5-22-2016

Whiteboard Scanning Using Super-Resolution

Wode Ni
Dickinson College

Whiteboard Scanning using Super-Resolution

By

Wode Ni

Submitted in partial fulfillment of the requirements
for departmental honors in Computer Science
Dickinson College, 2015-2016

John MacCormick, Adviser
Grant Braught, Reader
Timothy Wahls, Reader

April, 25, 2016

The Department of Mathematics and Computer Science at Dickinson College hereby accepts this senior honors thesis by Wode Ni, and awards departmental honors in Computer Science

---

John MacCormick (Advisor) Date

Grant Braught (Reader) Date

Timothy Wahls (Reader) Date

Richard Forrester (Department chairperson) Date

Department of Mathematics and Computer Science
Dickinson College

April, 25, 2016

Abstract

Whiteboard Scanning using Super-Resolution

by

Wode Ni

In this project, we investigated the effectiveness of the Super-Resolution algorithm on a distant whiteboard scenario. Imagine a person taking a video or an image sequence of a whiteboard from a distance. Due to the limitation of camera resolution and the distance, the words in the whiteboard images are sometimes illegible. Though there exist applications to enhance the image quality, the resolution limit caused by the distance is difficult to overcome. Super-resolution, a class of techniques in the field of Computer Vision, enables us to enhance the quality of the images by utilizing the information of multiple low quality images and fuse them to obtain an higher quality image. In this project, we study spatial domain SR algorithms, experiment on an implementation of it in OpenCV, and try to apply it to our distant whiteboard data set. Due to the complexity of the OpenCV implementation, we first perform black-box empirical analysis on the running time of the algorithm. Quantitative analysis on the quality of the resulting images are then conducted to understand the effect of different parameter values for the algorithm. Finally, we attempt to improve the quality of the output by adjusting the algorithm to fit the whiteboard scenario and experimenting on different types of optical flow algorithms.

Acknowledgments

I would like to thank professor John MacCormick for his expertise and support that greatly assisted the research. Also, I am grateful for the comments from professor Timothy Wahls and Grant Braught, which immensely improved the quality of the thesis. Finally, I appreciate my family and dear friends for their support.

Contents

1. Motivation

The following scenario often happens in college classrooms: the professor talks hastily and writes down theorems, formulae, and notes all over the whiteboard. It is usually hard for students to pay attention to the lecture and record all the information simultaneously. Therefore, assuming the professor has provided permission, we take pictures of the whiteboard and try to take notes after class. Due to the limitations of camera limitation, light condition, and the angle from which the photos are taken, the words in the whiteboard image are sometimes illegible. Applications have been developed to rectify and enhance the picture to generate a nice scanning of the whiteboard [1, 2]. Most of them, however, do not work quite well when the picture is taken from a distance. Information is lost because the whiteboard does not take as many pixels in the image as when it is close to the camera. In this year-long research project, we utilize Multi-frame Super-Resolution(SR) algorithms to transcend the resolution limitation, and desirably develop a version of SR algorithm that will work well with data sets of whiteboard images taken from a distance.

2. Background

In this section, we will define the SR problem, simplify it to an energy minimization problem, and present a well-studied solution to this problem. The implementation that we will introduce in the next section bases on this mathematical solution to the SR problem.

2.1. **Super-Resolution and the Imaging Model.**

Super-resolution, a class of techniques in the field of Computer Vision, enables us to enhance the quality of low-resolution (LR) images. Multi-frame SR algorithms take multiple LR frames from a video or photos taken in burst mode, register them to a high-resolution (HR) framework, and attempt to infer the HR scene using the LR observations[3]. As a result, the HR image will have more pixels per unit area and finer quality, therefore enabling us to extract information hidden in the LR observations. SR has already been applied in many real-world situations, such as medical image processing and license plate recognition[4].

Since we are trying to infer the original input by manipulating the end-products of the image formation process, this problem falls into the category of so-called "inverse problems". We do not have enough information about how the images are obtained, but it is possible for us to make assumptions about the process, i.e. an imaging model. Here we will present the one used by the algorithms we studied in this project, which closely resembles the one introduced in [5]. A thorough survey of commonly used imaging models can be found in [3].

Let us denote the original HR scene we are taking a photograph of as I_H. The LR observation I_L of I_H can be obtained by a series of processes that can be represented by

a chain matrix multiplication:

$$I_L = DBWI_H + e \tag{2.1}$$

where D represents the downsampling process of the camera, B blurring matrix, W warping caused by the movement of the camera, and finally e a noise vector which is normally an additive Gaussian noise. The process can be visualized as shown in Figure 2.1.

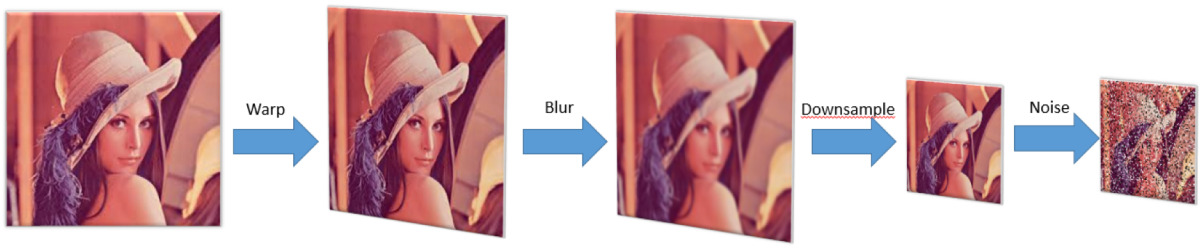


FIGURE 2.1. Image formation process

In the next sections, we will present a mathematical model for the solution of SR problem, which is largely based on [5] and related mathematical techniques from [6].

