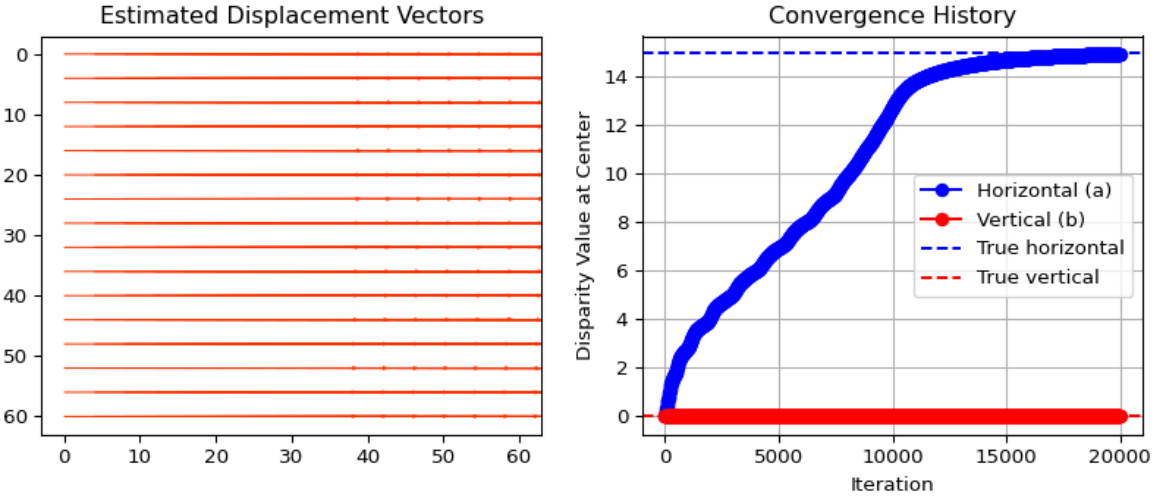

Figure 5.15: Convergence with 20,000 iterations

Case 3.2: 0px, 10px shift

- **Iterations:** 20,000 — λ : 0.5
- **Result :** a(x,y) at center = 0.2px and b(x,y) at center = -2.85px

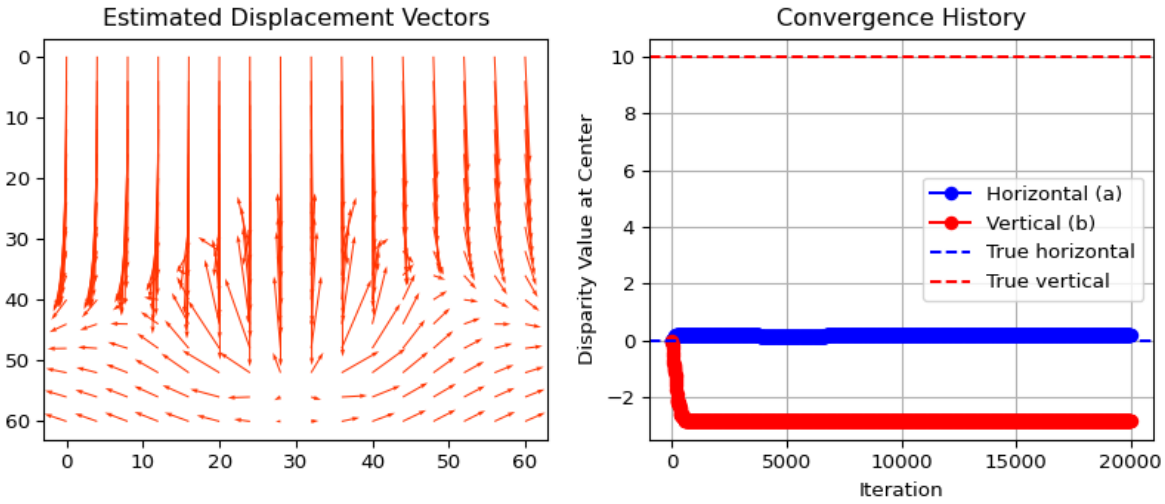

Figure 5.16: Convergence with 20,000 iterations

The issue arises because when vertically stacked boxes are shifted up or down, they can overlap with adjacent boxes. If each box is displaced by the same amount in both upward and downward directions (creating symmetrical movement), this balanced shift can make it difficult to determine the actual direction of motion along the vertical (y-axis). This creates a symmetrical vertical pattern that is due to overlap. The horizontal vectors are totaling to 0 which means that initially they were 0 but due to the diffusion term, the symmetrical vertical vector directions is induced in the horizontal.

