Reflection Recovery and Removal via Image Edge Parallax

G. Kocher
Stanford University
Stanford, CA 94305
kocher@stanford.edu

Abstract

Reflections are very common in photographs, and depending on the application, they may be a nuisance you want to remove or an interesting signal to isolate and explore further. I demonstrate a method for separating the reflection and background components of images taken through reflective objects like windows. By taking a series of images at slightly different camera orientations, differences in the motion of the background vs. reflective layers can be exploited to separate the reflection from the background in a reference image chosen from the series. The two layers recovered this way are used as initial values in a nonlinear optimization problem which further refines the quality of the separated component images. This work builds on the work of others. I implemented the entire pipeline from start to finish and made different algorithm design decisions, used different optimization techniques, and imposed different constraints. I demonstrate the success of the algorithm on real data collected by myself and others in various situations, and discuss potential future improvements.

1. INTRODUCTION

Reflections are very common in everyday photographs, for instance when taking pictures through windows. Sometimes one may want to remove the reflection to have a clear, unobstructed view of the background scene. For example, if one is standing on the sidewalk taking a picture of something inside a store window, she may wish not to have reflections of cars or pedestrians walking behind her show up in the photograph. It often is not possible to simply change the camera orientation because other reflected objects may come into view, or the scene of interest in the window display may become occluded due to the geometry of the spatial layout.

Alternatively, reflections may be viewed as an interesting additional layer of information you want to isolate and explore further. For instance, if someone is standing indoors and taking a picture of an outdoor scene through a window, it could be very interesting to see what is around him in the room. Or a photographer at an art museum could see what other works were in that gallery by extracting the reflection from a picture of a framed photograph on the wall. Finally, it's easy to see how this could be useful for forensic or legal purposes to identify clues in the scene environment when someone takes a photograph.

Therefore, this paper explains the theory and algorithm details of an automated end-to-end MATLAB pipeline that separates the reflection and background components of a reference image, where that image is one of a series of photographs of a scene taken through a reflective plane like a window.

The outline of this paper is as follows: in Section 2, I describe prior work on this topic and the novelties used here. In Section 3, I provide a detailed explanation of the algorithm pipeline and design decisions. Section 4 shows experimental results on various data sets, and Section 5 presents some final conclusions and discusses potential future improvements.

2. PREVIOUS WORK & NOVEL CONTRIBUTIONS

2.1. Review of Previous Work

Previous work in the area of reflection removal can roughly be divided into three major categories. Work here is often closely associated with the removal of obstructions in general, not just reflecting elements.

The most straightforward but also least flexible solution is to change the physical environment during image acquisition. It's possible that by adjusting the camera orientation, the reflection can be lessened. However, this is not always possible, and restricts the picture taking process. Another physical solution is to take the exact same image twice but using lenses with different polarizations [3]. However, the requirements to take images with the same viewpoint and to have multiple specialized lenses is too limiting and is not suitable for everyday use with mobile phone cameras.

A second class of solutions tries to separate the background and reflection from a single image. However, without any other knowledge, this is an ill-posed problem because a single image could be separated into two

different additive layers in arbitrarily many ways. To get around this, people have imposed various constraints, for instance by assuming that if the user focuses on the background, then the foreground reflection will be blurrier [4], or taking advantage of flash on cameras [1]. Other approaches include manually marking reflection and background regions, but this is not convenient since it is not automated.

The third category is multiple-view recovery, where a series of images is acquired at slightly different camera orientations such that the background and reflection move differently and can be separated based on motion. Most prior research in this area has assumed some type of simplistic parametric motion between images, for instance assuming that both layers follow affine or projective transformations between images. The problem with these assumptions is that they cannot properly handle variation in scene depth. To circumvent this, optical flow has been used to find a dense motion field defined uniquely for each pixel. The main issue here is that optical flow typically assumes that layers are opaque, whereas in the reflection problem, each pixel has contributions from a background layer and a transparent reflection layer. This results in a poor initialization that contains visible oddities. For this reason, Xue et al. [7] proposes a method called "edge flow" to provide better initial estimates on the separated image layers and their motion fields. Using edge flow in combination with a constrained nonlinear optimization setup, they achieve respectable results.