2.2. **The Energy Function.** Suppose we have a video of N frames, denoted as $I_L^{(k)}, k = 1 \dots N$. Then we can model the SR problem as a process in which we attempt to minimize the energy function:

$$E(I_H) = \sum_{k=1}^{N} ||D_k B_k W_k I_H - I_L^{(k)}|| \tag{2.2}$$

Essentially, we are trying to find an HR estimation derived from a LR sequence that most closely resembles all of the images. Since I_H and I_L are of different sizes, we apply downsampling, blurring, and warping operations to I_H and the resulting matrix

represents the LR estimation of I_H using to our assumed imaging model. If we have a model that corresponds exactly to the imaging process of I_L and an HR result that is identical to the original scene, we should obtain $E(I_H) = 0$. Also note that $||\cdot||$ can be in any norm here and it determines how we compute the difference between I_H and I_L. The use of different norms will affect the quality of the output image. L^2 norm, also known as Euclidean norm, tends to give output images with smoother edges. For example, if we apply L^2 on (2.2), we obtain:

$$E(I_H) = \sum_{k=1}^{N} ||D_k B_k W_k I_H - I_L^{(k)}||_2 = \sum_{k=1}^{N} \sum_{i,j} |(D_k B_k W_k I_H)(i,j) - I_L^{(k)}(i,j)|^2 \tag{2.3}$$

where we compute the summation of squared differences of pixel intensities between the downgraded HR image and the original LR image. On the other hand, L^1 norm, which sums up the differences without squaring them, can preserve the sharpness of the image. Both [5] and [7] claim that the use of L^1 norm will result in better preservation of edges in the output. In this section, derivation of the solution of SR problem will be in L^2 norm for simplicity. L^1 norm's solution can be found in [5] and the corresponding algorithm that computes the solution is similar to L^2 norm's.

Similar to some other inverse problems, in most cases, the SR problem is ill-posed, i.e. there are multiple solutions to the problem or the solution's behavior does not change continuously with the change of the initial conditions. Therefore, a regularization term, or "smoothness term", can be introduced into the energy function in order to prevent the algorithm from producing extreme solutions, and the original function will be called the "data term" in the new equation. For instance, [5] uses Total Variation(TV) regularization for the smoothness term, which is commonly used in denoising algorithms,

penalizing the total gradient of the HR estimation[8]. After adding the TV regularization term, we obtain:

$$E(I_H) = \frac{1}{2}\sum_{k=1}^{N}\sum_{i,j} |(D_k B_k W_k I_H)(i,j) - I_L^{(k)}(i,j)|^2 + \lambda|\nabla I_H|^2 \tag{2.4}$$

where ∇ is the gradient operator, which will compute the sum of difference between the current and neighboring pixels for every pixel. λ is a value that controls the weighting of the smoothness term. In other words, this regularization term penalizes large local variations.

2.3. **Minimizing The Energy Function.** Once the energy function defined, we can simplify the Super-Resolution problem to the following: given N LR images, find I_H that will minimize $E(I_H)$. To find the minimum, techniques in the field of Calculus of Variation must be used. Here we rewrite (2.4) by replacing the summation sign by an integral sign and using a 2-vector $\mathbf{x} = (i, j)$ to represent coordinates of pixels:

$$E(I_H) = \frac{1}{2}\sum_{k=1}^{N}\int_{\Omega} |D_k B_k W_k I_H - I_L^{(k)}|^2 + \lambda|\nabla I_H|^2 d\mathbf{x} \tag{2.5}$$

Note that the integration is simply an abstraction of the summation over all pixels. (2.5) is a functional of the following form:

$$I = \int_b^a F(y, y', x)dx \tag{2.6}$$

and the function y, which will give the extrema of (2.6), can be found using the Euler-Lagrange equation:

$$\frac{dI}{dy} = \frac{\partial F}{\partial y} - \frac{d}{dx}(\frac{\partial F}{\partial y'}) = 0 \tag{2.7}$$

We apply the Euler-Lagrange equation to (2.5) and find the derivative of $E(I_H)$ with respect to $\mathbf{x}$. Replace I, y, y', and x by E, I_H, ∇I_H, and $\mathbf{x}$, respectively. Also denote $\frac{1}{2}|D_k B_k W_k I_H - I_L^{(k)}|^2 + \lambda|\nabla I_H|^2$ as F to simplify the notation:

$$\frac{dE}{dI_H} = \frac{\partial F}{\partial I_H} - \frac{d}{d\mathbf{x}}(\frac{\partial F}{\partial \nabla I_H}) = 0 \tag{2.8}$$

Note that here D, B, W, I_H, and I_L are all matrices and we are taking partial derivatives of scalars with respect to matrices. When performing the differentiation, we vectorize the matrices by applying the *vec* operator, which will stack the columns of the matrix and generate a big vector. We then differentiate according to rules in vector calculus and the result in matrix form can be reconstructed by undoing the stacking process.

We compute the partial of the first term by applying differentiation rules in vector calculus. For the second term, the gradient sign turns into the negative of the Laplacian Operator by Green's First Identity. After finding the partials, we obtain the Euler-Langrange equation:

$$\frac{dE}{dI_H} = \sum_{k=1}^{N} W_k^T B_k^T D_k^T (D_k B_k W_k I_H - I_L^{(k)}) - \lambda \Delta I_H = 0 \tag{2.9}$$

Note that here D^T, B^T, and W^T, due to their mathemetical natures. correspond to upsampling, blurring, and backward warping (reversing the geometric displacement from

a frame to another)[5, 7, 9]. Due to time constraint, we only attempted to proof a special case of W^T and tried to show that it indeed reversed a certain type of geometric transformation done on an image.

(2.9) can be solved numerically using the steepest descent method, which computes a local minimum of a function by taking small steps proportionate to the negative of the gradient and approach the local minimum gradually. In our case, we start with an estimation of I_H, and compute I_H by iteratively stepping towards to the solution. Formally, in an iteration, we compute the next estimation of a local minimum, denoted by I_H^{new}, based on the last estimation I_H^{old}:

$$\begin{aligned} I_H^{new} &= I_H^{old} + \tau(\sum_{k=1}^{N} W_k^T B_k^T D_k^T (D_k B_k W_k I_H^{old} - I_L^{(k)}) - \lambda \Delta I_H^{old}) \\ &= I_H^{old} - \tau(\sum_{k=1}^{N} W_k^T B_k^T D_k^T (I_L^{(k)} - D_k B_k W_k I_H^{old}) + \lambda \Delta I_H^{old}) \end{aligned} \tag{2.10}$$

where τ is a small number controlling the size of each step. In an ideal case, the estimation of I_H will eventually converge after a certain number of iterations, reaching a local minimum.