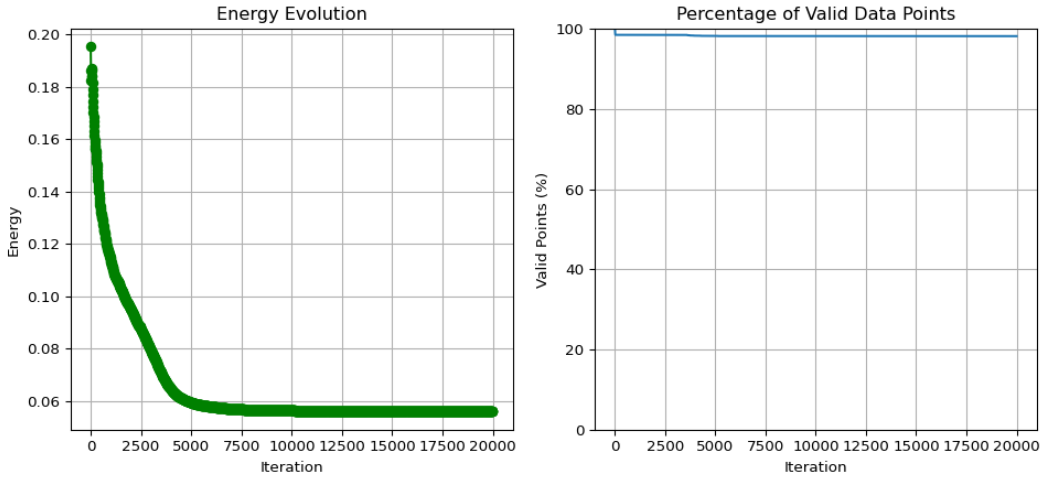

Figure 5.17: Valid points detected from which data fidelity term is calculated is near 100% which is wrong (should be lesser)

Case 3.3: 0px, 10px shift

- **Iterations:** 20,000
- **Result :** a(x,y) at center = -1.57px and b(x,y) at center = -4.67px
- **Regularization λ:** 0.6
  Note that coefficient of data-fidelity term is 1 and regularization term is λ

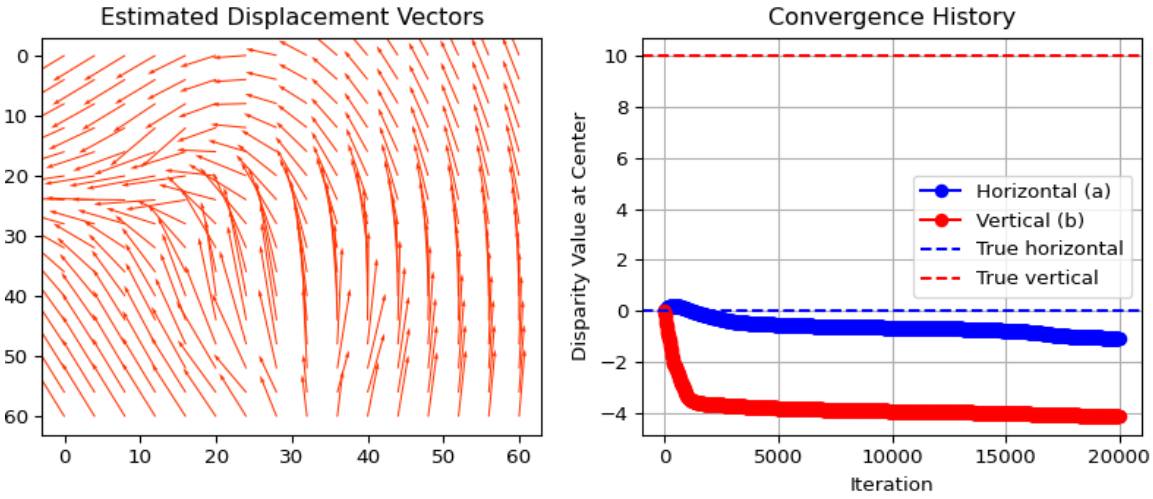

Figure 5.18: Convergence with 20,000 iterations

The issue here is the same as above. However, the difference here is that the regularization parameter was bumped up to 0.6 from 0.5. This creates a smoothing effect but rather diffuses the strong symmetrical vertical pattern in the lower region into above part. As can be seen that the horizontal line in right graph inclines itself towards the left direction since the top box is present to the left of other two.

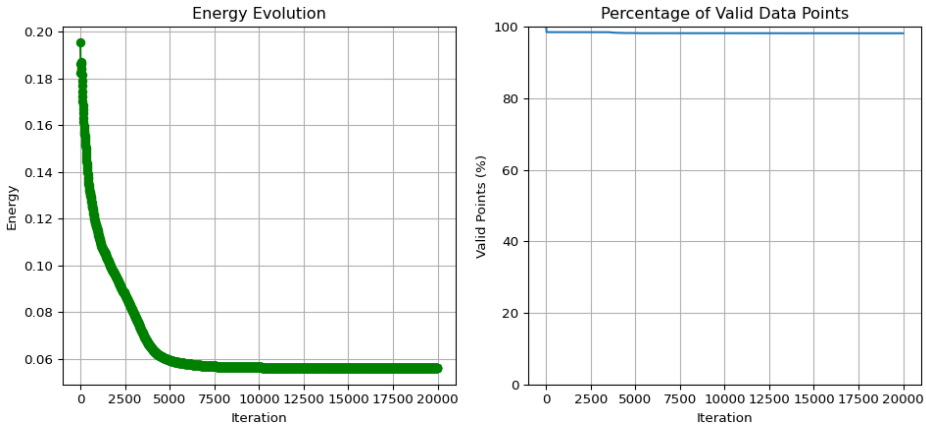

Figure 5.19: Valid points detected from which data fidelity term is calculated is near 100% which is wrong

