Image Processing and Restoration under Atmospheric Turbulence

Lucky-Region Fusion and Motion Detection through Atmospheric Turbulence

Francis Alvarez
Nicholas Ferrante
Margaret Fortman
Lena Tahir
Alex Tarter
Anneke von Seeger

Project Advisor: Dr. Jérôme Gilles

A report presented for program of the 2016 Mathematics Research Experience for Undergraduates

Department of Mathematics
San Diego State University
USA

12 August 2016

Image Processing and Restoration

Lucky-Region Fusion and Motion Detection through Atmospheric Turbulence

Francis Alvarez
Nicholas Ferrante
Margaret Fortman
Lena Tahir
Alex Tarter
Anneke von Seeger

Project Advisor: Dr. Jérôme Gilles

Abstract

Clear, high-resolution images and video are useful in a wide range of applications, from finding speeding cars to identifying faces or people over a long distance. Several approaches exist for improving the quality of images. One problem that exists in the image processing field is the restoration of an image that undergoes distortion due to turbulence. The goal of this paper is to outline three topics within the image turbulence mitigation community. We discuss the creation of an open source dataset, designed to bring better communication among the image processing community. We also demonstrate methods involving deformation flow for static image restoration and motion detection for a sequence of images with turbulence present. Assorted methods and experiments are outlined demonstrating results of the algorithms we use.

A more complete version of this report (with more experiments and illustrations) is available at `http://www-rohan.sdsu.edu/~jegilles/doc/REU2016_Imaging_Final_Report.pdf`.

Acknowledgements

We would like to thank Dr. Jérôme Gilles and Dr. Vadim Ponomarenko for their continuous mathematical, as well as moral, support throughout the program.

Contents

Introduction

A second approach to improving image quality is processing the image after it has been captured by the camera. Image processing can be applied in two steps: removing blur and correcting geometric distortion.

Taking images over a long distance can exacerbate both blurring and geometric distortion. Images taken over a long distance also generally show stronger effects from turbulence, especially when taken somewhere hot, such as in a desert or on a highway. Under those conditions, the heat waves visible over a hot road in summer are captured by the camera and distort the image.

In addition to making it more difficult to recognize shapes and objects in distorted images, turbulence can also make it more difficult to recognize moving objects in videos. In a turbulent video, the entire scene appears to move; it can be extremely difficult for both human observers and computers to identify real moving objects.

This report will first attempt to address the lack of a common dataset for testing image processing methods. The dataset will allow easy comparison of different image processing algorithms. It then presents methods attempting to address turbulence in both still images and video sequences. The methods for still images will focus on image restoration. Optical flows are used to calculate divergence at each pixel in an image. The intuition that positive divergence suggests a zoomed-in area while negative divergence indicates a zoomed-out area, provides motivation for incorporating divergence into image restoration. The first set of methods in this report incorporates divergence into lucky-imaging restoration techniques, based on the methods presented in [12] and [1]. The last method presented is based on work in [7] and in [5], which focuses on super-resolution of images using zoom as a cue. The method presented here incorporates divergence as an indicator of zoom.

The methods for video sequences focus on motion detection in turbulent sequences. Many algorithms exist, with varying degrees of success, that display the motion of objects across a frame in a sequence. One algorithm, outlined in [6], works well for non-distorted sequences, but our results show that the algorithm is less successful when applied to turbulent sequences. Further, decomposing an image into its structure and texture components allows the separation of bounded variation from highly oscillatory components. In noisy images, decomposition removes the noise with the texture. Though methods exist to separate noise from single images, little work has been done to remove turbulence from entire sequences of images. We extend a known algorithm, outlined in [2], to incorporate time evolution of a turbulent sequence and track constant linear motion across a frame.

Chapter 1

Open Turbulence Image Set

1.1 Introduction to the dataset (OTIS)

Long distance imaging is subject to the impact of the turbulent atmosphere. This results into geometric distortions and some blur effect in the acquired frames. Despite the existence of several turbulence mitigation algorithms in the literature, no common dataset exists to objectively evaluate their efficiency. In this paper, we describe a new dataset called OTIS (Open Turbulent Images Set) which contains several sequences (both static or with a moving target) acquired through the turbulent atmosphere. For many sequences, we provide the corresponding groundtruth in order to make the comparison easier.

1.1.1 Equipment

All sequences were acquired by a GoPro Hero 4 Black camera modified with a RibCage Air chassis permitting to adapt several type of lenses. We always used a 25mm, f/2.0 14d HFOV 3MP lens. The camera was setup at a 1080p resolution and a framerate of 24 frames per second (fps). A small tripod was used to hold the camera (see Figure 1.1). The camera was controlled by the usual GoPro App on a Samsung Galaxy tablet.

Figure 1.1: RibCage Air Modified GoPro Hero 4 Black camera with a 25mm f/2.0 14d HFOV 3MP lens on its tripod.

The acquired sequences contain both natural elements from the observed scene

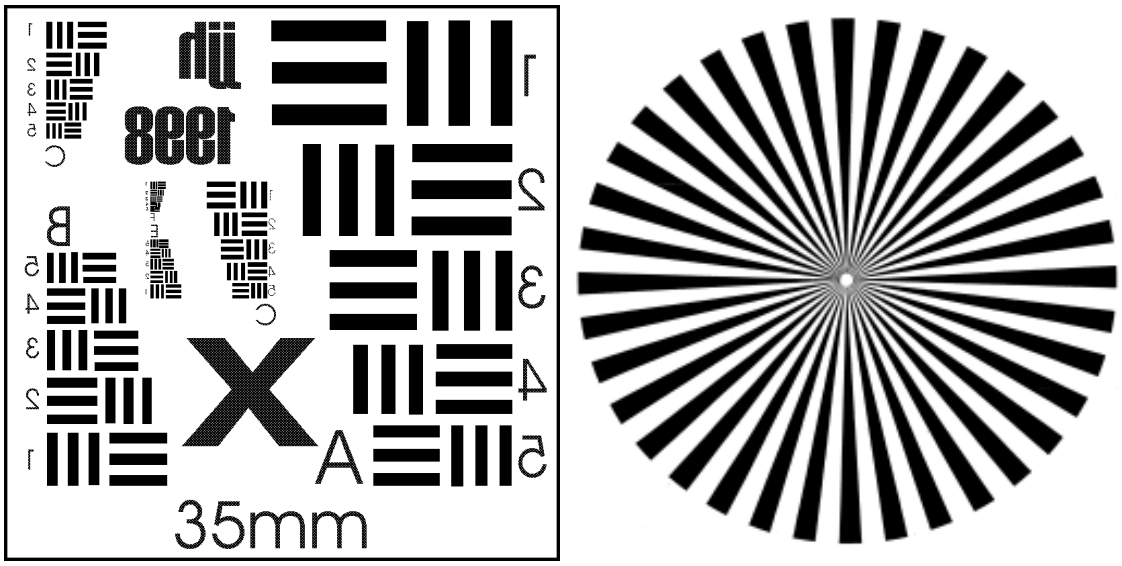


Figure 1.2: The two charts serving as our static artificial targets after being printed on a poster.

Figure 1.3: Remote Controlled car utilized as our moving target for the dynamic sequences.

as well as artificial "targets". For the static sequences, we used two charts containing some geometric patterns at different spatial frequencies and orientations (see Figure 1.2). These charts were printed on a poster and hold by a homemade wooden stand. For the dynamic sequences, we used a standard remote controlled car (see Figure 1.3).

1.1.2 Procedures

All acquisitions were made on hot sunny days in order to guaranty a certain amount of atmospheric turbulence. Moreover, all equipments were setup on a practice field

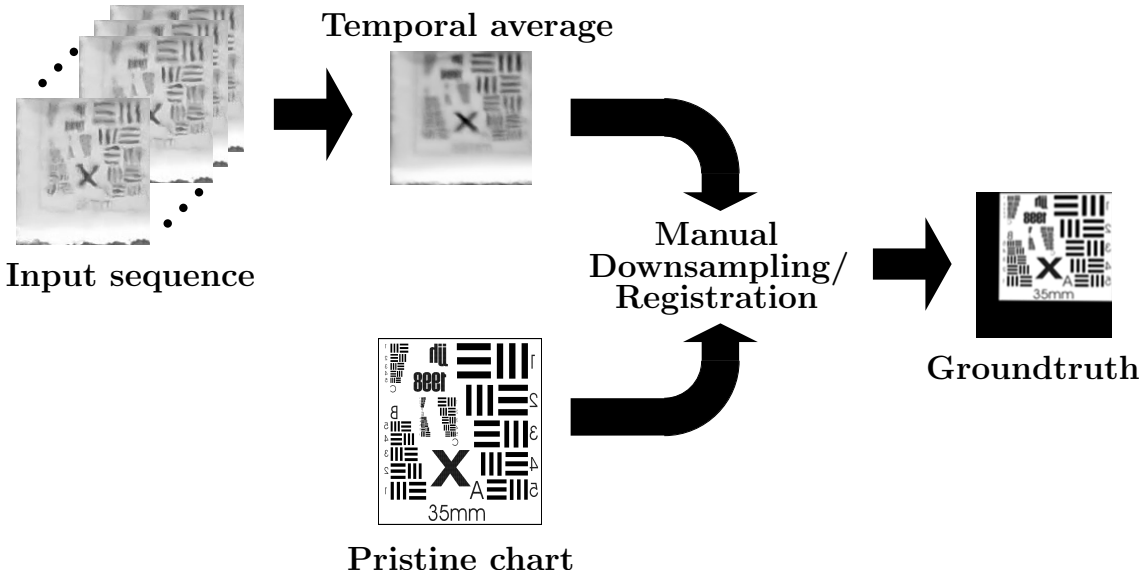


Figure 1.4: Groundtruth images creation procedure.

equipped with artificial turf since such material provide higher temperature gradient leading to higher level of turbulence. The camera stood at about 10cm above the ground observing the target positioned at several distances.

After all acquisitions are done, the different recorded movies were downloaded on a Linux computer and split into sequences of PNG image files. The different region of interest are finally cropped and saved as individual PNG sequences and stored. Since the Matlab® software is widely used by the community, we also provide each sequence as a Matlab 3D matrix (the first two coordinates are the spatial coordinates while the third one corresponds to time) saved in a .mat file.

Since the purpose of this dataset is to be used for evaluating turbulence mitigation algorithms, all sequences containing the two above mentioned charts are provided with a ground truth image. This groundtruth image contains the pristine chart after being downsampled and registered to the actual sequence (in practice, we manually register the pristine chart on a temporal average of the input sequence using the GIMP [1] software). This procedure is summarized in Figure 1.4.

1.2 Collected data

1.2.1 Static sequences

We can distinguish two types of static sequences, those containing observations of natural elements in the scene (doors, steps, signs,...) and those made of the two previous charts. The former ones are then provided with a groundtruth image in order facilitate future algorithm evaluation.

[1] `https://www.gimp.org/`

1.2.2 Dynamic sequences

Dynamic sequences contain a moving remote controlled car at different distances and directions. Our goal was to create both sequences where the moving car is still easy to detect even if it is affected by geometric distortions, as well as sequences where the car movement magnitude is close to the magnitude of the turbulence hence making them more challenging to distinguish.

1.2.3 Information about sequences

Within this paper, sequences are utilized from OTIS. These are listed below:

Table 1.1: Summary of the different static scenes

Filename	Type	Size (W×H)	# frames	Distance (m)	Turbulence level
Steps	Static	400 × 500	493	100	Moderate
Barchart 1	Static	271 × 150	622	50	Moderate-Heavy
Barchart 4	Static	366 × 201	486	30	Light-Moderate
Barchart 4 Box	Static	182 × 201	486	30	Light-Moderate
Barchart 6	Static	482 × 260	592	15	Light
CarSeq 2A	Dynamic	360 × 100	100	40	Moderate

1.3 Continuing work

OTIS's construction will continue throughout the academic school year for 2016/2017 as the dataset manager and adviser both attend and work at San Diego State University. It is hoped to be published soon in an academic journal and a public Beta of the functioning dataset. More sequences of varying turbulence and background will continue to be created, as the weather in August and September is significantly more conducive to atmospheric turbulence.

Chapter 2

Image Fusion

2.1 Lucky-region fusion (LRF)

2.1.1 Lucky-region fusion method

While a sequence of images are distorted due to atmospheric turbulence the lucky-region fusion method aims to correctly fuse the sequence into one clear compiled image. This process is possible since the LRF algorithm [1] assumes that there are certain regions in the sequence which are of higher quality, "lucky-regions". After iterating throughout the whole sequence only the portions with better image quality are taken and updated to the single fused synthetic image.

2.1.2 LRF algorithm

The vector r is defined as $r = (x, y)$ where (x, y) are the coordinate pixels in our image matrix.
$I^{(t)}(r) = I(r, t)$ is our original image at the coordinate (x, y) during frame t.

In order to determine what makes a sharper image there is the need to accurately detect edges. This is done by taking the norm of the gradient of our image, $|\nabla I(r, t)|$. After evaluating, we then scale this term by normalizing it over the entire intensity of the whole image. This procedure defines our image quality metric, J(r).

$$J(r) = \frac{|\nabla I(r, t)|}{\int I(r)dr} \tag{2.1}$$

Since there may be some noise after determining the Image Quality Metric the following step is needed to smooth J. This is done by taking the 2D convolution of J with a Gaussian Kernel, $G(r, a) = exp[-(x^2 + y^2)/a^2]$. We define this as the Image Quality Map (IQM), $M(r)$.

$$M(r) = \int J(r')G(r - r', a)dr' \tag{2.2}$$

The difficult part is choosing the scalar a, which is referred to as the kernel size. This is done through trial and error. After acquiring the IQM the next step is the iterative process to obtain our synthetic image of the fused sequence.

$$I_s^{(n+1)}(r) = [1 - \Delta^{(n)}(r)]I_s^{(n)}(r) + \Delta^{(n)}(r)I^{(n)}(r) \tag{2.3}$$

$I_s^{(n)}(r)$ is the current synthetic image at the n^{th} iteration. $\Delta(r,t)$ is the normalized anisotropic gain defined by $\Delta(r,t) = \delta(r,t)/max_{(r,t)}[\delta(r,t)]$ where

$$\delta^{(n)}(r,t) = \begin{cases} M(r,t_n) - M_s(r,t) & \text{for } M(r,t_n) > M_s(r,t) \\ 0 & \text{otherwise} \end{cases}$$

There are a few options as to what we can select for our initial fused image, $I_s^{(0)}$. The temporal average over the entire sequence of images is a typical choice and that is the method that was used in the experiments. Other options include using the first frame $I^{(0)}$ or using the image with the best image quality J_{img} which is defined as $J_{img} = \int J(r)dr$.

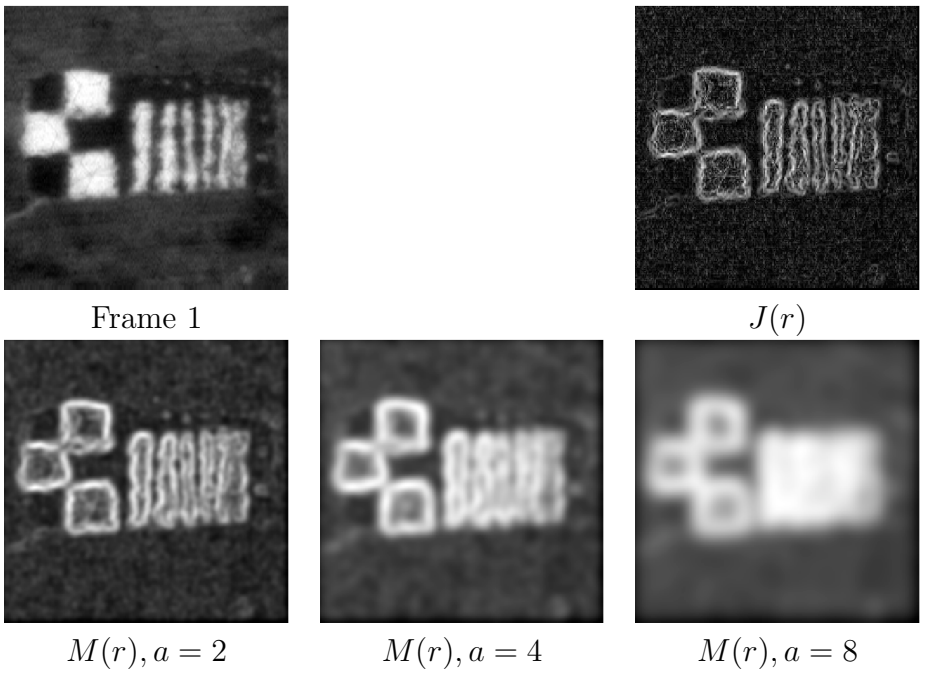


Figure 2.1: Initial steps for LRF algorithm

In Figure 2.1 we can notice how the different kernel sizes affect the amount of blur induced on the image. We can then see how that affects the final output image.

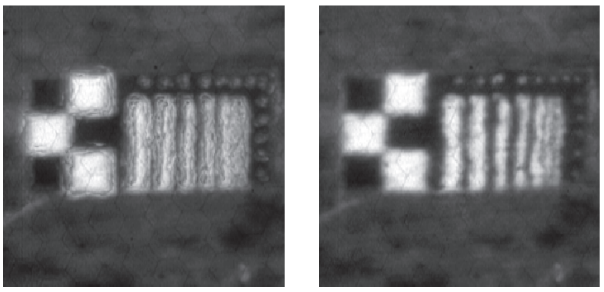


Figure 2.2: LRF fused images with different kernel sizes

If the kernel size is too small then the image does not receive much blur. This in turn leads to extra artifacts around hard edges when fusing the images together.

If the kernel size is too large then the image loses its sharpness and we are returned a blurred image. These results can be seen in Figure 2.2.

2.1.3 Implementation

Included are LRF results from various image sequences, all of which are undergoing varying intensities of turbulence.