This process can be then translated to an algorithm. Code 1 illustrates the SR algorithm originally presented in [5]. Some of the notations are modified for clarity. Note that, in the actual implementation, the motion fields used by W are pre-computed by a dense optical flow algorithm, which takes a series of images as input and output motion fields between consecutive images.

Therefore, in this section, we defined the criteria for the solution of SR problem and presented a method of finding the solution based on our imaging model. We defined an energy function (Equation 2.8) with (1) a data term that favors SR results, after

downsampling, blurring, and warping, resemble the input data the most, and (2) a regularization term that penalize the gradient of the result, forcing the output image to be smoother. We then showed how to approach the optimal solution using gradient descent. Lastly, an SR algorithm was presented to illustrate an algorithmic version of the energy minimization.

Code 1. the SR algorithm

```
// INPUT: a set of N low resolution images - lowResImages
// PARAMETER: T - the number of iterations; tau - weight for steepest
    descent
// OUTPUT: a high resolution image - highres
Choose an image lowResImages(r) from the input images as the reference
    image, where r is the index of the reference image
Estimate motion fields between consecutive frames
Using the motion fields, produce relative motion maps with respect to the
    lowResImages(r)
Initialize highres by upsampling lowResImages(r) using some form of
    interpolation
Upsample all motion fields and relative motion maps to the size of highres
for t = 1 to T:
    sum = 0
    for k = 1 to N:
        // W, B, D are warping, blurring, and downsampling operations,
            respectively.
        a = W * highres
        b = B * a
        c = D * b
        c = lowResImages(k) - c
        a = transpose(D) * c // transpose() takes the transpose of a matrix
        b = transpose(B) * a
        c = transpose(W) * b
        sum = sum + c
    highres = highres + tau * (sum - lambda * (Laplacian(highres)))
return highres
```

3. Implementation and Experiment

A fully functional implementation of an SR algorithm using [7] and [5] can be found in Open Source Computer Vision (OpenCV) library and it is included in the `superres` module. OpenCV contains a large number of image processing and computer vision algorithms and a set of built-in data types. The implementation of SR is built on top of this library and contains both CPU and GPU versions, where the GPU version takes advantage of CUDA programming to enhance performance. It uses dense optical flow algorithms in OpenCV to estimate motions and compute the SR image using steepest

descent method described in the Background section. The program produces a super-resolved video sequence by executing the SR algorithm multiple times with respect to each frame of the input file. A sample of the usage of OpenCV implementation is shown in Code 2 and result of this code in Figure 3.2. This implementation is a hybrid of methods from [7] and [5]: it has the general structure of the algorithm proposed in [5] and uses optical flow to estimate motion fields; it also applies the robustness technique presented in [7]. Code 3 shows a part of the source code that mimics the algorithm shown in Code 1, which is equivalent to the algorithm presented in [5]. Line 11 utilizes robustness since `diffSign()`, which is meant to compute the difference between matrix `src(k)` and `c_`, will only output -1, 0, or 1 depending on the sign of the result of subtraction rather than returning the actually result. Consequently, we obtain a matrix of integers in the range $[-1, 1]$ and according to [7], this technique will speed up the algorithm while retaining the quality of the result.

The majority of the coding effort in the project was done studying C++, OpenCV syntax, reading OpenCV source code, and writing codes for experiments. OpenCV is a vast library with enormous distinct features such as (1) built-in data types for images, points, vectors and etc., (2) smart pointers that will automatically get deleted after used, and (3) a series of classes and functions that can generate GUI components. Additionally, the `superres` module, in particular, was not sufficiently documented. As a result, we experienced a big learning overhead when dealing with this new library. Also, we have written some basic image/video processing functions to generate our datasets and obtain certain information about videos. We also tried to make the experiment codes as

automated as possible, which include a logging feature and encapsulation of functions in `superres`.

CODE 2. Sample Code Using superres

```
Ptr<FrameSource> frames = createFrameSource_Video( "video.mp4" );
Ptr<SuperResolution> super_res = createSuperResolution_BTVL1();
super_res->setInput( frames );
Mat result;
super_res->nextFrame( result );
imshow( "Super Resolved Output", result );
waitKey( 0 );
```

CODE 3. Source code of OpenCV implementation

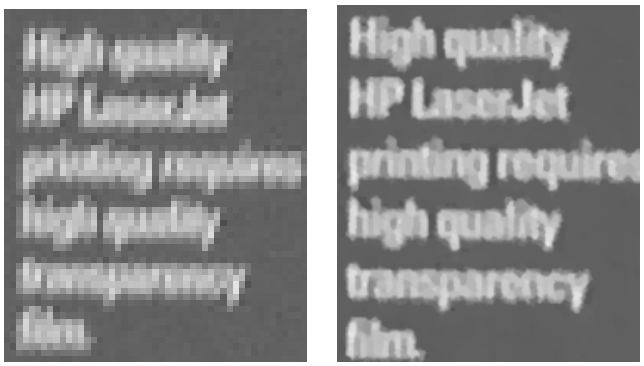
```
for (int i = 0; i < iterations_; ++i)
{
    diffTerm_.setTo(Scalar::all(0));

    for (size_t k = 0; k < src.size(); ++k)
    {
        remap(highRes_, a_, backwardMaps_[k], noArray(), INTER_NEAREST);
        GaussianBlur(a_, b_, Size(blurKernelSize_, blurKernelSize_),
            blurSigma_);
        resize(b_, c_, lowResSize, 0, 0, INTER_NEAREST);

        diffSign(src[k], c_, c_);

        upscale(c_, a_, scale_);
        GaussianBlur(a_, b_, Size(blurKernelSize_, blurKernelSize_),
            blurSigma_);
        remap(b_, a_, forwardMaps_[k], noArray(), INTER_NEAREST);

        add(diffTerm_, a_, diffTerm_);
    }
    if (lambda_ > 0)
    {
        calcBtvRegularization(highRes_, regTerm_, btvKernelSize_,
            btvWeights_, ubtvWeights_);
        addWeighted(diffTerm_, 1.0, regTerm_, -lambda_, 0.0, diffTerm_);
    }
    addWeighted(highRes_, 1.0, diffTerm_, tau_, 0.0, highRes_);
}
```