2.2. Novelties in this Work

This paper mostly parallels Xue et al. [7]. It discusses my end-to-end MATLAB implementation of an automated pipeline to separate the reflection and background components of a reference image taken as part of a series of photographs of a scene through a reflecting element like a window. Along the way, I made different design decisions than those authors and thought of improvements that could be made to their work. Specifically, although I also use Normalized Cross-Correlation to estimate the initial motion field, my procedure for finding the most likely motion vectors is different, and I filter out spurious pairings in a more methodical way. I also propose a potentially better way of using RANSAC to get initial motion field estimates for the reflection and background layers. Finally, I solve a modified version of the optimization problem, use different optimization techniques for added stability, and propose additional constraints to improve the optimized result.

3. TECHNICAL APPROACH

3.1. Technical Summary

The algorithm implemented here ingests a series of images taken from different viewpoints through a reflective element like a window. The output is a pair of images: one for the background layer of the reference input image and one for the reflection layer of the reference image. To achieve this decomposition into two separate layers, there are two major steps. The first is to get a good initial decomposition of the background and reflection layers (Section 3.3), and the second is to solve an optimization problem that refines those initial estimates (Section 3.4).

More specifically, the first task of finding an initial estimate consists of the following steps: for each image relative to the reference image, determine a sparse motion field that transforms the reference image into the target image. This is done by calculating the normalized cross-correlation (NCC) for edge pixels in image1 to image2 to define a motion vector. RANSAC is performed to fit a projective transformation to background and reflection edge pixels as a way of assigning pixels to either of the two layers. The sparse motion fields are then interpolated to get dense motion fields. The initial estimates of the background and reflection layers are calculated using pixel-wise operations as detailed in Section 3.3.

The optimization step consists of minimizing the objective function defined in Section 3.4. The initial estimates for the background and reflection layers and their motion fields are used as the starting point of the nonlinear optimization problem. Constraints are imposed to ensure valid results and improve the quality of the final solution.

3.2. Technical Background

For planar reflective elements like windows, the geometry of reflections is as simple as that shown in Figure 1. The reflective plane is between the camera and the background, and some objects (represented by a chair in Figure 1) are on the same side of the reflective plane as the camera. Photons from those objects reflect of the window and enter the camera lens, appearing to originate from a virtual object on the other side of the reflecting plane. A fair assumption is that each pixel in the final image is an additive composition of a background image and a reflection image. Depending on lighting conditions, the background or the reflection component may be stronger (e.g. if you are indoors taking a picture facing outside with interior lighting, during the day the background will be stronger but at night the reflection will be stronger). The relative contributions of the background and reflection can be dealt with by introducing an opacity or alpha matte constant.

In the case of multiple images with static background and reflection layers, one of the images can be chosen as a

Figure 1. Reflection Geometry.

reference image and the other images can be described as warped versions of that reference image. Equation 1 expresses the index *t* image, I^t, as a sum of two parts:

$$I^t = (1-\alpha)\boldsymbol{W}(V_R^t)I_R + \alpha\boldsymbol{W}(V_B^t)I_B \qquad (1)$$

There is I_R, where the subscript R denotes the reflection layer and absence of superscript denotes the reference image, and there is I_B where the subscript B means background layer. Each image is a flattened vector of length Npixels, and the variables V_R^t and V_B^t are the dense motion field vectors determined via NCC, and are defined at every pixel and have both an x and y component. They dictate where a pixel in one image moves to in a second image. $\boldsymbol{W}$ is an Npixels x Npixels warping matrix that transforms I_R into I_R^t and I_B into I_B^t. The $\boldsymbol{W}$ matrices are functions of the motion field between the reference image and the index t image, and are matrices of bilinear interpolation weights. Details of how I calculated the warping matrices are given in Section 3.4. The constants α and $(1-\alpha)$ are the alpha matte, such that a pure reflection would have $\alpha = 0$ and a pure background would have $\alpha = 1$. Finally, note that there is a corresponding equation for each of the 3 color channels (or alternatively, each flattened image can be thought of as a length 3*Npixels vector, and the warping matrix can be written as a 3*Npixels x 3*Npixels matrix).