Case 3.4: 0px, 10px shift

- **Iterations:** 20,000
- **Regularization λ:** 0.75
- **Result :** a(x,y) at center = -10.8px and b(x,y) at center = -4.96px

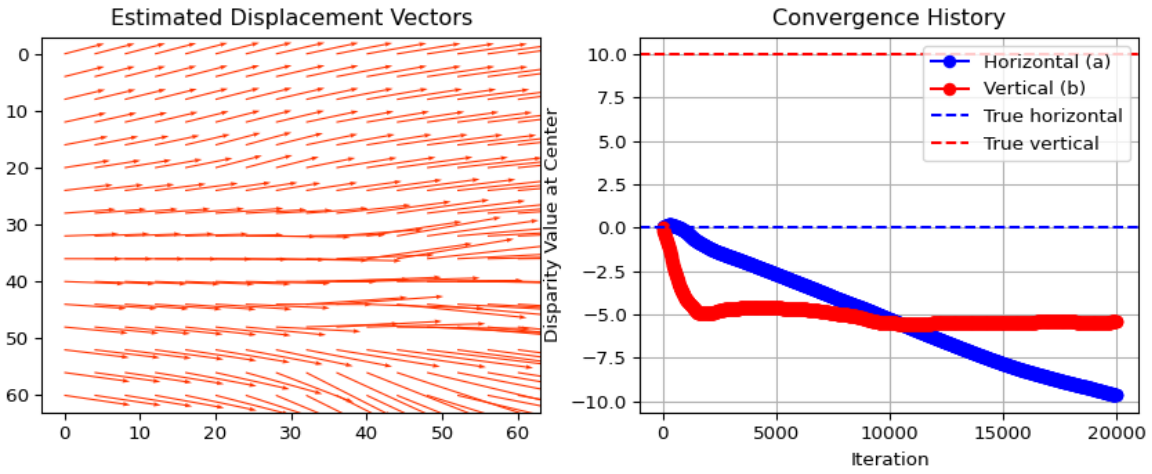

Figure 5.20: Convergence with 20,000 iterations

We observe that there is no way out for this case since the energy functional is getting stuck at a local minima (as seen previously). The vertical displacement field tries to go to the displaced value however the strong regularization **dominates** the vertical deviation due to the definite horizontal displacement. Since the top left box is creating a horizontal field, a $\lambda = 0.75$ induces a strong effect of horizontal on the symmetric vertical field to completely bias it towards horizontal direction.

4. Quantitative Results

In this section, I would talk about comparing the $a(x, y)$ and $b(x, y)$ against ground truth values which is essentially Pixel-to-Pixel squared error. I have earlier presented a similar comparison of $a(x, y)$ and $b(x, y)$ values at the center of the image with the ground truth shift.

One can calculate the RMSE and Pixel-to-Pixel error of the $a(x, y)$ and $b(x, y)$ with the ground truth, however I have not provided that. I believed it would be less descriptive of what is happening under the hood if a single scalar metric is provided for a whole grid. Therefore, I have provided the learned $a(x, y)$ and $b(x, y)$ values and compared with the ground truth at the center of the grid in the above section and have provided with a map of displacement vectors all over the grid giving a great qualitative overview of the spread of the $a(x, y)$ and $b(x, y)$ field.

5. Experiment with real-world images

In this final section, I would now just lay out PDE convergence results on real-world stereo image. I unfortunately don't have ground truths for the images. But the analysis is quite critical in terms of understanding a different effect of PDE. I have divided the testing into 3 cases as below.

Case 1: Grayscale Stereo Image downsampled to 64px X 64px

In this scenario, I have taken an image and down-sampled it to 64px X 64px from 720px X 720px resolution. Here is the PDE convergence plots for the earlier decided optimal parameters (in section 2 of evaluation).