(a) A frame from input video (b) SR output

FIGURE 3.2. Result Using the Sample Code

3.1. **Black-box Analysis.** To fully understand efficiency and behavior of the OpenCV implementation, we decided to run the code while changing one parameter at a time while making the rest default. There are 10 parameters defined in the OpenCV official documentation on Super Resolution module[1], so shown in Table 1:

After our initial inspection of the source code, we selected `temporalAreaRadius`, `iterations`, and `btvKernelSize` as the subjects of our experiment, since they seemed to be the ones that influence the quality of the output and/or running time of the program. The results are provided in Figure 3.3. The input of the program is an 100×100 video of 40 frames. Here we will analyze the result and explain the functions of these parameters.

Obviously, as we increase `iterations` or `temporalAreaRadius`, the running time increases almost linearly. First, we conclude that `iterations` plays the same role as `T` in Code 1, which is the number of HR estimations computed. The more iterations the algorithm goes through, the more likely the result will converge to an output that would balance the data term and the smoothness term and give desirable solution. Second,

[1] `http://docs.opencv.org/2.4/modules/superres/doc/super_resolution.html`

TABLE 1. Parameters for the OpenCV SR algorithm

`int scale`	Scale factor.
`int iterations`	Iteration count.
`double tau`	Step size of steepest descent method.
`double lambda`	Weight parameter to balance data term and smoothness term.
`double alpha`	Parameter of spacial distribution in Bilateral-TV.
`int btvKernelSize`	Kernel size of Bilateral-TV filter.
`int blurKernelSize`	Gaussian blur kernel size.
`double blurSigma`	Gaussian blur sigma.
`int temporalAreaRadius`	Radius of the temporal search area.
`Ptr<DenseOpticalFlowExt> opticalFlow`	Dense optical flow algorithm.

`temporalAreaRadius` is the same as the the total image count **N** here. Moreover, the OpenCV implementation can produce a video of SR frames by moving a search window, which has the size of the `temporalAreaRadius`. It keeps a pointer to the current reference frame, i.e. the frame that is upsampled to produce the initial estimation of the super-resolved image. Starting from the first frame of the video, the program will execute the SR algorithm whenever $nextFrame()$ is called and the pointer will also move forward, appointing the next frame to be the reference frame. Therefore, in this context, `temporalAreaRadius` is the number of frames before and after the reference frame.

The formal definition of the Bilateral Total Variation(BTV) kernel is first introduced in [7], which is a "robust regularizer" to compute the smoothness term in the energy function minimization step. In Equation (2.4), we penalize the gradient of the image, i.e. differences of intensities between neighbouring pixels, and this method is called Total Variation(TV) regularization. BTV is a generalized form of TV regularization. Instead of just penalizing the gradient, it will compute the difference in a larger scale by treating pixels within a search window as a unit, translating the input image, and taking the

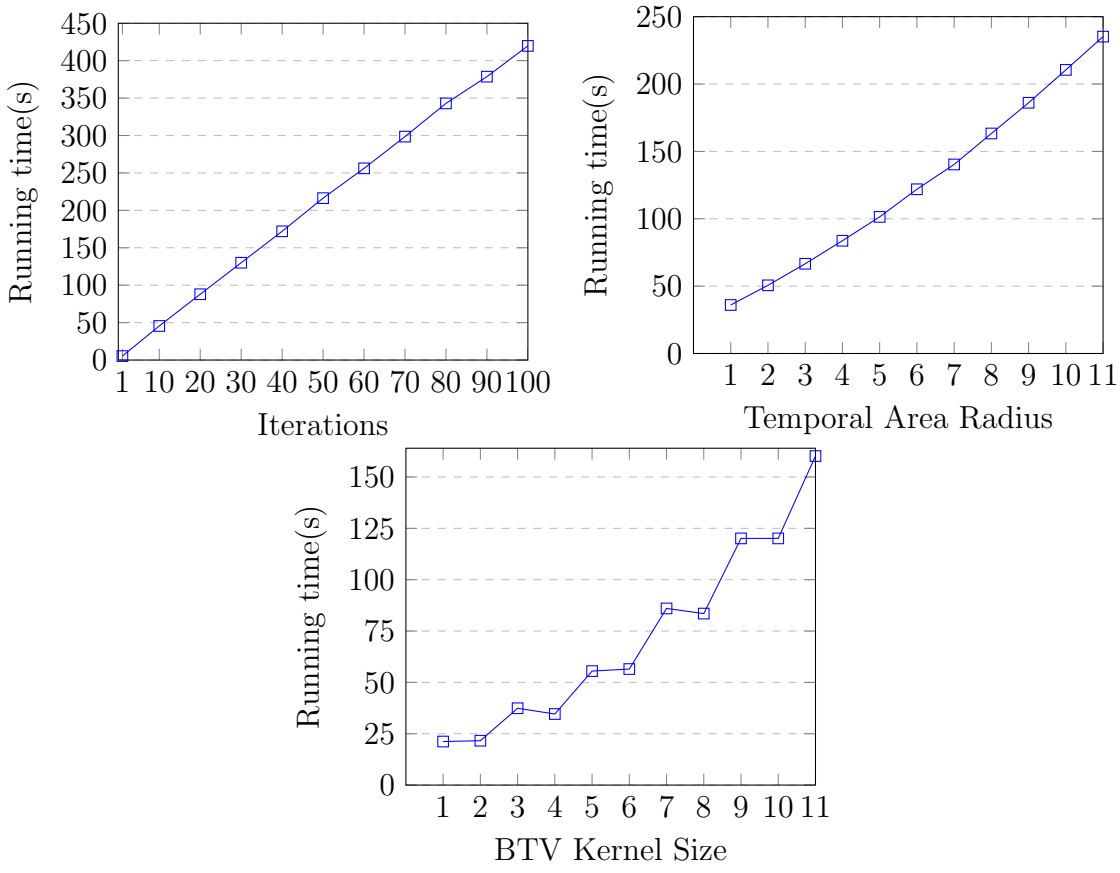
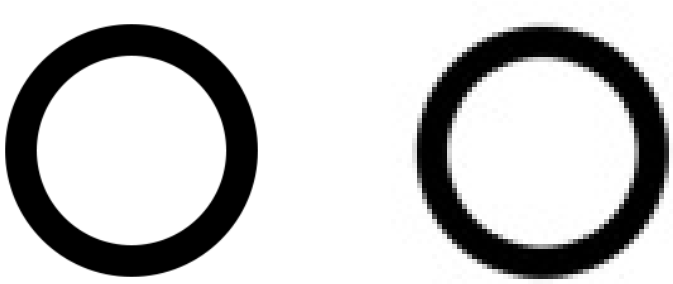