Therefore, the goal is to obtain the best possible estimates of I_B and I_R (which also requires finding α and the two sets of motion fields $\{V_R^t\}$ and $\{V_B^t\}$). The cost function in Section 3.4 follows directly from this, and the problem now becomes one of finding good initial estimates, which is the topic of the next section.

3.3. Initial Solution

A good initial solution for the two separated layers and their motion fields is important because the final step is to solve a large nonlinear optimization problem which is liable to get stuck at local minima or saddle points. As discussed earlier, such techniques as optical flow may fail and result in weird artifacts in the images. Therefore, I chose to implement a modified version of "edge flow" proposed in Xue et al. [7].

Edge flow relies on the key observation that edge pixels usually belong to *either* a background edge *or* a reflection edge, but not both. For an edge pixel to belong to both the background and the reflection, those edges would have to overlap perfectly in the image, so in general, most edge pixels will not belong to both components since the alignment between layers won't be perfect. Also, since the background and reflection are independent, those two layers move differently, so edge pixel motion-based separation is a robust way of distinguishing the 2 layers.

The procedure goes as follows. First, extract the Canny edges of each of the images of the series. The absolute number of edge pixels will depend on the image dimensions, as well as the parameters used in Canny edge detection (e.g. the sigma sizes). For the datasets used here, with image dimensions on the order of several hundred to one or two thousand pixels on a side, there were usually too many edge pixels to use all of them in later computation. Therefore, in large images, I randomly sampled 20% of the edge pixels to keep for the next step.

Once a (sub)set of image edge pixels is obtained, a sparse motion field is determined by calculating the Normalized Cross-Correlation (NCC) between image edges. Specifically, for a window centered on each of the edge pixels in the first image, the NCC between that window and all areas of the reference image is computed. Peaks are found in the NCC space, and the location of the maximum peak is recorded. To get subpixel estimates of the x and y components of the motion vector, the centroid of a very small window centered on the peak is calculated. The x and y coordinate offsets of the peak in NCC space give the x and y components of the motion vector for the edge pixel in the first image. This procedure is repeated for the whole (sub)set of edge pixels in each image. I ended up using the OpenCV version of NCC and mexing it, which provided at least an order of magnitude speed up over the built-in MATLAB NCC function. The output of this step is a set of motion vectors defined at each of the (sub)set of edge pixels, representing the flow of that edge pixel in the first image to a new location in the second image. This motion field is sparse in the sense that it is only defined at edge pixels (not every pixel of the image), and it is still a single motion field which has not yet been split into background or reflection components. Representative images are shown at center of Figure 2. Note that there are still some spurious motion vectors.

The next step is to decompose the combined motion field into background and reflection motion components. In most common imaging scenarios, either the background or reflection will be stronger (the side of the reflective plane with stronger lighting will dominate when the two layers are additively combined). Therefore, either background or reflection edges will be dominant, and random selection of edge pixels will more likely choose pixels from that layer. Combining this observation with the fact that the two layers move differently gives an approach to separate the two motion layers. First,