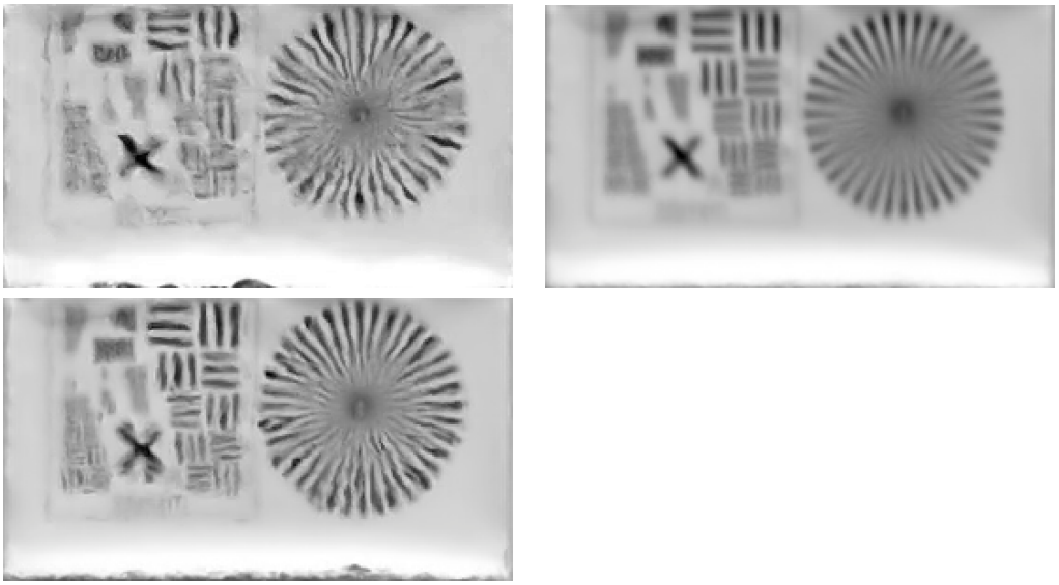


Figure 2.3: Topleft: Barchart1 Frame1, Topright: Average, Bottomleft: LRF $a = 4$

Figure 2.4: Barchart4Box Frame1, Average, LRF $a = 4$

Immediately after viewing the images one can see how the effect of turbulence influences the result of LRF. Sequences with minor turbulence are still geometrically visible, Figure 2.5. Images with high turbulence, Figures 2.3, 2.4, lead to an unsatisfactory result with the LRF approach. Because the image is shifting too much there are many edges and figures that do not line up which is why there is still high distortion in the final synthetic image. In order to improve this method the next step will be looking at the deformation flows and seeing if we can utilize those in any manner.

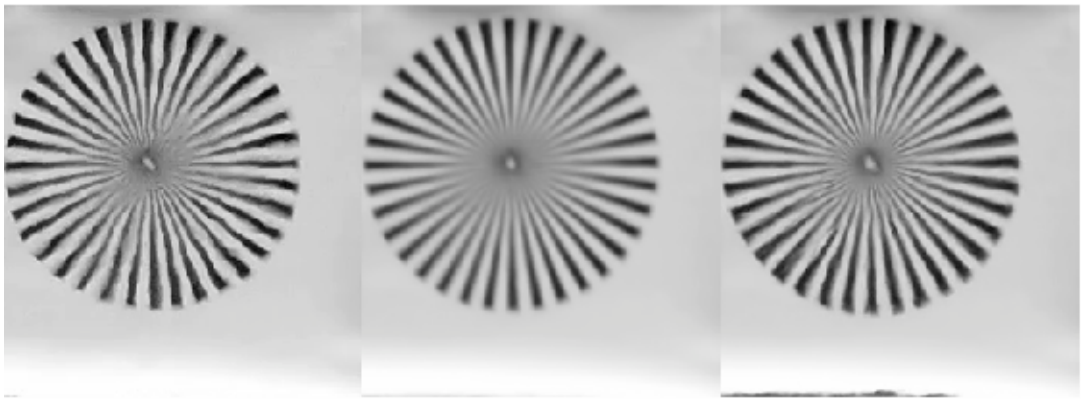


Figure 2.5: Barchart6Circle Frame1, Average, LRF $a = 4$

2.2 SSIM and MSE

Two metrics to measure the quality of the resulting images are mean-squared error (MSE) and structural similarity index (SSIM). We used the MSE and SSIM functions in Matlab to compare our images with the ground truths of each image.

The mean-squared error is a simple way to measure the quality of an image in comparison to a reference image. The MSE compares each pixel of the input image to the corresponding pixel in the reference image, calculating the difference between their intensities. The difference is then squared and the average of the squared intensities yields the mean-squared error.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\mathrm{I} - \mathrm{I_R})^2, \tag{2.4}$$

where I is the intensity of the input image and $\mathrm{I_R}$ is the intensity of the reference image. As the formula suggests, a lower MSE is desired, as it indicates that there is less difference between the input image and reference image.

As opposed to mean-squared error, the structural similarity index does not just measure absolute error but looks at the luminance, contrast, and structure of the image. The luminance measurement of the images is estimated by comparing the mean intensities of the image signals. Then the mean intensity is removed from each signal, and the contrast measurement is calculated by comparing the standard deviation of the resulting signals. Lastly, the signals are normalized, and the structure measurement is defined by the correlation between the normalized signals. The formulas for each of these comparisons are found in [3]. Finally, the components are combined to get the SSIM:

$$\mathrm{SSIM(I,I_R)} = [\ell(\mathrm{I},\mathrm{I_R})]^{\alpha} \cdot [\mathrm{c}(\mathrm{I},\mathrm{I_R})]^{\beta} \cdot [\mathrm{s}(\mathrm{I},\mathrm{I_R})]^{\gamma} \tag{2.5}$$

where $[\ell(\mathrm{I},\mathrm{I_R})]$ is the luminance comparison, $[\mathrm{c}(\mathrm{I},\mathrm{I_R})]$ is the contrast comparison, and $[\mathrm{s}(\mathrm{I},\mathrm{I_R})]$ is the structure comparison. The parameters α, β, γ adjust the importance of each measurement. The default for these values is $\alpha = \beta = \gamma = 1$. The output of the SSIM will be between -1 and 1, with a value of 1 meaning that there is no difference between the input and reference images.

Although MSE is easier to calculate and has a clear physical meaning, there are advantages of SSIM that justify using it. One advantage of using SSIM over MSE is that if the images are not aligned, the MSE may give an inaccurate measurement of the image quality. Also, MSE does not measure characteristics that are perceived in the human visual system. SSIM, however, aims to weight different aspects of the error according to how well they are seen by humans. The primary advantage of the SSIM is that it takes into account the geometry of the image.

2.3 Deformation flow and stabilization

2.3.1 Deformation flow explained

When working with a sequence of images it is possible that the original location of a certain pixel may translate from frame to frame. This is always the case when dealing with sequences involving movement, but also the case when images endure atmospheric turbulence. If the pixels are not in the same location in the following frame, $I_n(x, y) \neq I_{n+1}(x, y)$, then we can stabilize our images and find the deformation flow, ϕ, by solving

$$I_n(x, y) = I_{n+1}(x + \varepsilon_{x_n}, y + \varepsilon_{y_n}),$$

where $\phi_n := (\varepsilon_{x_n}, \varepsilon_{y_n})$. We can also determine the deformation flow with respect to a reference frame, φ. The reference frame used is either the temporal median or temporal average of the sequence. Those two choices are used because they give a reasonable shape of the how the sequence would appear without turbulence. Figure 2.6 illustrates the two different flow options.

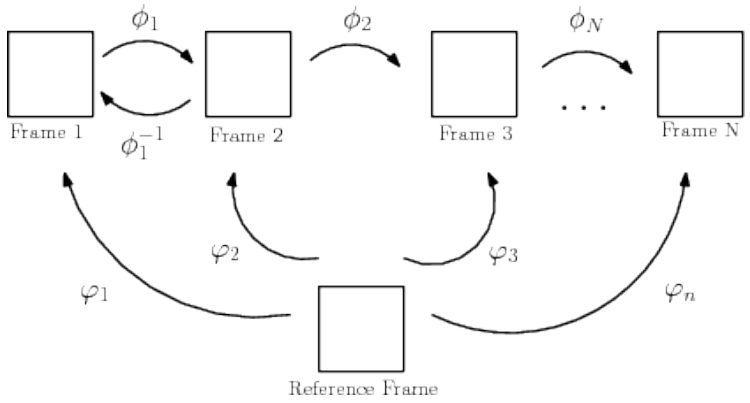


Figure 2.6: Deformation flow chart

2.3.2 Importance

Once the deformation flows are obtained the sequence can then be stabilized to a reference image or from frame to frame. There is also more that can be gained from acquiring the deformation flow. When looking at a vector field we can see that there are certain regions with an inward flow and an outward flow, for example Figure 2.7.

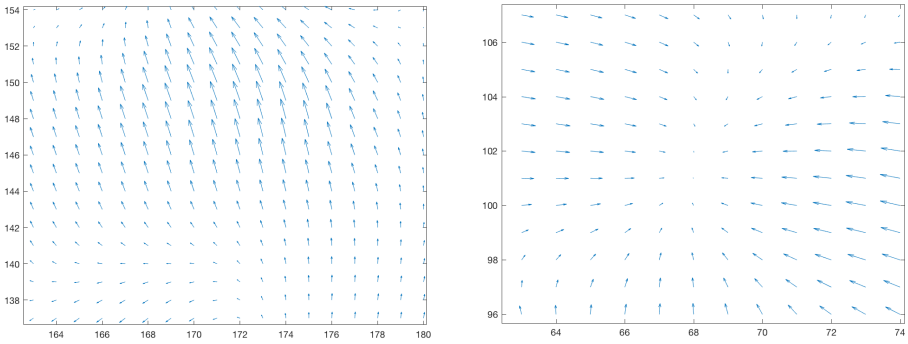


Figure 2.7: Deformation flow vector fields

At locations with outward flow we can expect there to be a zooming in effect while at locations with inward flow we expect there to be a zooming out effect. To determine the type of zoom that is occurring we can look at the divergence.

$$\begin{cases} div(x,y) > 0 & : \text{zooming in} \\ div(x,y) < 0 & : \text{zooming out} \end{cases}$$

2.3.3 Deformation flow methods used

In the context of the work done later we had only used two methods for estimating the deformation flow. We had used the Lucas-Kanade[14] (LK) method and Demon's Algorithm[10]. Both methods perform slightly differently which can be illustrated by their deformation flows in Figure 2.8. For that reason we will see how that difference impacts our results. In Figure 2.9 we can see how the LK method and Demon's algorithm perform to stabilize the sequence in regards to the temporal average reference frame.

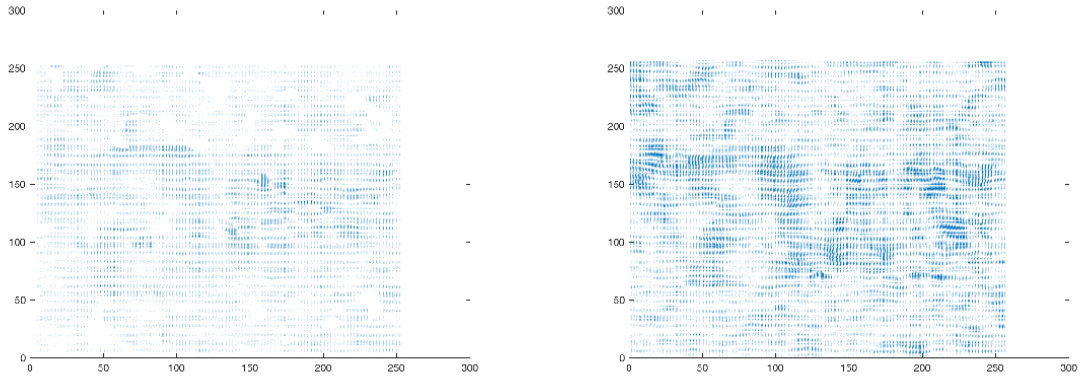


Figure 2.8: φ Deformation flow frame 1, LK method and Demon's algorithm

2.3.4 Comparing deformation flow methods

We can see that the Demon's algorithm performs significantly better with respect to the LK method. In regards of running time the LK method runs drastically quicker.

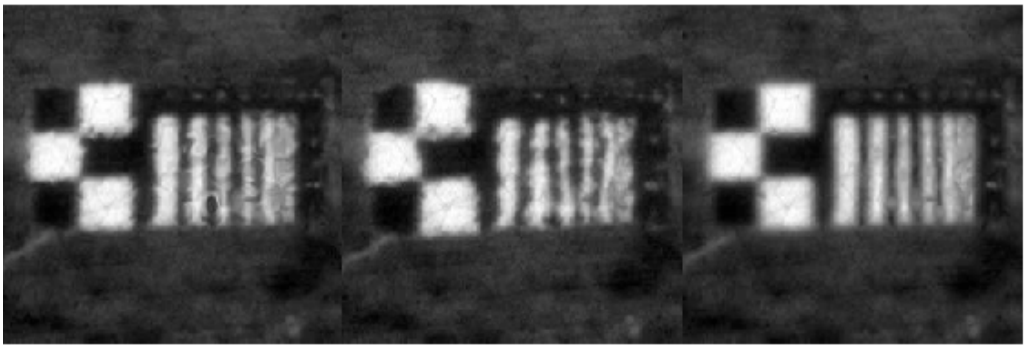


Figure 2.9: Fr 1: LK stabilized to reference, frame 1, Demon's stabilized to reference

The LK method was also used in part with the Mao-Gilles method to further stabilize the sequence which has comparable results to the Demon's algorithm.

2.3.5 Mao-Gilles method

The final technique used in this report was developed in [13]. The authors model the observed images as a version of the true image after being blurred and distorted, with the addition of noise.

$$f_i(x) = D_i(H(u(x))) + \text{noise} \tag{2.6}$$

Here, $f_i(x)$ is the observed image, $u(x)$ is the true image, H is a blurring kernel, and D_i is an operator that geometrically distorts the image. Geometric distortions are often modeled using optical flow algorithms. Many methods calculate optical flow from frame to frame; this method calculates the optical flow between each frame, $f_i(x)$, and a reference image, denoting the optical flow vectors as φ_i. With φ_i fixed, the deformation vectors can be treated as a linear operator on $u(x)$, and the general model becomes:

$$f_i = \varphi_i(u) + \text{noise}, \forall i \tag{2.7}$$

After including a regularization term, denoted $J(u)$, usually nonlocal total variation, φ_i can be estimated by minimizing the following equation:

$$\min_u \quad J(u) \text{ s.t. } f_i = \varphi_i(u) + \text{noise}, \forall i \tag{2.8}$$

The image is then stabilized by applying the inverse flows, φ^{-1}, to the input images. Full details of the implementation can be found in [13]. When using the method to stabilize images, the optical flows are calculated and the inverse flow vectors are applied to the distorted frames, resulting in an image that looks more like the reference image.

2.3.6 Comparing stabilization methods

Both the imregdemons and the Mao-Gilles algorithms can be used to stabilize images. Both methods can calculate optical flows between a reference image or frame-to-frame. Both also remove most of the geometric distortions. However, the image produced by imregdemons is slightly steadier and the imregdemons algorithm is much faster than the Mao-Gilles algorithm. On the other hand, the output of the

imregdemons algorithm is often noisier than the images produced by the Mao-Gilles algorithm and the imregdemons algorithm can result in artifacts near the edges of the frames which are not observed with the Mao-Gilles algorithm. Another issue for concern is when applying the imregdemons algorithm the stabilized image does not correctly adjust for shifted pixels along the border which leads to black noise on the edges. The choice of which stabilization method to use depends on the specific application; both are shown in most of the following sections.

2.4 Method 1

2.4.1 Notation

u is the restored image, or the estimate of the true image
$\{f_n\}$ is the sequence of input images
$\{\phi_n\}$ is the optical flow from frame to frame (flow from f_n to f_{n+1})
$\{\varphi_n\}$ is the optical flow between a reference frame (usually the temporal mean) and f_n
$\{D_n\}$ is the divergence for each optical flow (can be calculated using either ϕ or φ)
$\{B_n\}$ is the binary divergence map (1 if $D_n(x) > 0$, 0 otherwise)
$\{D_{n,+}\}$ is the divergence map for optical flow n, but with $D_n(x) < 0$ set to 0
K is the total number of frames in the input sequence
$\cdot\,(x)$ refers to the x^{th} pixel in that image (linear indexing)

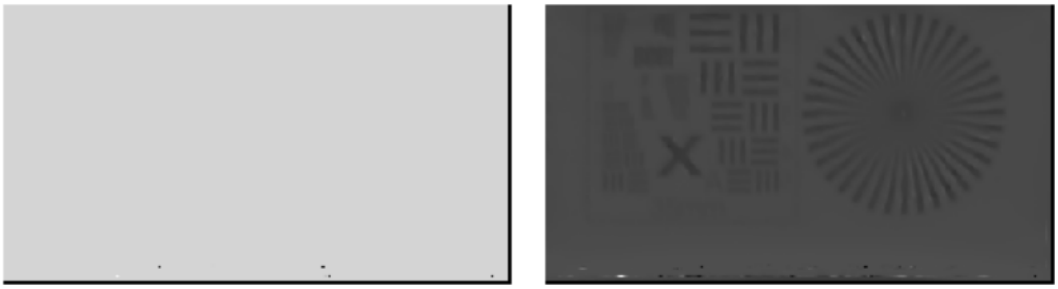
2.4.2 Divergence map (frame to frame, ϕ)

For $k \in \{1, \ldots, K\}$

$$u^{k+1} = (1 - R) * D_k(x)) * u^k(x) + R * D_k(x) * f_k(\phi^{-1}(x)) \tag{2.9}$$

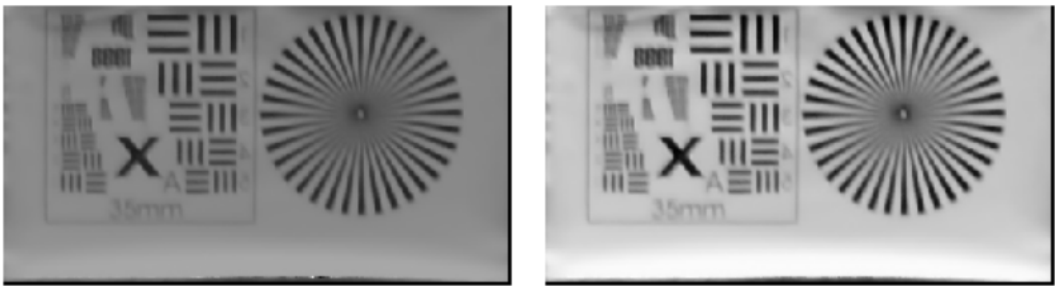
Initialize u^0 with the temporal median of the input sequence. The parameter R controls how much weight is put on the current image $u_k(x)$ versus $f_k(x)$, the current input frame. When R = 1, each new frame completely wipes out the current $u_k(x)$; when R = 0, the input frames have no effect on updating the image, i.e. $u_{k+1}(x) = u_k(x)$. In the gilles1fun function, this was produced using the method 'div', with ref 'NN', indicating it was calculated without respect to a reference image. The method was repeated using images stabilized with the Mao-Gilles algorithm and using optical flows calculated using the MATLAB function imregdemons. The method is demonstrated below using sample images.