FIGURE 3.3. Results of varying different parameters

difference between original and translated image inside the same window. Multiple differences will be taken and weighted negatively proportional to the displacement from the translated image to the original. The output will be the summation of these differences. Given increasing BTV kernel size, the running time of the algorithm only increases if the kernel size is odd. As we later verified in the code, the program only permit odd kernel sizes, and even input will be rounded to the nearest odd number that is less than the input. The performance of it is estimated to $O(K^2)$, where K is the size of the kernel matrix that is applied to a certain image, since applying a kernel of size K to an image means to convolve the image with a $K \times K$. Matrix convolution can be done in $O(K^2)$ time, and some algorithm such as Fast Fourier Transform(FFT) can be faster.



(a) Original image (b) Degraded

FIGURE 3.4. Synthesized image and sample frame of the degraded dataset

3.2. **SSDs of Changing Parameters.** Sometimes it is hard and unreliable for us to visually measure the quality of the output. Therefore, we decided to create `circle`: a synthesized, simple image, as shown in Figure 3.4. We then manually downsampled and blurred the image, and used the degraded image to create a 40-frame video of that image. The original synthesized image is used as the ground truth and the degraded video of it as the input of the SR algorithm. In order to correlate the the quality of the output with values of parameters, we compute the Sum of Squared Difference(SSD) between an output and the ground truth:

$$SSD(I_1, I_2) = \sum_{i=1}^{m} \sum_{j=1}^{n} (I_1(i,j) - I_2(i,j))^2 \tag{3.1}$$

where I_1 and I_2 are both $m \times n$ images and $I(i,j)$ denotes the intensity value or the sum of intensity values of RGB channels of the pixel of the (i,j)th coordinate. Note that in the case when I_1 and I_2 are of different sizes, which is often the case for outputs of the OpenCV implementation since the Gaussian blur kernel shrinks both the width and the

height of the output by a small number of pixels, we will resize the ground truth using area interpolation algorithm provided in OpenCV [2] to match the size of the output.

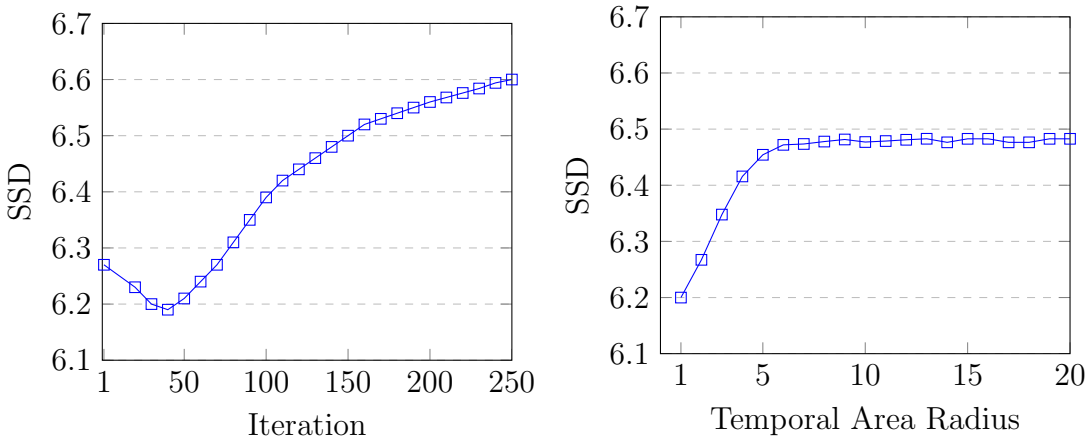


FIGURE 3.5. SSDs of varying different parameters

Using the synthesized dataset, we run the algorithm using varying iterations or temporal area radii, and then compute the SSD for each result, which is shown in Figure 3.5. Intuitively, it will make sense of we have a smooth curve of a decreasing negative slope for increasing iterations, since with more iterations the result will be more accurate and then at some point the solution to the energy function will converge, resulting in a slope close to zero. However, the result we got seems to contradict our prediction. There seems to be a point where the quality of the output suddenly starts to degrade, and converge to a higher plateau. The result obtained by increasing temporal area radius is more interesting. The input of the program, as mentioned in the Method section, is a video of 40 identical frames. Therefore, intuitively, the result will not fluctuate at all since we are processing the same image. However, as shown in the figure, the result gets worse as we increase temporal area radius.