Optimal Parameter Choice: $\lambda = 0.5$ and $T = 20,000$

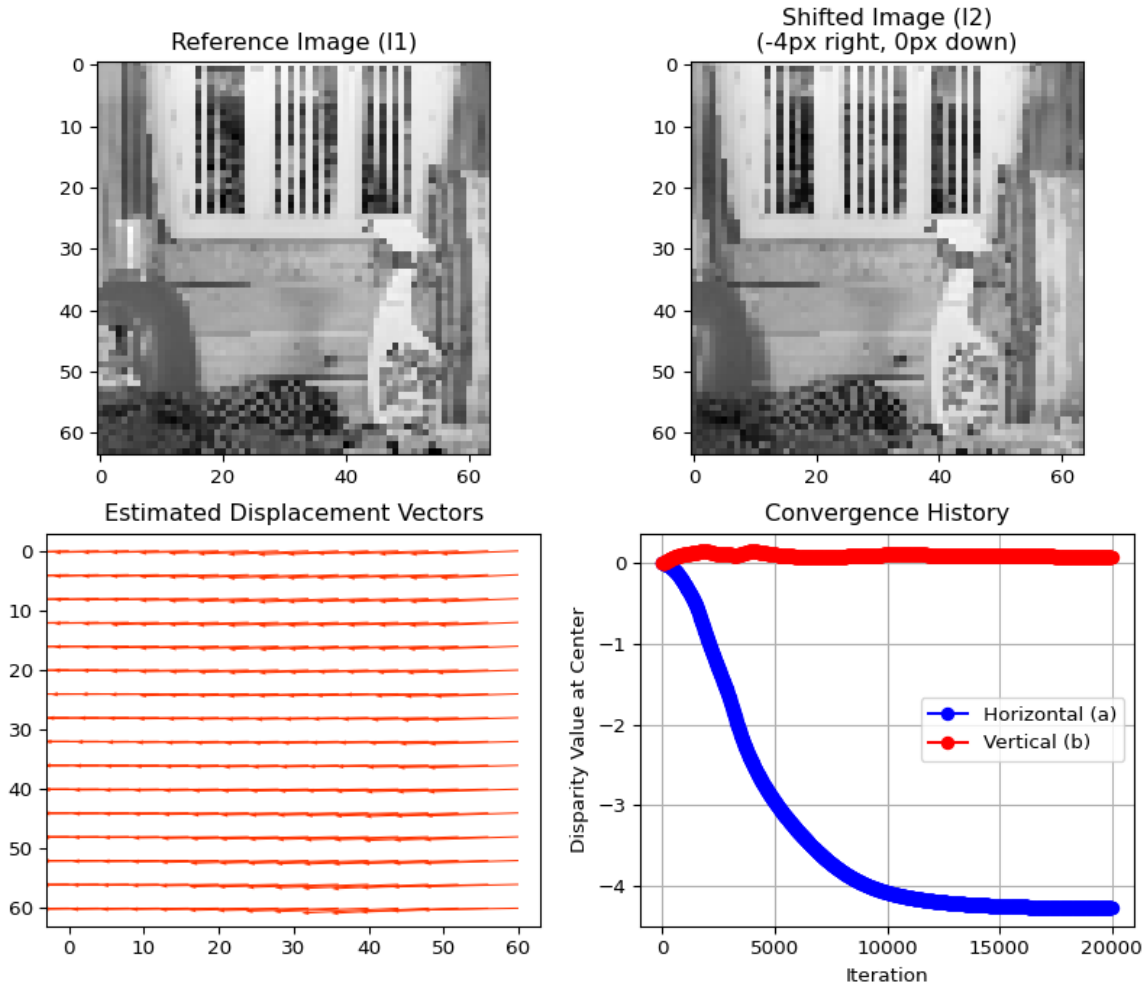

Figure 5.21: Grayscale Image

As can be seen that the horizontal disparity is **beautifully** captured to be around -4px which seems about right qualitatively. The vertical disparity is 0 as well!

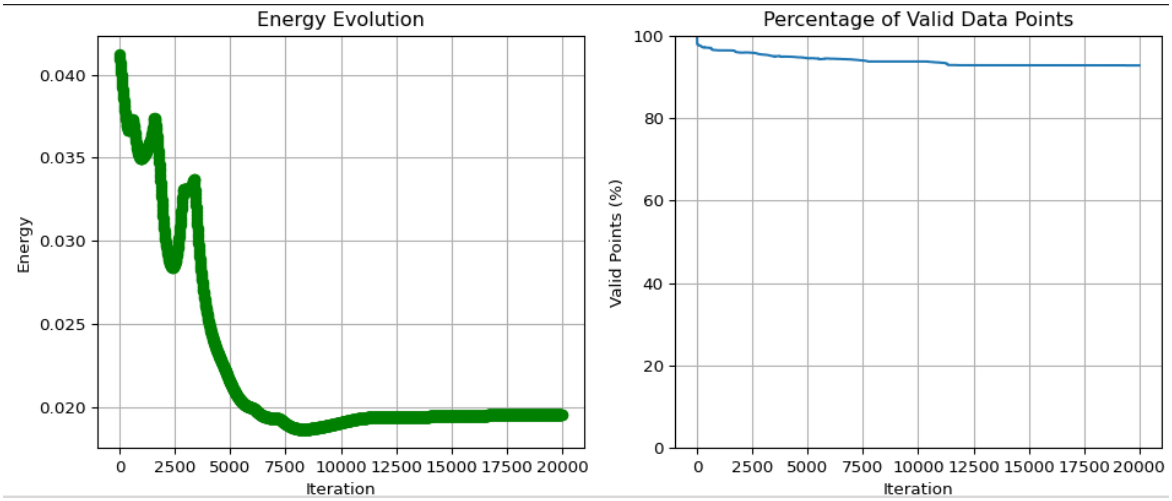

Figure 5.22: Energy Function plot and Points used for Data Fidelity Calculation

The energy plot is seen to be decreasing to a stable minimum value and holding it after 12,500 timestep. This indicates that minimum value is acheived.

Case 2: Grayscale Stereo Image downsampled to 128px X 128px

In this scenario, I have taken an image and down-sampled it to 128px X 128px from 720px X 720px resolution. Here is the PDE convergence plots for optimal parameters decided in section 2 of evaluation.