Since this method, using the unadjusted divergence, gives results very different from those of the other methods (the output is usually a gray or white box, depending on the value of R), it is noted here that the most likely cause is the outliers in the divergence map. The outliers have large positive or negative values which cause a few pixels to have a very different intensity than the average; when the image is scaled to be displayed on a screen, most of the pixels, which are not outliers and are close to the average, get scaled to a mid-range intensity and most of the image appears as gray.



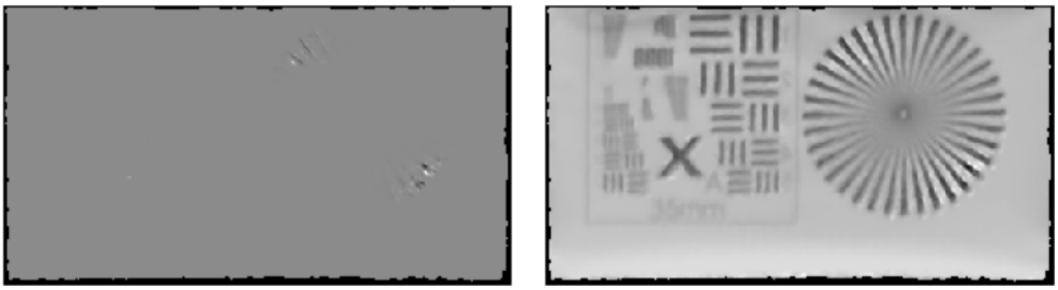
SSIM = 0.1026 SSIM = 0.1489

Figure 2.10: Divergence: R = 0.5 | R = 0.1



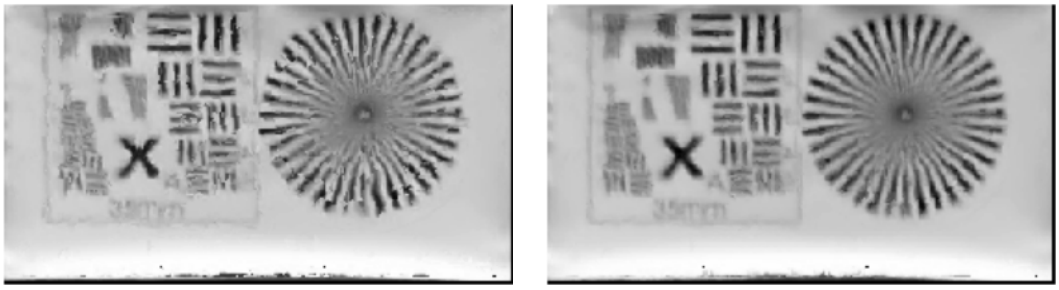
0.1595 0.1595

Figure 2.11: Divergence, stabilized with Mao-Gilles: R = 0.5 | R = 0.1



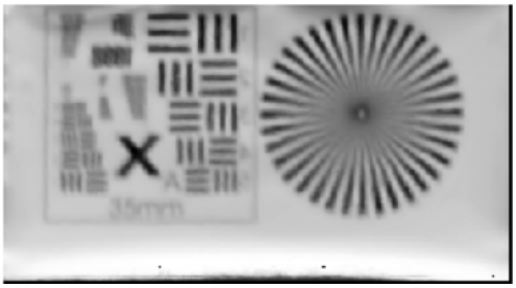
0.0938 0.1491

Figure 2.12: Divergence, ϕ from imregdemons: R = 0.5 | R = 0.1



0.1101 0.1145

Figure 2.13: Binary divergence: R = 1 | R = 0.5



0.1234

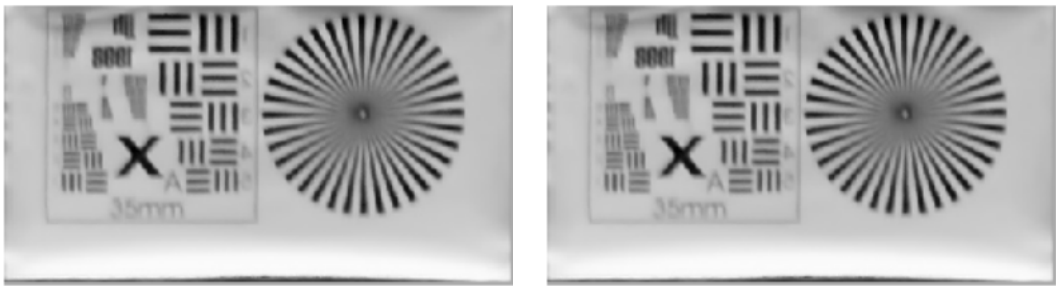
Figure 2.14: Binary divergence: R = 0.1

2.4.3 Binary divergence map (frame to frame, ϕ)

For $k \in \{1, \ldots, K\}$

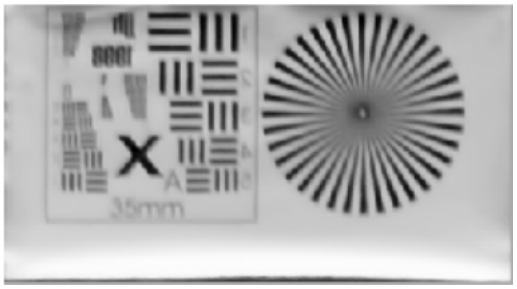
$$u^{k+1} = (1 - R) * B_k(x)) * u^k(x) + R * B_k(x) * f_k(\phi^{-1}(x)) \tag{2.10}$$

Initialize u^0 with the temporal median of the input sequence. The parameter R performs the same function as before. In the gilles1fun function, this was produced using the method 'divBinary', with ref 'NN', indicating it was calculated without respect to a reference image. The method was repeated using images stabilized with the Mao-Gilles algorithm and using optical flows calculated using the MATLAB function imregdemons. The method is demonstrated below using sample images.



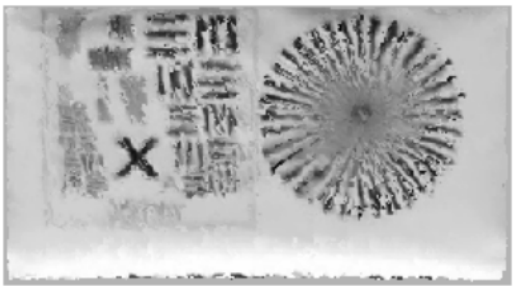
0.1669 0.1676

Figure 2.15: Binary divergence, stabilized with Mao-Gilles: R = 1 | R = 0.5

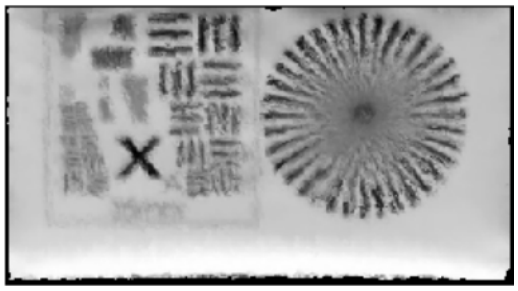


0.1707

Figure 2.16: Binary divergence, stabilized with Mao-Gilles: R = 0.1

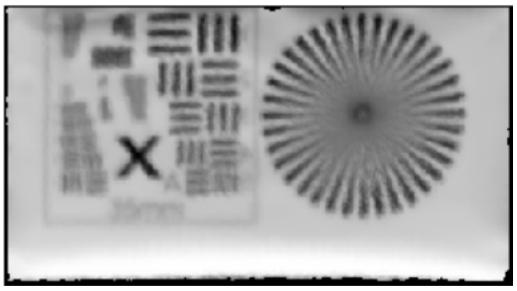


0.1038



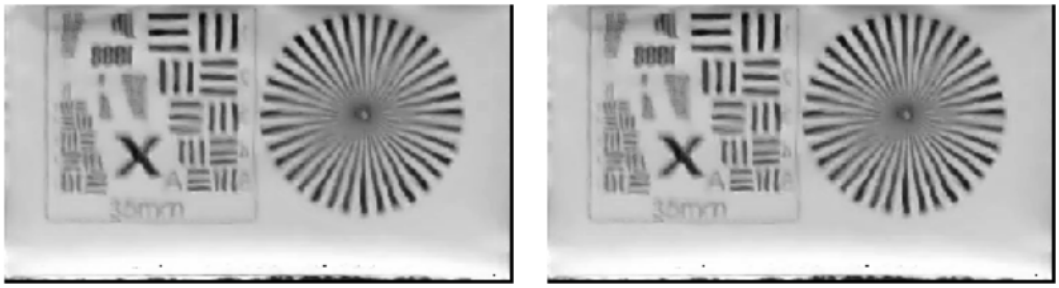
0.1067

Figure 2.17: Binary divergence, ϕ from imregdemons: R = 1 | R = 0.5



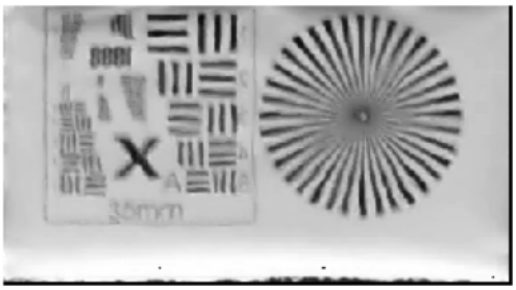
0.1141

Figure 2.18: Binary divergence, ϕ from imregdemons: R = 0.1



0.1440 0.1455

Figure 2.19: Scaled positive divergence: R = 1 | R = 0.5



0.1470

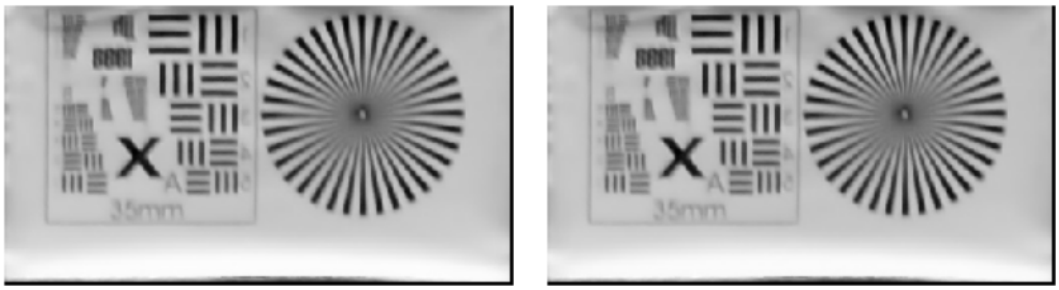
Figure 2.20: Scaled positive divergence: R = 0.1

2.4.4 Scaled positive divergence map (frame to frame, ϕ)

First, scale $\{D_{n,+}\}$ between 0 and 1. Then, for $k \in \{1, \ldots, K\}$

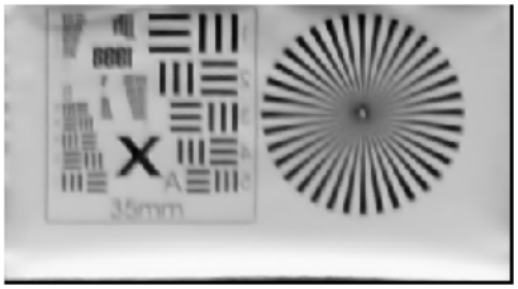
$$u^{k+1} = (1 - R) * D_{k,+}(x)) * u^k(x) + R * D_{k,+}(x) * f_k(\phi^{-1}(x)) \tag{2.11}$$

Initialize u^0 with the temporal median of the input sequence. The parameter R performs the same function as before. In the gilles1fun function, this was produced using the method 'divScaledpos', with ref 'NN', indicating it was calculated without respect to a reference image. The method was repeated using images stabilized with the Mao-Gilles algorithm and using optical flows calculated using the MATLAB function imregdemons. The method is demonstrated below using sample images.



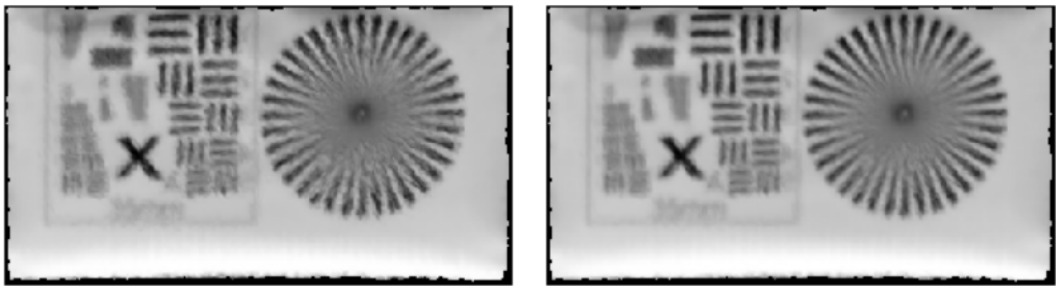
0.1644 0.1646

Figure 2.21: Scaled positive divergence, stabilized with Mao-Gilles: R = 1 | R = 0.5



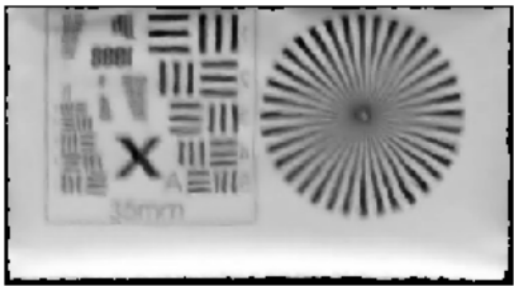
0.1649

Figure 2.22: Scaled positive divergence, stabilized with Mao-Gilles: R = 0.1



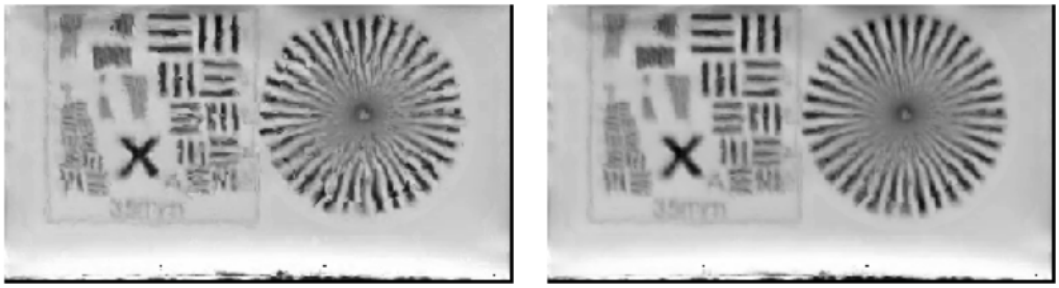
0.0992 0.1066

Figure 2.23: Scaled positive divergence, ϕ from imregdemons: R = 1 | R = 0.5



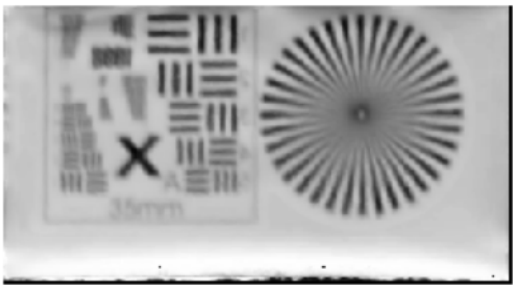
0.1323

Figure 2.24: Scaled positive divergence, ϕ from imregdemons: R = 0.1



0.1118 0.1145

Figure 2.25: Binary divergence, ϕ^{-1} applied to D and f_k: R = 1 | R = 0.5



0.1219

Figure 2.26: Binary divergence, ϕ^{-1} applied to D and f_k: R = 0.1

2.4.5 Binary divergence map, ϕ^{-1} applied to divergence and input frames

This method applies ϕ^{-1} only to the input frames. Then, for $k \in \{1, \ldots, K\}$

$$u^{k+1} = (1 - R * B_k(\phi^{-1}(x))) * u^k(x) + R * B_k(\phi^{-1}(x)) * f_k(\phi^{-1}(x)) \tag{2.12}$$

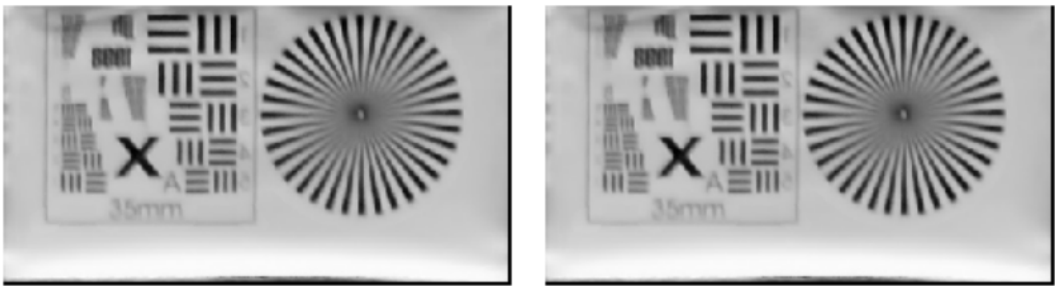
Initialize u^0 with the temporal median of the input sequence. The parameter R performs the same function as before. In the gilles1fun function, this was produced using the method 'binaryBoth', with ref 'NN', indicating it was calculated without respect to a reference image. The method was repeated using images stabilized with the Mao-Gilles algorithm and using optical flows calculated using the MATLAB function imregdemons. The method is demonstrated below using sample images.