[2] `http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html`

TABLE 2. Optical Flow implementation provided by OpenCV and the related papers

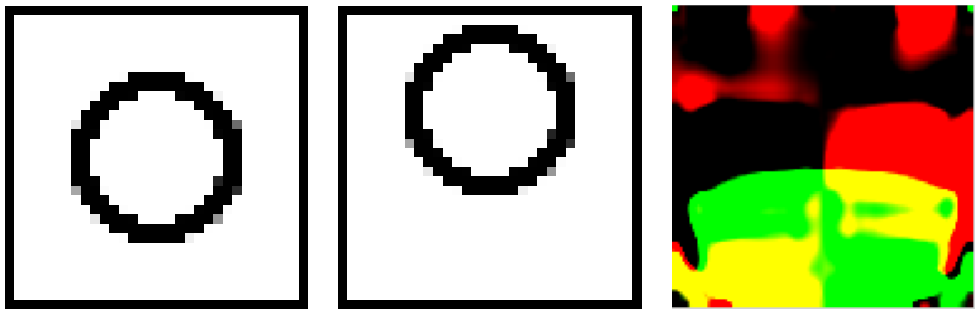
Class	Papers
`BroxOpticalFlow`	[10]
`DensePyrLKOpticalFlow`	[11]
`FarnebackOpticalFlow`	[12]
`OpticalFlowDual_TVL1`	[13]

To understand this result, we studied the optical flow algorithm that OpenCV uses. Since the synthesized video have all identical frames and there is no motion between any two of them, we expected the motion fields computed by optical flow to be matrices with all zero entries. This turned out to be true. At this stage, the exact reason behind the SSD results is still unknown. One possible reason could be that BTV regularization will add the weighted difference between an area in the original image and a different area in the translated image. Outliers can be introduced and the result therefore gets worsened.

3.3. **Window Size of Optical Flow.** As our attempt to improve the output, we investigated the accessible types of optical flows and the modifiable parameters associated with them. As for version 3.0.0, OpenCV provides five implementation of optical flow as shown in Table 2. By default, `superres` uses `FarnebackOpticalFlow`, which is a version of dense optical flow.

First, to see whether the optical flow algorithm produces accurate motion estimation, we created a dataset `circle_motion`: the same `circle` dataset with circles in even-numbered frames translated 10 pixels upwards. The first and second frames of `circle_motion` dataset are shown in Figure 3.6. We managed to print out the relative motion fields computed by the optical flow. Ideally, the optical flow will have vectors at each pixel denoting the vertical and horizontal displacement from the first frame to the second frame. Therefore, we would expect the relative motion from the first frame

to second frame to be a set of vectors pointing upwards. However, as shown in Figure 3.6, we observed vectors both pointing upwards and downwards. Note that red color represents a negative vertical displacement, i.e. moving downwards, and green a positive one, i.e. moving upwards.

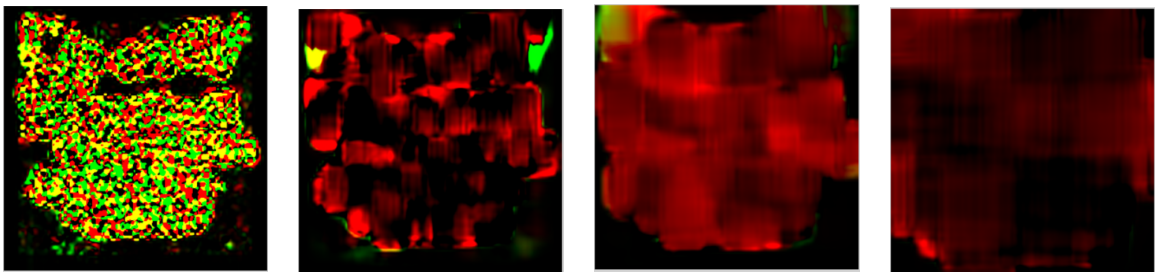


(a) 1^{st} frame (b) 2^{nd} frame (c) Motion estimation between these frames

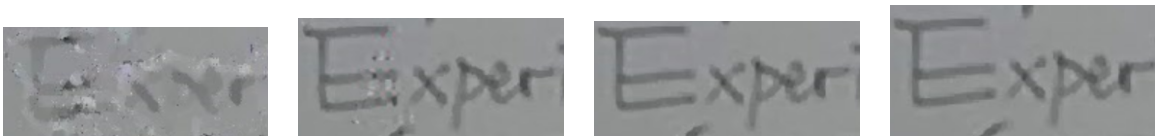
FIGURE 3.6. Frames from circle_motion dataset

This phenomenon is commonly known as the aperture problem[14]. If we observe a global motion of an object from a smaller local window, the moving contour of the object cannot provide enough information for us to determine the direction of the global motion. The dense optical flow in OpenCV suffers from this problem since it processes the image by looking at areas within a fixed size window.

Attempting to avoid the aperture problem, we experimented on changing the window size parameter in `FarnbackOpticalFlow` class, which is 13 by default. We apply the SR algorithm with varying optical flow window size on the "tome" dataset we created, which is a 90×90 video taken in Tome hall in Dickinson College. In (a)-(d) of Figure 3.7, when the window size is small, the estimation of optical flow tend to be inconsistent in different areas due to the aperture problem. As we increase the window size from 1, we observed that the blocky articles possibly created by the erroneous motion estimations faded away, as shown in (e)-(h) in Figure 3.7. However, when window size exceeded the



(a) window size = 1 (b) window size = 6 (c) window size = 13 (d) window size = 25