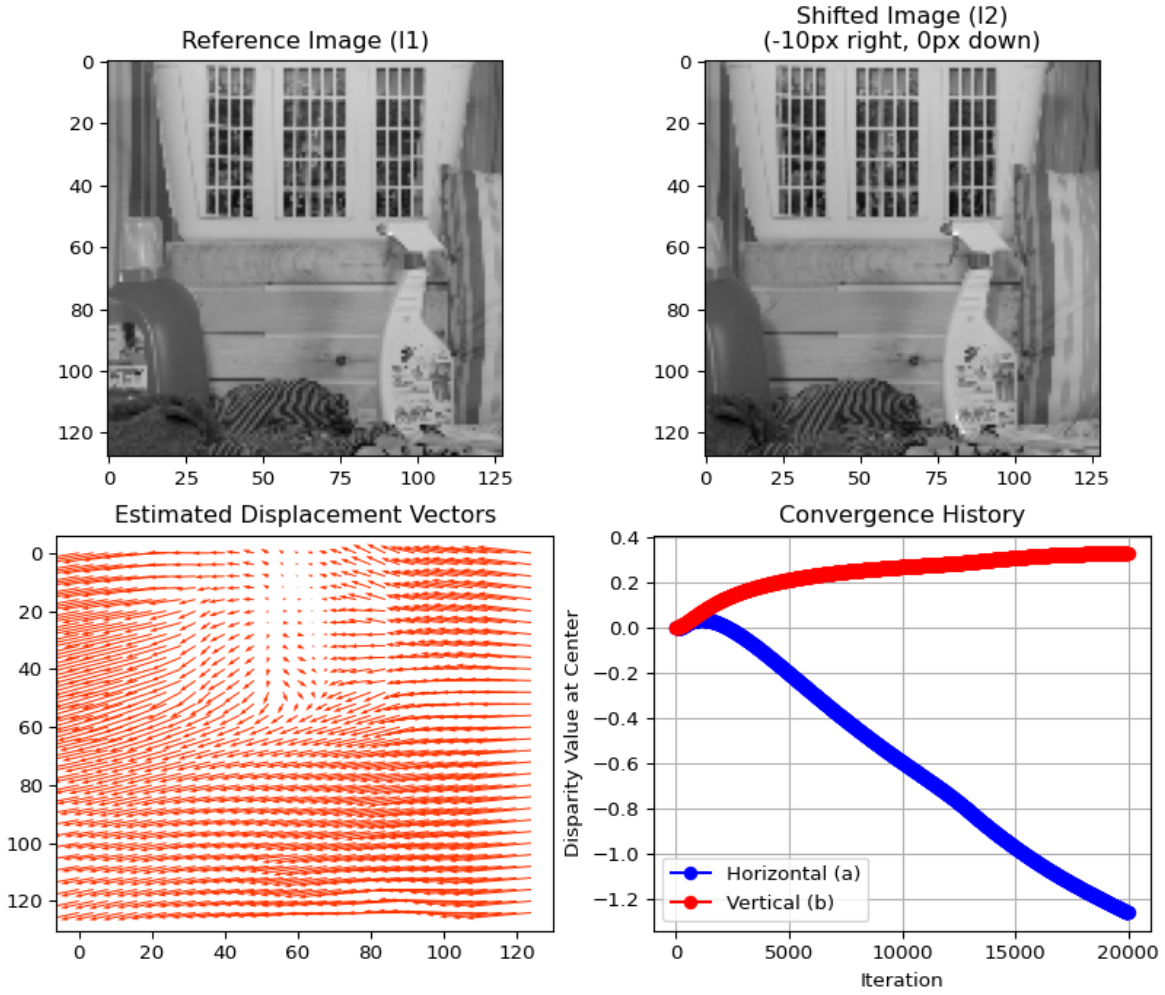

Figure 5.23: Image with its stereo pair

In this case, we can comment that direction of the displacement vectors is almost correct as it is majorly in the horizontal direction. However, carefully analysing the plot we can see that the displacement vectors near the window are symmetrical and rather of small magnitude which is due to the same symmetry fact that was discussed in Case 3.3 above in section 3.

This performs poorer than 64px X 64px since there are way too many points in the data fidelity term to compute from. Therefore, **stereo matching is quite sensitive to high resolution features**. Downsampling it to lower values helped capture low-frequency dominant features and therefore disparity could converge to a stable global value.

Case 2: Grayscale Stereo Image downsampled to 256px X X 256px

In this scenario, I have taken an image and down-sampled it to 256px X 256px from 720px X 720px resolution. Here is the PDE convergence plots for optimal parameters decided in section 2 of evaluation.