2.4.6 Divergence map (w.r.t. reference image), φ

For $k \in \{1, \ldots, K\}$

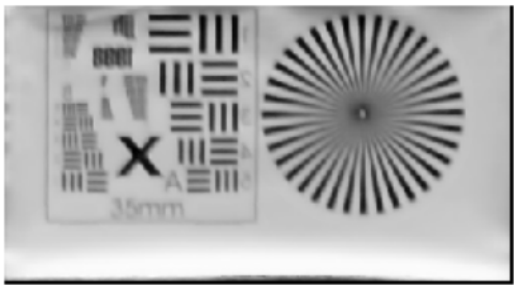
$$u^{k+1} = (1 - R) * D_k(x)) * u^k(x) + R * D_k(x) * f_k(\varphi^{-1}(x)) \tag{2.13}$$

Initialize u^0 with the temporal median of the input sequence. As before, the parameter R controls how much weight is put on the current input frame, f_k.In the gilles1fun function, this was produced using the method 'div', with ref 'YY', indicating it was calculated with respect to a reference image. The method was repeated using images stabilized with the Mao-Gilles algorithm and using optical flows calculated using the MATLAB function imregdemons. The method is demonstrated



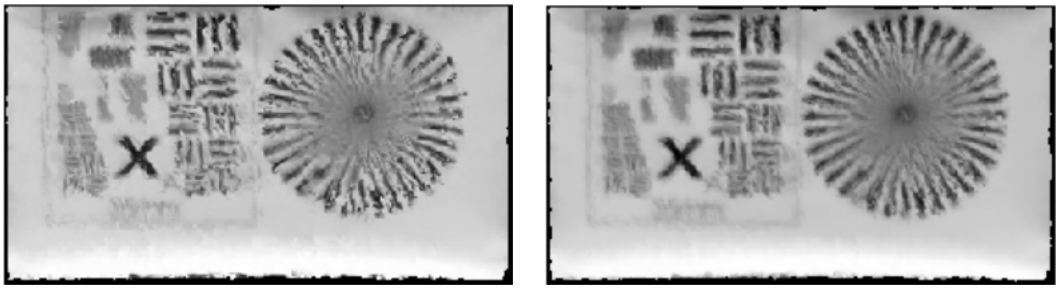
0.1544 0.1548

Figure 2.27: Binary divergence, ϕ^{-1} applied to D and f_k, stabilized with Mao-Gilles: R = 1 | R = 0.5



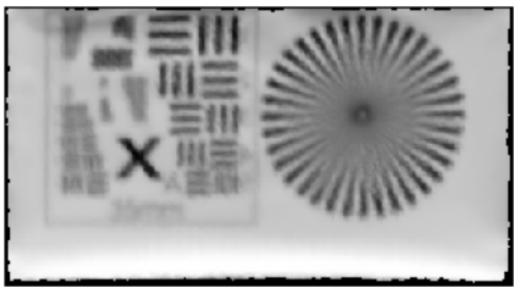
0.1579

Figure 2.28: Binary divergence, ϕ^{-1} applied to D and f_k, stabilized with Mao-Gilles: R = 0.1



0.0942 0.0950

Figure 2.29: Binary divergence, ϕ^{-1} applied to D and f_k, ϕ from imregdemons: R = 1 | R = 0.5

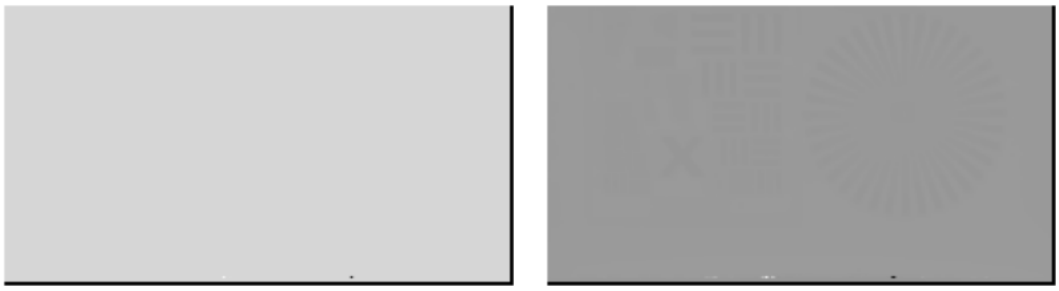


0.1010

Figure 2.30: Binary divergence, ϕ^{-1} applied to D and f_k, ϕ from imregdemons: R = 0.1

0.1143 0.1479

Figure 2.31: Divergence (φ): R = 0.5 | R = 0.1



0.1559 0.1592

Figure 2.32: Divergence (φ), stabilized with Mao-Gilles: R = 0.5 | R = 0.1

below using sample images.

As before, we note here the unusual results the unadjusted divergence produces, due to outliers in the divergence map. For a full explanation, please see section 2.4.2

2.4.7 Binary divergence map (w.r.t. reference image, φ)

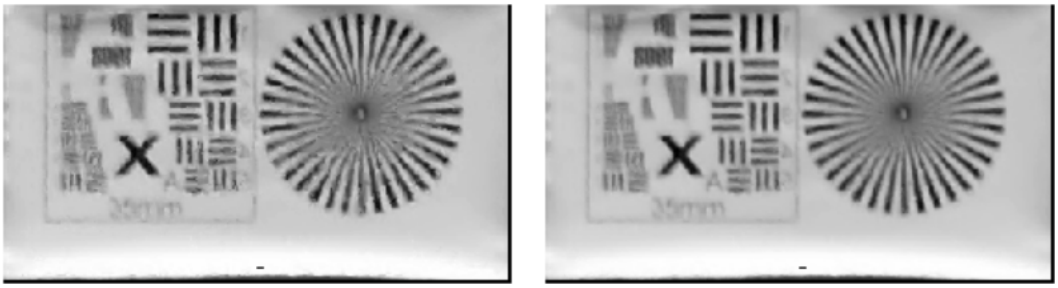
For $k \in \{1, \ldots, K\}$

$$u^{k+1} = (1 - R * B_k(x)) * u^k(x) + R * B_k(x) * f_k(\varphi^{-1}(x)) \tag{2.14}$$

Initialize u^0 with the temporal median of the input sequence. The parameter R performs the same function as before. In the gilles1fun function, this was produced using the method 'divBinary', with ref 'YY', indicating it was calculated with respect to a reference image. The method was repeated using images stabilized with

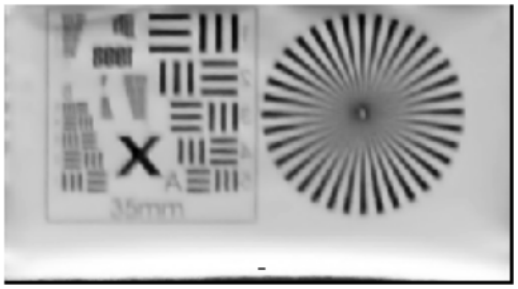
0.0937 0.1491

Figure 2.33: Divergence, φ from imregdemons: R = 0.5 | R = 0.1



0.1402 0.1422

Figure 2.34: Binary divergence (φ): R = 1 | R = 0.5



0.1483

Figure 2.35: Binary divergence (φ): R = 0.1

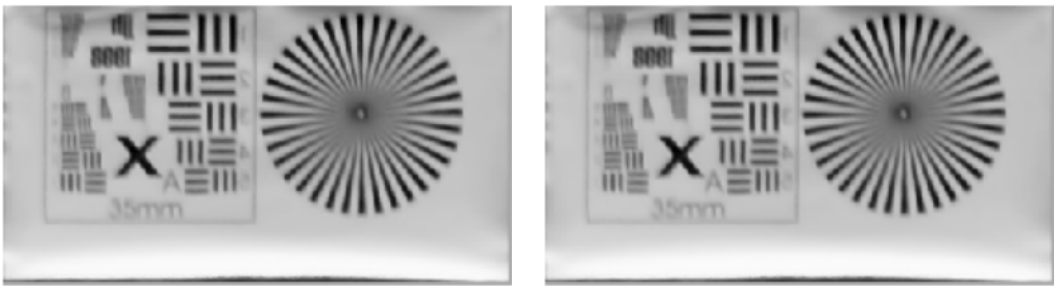
the Mao-Gilles algorithm and using optical flows calculated using the MATLAB function imregdemons. The method is demonstrated below using sample images.

2.4.8 Scaled positive divergence map (w.r.t. reference image, φ)

For $k \in \{1, \ldots, K\}$

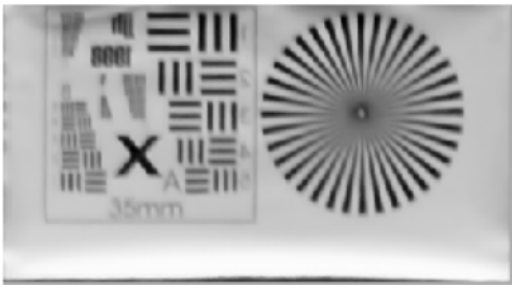
$$u^{k+1} = (1 - R * D_{k,+}(x)) * u^k(x) + R * D_{k,+}(x) * f_k(\varphi^{-1}(x)) \tag{2.15}$$

Initialize u^0 with the temporal median of the input sequence. The parameter R performs the same function as before. In the gilles1fun function, this was produced using the method 'divScaledpos', with ref 'YY', indicating it was calculated with



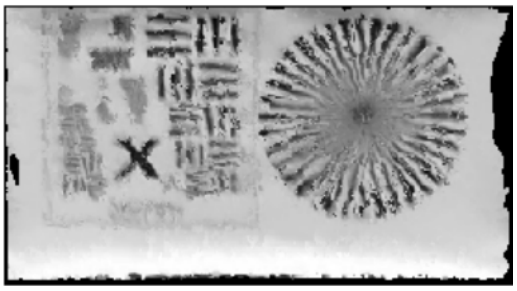
0.1689 0.1690

Figure 2.36: Binary divergence (φ), stabilized with Mao-Gilles: R = 1 | R = 0.5

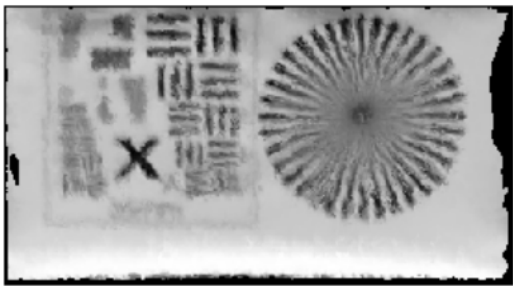


0.1698

Figure 2.37: Binary divergence (φ), stabilized with Mao-Gilles: R = 0.1

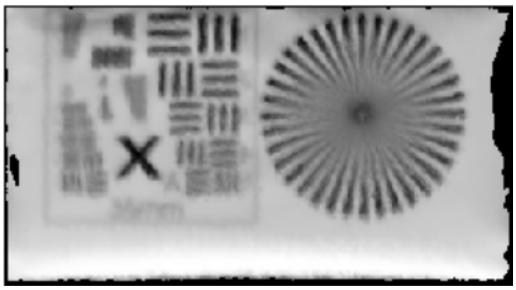


0.1029



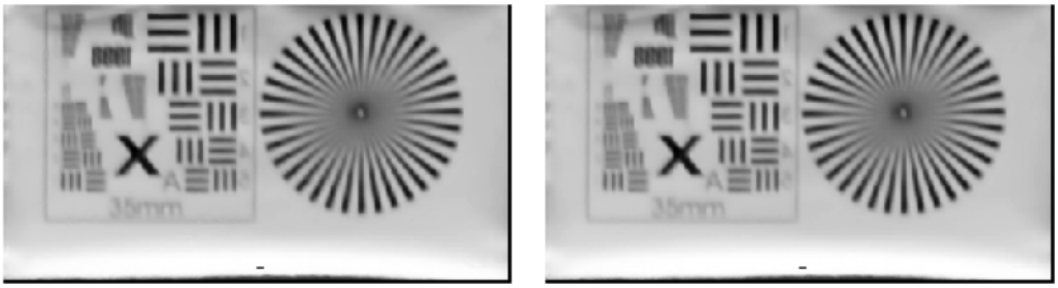
0.1055

Figure 2.38: Binary divergence, φ from imregdemons: R = 1 | R = 0.5



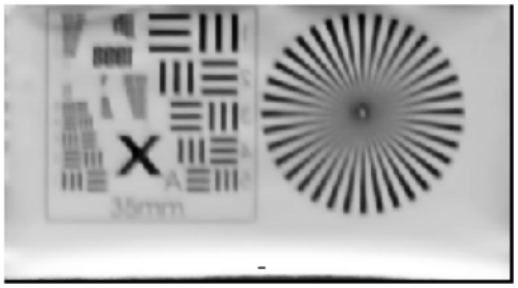
0.1178

Figure 2.39: Binary divergence, φ from imregdemons: R = 0.1



0.1497 0.1509

Figure 2.40: Scaled positive divergence (φ): R = 1 | R = 0.5



0.1524

Figure 2.41: Scaled positive divergence (φ): R = 0.1

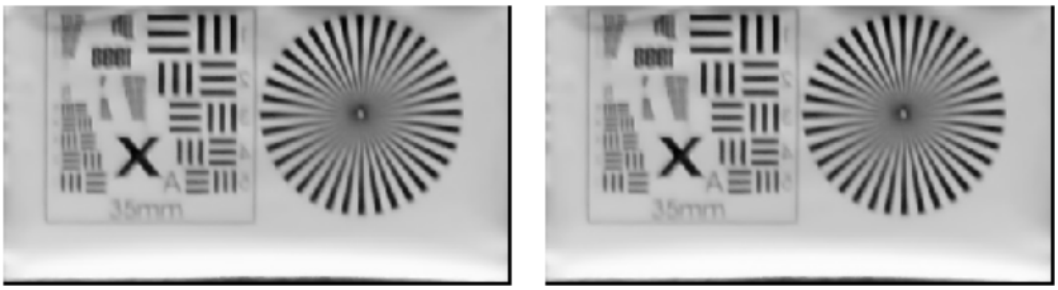
respect to a reference image. The method was repeated using images stabilized with the Mao-Gilles algorithm and using optical flows calculated using the MATLAB function imregdemons. The method is demonstrated below using sample images.

2.4.9 Binary divergence map, φ^{-1} applied to divergence and input frames

This method applies φ^{-1} only to the input frames. Then, for $k \in \{1, \ldots, K\}$

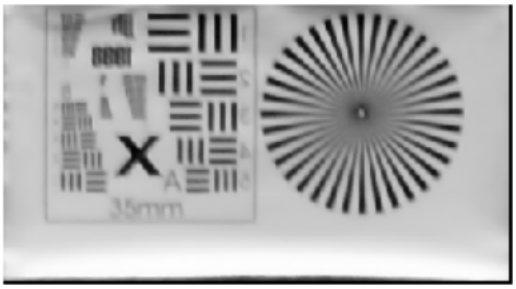
$$u^{k+1} = (1 - R * B_k(\varphi^{-1}(x))) * u^k(x) + R * B_k(\varphi^{-1}(x)) * f_k(\varphi^{-1}(x)) \tag{2.16}$$

Initialize u^0 with the temporal median of the input sequence. The parameter R performs the same function as before. In the gilles1fun function, this was produced



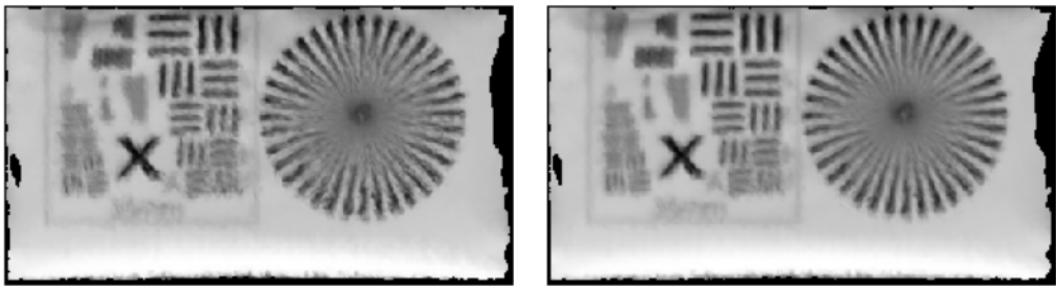
0.1585 0.1589

Figure 2.42: Scaled positive divergence (φ), stabilized with Mao-Gilles: R = 1 | R = 0.5