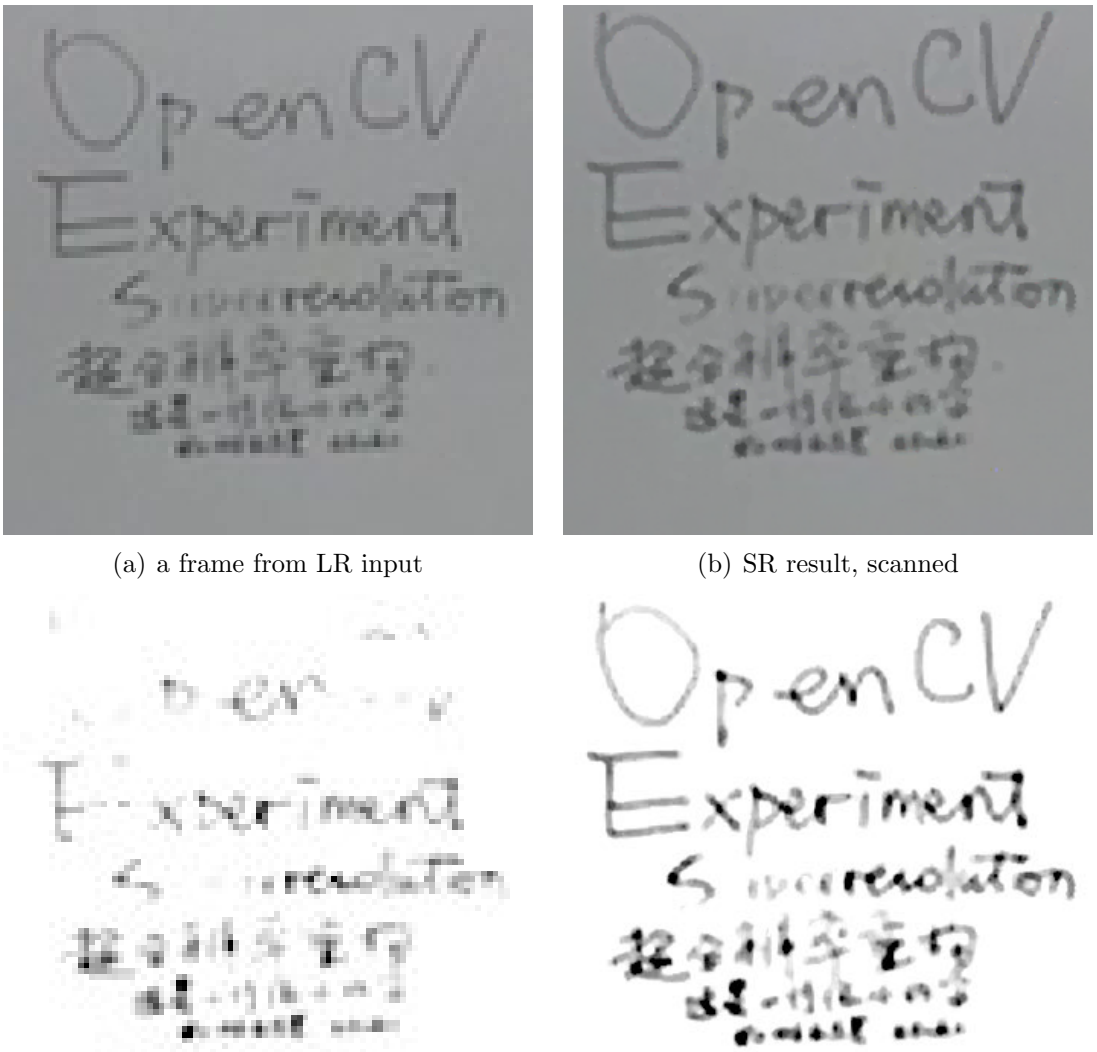
(e) window size = 1 (f) window size = 6 (g) window size = 13 (h) window size = 25

FIGURE 3.7. Result of Optical Flow on "tome" dataset.

default value, no visual improvement on the output was observed. Therefore, we conclude that there will not obvious improvement on the output by merely increasing the window size.

3.4. **Visual improvement with CamScanner.** To test the effect of SR algorithm, we process the outputs using a commercial application CamScanner[2]. CamScanner takes in a normal photograph and does a series of image enhancement techniques such as brightening and sharpening to create the effect of scanning. We use (1) one of the frame from the tome dataset and (2) super-resolved image generated from the same dataset as inputs to CamScanner and find out that texts are better preserved in the SR output. The result is shown in Figure 3.8, where (a) is an LR frame from the input video and (b) is the SR output, and (c) and (d) are scanned versions of (a) and (b), respectively. We observe a substantial improvement in terms of the information preserved, i,e, words on the whiteboard, and the sharpness of the scanned images (from (c) to (d) in Figure 3.8).

Since the imaging filters used by CamScanner seem to be proprietary, we do not have enough information yet to fully understand what improved the quality of the output.



(a) a frame from LR input

(b) SR result, scanned

(c) a frame from LR input

(d) SR result, scanned

FIGURE 3.8. Scanned comparison: LR input and SR output

Our hypothesis is that the SR algorithm improves the sharpness of the image by fusing the input frames and increases the intensity of pixels representing the words on the whiteboard, and therefore less information get filtered out by CamScanner. Another way to fully test this is to implement an image scanning algorithm similar to that of CamScanner, which will be addressed in the Future Work section.

4. Conclusion, and Future work

Our objective of the research is to apply SR algorithms on the proposed distant whiteboard scenario and seek further improvement on the algorithm to increase the quality of the result. First, it seems that using SR to pre-process a video from the distant whiteboard will improve the quality of the scanned result. However, due to the nature of the scanning software we use, it remains difficult to analyze the improvement numerically. We also gained a better understanding of both SR technique and optical flow algorithms when studying the OpenCV source code. We designed experiments to show the correlation between the running time of the algorithm and the values of certain parameters. Moreover, we attempted to numerically evaluate the result of the algorithm by calculating SSDs, which produced unexpected result. To improve the quality of the output, we tried to avoid the aperture problem that the optical flow algorithm has. First, we run the program with difference window sizes, the result showed that the effect of aperture problem will decrease as the window size increases. However, no visual improvement was observed for window sizes beyond the default value.

Although we managed to improve the scanned whiteboard image by using SR, more works need to be done to justify this improvement. For future work, numerical methods can be developed to evaluate the quality of the output and different motion estimation methods can be tested. Also, one can also develop an image processing algorithm that mimics that of CamScanner to analyze the visual improvement numerically.

Appendix A. W^T as an inverse operation

Here we will show one special case of the warping operation, horizontal displacement, and prove that the transpose of W indeed re-do what is done by W. Similar proofs can be done for other cases.