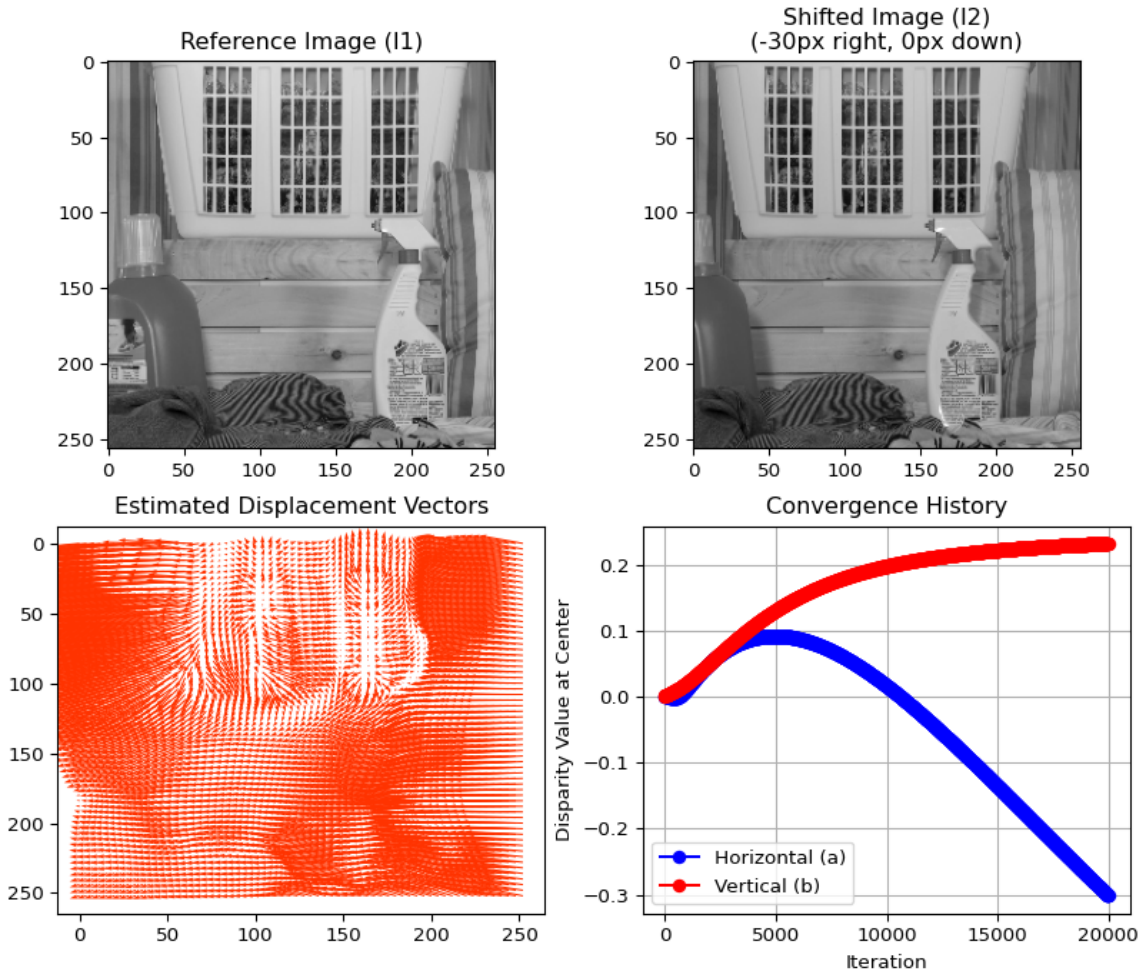

Figure 5.24: Image with its stereo pair

In this case, we can observe that the image at this resolution performs poorer than lower resolution images thus proving the **fact stated above**. It can be seen that now even the vertical disparity is non-zero which is false since the image primarily has only horizontal disparity.

Key Observation: The energy formulation stated above works best for low resolution images using the discretization schemes provided.

Chapter 6

Additional Learning

In this section, I would like to discuss a few observations I noticed due about the generalized 2D stereo disparity PDE. Below are the comments.

Some **Positive Points** about the PDE.

- The energy formulation is quite simple and works brilliantly for images which lack complex symmetries that confuse the PDE to optimize to a different solution.
- The smoothness term used is a laplacian. This is quite intuitive and works as a very reliable diffuser to impart the effect onto neighboring pixels.
- We saw low-resolution images benefit from this PDE formulation since they capture more dominant behavior and changes in the image. This is desirable since low-resolution images are computationally less intensive to work with!
- The tradeoff between regularization term and data-fidelity term allows to balance for noisy images and lets them converge to a stable solution.

Some **Negative Points** about the PDE.

- The energy formulation reaches its limitation to accurately estimate displacement field for few complex images as was discussed earlier. More better discretization techniques must be used inorder to overcome that.
- The diffusion term evolves very slowly which means that the PDE has to be run for a whole lot timesteps inorder to let it converge.
- High-Resolution Images that are too cluttered fail to benefit from the PDE formulation due to a highly sensitive data-fidelity term which can take large amount of time to stabilize.

Some **Next Steps** to explore.

- Use a non-constant regularization parameter that adaptively reduces contribution of similar looking pixels and increases contribution of diverse ones.
- Since high-frequency features disturb the solution to which disparity converges, use a smoothing filter to smoothen out the high frequency features and retain only the low-resolution dominant ones.