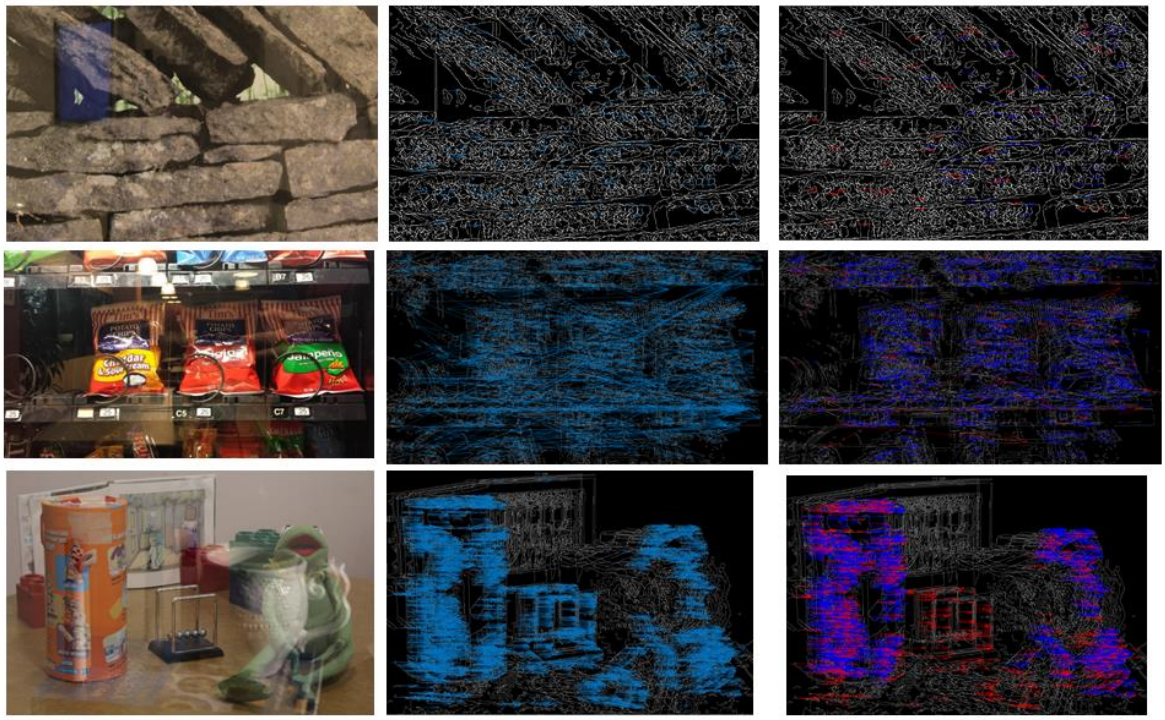

Figure 2. *Left column*: Input images. *Center column:* Motion field vectors made from NCC on a random sample of edge pixels. *Right column*: Background (dark blue) and reflection (red) motion field components.

RANSAC is used on the full set of edge pixels to fit a projective transformation. Depending on which layer dominates, this will fit a projective transformation to either the background or reflection. Then the remaining edge pixels (outliers of the RANSAC model) will mostly belong to the other layer or be noise. So RANSAC is used again to fit a projective transformation to the second layer. The output of this step is, for each image with respect to the reference image: a sparse motion field for the background and another sparse motion field for the reflection. Examples are seen at right of Figure 2.

Some comments about the RANSAC procedure are useful here. First, the greedy RANSAC approach used by Xue et al. [7] (sequentially fitting a homography to one layer, then fitting a second homography to the outliers) works well in general, but I think it can be improved by fitting the two models in parallel. In this case, 4 points are randomly chosen to fit a homography, then another 4 points are chosen to fit a second homography. Inliers are uniquely assigned to only one of the transformations, and a score is used to determine the goodness of the combined two transformation model, where the score is a function of the number of inliers of both transformations (e.g. the harmonic mean like the F-score used in binary classification). Finally, the joint model is valid only if the two individual transformations are at least as different as some threshold. That threshold is a function of the difference of the two components of the model, e.g. a distance metric, an information theoretic measure related to how independent the individual transformations are, or a measure of how far apart on average the transformations of the same point would be under the two transformations. This version of joint RANSAC should improve the initial decomposition even further, at the cost of being more computationally expensive because two transformations would need to be fit for each model.

Another point to consider about the RANSAC procedure is that the actual projective transformations calculated for each layer are not important, they simply are a way of splitting the motion field vectors into a unique set of background and reflection pixels, but the transformation matrices are not used again in the algorithm pipeline.

Finally, there are some minor steps that can be taken to improve the inlier to outlier ratio before employing RANSAC. In particular, since I ask the user to move their camera roughly horizontally to acquire images at different orientations, this means that motion vectors with large vertical components are probably due to spurious NCC matches, and are excluded. A possible minor improvement that would add very little burden to the user would be to tell the user to move their camera always from left to right, and exploit this by also excluding NCC matches with the opposite direction. These minor operations improve the success of RANSAC by increasing the proportion of inliers, which is especially important if only a subset of edge pixels are used in order to reduce computation time.