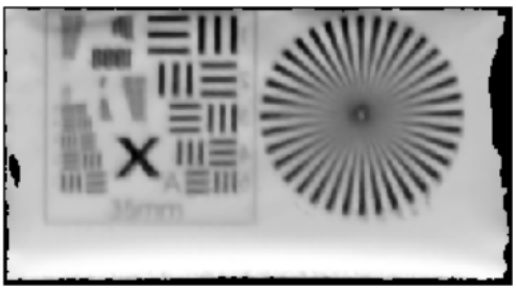
0.1594

Figure 2.43: Scaled positive divergence (φ), stabilized with Mao-Gilles: R = 0.1



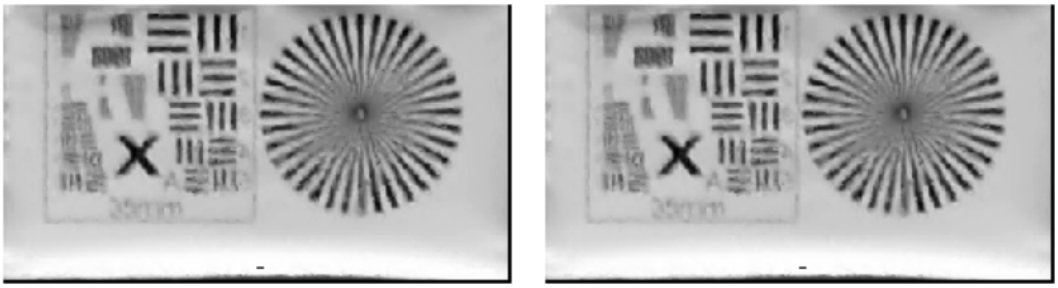
0.1027 0.1107

Figure 2.44: Scaled positive divergence (φ): R = 1 | R = 0.5



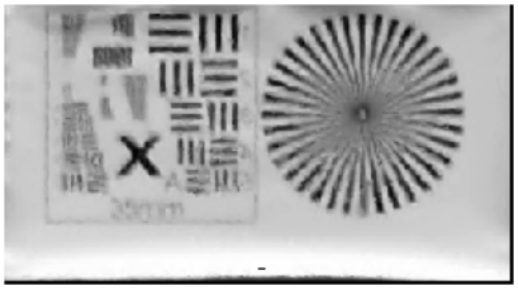
0.1389

Figure 2.45: Scaled positive divergence (φ): R = 0.1



0.1403 0.1421

Figure 2.46: Binary divergence, φ^{-1} applied to D and f_k: R = 1 | R = 0.5



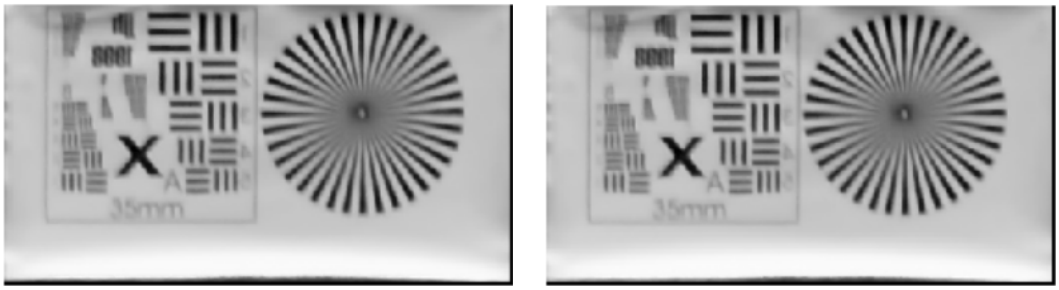
0.1482

Figure 2.47: Binary divergence, φ^{-1} applied to D and f_k: R = 0.1

using the method 'binaryBoth', with ref 'YY', indicating it was calculated with respect to a reference image. The method was repeated using images stabilized with the Mao-Gilles algorithm and using optical flows calculated using the MATLAB function imregdemons. The method is demonstrated below using sample images.

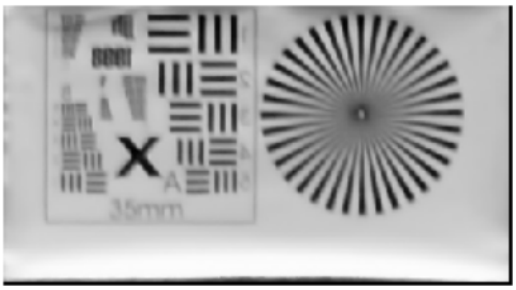
2.4.10 Binary map (w.r.t reference image, φ), iterate and update optical flows

Initialize $u^0(x)$ with the temporal median of the input sequence, and use the binary divergence map to create a fused image according to the following equation (the



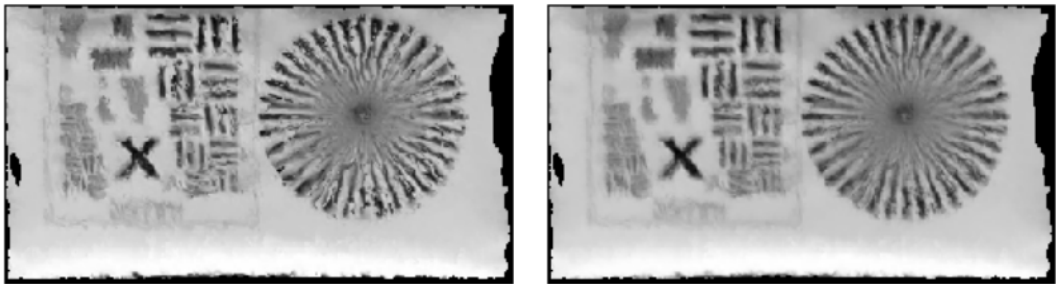
0.1559 0.1561

Figure 2.48: Binary divergence, φ^{-1} applied to D and f_k, stabilized with Mao-Gilles: R = 1 | R = 0.5



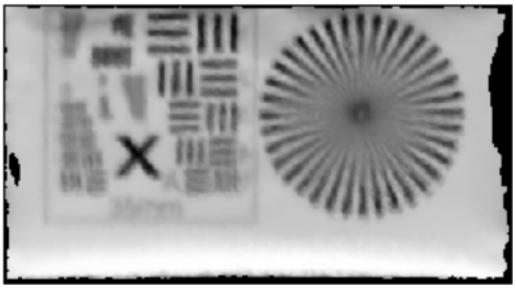
0.1569

Figure 2.49: Binary divergence, φ^{-1} applied to D and f_k, stabilized with Mao-Gilles: R = 0.1



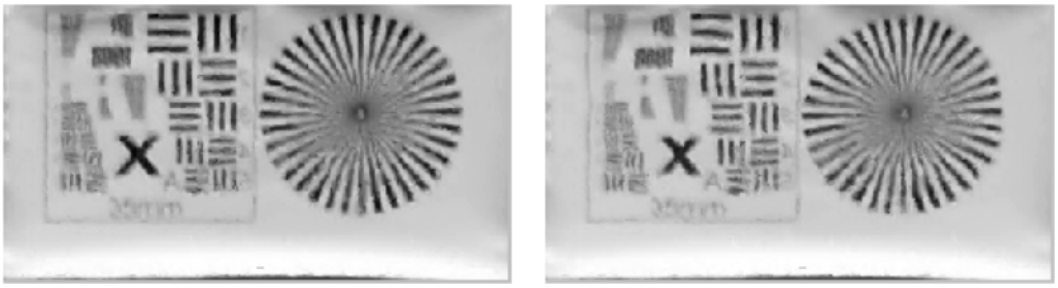
0.0945 0.0948

Figure 2.50: Binary divergence, φ^{-1} applied to D and f_k, φ from imregdemons: R = 1 | R = 0.5



0.1052

Figure 2.51: Binary divergence, φ^{-1} applied to D and f_k, φ from imregdemons: R = 0.1



0.1401 0.1207

Figure 2.52: Eq 2.17 R = 1; 1 iteration | 25 iterations

0.1482 0.1377

Figure 2.53: Eq 2.17 R = 0.1; 1 iteration | 25 iterations

same method as in 2.4.7):

$$u^{k+1} = (1 - R * B_k(x)) * u^k(x) + R * B_k(x) * f_k(\varphi^{-1}(x)) \tag{2.17}$$

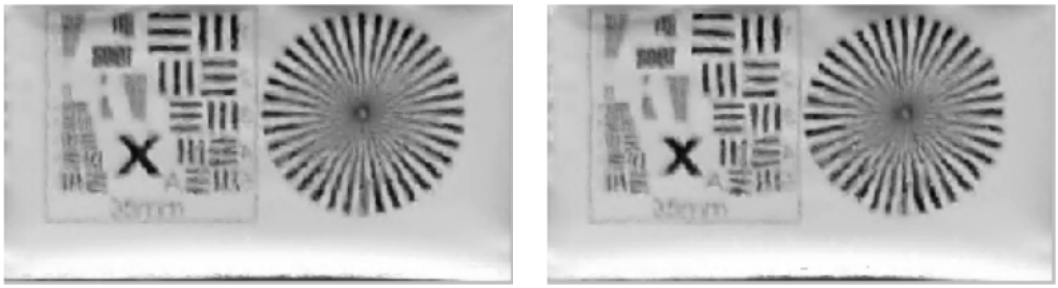
Then calculate the new optical flows, φ, with respect to $u^k(x)$ and repeat the iteration, using the updated optical flows. As before, the parameter R controls how much weight is put on the current input frame, f_k. Surprisingly, more iterations seemed to make the image worse, as seen in the SSIM values in section 2.4.13. This is u7 in the Matlab script gilles1. The method is demonstrated below using sample images.

2.4.11 Binary divergence map (w.r.t reference image, φ), updating optical flows, and applying φ^{-1} to both the binary map and to the input frames

Initialize $u^0(x)$ with the temporal median of the input sequence, and use the binary divergence map, with φ^{-1} applied to both the divergence and the input frames, to create a fused image according to the following equation (the same method as in 2.4.9):

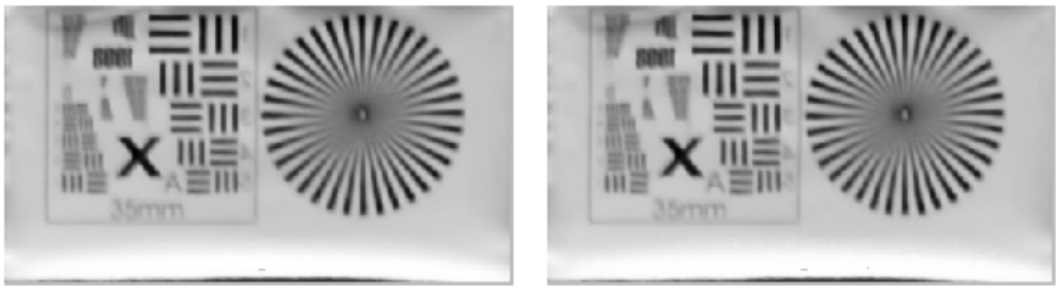
$$u^{k+1} = (1 - R * B_k(\varphi^{-1}(x))) * u^k(x) + R * B_k(\varphi^{-1}(x)) * f_k(\varphi^{-1}(x)) \tag{2.18}$$

Then calculate the new optical flows, φ, with respect to $u^k(x)$ and repeat the iteration, using the updated optical flows. The parameter R performs the same function as before. As before, more iterations seemed to make the image worse. This is u8 in the Matlab script gilles1. The method is demonstrated below using sample images.



0.1402 0.1199

Figure 2.54: Eq 2.18 R = 1; 1 iteration | 25 iterations



0.1481 0.1365

Figure 2.55: Eq 2.18 R = 0.1; 1 iteration | 25 iterations

2.4.12 Flip and scale divergence, and update and iterate flows (w.r.t. reference image, φ)

Create a divergence metric, $\{D_{n,m}\}$ that favors divergence near 0 instead of large divergence. $\{D_{n,m}\}$ is described by the following equation:

$$\{D_{n,m}\} = ||B_n| - 1| \tag{2.19}$$

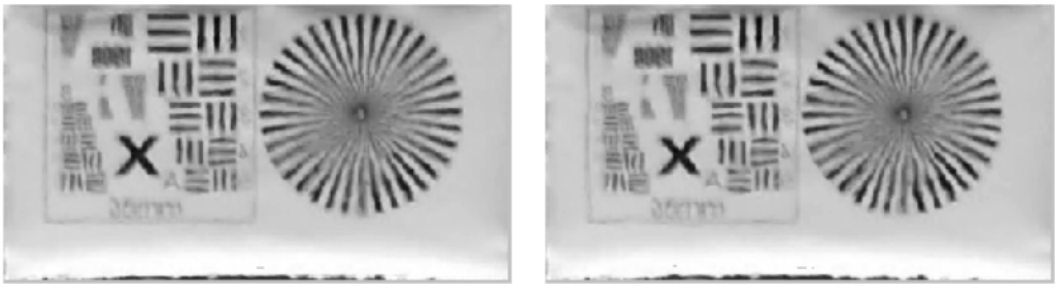
Then use $\{D_{n,m}\}$ instead of the divergence map.
Initialize u^0 with the temporal median of the input sequence.
Then, iterate to create $u^k(x)$, according to the following formula:

$$u^{k+1} = (1 - R * B_k(\varphi^{-1}(x))) * u^k(x) + R * B_k(\varphi^{-1}(x)) * f_k(\varphi^{-1}(x)) \tag{2.20}$$

For each iteration, calculate the optical flows φ with respect to the new reference image, u^k, and recalculate the divergence, $\{D_{n,m}\}$. The parameter R performs the same function as before. This is u9 in the Matlab script in gilles1. The method is demonstrated below using sample images.

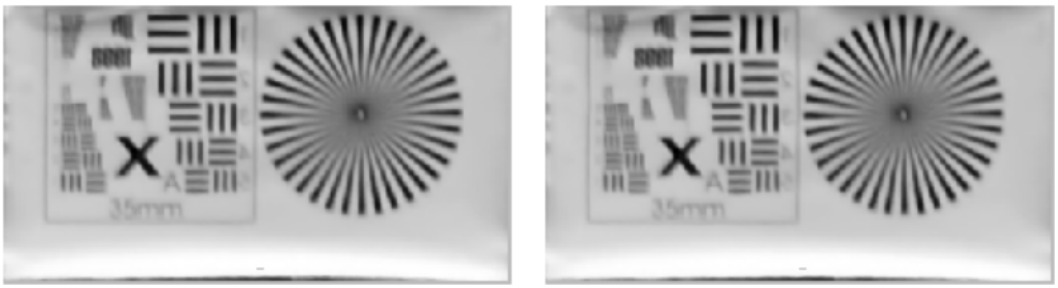
2.4.13 Results and comparisons

The SSIM values allow quantitative comparisons between the images created by the methods tested here. The Mao-Gilles algorithm is used to calculate the optical flows used in creating the divergence maps in all methods except where imregdemons is specified. By comparing the SSIM values, it is clear that the Mao-Gilles optical flow vectors are much more effective than imregdemons in these methods. Stabilizing the



0.1364 0.1241

Figure 2.56: Eq 2.20 R = 1; 1 iteration | 25 iterations



0.1454 0.1378

Figure 2.57: Eq 2.20 R = 0.1; 1 iteration | 25 iterations

sequence before applying the methods also significantly improves the results. Calculating ϕ^{-1} and φ^{-1} with respect to a reference image rather than frame to frame also generally improves the results. Results are also better when a smaller value of R, such as 0.1 rather than 1, is used. Finally, by comparing the plain divergence, binary divergence, scaled positive divergence, and applying ϕ^{-1} or φ^{-1} to the binary divergence map and the input frames, we see that the scaled positive divergence usually gives the best results. We also see that updating the optical flows does not improve results; the SSIM values decrease as the number of iterations increases.

The SSIM values were all obtained by comparing the output of the method described with a ground truth image. Note that the SSIM value of the temporal median of the unstabilized sequence is 0.1532, and the SSIM of the temporal median of the stabilized sequence is 0.1597. These methods only improve on the temporal median when applied to the stabilized sequence.

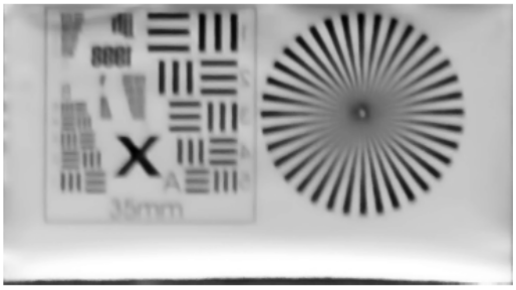
8cm

2.4.14 Adding divergence as weights to lucky-region fusion IMQ

The same lucky-imaging technique presented above is modified here to incorporate divergence. The method is reviewed for clarity. First, the image quality map, $M(x)$, is created:

$$M(r) = \int J(r')G(r - r')dr' \tag{2.21}$$

Ground truth
SSIM = 1



Temporal median (unstabilized)
SSIM = 0.1532

	R = 1	R = 0.5	R = 0.1
Divergence (ϕ)		0.1026	0.1489
Binary Divergence (ϕ)	0.1101	0.1145	0.1234
Scaled Positive Divergence (ϕ)	0.1440	0.1455	0.1470
ϕ^{-1} to D and $f_k(x)$	0.1118	0.1145	0.1219
Divergence (φ)		0.1143	0.1479
Binary Divergence (φ)	0.1402	0.1422	0.1483
Scaled Positive Divergence (φ)	0.1498	0.1509	0.1524
φ^{-1} to D and $f_k(x)$	0.1403	0.1421	0.1482

Table 2.1: SSIM Values for Unstabilized Sequences

	R = 1	R = 0.5	R = 0.1
Divergence (ϕ)		0.1595	0.1595
Binary Divergence (ϕ)	0.1549	0.1554	0.1581
Scaled Positive Divergence (ϕ)	0.1644	0.1646	0.1649
ϕ^{-1} to D and $f_k(x)$	0.1544	0.1548	0.1579
Divergence (φ)		0.1559	0.1592
Binary Divergence (φ)	0.1560	0.1561	0.1569
Scaled Positive Divergence (φ)	0.1585	0.1589	0.1594
φ^{-1} to D and $f_k(x)$	0.1559	0.1561	0.1569

Table 2.2: SSIM Values for Stabilized Sequences

	R = 1	R = 0.5	R = 0.1
Divergence (ϕ)		0.0938	0.1491
Binary Divergence (ϕ)	0.1038	0.1067	0.1141
Scaled Positive Divergence (ϕ)	0.0992	0.1066	0.1323
ϕ^{-1} to D and $f_k(x)$	0.0942	0.0950	0.1010
Divergence (φ)		0.0937	0.1491
Binary Divergence (φ)	0.1029	0.1055	0.1178
Scaled Positive Divergence (φ)	0.1027	0.1107	0.1389
φ^{-1} to D and $f_k(x)$	0.0945	0.0948	0.1052

Table 2.3: SSIM Values for Sequences Using imregdemons

	# of iterations	R = 1	R = 0.1
Binary Divergence (φ)	1	0.1401	0.1482
Binary Divergence (φ)	25	0.1207	0.1377
φ^{-1} to D and $f_k(x)$	1	0.1402	0.1481
φ^{-1} to D and $f_k(x)$	25	0.1199	0.1365
Flipped Divergence (φ)	1	0.1364	0.1454
Flipped Divergence (φ)	25	0.1241	0.1378

Table 2.4: SSIM Values for Sequences, Iterate and Update Flows

$$J(r) = \frac{|\bigtriangledown I(r,t)|}{\int I(r,t)dr} \tag{2.22}$$

$$G(r,a) = exp(-\frac{x^2+y^2}{a^2}) \tag{2.23}$$

Here, $I(r)$ is the image, and $r = (x, y)$ denotes the current pixel. $G(r, a)$ is a Gaussian blur kernel with kernel size a, and $J(r)$ is the image quality metric, which detects sharp edges.