Suppose we have the original image $A_{M,N} = [a_{ij}]$ that is of M pixels height and N pixels width. We warp it using a warping matrix W, moving it x pixels to the right. The warping can be done on the vectorized version of $A_{M,N}$, and the we obtain the warped image by de-vectorizing the result. First, we stack the columns of $A_{M,N}$ to form an MN-vector:

$$vec(A_{MN}) = \begin{bmatrix} col_1(A_{M,N}) \\ col_2(A_{M,N}) \\ \vdots \\ col_N(A_{M,N}) \end{bmatrix} \tag{A.1}$$

Then W will be a matrix of size $MN \times MN$:

$$W_{MN,MN} = \begin{bmatrix} & \mathbf{0}_{Mx \times MN} & \\ 1 & \mathbf{0}_{1 \times MN-1} & \\ 0 & 1 & \mathbf{0}_{1 \times MN-2} \\ \vdots & \vdots & \vdots \\ \mathbf{0}_{1 \times MN-Mx-1} & 1 & \mathbf{0}_{1 \times Mx} \end{bmatrix} \tag{A.2}$$

Then we obtain the warped image $A'_{M,N}$ by applying $W_{MN,MN}$ on $vec(A_{M,N})$:

$$
\begin{aligned}
vec(A'_{M,N}) &= \begin{bmatrix} & \mathbf{0}_{Mx\times MN} & \\ 1 & \mathbf{0}_{1\times MN-1} & \\ 0 & 1 & \mathbf{0}_{1\times MN-2} \\ \vdots & \vdots & \vdots \\ \mathbf{0}_{1\times MN-Mx-1} & 1 & \mathbf{0}_{1\times Mx} \end{bmatrix} \begin{bmatrix} col_1(A_{M,N}) \\ col_2(A_{M,N}) \\ \vdots \\ col_N(A_{M,N}) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{0}_{1\times Mx} \\ col_1(A_{M,N}) \\ col_2(A_{M,N}) \\ \vdots \\ col_{N-x}(A_{M,N}) \end{bmatrix}
\end{aligned} \tag{A.3}
$$

Take of transpose of W and apply $W^T_{MN,MN}$ on $A'_{M,N}$:

$$
W^T_{MN,MN} = \begin{bmatrix} \mathbf{0}_{1\times Mx} & 1 & \mathbf{0}_{1\times MN-Mx-1} \\ \mathbf{0}_{1\times Mx+1} & 1 & \mathbf{0}_{1\times MN-Mx-2} \\ \vdots & \vdots & \vdots \\ & \mathbf{0}_{1\times MN-1} & 1 \\ & \mathbf{0}_{Mx\times MN} & \end{bmatrix} \tag{A.4}
$$

$$
\begin{aligned}
vec(A''_{M,N}) &= W^T_{MN,MN} vec(A'_{M,N}) \\
&= \begin{bmatrix}
\mathbf{0}_{1\times Mx} & 1 & \mathbf{0}_{1\times MN-Mx-1} \\
\mathbf{0}_{1+1} & 1 & \mathbf{0}_{1\times MN-Mx-2} \\
\vdots & \vdots & \vdots \\
 & \mathbf{0}_{1\times MN-1} & 1 \\
 & \mathbf{0}_{Mx\times MN} &
\end{bmatrix}
\begin{bmatrix}
\mathbf{0}_{1\times Mx} \\
col_1(A_{M,N}) \\
col_2(A_{M,N}) \\
\vdots \\
col_{N-x}(A_{M,N})
\end{bmatrix} \\
&= \begin{bmatrix}
col_1(A_{M,N}) \\
col_2(A_{M,N}) \\
\vdots \\
col_{N-x}(A_{M,N}) \\
\mathbf{0}_{1\times Mx}
\end{bmatrix}
\end{aligned}
\tag{A.5}
$$

As shown above, $W^T_{MN,MN}$ reverses the operation done by $W_{MN,MN}$, which is shifting $A_{M,N}$ x pixels to the right. $vec(A''_{M,N})$ is indeed equal to $A_{M,N}$, but note that $N-x+1$th to Nth columns were lost in the initial warping operation. Hence, without an appropriate paddling rule (e.g.: paddle x zero columns to the right of $A_{M,N}$), it is impossible to recover the lost columns.

References

[1] Li-wei He, Zicheng Liu, and Zhengyou Zhang. Why take notes? use the whiteboard capture system. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 5, pages V–776. IEEE, 2003.

[2] Ltd. NTSIG Information Co. Camscanner official website. vers. 3.8.1. `https://www.camscanner.com/`. Accessed: 2015-09-11.

[3] Kamal Nasrollahi and Thomas B Moeslund. Super-resolution: a comprehensive survey. *Machine vision and applications*, 25(6):1423–1468, 2014.

[4] Jie Yuan, Si-dan Du, and Xiang Zhu. Fast super-resolution for license plate image reconstruction. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

[5] Dennis Mitzel, Thomas Pock, Thomas Schoenemann, and Daniel Cremers. Video super resolution using duality based tv-l 1 optical flow. In *Pattern Recognition*, pages 432–441. Springer, 2009.

[6] KF Riley, MP Hobson, and SJ Bence. Mathematical methods for physics and engineering. *American Journal of Physics*, 67(2):165–169, 1999.

[7] Sina Farsiu, M Dirk Robinson, Michael Elad, and Peyman Milanfar. Fast and robust multiframe super resolution. *Image processing, IEEE Transactions on*, 13(10):1327–1344, 2004.

[8] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[9] Assaf Zomet and Shmuel Peleg. Efficient super-resolution and applications to mosaics. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 1, pages 579–583. IEEE, 2000.

[10] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *Computer Vision-ECCV 2004*, pages 25–36. Springer, 2004.

[11] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[12] Gunnar Farnebäck. Fast and accurate motion estimation using orientation tensors and parametric motion models. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 1, pages 135–139. IEEE, 2000.

[13] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Pattern Recognition*, pages 214–223. Springer, 2007.

[14] Josh McDermott, Yair Weiss, and Edward H Adelson. Beyond junctions: nonlocal form constraints on motion interpretation. *Perception*, 30(8):905–923, 2001.