Chapter 7

Appendix (Code)

The link for the code is available on: link
Here is the python code attached below used for the result generation.

```
"""
Title: 2D generalised stereo disparity flow implementation for displacement estimation
Final Version - Updated 2025.04.22 (had issues with boundary conditions)
Name: Rahul Rustagi
GTID: 904024521
"""

import numpy as np
from matplotlib import pyplot as plt
from time import time
import cv2

### Helper functions section ###

def get_interpolated(image, x, y):
    """Bilinear interpolation with zero-pad for OOB
    (Tried different padding first, zero worked best)
    """
    H, W = image.shape
    x_floor = np.floor(x).astype('int32')
    y_floor = np.floor(y).astype('int32')
    x_ceil = x_floor + 1
    y_ceil = y_floor + 1

    # Weights calculation
    dx = x - x_floor
    dy = y - y_floor
    wt_tl = (1-dx)*(1-dy)  # top-left weight
    wt_tr = dx*(1-dy)
    wt_bl = (1-dx)*dy
    wt_br = dx*dy

    # Initialize with zeros (handles OOB automatically)
    interpolated = np.zeros_like(x)

```

```
    # Check each corner with boundary constraints
    # (This mask approach was faster than np.clip)
    valid_tl = (x_floor >= 0) & (x_floor < W) & (y_floor >= 0) & (y_floor < H)
    interpolated[valid_tl] += wt_tl[valid_tl] * image[y_floor[valid_tl], x_floor[valid_tl]]

    valid_tr = (x_ceil >= 0) & (x_ceil < W) & (y_floor >= 0) & (y_floor < H)
    interpolated[valid_tr] += wt_tr[valid_tr] * image[y_floor[valid_tr], x_ceil[valid_tr]]

    valid_bl = (x_floor >= 0) & (x_floor < W) & (y_ceil >= 0) & (y_ceil < H)
    interpolated[valid_bl] += wt_bl[valid_bl] * image[y_ceil[valid_bl], x_floor[valid_bl]]

    valid_br = (x_ceil >= 0) & (x_ceil < W) & (y_ceil >= 0) & (y_ceil < H)
    interpolated[valid_br] += wt_br[valid_br] * image[y_ceil[valid_br], x_ceil[valid_br]]

    return interpolated

def apply_laplacian(matrix):
    """5-point stencil laplacian with reflection BCs
    (Experimented with zero-padding first, reflection gave better edge behavior)

    The formulation is:    f    = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) -4f(x,y)
    """
    # Add reflective padding to handle borders
    padded = np.pad(matrix, pad_width=1, mode='reflect')

    # Stencil implementation
    laplacian = (padded[:-2, 1:-1]    # i-1,j
                + padded[2:, 1:-1]    # i+1,j
                + padded[1:-1, :-2]   # i,j-1
                + padded[1:-1, 2:]    # i,j+1
                - 4*padded[1:-1, 1:-1])
    return laplacian

def calculate_gradients(img):
    """Central difference gradients with simple edge handling
    (Tried forward diff first but central worked better)
    """
    grad_x = np.zeros_like(img)
    grad_y = np.zeros_like(img)

    # X-direction (columns)
    grad_x[:, 1:-1] = (img[:, 2:] - img[:, :-2]) / 2.0
    # Y-direction (rows)
    grad_y[1:-1, :] = (img[2:, :] - img[:-2, :]) / 2.0