Then use the image quality map, $M(r)$ to create the synthetic image, $I_s(r)$.

$$\frac{\partial I_s(r,t)}{\partial t} = -k\Delta(r,t)[I_s(r,t) - I(r,t_n)] \tag{2.24}$$

$$\Delta(r,t) = \begin{cases} M(r,t_n) - M_s(r,t) & \text{for } M(r,t_n) > M_s(r,t) \\ 0 & \text{otherwise} \end{cases} \tag{2.25}$$

$$I_s^{(n+1)} = [1 - \Delta^n(r)]I_s^n(r) + \Delta^n(r)I^n(r) \tag{2.26}$$

Eq. 2.24 describes the evolution of the image over time. Eq. 2.25 gives the weight for each pixel. Eq. 2.26 describes how the fused image is updated according to the weight on the current image, $\Delta(x, t)$.

Divergence is incorporated by adding an extra weight D to $\Delta(x, t)$; equation 2.26 is updated to be:

$$I_s^{(n+1)} = [1 - D * \Delta^n(x)]I_s^n(x) + D * \Delta^n(x)I^n(x) \tag{2.27}$$

We determine D from the divergence. First, we use just the plain divergence map, where $D = div(\phi)$ or $D = div(\varphi)$. Then we use a binary divergence map, where $D = 1$ for pixels with positive divergence in the current frame, and $D = 0$ for pixels with negative divergence in the current frame. We also use a divergence map with only the positive divergence, with the values scaled between 0 and 1, which determines D at each pixel. We also use a divergence metric that favors divergence near 0 and penalizes divergence far from 0 to determine D, by taking the absolute value of the divergence, scaling it between 0 and 1, subtracting 1, and flipping it by taking the absolute value a second time. These divergence maps were all calculated using ϕ, the optical flow between frames. Finally, we repeat these algorithms, replacing the optical flows used to calculate divergence with φ, the optical flow between each frame and a reference image. We use two kernel sizes, $a = 6$ and $a = 10$. The method was repeated using images from a sequence stabilized with the MATLAB

function imregdemons. We also include SSIM values (comparing each image to the ground truth) for every method. All of this is in the Google Drive in the Matlab script luckyDivergenceFun.m

By visually comparing the images as well as the SSIM values, it can be seen that the different divergence maps do not have a significant impact on the quality of the reconstructed image. The kernel size also has a much smaller impact than in the original lucky-imaging method. On the other hand, stabilizing the images before applying the algorithm seems to improve the quality of the reconstructed image.

2.4.15 Results and comparisons

The SSIM values allow us to quantitatively compare the methods shown above. As in the previous section, the SSIM values were calculated using the ground truth image shown in the ground truth image in section 2.4.13. The SSIM values between the ground truth and the temporal median of the unstabilized sequence and the temporal median of the stabilized sequence are 0.1532 and 0.1597, respectively.

After comparing the SSIM values, it is clear that the images produced by the methods outlined above produce the temporal average. The SSIM values are all the same, with no difference depending on the kernel size, and they match the SSIM values between temporal median and the ground truth. The SSIM values between the images produced here and the temporal median (unstabilized or stabilized, as appropriate), are 1. These methods are not very effective; they take longer and involve a lot of extra calculations to produce the temporal median, which is actually one of the fastest, easiest ways to stabilize an image, and a method we had hoped to improve on.

	a = 6	a = 10
Divergence (ϕ)	0.1532	0.1532
Binary Divergence (ϕ)	0.1532	0.1532
Scaled Positive Divergence (ϕ)	0.1532	0.1532
Flipped Divergence (ϕ)	0.1532	0.1532
	a = 6	a = 10
Divergence (φ)	0.1532	0.1532
Binary Divergence (φ)	0.1532	0.1532
Scaled Positive Divergence (φ)	0.1532	0.1532
Flipped Divergence (φ)	0.1532	0.1532

Table 2.5: Lucky Divergence Applied to Unstabilized Sequence

	a = 6	a = 10
Divergence (ϕ)	0.1597	0.1597
Binary Divergence (ϕ)	0.1597	0.1597
Scaled Positive Divergence (ϕ)	0.1597	0.1597
Flipped Divergence (ϕ)	0.1597	0.1597
	a = 6	a = 10
Divergence (φ)	0.1597	0.1597
Binary Divergence (φ)	0.1597	0.1597
Scaled Positive Divergence (φ)	0.1597	0.1597
Flipped Divergence (φ)	0.1597	0.1597

Table 2.6: Lucky Divergence Applied to Sequence Stabilized with Mao-Gilles Algorithm

	a = 6	a = 10
Divergence (ϕ)	0.1532	0.1532
Binary Divergence (ϕ)	0.1532	0.1532
Scaled Positive Divergence (ϕ)	0.1532	0.1532
Flipped Divergence (ϕ)	0.1532	0.1532
	a = 6	a = 10
Divergence (φ)	0.1532	0.1532
Binary Divergence (φ)	0.1532	0.1532
Scaled Positive Divergence (φ)	0.1532	0.1532
Flipped Divergence (φ)	0.1532	0.1532

Table 2.7: Lucky Divergence Applied to Unstabilized Sequence, with ϕ and φ from imregdemons

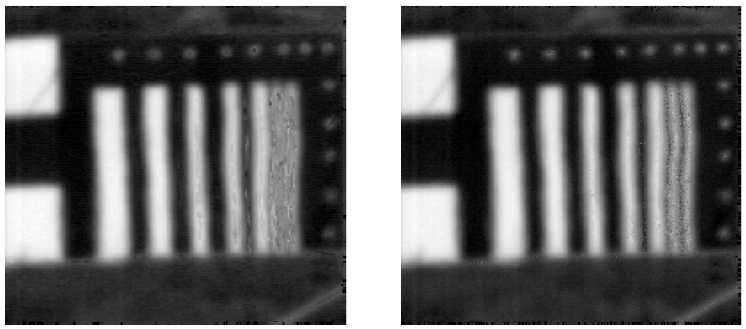
2.5 Method 2

Our second method will stray away from the LRF method and rather focus solely on the deformation flows to obtain a fused synthetic image. We obtain the synthetic image by first stabilizing each frame in our sequence to a reference frame. In this case the reference was the temporal average of the sequence. After each frame has been shifted the maximum temporal divergence is found for each location. The pixel located at $r = (x, y)$, with the maximum temporal divergence is then selected to be the pixel used for the final synthetic image at location $r = (x, y)$. This can be expressed as

$$u(r) = f_{\{\arg\max_n D_n(\varphi_k^{-1}(r))\}}(\varphi_k^{-1}(r))$$

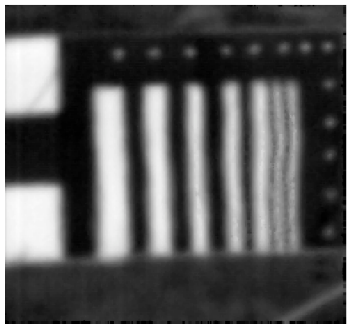
Rather than taking the argmax, the argmedian may also be taken. In both cases only values with positive divergence are considered. The intuition is that those regions with positive divergence will have some scale of zooming in which could translate to higher quality images.

The method was tested on two sequences: one sequence without a ground truth which shows how the method performs visually, another sequence from OTIS dataset with a ground truth to see how it performs under the SSIM.

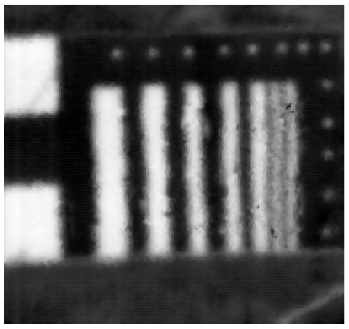


Demons Argmax

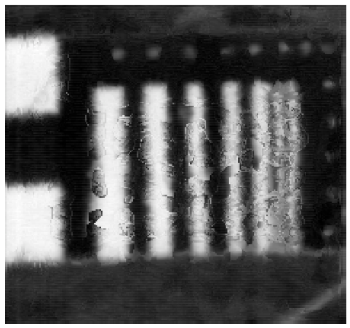
Demons Argmedian



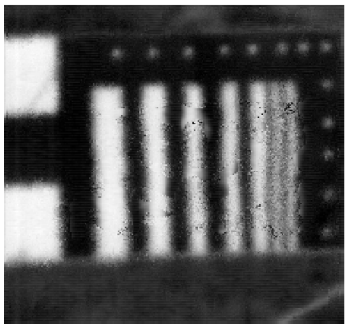
Demons Argmedian Filtered



LK Argmedian Filtered



LK Argmax

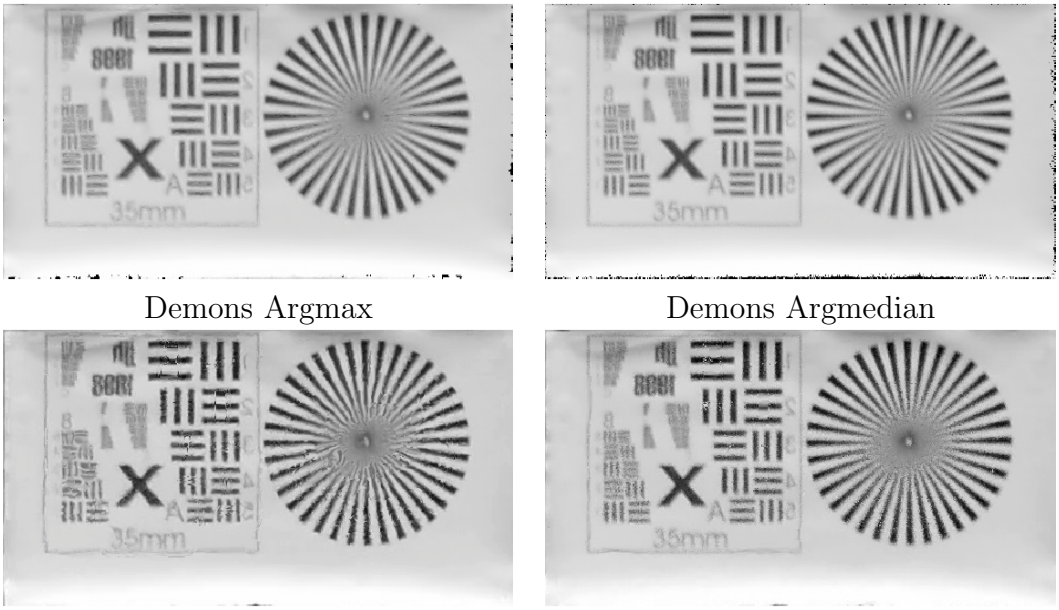


LK Argmedian

Figure 2.58: Results for method 2

From Figures 2.58, 2.59 we can see that both methods yielded the similar results for either deformation method. The argmax method led to random artifacts near edges as well as missing pixels in low texture regions. The barchart sequence has extra elements in between the bars. These extra artifacts are reduced when using the argmedian and the image quality improves greatly. With the argmedian there is still the issue of "grainy" edges. This is apparent on the last two bars from the right.

After examining Table 2.8 above we can see that the Demons method outperformed the LK method. This result was expected after seeing visually how the deformation flows for the LK method stabilized the sequence. In the case of applying the median filter the results were dependent on which method was used. When applying the median filter to the demons algorithm the results had worsened. Although some noise was removed the image was blurred which in turn lowered the



Demons Argmax | Demons Argmedian

LK Argmax | LK Argmedian

Figure 2.59: Results for method 2 using Barchart6

Method	Demons	LK
Average	0.2009	0.2009
Argmax	0.2024	0.1682
Argmax Filtered	0.1995	0.1726
Argmed	0.2085	0.1720
Argmed Filtered	0.2051	0.1783

Table 2.8: SSIM values for method 2

SSIM value. On the other hand, the LK method improved.

2.6 Method 3

2.6.1 Binary map (ϕ)

The third method we tested uses the equation

$$u(x) = \sum_{n=1}^{N} B_n(\phi_n^{-1}(x)) f_n(\phi_n^{-1}(x)). \tag{2.28}$$

This method is performed in the GillesMethod3 script. Results were produced using the original deformed sequence as the input sequence (f_n) and was repeated with the stabilized sequence (using the Mao Gilles Stabilization function and imregdemons function) as the input sequence.

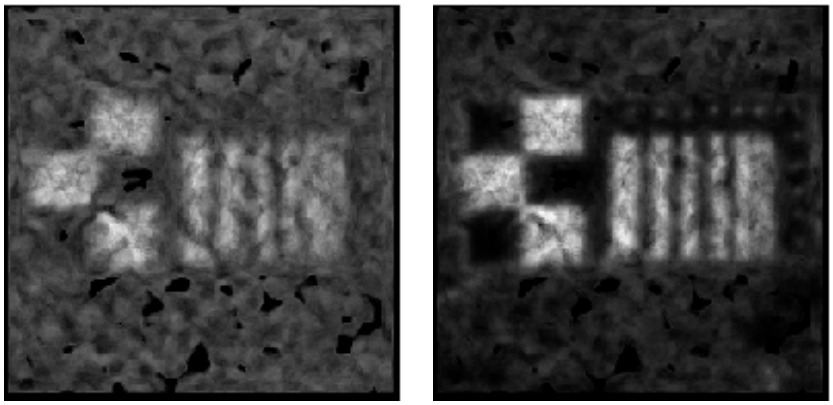


Figure 2.60: Barchart1: Unstabilized (left) and stabilized (right)

2.6.2 Divergence map (ϕ)

Since using the binary divergence map did not give good results, method 3 was repeated using the divergence map instead,

$$u(x) = \sum_{n=1}^{N} D_n(\phi_n^{-1}(x)) f_n(\phi_n^{-1}(x)). \tag{2.29}$$

This method is performed in the GillesMethod3 script. Results were produced using the original deformed sequence as the input sequence (f_n) and was also repeated with the stabilized sequence (using the Mao Gilles Stabilization function and imregdemons function) as the input sequence. The results are very good geometrically.

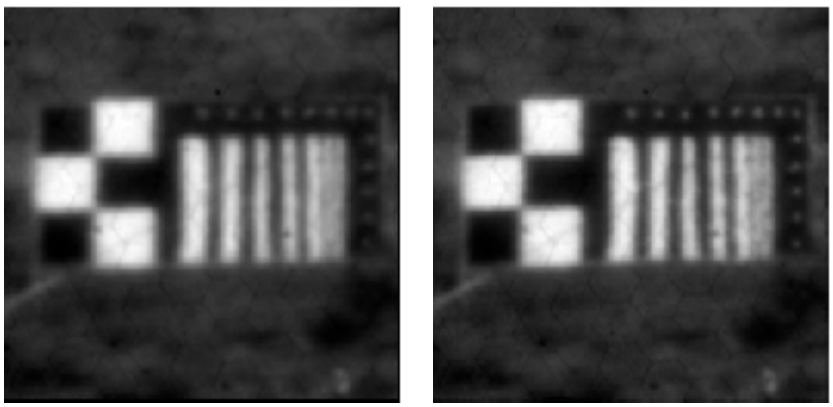


Figure 2.61: Barchart1: Unstabilized (left) and stabilized (right)

Figure 2.62: Barchart29JUN4box: Frame 15 of original sequence (left) and Method 3 stabilized (right)

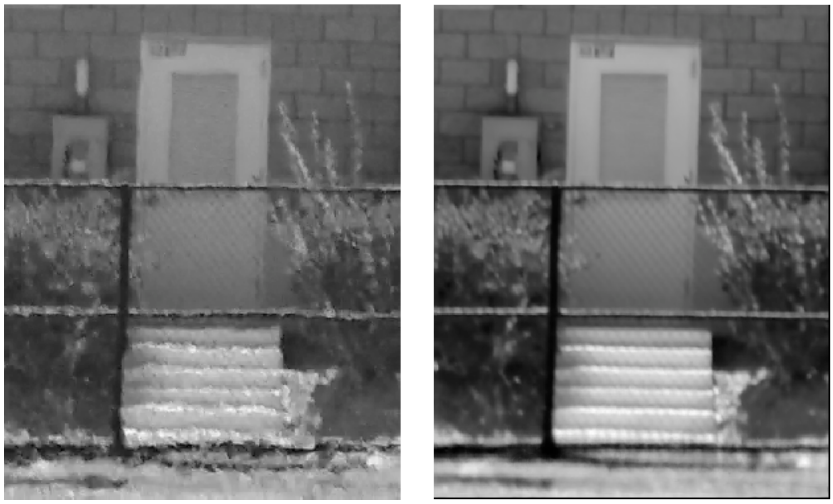


Figure 2.63: Steps100m: Frame 15 of original sequence (left) and Method 3 stabilized (right)

2.6.3 Divergence map (φ)

Lastly, we repeated the equation using the optical flows between each frame and a reference image instead of the optical flows from frame to frame. The equation is as shown:

$$u(x) = \sum_{n=1}^{N} D_n(\varphi_n^{-1}(x)) f_n(\varphi_n^{-1}(x)) \tag{2.30}$$

This method is performed in the GillesMethod3 script. Results were produced using the original deformed sequence as the input sequence (f_n) and was also repeated with the stabilized sequence (using the Mao Gilles Stabilization function and imregdemons function) as the input sequence. The results are almost identical to the previous equation when using optical flows from frame to frame.

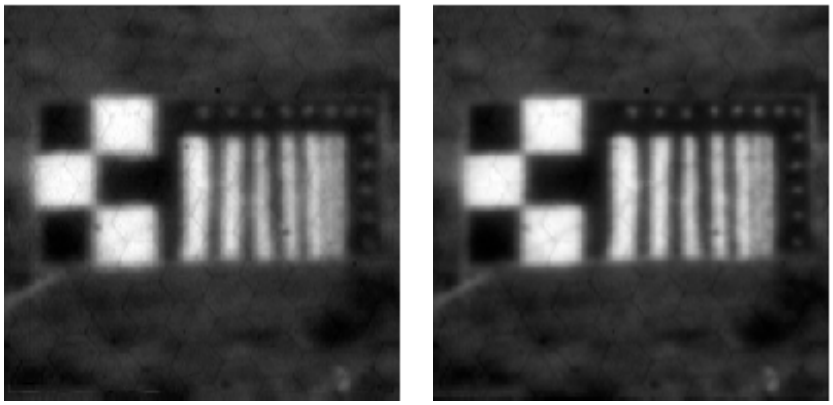


Figure 2.64: Barchart1: Unstabilized (left) and stabilized (right)

2.7 Image fusion future work

One topic to explore in the future is to analyze the varying deformation flows. We used the optical flows from frame to frame (ϕ) and the optical flows from the frames to a reference image (φ), however, other deformation flows, such as B-Spline, Horn-Schunck, etc., should be analyzed to verify that the best one is used. Another topic to pursue is the use of a blur deconvolution method. Our image fusion algorithms improved the geometry of the images, however the resulting images were still affected by blur. Combining a blur deconvolution method with the image fusion methods could significantly improve the quality of the images. Lastly, the image quality metrics used, mean-squared error and structural similarity index, require ground truths of each image as references. It would be helpful to have an image quality metric that did not require having the ground truths, so that the method could be used and evaluated on any image.

2.8 Using Zoom in Super-Resolution

2.8.1 Chaudhuri and Manjunath algorithm

A possible next step in incorporating divergence into image processing is using it as an indicator for the amount that an area has been zoomed. Intuition suggests that areas with a positive divergence may be zoomed in, while areas with negative divergence are zoomed out.

Previous research has investigated the use of zoom as a cue in super-resolution. Most super-resolution techniques depend on sub-pixel shifts between low-resolution frames, so that each frame contains new information. The information from each frame is then combined to form a single high-resolution frame. Research on motion-free super-resolution considers the use of cues other than motion, such as zoom, in super-resolving images.

In [7], the authors present a method for super-resolving images using zoom as a cue. The method combines several images, ranging from Y_1 to Y_K, where Y_1 has the widest field of view but the lowest resolution and Y_K has the smallest field of view

but the greatest resolution. Between each image Y_n and Y_{n+1} there's a zoom factor r_n, and the total zoom factor between Y_1 and Y_K is $r_1 r_2 \dots r_{k-1}$. The goal is to construct a super-resolved image Z with the field of view of Y_1 and the resolution of Y_K.

After converting each image Y_m of size M_1xM_2 to a lexicographically ordered vector of size $M_1 M_2$x1, we can model each image as a noisy, decimated version of the super-resolved image Y_K.

$$y_m = D_m R_m z_{\alpha m} + n_m, \qquad m = 1, \dots, K \tag{2.31}$$

y_m is the observed image and z_{α_m} is the super-resolved image, with a shift α from the optical center given by α_m, where $\alpha_m = (\alpha_{m_x}, \alpha_{m_y})$ and $z_{\alpha_m} = z(x - \alpha_{m_x}, y - \alpha_{m_y})$. R_m is a cropping operator that crops z_{α_m} to the field of view of y_m, and D_m is a decimation operator that reduces the resolution of the cropped z_{α_m} to the resolution of y_m.

Given the set of y_m, $m = 1, \dots, K$ the high-resolution image z can be modeled as a Markov Random Field (MRF). The high-resolution image is estimated using maximum *a posteriori* (MAP) techniques.

$$\hat{z} = \arg\max_z P(z|y_1, y_2, \dots, y_k)$$

It is possible to show that the this can be reformulated as

$$\hat{z} = \arg\max_z \left[\sum_{m=1}^{k} \frac{||y_m - D_m R_m z_{\alpha_m}||^2}{2\sigma^2} + U(z) \right] \tag{2.32}$$

where $U(z)$ is a regularization term. The full details are given in [7].

[7] gives several possible regularization terms. The first, the ℓ^2-norm, tends to over-smooth the image. However, using the ℓ^2-norm makes the minimization convex, allowing the function to be optimized using gradient descent, which allows a very fast optimization in most cases. In order to avoid over-smoothing, the authors suggest an alternative regularization term using line fields. Since this makes the function non-convex, they suggest simulated annealing in order to minimize. However, this method was not implemented here, since it's too slow.

The original method, first presented in [7], requires the zoom factors to be known integers. The authors expand the method to allow for unknown rational zoom in [5]. The zoom factor is estimated basically by guess-and-check. The image is successively resized by a series of possible zoom factors (using the MATLAB function imresize), the shift, α_m, is determined by block-matching, and calculating the error between the zoomed-in observation and the corresponding portion of the resized original image. The factor which produces the lowest error between the two images is taken as the zoom factor.

2.8.2 Implementing Chaudhuri and Manjunath algorithm

Eq. 2.32 is demonstrated below using an image with known integer zoom factors, as presented in [7]. The initial estimate was created by resizing the widest field

of view by 4, the zoom factor between it and the highest resolution image. Then the center was replaced with the appropriately resized images with greater zoom. For the final, center zoomed-in piece, no resizing was needed since it is already at the highest resolution. Note that the center parts of the initial estimate are sharper than the center of Y_1. The gradient descent algorithm was then implemented as in [9].

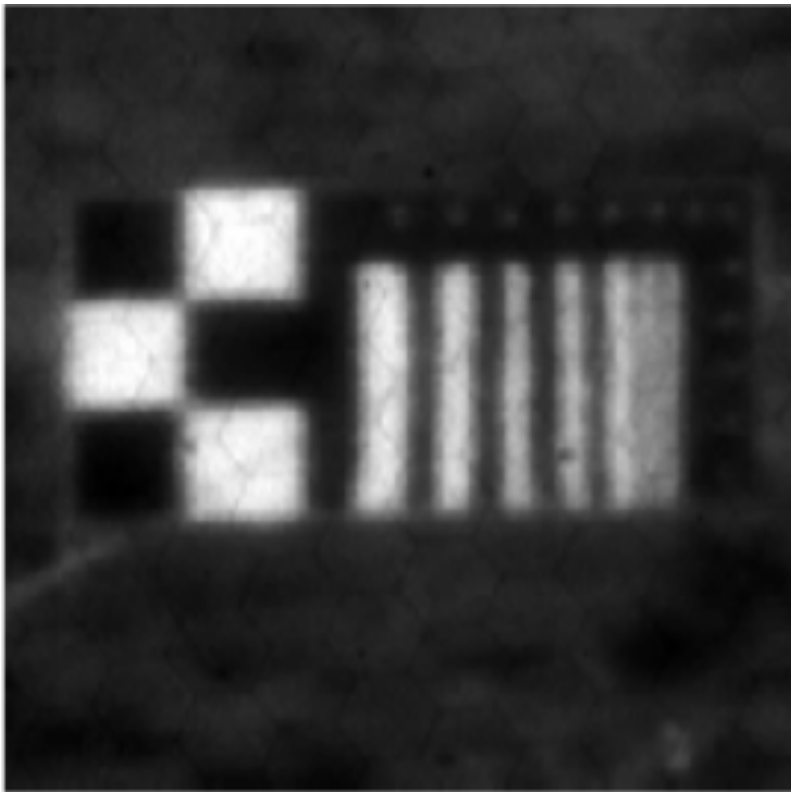


Figure 2.65: Original image (ground truth),
256×256 pixels

The super-resolved image is sharper near the center, where there is more information from the zoomed-in frames. Also note that some of the edges have been over-smoothed, most likely as a result of using the ℓ^2-norm.

2.8.3 Incorporating divergence into Chaudhuri and Manjunath algorithm

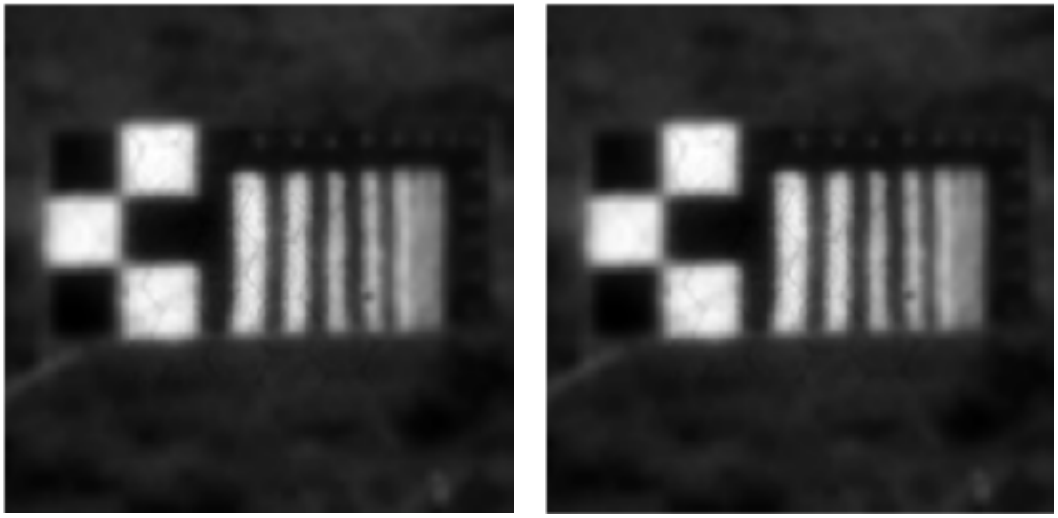
After implementing the algorithm from [5], the algorithm was altered to allow the incorporation of divergence. After calculating the divergence, the sections with divergence above a certain threshold, usually 0.2 for unstabilized images and 0.1 for stabilized images, were smoothed using morphological opening and closing, and labeled as objects.
For each object, the zoom was estimated as in [5]. All of the objects that were zoomed by a given factor were then fused into one frame. For example, there might be a frame zoomed at a factor of 1.2 with 3 sections fused into the resized frame, and another similar frame containing the fused sections with a zoom factor of 2).

Decimated by 4
Zoom factor = 0

Decimated by 2, cropped
Zoom factor = 2

Undecimated, cropped
Zoom factor = 4

Figure 2.66: Input frames for minimization, all 64 × 64 pixels



Initial Estimate

Super-resolved Image

Figure 2.67: 256 × 256 pixels

Those frames are then used as the observations Y_m, $m = 1, \ldots, K$, and the following equation is minimized as before.

$$\hat{z} = \arg\max_z \left[\sum_{m=1}^{K} \frac{||y_m - D_m R_m z_{\alpha_m}||^2}{2\sigma^2} + U(z) \right] \tag{2.33}$$

Again, U(z) is a regularization term. In these examples, the ℓ^2-norm was used for speed of minimization but using another regularization term might better preserve discontinuities. For the following examples, the parameter σ^2, which estimates noise in the image, was set to 0.8, and λ, the parameter controlling the weight on the regularization term, was set to 80.

2.8.4 Unstabilized sequence examples

We compared the super-resolved image with a state-of-the-art super-resolution algorithm by Šroubek from [11]. The algorithm deconvolves and super-resolves the images in one step, so the super-resolved frames shown above were deconvolved

without super-resolution using Šroubek's algorithm in order to compare them with the image produced by deconvolving and super-resolving using Šroubek's method. When using the The results of the algorithm incorporating divergence and zoom compare favorably with the results of the Šroubek algorithm. The output from Šroubek's algorithm shows cross-hatching, especially near the edges between the black and white sections, which is not evident in the output of the algorithm which depends on divergence.

2.8.5 Stabilized sequence examples

As in the previous section, we compare the results of our super-resolution algorithm with Šroubek's state-of-the-art deconvolution and super-resolution algorithm. Note that we were not able to achieve as great a degree of super-resolution using the stabilized sequence (increasing resolution from 256×256 to 384×384 instead of to 2048×2048). When comparing these results, the output of Šroubek's algorithm does not show as significant cross-hatching as in the unstabilized sequence, probably because it was super-resolved at a lower factor. Šroubek's algorithm does have the advantage of being significantly faster than the algorithm incorporating divergence, for both the stabilized and the unstabilized sequence.

Stabilizing the sequence reduced the divergence as well as reducing the amount of zoom, so the maximum zoom in most frames of the sequence was a factor of 1.5 rather than 8 in the stabilized sequence. Since Šroubek's algorithm only accepts integer values for super-resolution factors, the images super-resolved using divergence and zoom were resized to 521×512 pixels (equivalent to a zoom factor of 2 instead of the original zoom factor of 1.5) using MATLAB imresize. Imresize fills in pixels using cubic interpolation.

These experiments demonstrate that it is important to find a balance between stabilizing the sequence enough to reduce geometric distortions while not over-stabilizing the sequence. The unstabilized sequence allowed super-resolution by a factor of 8, which is greater than most applications would require. However, stabilizing the sequence reduces the amount of divergence. If the sequence were extremely stabilized, there would be no divergence and no zoom to use in the super-resolution algorithm.

2.8.6 Future work

Future work could examine the effect of different regularization terms as well as methods of faster and more efficient methods of minimizing the function, especially with non-convex regularization terms. Further work might also be done on the effects of the parameters, σ^2 and λ. Finally, future work on determining better methods to estimate the zoom factor would be interesting and could be very useful.

Chapter 3

Motion Detection Algorithms

3.1 Turbulence-free motion detection

One important element of motion detection is differentiating between an object in motion, and elements of a stable background. This distinction allows for detection of a moving target, or object of interest. In his 2014 paper, Chen proposed an algorithm to distinguish between stationary elements of a frame (background), and objects moving across a frame (foreground). Our first major trial implemented Chen's algorithm on both original and turbulent image sequences.

Chen's algorithm features a method in which a background image is iteratively composed from a selected n number of images. Once a background frame is determined, Chen's algorithm cycles through each image in the original sequence, and selects sections of pixels which do not match the corresponding space in the background image. A pixel is determined as different, depending upon a declared threshold. If the pixel's value differs from the background image's corresponding pixel value by a number greater than the threshold, the pixel is labeled as foreground. These foreground pixels are reassigned a value of 1 (showing as white), while all other pixels are reassigned a value of 0 (showing as black). Through a sequence of images, this can show an object (foreground) moving across the frame (background).
Chen's algorithm calls to first determine a composite background image for the sequence. First, a value α is declared, between 0 and 1. The algorithm cycles through the given sequence, updating the background image with each frame, and using α to determine the weight of each new frame I_k, on the current composite image B_k. The algorithm sets the initial background image to be the mean of the first three frames, then cycles through the sequence, updating this initial frame with each following image. We used an α value of .97, meaning that we gave very little weight to each new frame, updating the background image only slightly.

$$\mathrm{B_k}(\overrightarrow{x}) = \alpha \cdot \mathrm{B_{k-1}}(\overrightarrow{x}) + (1-\alpha) \cdot \mathrm{I_k}(\overrightarrow{x}) \tag{3.1}$$

Once a background sequence is determined, the algorithm calls to cycle through the original sequence, comparing each frame to its corresponding background frame. A difference sequence is constructed for each frame, where each pixel's value is subtracted from the background frame's corresponding pixel.

$$\mathrm{Diff_k}(\overrightarrow{x}) = |\mathrm{I_k}(\overrightarrow{x}) - \mathrm{B_k}(\overrightarrow{x})| \tag{3.2}$$

Next, a threshold value is determined for each frame. This threshold value is given by the median difference in the frame's difference sequence,multiplied by a scalar referred to as gain, and added to an offset value.

$$\mathrm{T}(\overrightarrow{x}) = \mathrm{Gain} \cdot \mathrm{median_k}[\mathrm{Diff_k}(\overrightarrow{x})] + \mathrm{Offset} \tag{3.3}$$

We used the gain value at 3, as done in the Chen paper. The offset that worked best was .05, as also suggested in the Chen paper.
Finally, the difference sequence is compared to the threshold sequence.

$$\mathrm{F_k}(\overrightarrow{x}) = \begin{cases} 1, & \mathrm{Diff_k}(\overrightarrow{x}) > \mathrm{T}(\overrightarrow{x}) \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

If the difference in pixel values is greater than the threshold, this pixel is set to 1, turning the pixel white. In turn, if the difference is less than the threshold, the pixel is set to 0, turning it black. In this way, when the sequence is played, white groups of pixels show object moving across the frame, while the steady background is seen as black.

(a) Frame from the original freeway sequence.

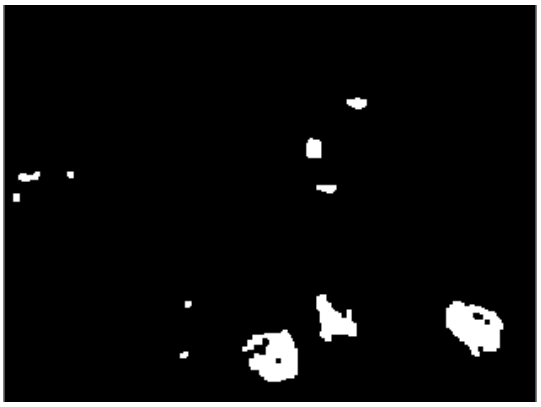
(b) Original frame, with Chen's algorithm applied.