```

```
    # Handle edges by replication (simple approach)
    grad_x[:, 0] = grad_x[:, 1]
    grad_x[:, -1] = grad_x[:, -2]
    grad_y[0, :] = grad_y[1, :]
    grad_y[-1, :] = grad_y[-2, :]

    return grad_x, grad_y

### Main algorithm ###

# Experiment parameters
GRID_SIZE = 64  # Changed from 128 to 64 for faster testing
LAMBDA = 0.5  # Regularization parameter
MAX_ITERS = 20  # Max iterations (usually converges faster)
TRUE_SHIFT_X = 0  # Ground truth X shift
TRUE_SHIFT_Y = 10   # Ground truth Y shift

# I1 = cv2.imread('/home/rrustagi7/Desktop/view0_1.png', cv2.
    IMREAD_GRAYSCALE) / 255
# I2 = cv2.imread('/home/rrustagi7/Desktop/view6_1.png', cv2.
    IMREAD_GRAYSCALE) /255
# # cv2.imshow('I1', I1)
# # Resize images to N x N
# I1 = cv2.resize(I1, (GRID_SIZE, GRID_SIZE))
# I2 = cv2.resize(I2, (GRID_SIZE, GRID_SIZE))

# base_img = I1
# shifted_img = I2

# Create test images with square pattern
base_img = np.zeros((GRID_SIZE, GRID_SIZE))
square_coords = slice(20,40), slice(20,40)
# base_img[square_coords] = 1.0
base_img[20:30, 20:40] = 1.0  # Centered box
base_img[35:45, 20:40] = 1.0  # Centered box
base_img[5:15, 10:20] = 1.0  # Centered box
# Shifted image (simulate translation)
shifted_img = np.roll(base_img, TRUE_SHIFT_Y, axis=0)
shifted_img = np.roll(shifted_img, TRUE_SHIFT_X, axis=1)

# Precompute gradients of shifted image
grad_x, grad_y = calculate_gradients(shifted_img)

# Initialize displacement fields (a=horizontal, b=vertical)
disp_x = np.zeros_like(base_img)
disp_y = np.zeros_like(base_img)

a_history = []  # Store horizontal displacement history
b_history = []  # Store vertical displacement history
E_history = []  # Store energy history
valid_points = [] # Store valid points for visualization


# Create coordinate grids (switched to meshgrid for readability)
col_grid, row_grid = np.meshgrid(np.arange(GRID_SIZE),
```

```
                                    np.arange(GRID_SIZE))
# Convert to float for displacement updates
col_grid = col_grid.astype('float64')
row_grid = row_grid.astype('float64')

# Optimization loop
start_time = time()

for iteration in range(MAX_ITERS):
    # Current displaced coordinates
    warped_cols = col_grid + disp_x
    warped_rows = row_grid + disp_y

    # Interpolate shifted image and gradients
    I2_warped = get_interpolated(shifted_img, warped_cols,
   warped_rows)
    I2_wx = get_interpolated(grad_x, warped_cols, warped_rows)
    I2_wy = get_interpolated(grad_y, warped_cols, warped_rows)

    # Calculate valid points percentage
    valid_mask = ((warped_cols >= 0) & (warped_cols < GRID_SIZE) &
                  (warped_rows >= 0) & (warped_rows < GRID_SIZE))
    valid_count = np.mean(valid_mask) * 100  # Percentage
    valid_points.append(valid_count)

    # Compute residual and gradient terms
    residual = I2_warped - base_img
    Rx = residual * I2_wx
    Ry = residual * I2_wy

    # Regularization terms
    lap_x = apply_laplacian(disp_x)
    lap_y = apply_laplacian(disp_y)

    # Dynamic time step calculation
    max_grad = max(np.abs(Rx).max(), np.abs(Ry).max())
    denominator = 4*LAMBDA
    dt = 2.0 / (denominator + 1e-12)  # Prevent division by zero
    dt = min(min(dt, 2/max_grad),0.25)  # Stability limit
    # print(dt)

    # Update displacement fields
    disp_x += dt * (LAMBDA * lap_x - Rx)
    disp_y += dt * (LAMBDA * lap_y - Ry)

    # Store history
    if iteration % 10 == 0:
        a_history.append(disp_x.copy())
        b_history.append(disp_y.copy())
        # Compute energy for convergence check
        E_history.append(np.mean(residual**2) + np.mean(lap_x**2) +
   np.mean(lap_y**2))

    # Optional: Add convergence check here
```

```
187     # if iteration % 100 == 0:
188     #     print(f"Iter {iteration}: max displacement {np.hypot(disp_x
    , disp_y).max():.3f}")
189
190 print(f"Computation time: {time()-start_time:.2f}s")
191
192 ### Visualization ###
193 plt.figure(figsize=(10, 8))
194
195 # Original images
196 plt.subplot(2, 2, 1), plt.imshow(base_img, cmap='gray')
197 plt.title('Reference Image (I1)')
198
199 plt.subplot(2, 2, 2), plt.imshow(shifted_img, cmap='gray')
200 plt.title(f'Shifted Image (I2)\n({TRUE_SHIFT_X}px right, {
    TRUE_SHIFT_Y}px down)')
201
202 # Displacement field (every 4th arrow)
203 skip = 4
204 plt.subplot(2, 2, 3)
205 plt.quiver(col_grid[::skip, ::skip],
206            row_grid[::skip, ::skip],
207            disp_x[::skip, ::skip],
208            -disp_y[::skip, ::skip],  # Y-axis flip for display
209            scale=25, color='#FF3300', width=0.003)
210 plt.gca().invert_yaxis()  # Match image coordinates
211 plt.title('Estimated Displacement Vectors')
212
213 # Convergence plots
214 plt.subplot(2, 2, 4)
215 plt.plot(np.arange(0, MAX_ITERS, 10), [disp_x[GRID_SIZE//2, GRID_SIZE
    //2] for disp_x in a_history], 'b-o', label='Horizontal (a)')
216 plt.plot(np.arange(0, MAX_ITERS, 10), [disp_x[GRID_SIZE//2, GRID_SIZE
    //2] for disp_x in b_history], 'r-o', label='Vertical (b)')
217 # plt.axhline(TRUE_SHIFT_X, color='b', linestyle='--', label='True
    horizontal')
218 # plt.axhline(TRUE_SHIFT_Y, color='r', linestyle='--', label='True
    vertical')
219 plt.xlabel('Iteration')
220 plt.ylabel('Disparity Value at Center')
221 plt.title('Convergence History')
222 plt.legend()
223 plt.grid(True)
224
225 plt.figure(figsize=(10, 6))
226
227 # plot the evolution of the energy
228 plt.subplot(1, 2, 1),
229 plt.plot(np.arange(0, MAX_ITERS, 10), E_history, 'g-o')
230 plt.xlabel('Iteration')
231 plt.ylabel('Energy')
232 plt.title('Energy Evolution')
233 plt.grid(True)
234
235 # Valid Points Tracking
```

```
236 plt.subplot(1, 2, 2),
237 plt.plot(valid_points)
238 plt.ylim(0, 100)
239 plt.xlabel('Iteration')
240 plt.ylabel('Valid Points (%)')
241 plt.title('Percentage of Valid Data Points')
242 plt.grid(True)
243
244 plt.tight_layout()
245
246 # Print center values for verification
247 center = GRID_SIZE//2
248 print(f"Center displacement - X: {disp_x[center,center]:.2f} (true {
    TRUE_SHIFT_X})")
249 print(f"Center displacement - Y: {disp_y[center,center]:.2f} (true {
    TRUE_SHIFT_Y})")
250
251 # Save figure for report
252 # plt.savefig('displacement_results.png', dpi=150)
253 plt.show()
```