Figure 3.1: Comparison of a turbulence-free frame.

This algorithm worked well to show movement across a frame in a non-distorted sequence, as seen in Figures 3.1a and 3.1b. Once turbulence was simulated, however, the difference between background and foreground was less distinct (see Figures 3.2a and 3.2b). Many false areas of movement were identified, due to the turbulence from frame to frame.

For this reason, our next step was to first control the turbulence in the sequence, then run the motion algorithm on the stabilized sequence. We used Gaussian crops, box crops, and mitigation on the optical flows, in attempts to initially stabilize the

(a) Frame from the simulated turbulent freeway sequence.



(b) Simulated Turbulence frame, with Chen's algorithm applied.

Figure 3.2: Comparison of a simulated turbulence frame.

sequences. While each of these worked reasonably well on a sequence with simulated turbulence, none were powerful enough to significantly stabilize sequences with true turbulence.

3.2 Gaussian filter

Proposed is a method to remove image turbulence from a set of images utilizing properties of the Fourier transform and the Gaussian kernel.

3.2.1 Method

Let I be the 3-D array containing an image sequence. A 1-D Fourier transform is taken for each pixel of the image sequence with respect to time.

$$I(x,y,t) \xRightarrow[Transform]{Fourier} \hat{I}(x,y,\xi)$$

A new matrix G of equal dimension to I is created. Let $h_G(t)$ be the Gaussian kernel in the 1-D case. G is filled such that,

$$\forall x,y \quad G(x,y,t) = h_G(t),$$

then G and $\hat{I}$ are pointwise multiplied to make a new matrix $\hat{I}_G$. It is assumed that the turbulence in the image is of high oscillation with respect to time, while the true image has low oscillation. Therefore, with this pointwise multiplication, the high oscillatory motion should be dampened, while the low oscillatory motion remain intact. The matrix $\hat{I}_G$ is then sent back to the time domain by a 1-D inverse Fourier transform with respect to ξ.

$$\hat{I_G}(x,y,\xi) \xRightarrow[Transform]{Inverse} \bar{I}(x,y,t)$$

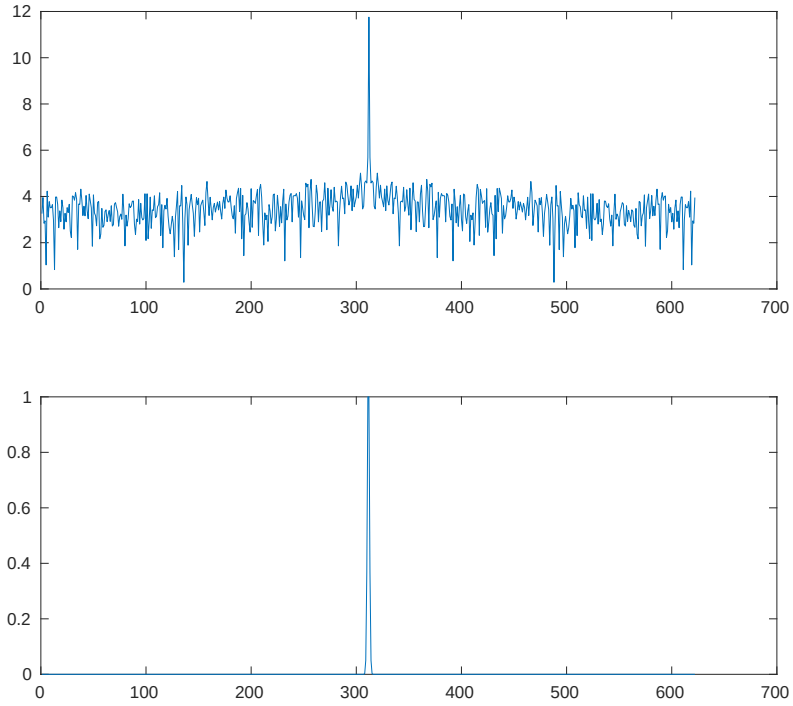


Figure 3.3: Above: A pixel with respect to time from I and a Pixel from G

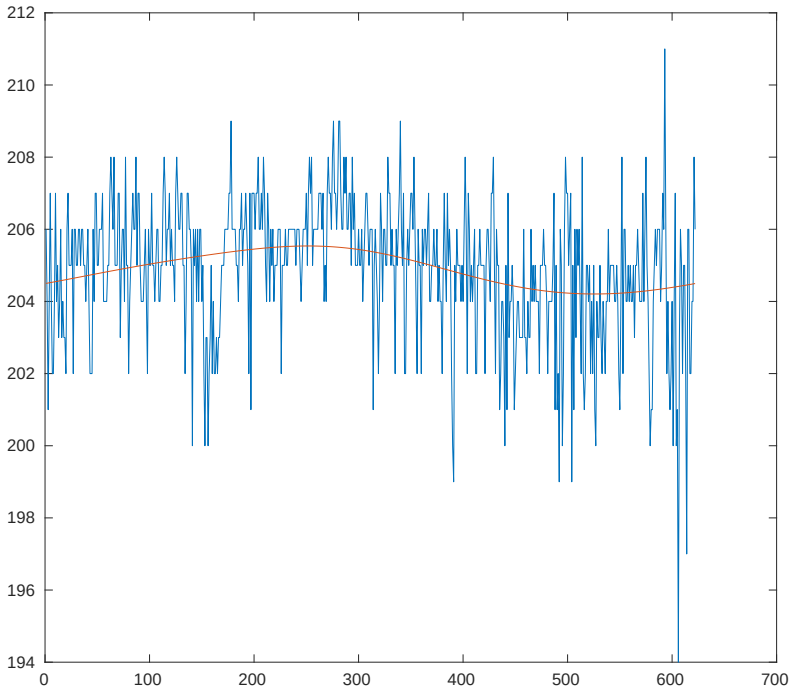


Figure 3.4: Large Signal: Pixel with respect to ξ from $\hat{I}$, Center signal: Pixel from $\bar{I}$

3.2.2 Results

After implementation of the method, it proved useful as a background for motion detection. The method was tested on image sequences containing simulated turbulence and faired well as a background image. When applied to a sequence of images containing real turbulence, the method did not perform as well as a background image.

3.3 Image decomposition

3.3.1 Chambolle projector

The total variation (TV) of a blurred image represents the difference between said image f and the 'true,' unblurred image. One can represent the TV as

$$\mathrm{J}(u) = \sum_{1 \le i,j \le N} |(\nabla u)_{i,j}|. \tag{3.5}$$

In the above equation u represents the structural component of the image in question. In [4] Antonin Chambolle finds a function that minimizes the functional

$$\mathrm{J}(u) + (2\lambda)^{-1}||f-u||^2, \tag{3.6}$$

where f is the original image and λ is a constant. The minimizer of Equation 3.6 above is given by

$$\hat{u} = f - \pi_{\lambda K}(f). \tag{3.7}$$

The author then proposes the following algorithm to calculate the above nonlinear projector $\pi_{\lambda K}$, which by the theorem below converges:

- Begin at p^0=0.
- Iterate the following: $p_{i,j}^{n+1} = \frac{p_{i,j}^n + \tau(\nabla(\mathrm{div}p^n - f/\lambda))_{i,j}}{1+\tau|(\nabla(\mathrm{div}p^n - f/\lambda))_{i,j}|}$

The following theorem, proven in [4], provides the importance of this algorithm.

Theorem 1. *If $\tau \le$ 1/8, then $\lambda \,\mathrm{div}\, p^n \xrightarrow[n\to\infty]{} \pi_{\lambda K}(f)$.*

We now have a method for calculating the projector $\pi_{\lambda K}$ in Equation 3.7, an algorithm that in practice rapidly converges. We will refer to $\pi_{\lambda K}$ as the Chambolle projector.

3.3.2 Two-dimensional Aujol algorithm

One can generally consider a clean image as a linear combination of structures and textures: $f = u + v$. In particular, the structure and texture of an image are oscillatory, with textures modeled as highly-oscillating functions. Many image decomposition algorithms exist that have shown the effectiveness and convergence of structure-texture models in separating these parts of an image (see [2]).

The aforementioned structure-texture algorithms have been used to show that the noise and texture components are similar. Further, in [2] Jean-Francois Aujol et al. use the Chambolle projector to minimize the functional

$$\mathrm{J}(u) + \mathrm{J}^*(v/\mu) + (2\lambda)^{-1}||f - u - v||^2, \tag{3.8}$$

where μ and λ are constants, and $\mathrm{J}^*(v/\mu)$ behaves as the dual of the TV. Based on the convergence shown in [2] the function is first minimized with respect to textures, then with respect to structures. The minimizers of Equation 3.8, through the projector outlined in [4] are given by

$$\hat{u} = f - v - \pi_{\lambda K}(f - v), \qquad \hat{v} = \pi_{\mu K}(f - u)$$

Since these functions do not yield an analytic solution, Aujol et al. propose the following algorithm to calculate the structure and texture components:

1. $u_0 = v_0 = 0$.
2. $v_{n+1} = \pi_{\mu K}(f - u_n)$.
3. $u_{n+1} = f - v_{n+1} - \pi_{\lambda K}(f - v_{n+1})$.
4. Stop when $\max(|u_{n+1} - u_n|,|v_{n+1} - v_n|) \leq \epsilon$.

We will refer to the algorithm above as the Aujol algorithm. Examples of the structure-texture separation are included in Figure 3.5 below, with values $\lambda = 1$ and $\mu = 0.1$.

(a) The original Barbara image.

(b) The Barbara image structures.

(c) The Barbara image textures.

Figure 3.5: The Barbara image decomposition.

3.3.3 Three-dimensional extension of Aujol algorithm

Given the previous work above, we seek to apply the structure-texture decomposition to a sequence of images. In particular our goal is to verify that turbulence mostly separates with the texture component of a sequence of images. Since atmospheric turbulence is highly oscillatory, like the texture component, it ought to separate from the structure and reveal most of the underlying structure of an image sequence when applied. In a sequence of moving objects with distortion due to turbulence,

our separation must take the two types of motion. Although possible to apply the Aujol algorithm to each individual image in the sequence, the distinction between real motion and turbulent motion would be lost in such a process.

The recent work of [8] builds upon the decomposition algorithm of [2]. By augmenting the Chambolle projector to include information on the time evolution of the sequence, effectively turning the 2D projector into a 3D projector (with time scale parameter $1/\beta$), the Aujol algorithm then produces two sequences: the structure, showing real motion in the sequence, and the texture-noise, containing the oscillatory behavior of the atmospheric turbulence.

Figure 3.6: A frame in the toy car sequence.

We have applied the spacetime Chambolle projection algorithm in combination with the Aujol algorithm to a sequence of turbulent images that depict a small toy car moving across a football field at San Diego State University. Figure 3.6 shows a sample frame from this sequence. Due to the heat rising from the field, there is a temperature gradient on the field that creates atmospheric turbulence visible in the original sequence. This turbulence makes the sequence ideal for testing the effectiveness of our algorithm, which we will refer to as the Gilles algorithm.

3.3.4 Decomposition results

To test the algorithm we use the optical flow of the sequence, which detects all motion in the sequence whether due to turbulence or real motion, as our input sequence. Once the algorithm has completed its run, the output structure sequence is viewed in the Slicer application on MATLAB to determine if the real motion of the toy car is visible in the sequence, as shown in Figure 3.7.

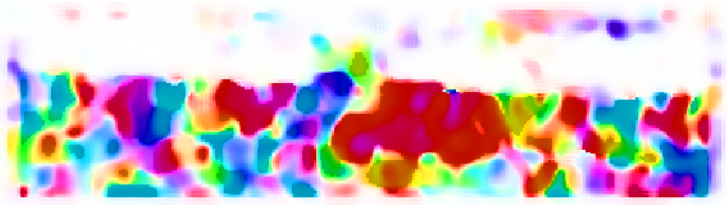


Figure 3.7: The optical flow of the previous frame.

Examples of the Slicer output are shown in Figures 3.8, 3.9 and 3.10. For this particular run of the algorithm, $\lambda = .075$, $\mu = 1.2$, and $\beta = 0.8$. As the frame

number increases along the vertical axis of the Slicer images, the broad red streak across the structure portion of the decomposition represents the straight-line motion of the toy from the right side to the left side of the frame.

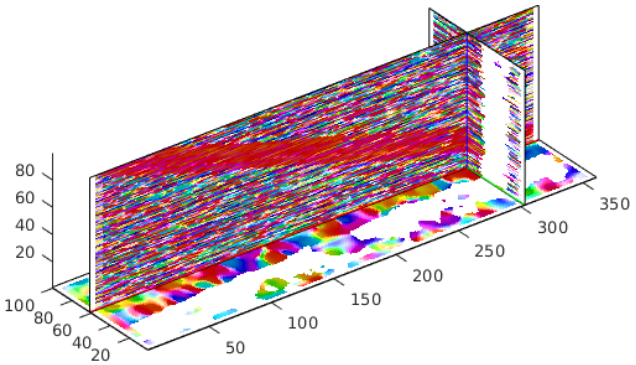


Figure 3.8: The optical flow of the sequence, with car slice at height 60px.

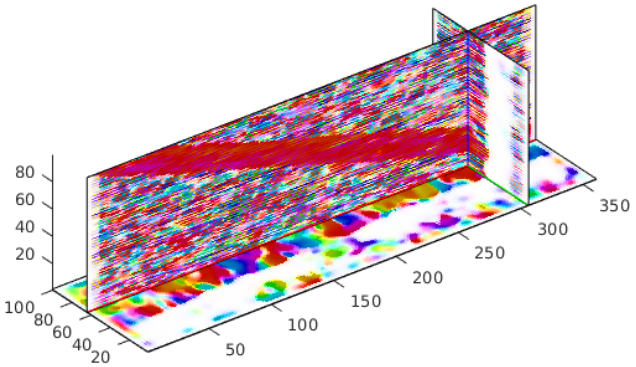


Figure 3.9: Structure portion of the sequence, with car slice at height 60px.

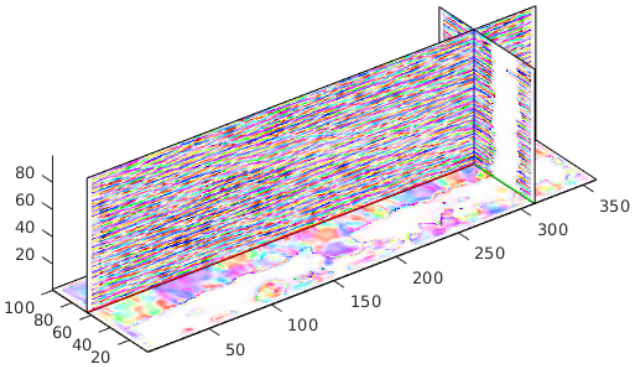


Figure 3.10: Texture portion of the sequence, with car slice at height 60px.

Conclusion

This paper presents the culmination of the San Diego Sate University Research for Undergraduate findings for the Summer of 2016. First, OTIS was introduced, demonstrating the practicality and use of a common source dataset. Second, the paper presented methods for static image restoration. Various methods involving deformation flow and divergence were implemented. Finally, methods of motion detection, such as foreground detection and cartoon-image decomposition were utilized. The results produced from this program will be carried on as a basis for research at San Diego Sate and other educational establishments.

Bibliography

[1] Mathieu Aubailly, Mikhail A. Vorontsov, Gary W. Carhart, and Michael T. Valley. Automated video enhancement from a stream of atmospherically-distorted images: the lucky-region fusion approach. *Proc. SPIE*, 7463:74630C–74630C–10, 2009.

[2] J.F. Aujol. Image decomposition into a bounded variation component and an oscillating component. *Journal of Mathematical Imaging and Vision*, 22 (1): 71–88, 2005.

[3] Alan Bovik, Hamid Sheikh, Eero Simoncelli, and Zhou Wang. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13 (4), 2004.

[4] Antonin Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20 (1): 89–97, 2004.

[5] Subhasis Chaudhuri and Joshi Manjunath. *Motion-Free Super-Resolution*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[6] Eli Chen. Detecting and tracking moving objects in long-distance imaging through turbulent medium. *Optical Society of America*, 53 (6): 1181–1190, 2014.

[7] Manjunath V. Joshi, Subhasis Chaudhuri, and Rajkiran Panuganti. Super-resolution imaging : Use of zoom as a cue. In *in Indian Conference on Computer Vision Graphics and Image Processing, Ahmedabad*, pages 439–444, 2002.

[8] Mathieu Lugiez. Dynamic color texture modeling and color video decomposition using bounded variation and oscillatory functions. *Lecture Notes in Computer Science*, 5099 (1): 29–37, 2008.

[9] Deepu Rajan and Subhasis Chaudhuri. An mrf-based approach to generation of super-resolution images from blurred observations. *J. Math. Imaging Vis.*, 16(1):5–15, jan 2002.

[10] J.P. Thirion. Image matching as a diffusion process: an analogy with maxwell's demons. *Medical Image Analysis*, 2(3):243 – 260, 1998.

[11] Filip Šroubek and Peyman Milanfar. Robust Multichannel Blind Deconvolution via Fast Alternating Minimization. *IEEE Transactions on Image Processing*, 21(4):1687–1700, 2010.

[12] B. Yang, W. Zhang, Y. Xie, and Q. Li. Distorted image restoration via non-rigid registration and lucky-region fusion approach. In *2013 IEEE Third International Conference on Information Science and Technology (ICIST)*, pages 414–418, March 2013.

[13] Yu Mao and Jérôme Gilles. Non rigid geometric distortions correction - Application to Atmospheric Turbulence Stabilization. *Inverse Problems and Imaging Journal*, 6(3):531–546, August 2012.

[14] Jean yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.