The steps so far have resulted in sparse motion fields for the background and reflection layers of each of the images (the motion fields for the reference image are all 0's since it doesn't move with respect to itself). Next, I

Figure 3. *Left*: Initial background estimate. Note problems with blue monitor. *Center*: Initial reflection estimate. *Right*: since reflection is hard to see, the right is the same image but contrast adjusted.

interpolated the sparse motion fields using bilinear interpolation to get dense motion fields. The set of motion fields for all index *t* background images, $\{V_B^t\}$, and the set of motion fields for the index *t* reflection layers, $\{V_R^t\}$, are saved for the final optimization step.

The last ingredient we need is an initial decomposition of the reference image into a background image and a reflection image. This is achieved by recalling that we assumed the reference image is an additive combination of a background image and a reflection image. Szeliski et al. [6] estimate the background image by taking the stack-wise minimum of the background aligned images. In other words, after warping each image into alignment with the reference image, for every pixel, take the minimum value over the aligned stack of images to create a new image which is the initial background estimate. In practice, I found that this worked fine, but occasionally the median over the image stack looked better even though there isn't as good theoretical justification. In particular, using the median produces a visually better initial background estimate than the minimum when the reflection layer has dark regions. If the minimum is used, those dark reflection layers can be repeated multiple times across the estimated image. The initial estimate of the reflection image follows directly by subtracting the background estimate from the reference image. So, the relevant set of equations is:

$$\begin{aligned} \hat{I}_B &= \min(I^t) \\ \hat{I}_R &= I - \hat{I}_B \end{aligned} \tag{2}$$

where the hat accents denote initial estimates, I is the reference input image., and the minimum is understood to be elementwise across the stack of images. We know have all of the required pieces to feed into the optimization problem discussed in the next section. An example of initial background and reflection estimates is shown in Figure 3. Because the reflection image is dark, the right side of Figure 3 also shows a contrast enhanced version. It is impressive that even from this initial estimate, you can clearly see details that were indiscernible in the original image, such as the coffee mugs at the lower left, or the large letters on the right side of the image.

3.4. Optimization

Everything up to now has been to get a good initial solution because the next step is to solve a large, nonlinear optimization problem. The objective function follows directly from the decomposition equation (Eq. 1), and is:

$$\min_{\alpha, I_R, I_B, \{V_R^t\}, \{V_B^t\}} \sum_t \| I^t - (1-\alpha)\boldsymbol{W}(V_R^t) I_R - \alpha \boldsymbol{W}(V_B^t) I_B \|_p$$

Subject to:

$$0 \le \alpha, I_R, I_B \le 1 \tag{3}$$

In other words, we need to minimize the sum of L^p errors between each index *t* image (I^t) and the decomposition of the reference image into a background and reflection layer (I_B and I_R). All terms here are the same as in Section 3.1. The reference background and reflection layers are each warped by warping matrices $\boldsymbol{W}^t$ to align with the index *t* background and reflection layers (details of the warping matrices will be discussed shortly). The reflection and background motion fields V_R^t and V_B^t are the same as described before: they are defined at every pixel and dictate how each pixel in the reference background and reflection layers transforms into the corresponding layer of the index *t* image. The constant α is the same alpha matte constant as before. The constraint is needed since α and the image vectors I_R and I_B all lie on [0,1].

Each $\boldsymbol{W}$ matrix is a function of a motion field between the reference image and the index *t* image (V_R^t or V_B^t), and is a matrix of bilinear interpolation weights. To derive the index *t* warping matrix for a given layer, I back-projected each pixel of the reference layer onto the index *t* image according to the dense flow field defined at that pixel location of the reference image. In general, the back-projected location falls between pixels of the index *t* image, and not exactly on any pixel. Therefore, interpolation is required. Because I earlier interpolated the sparse motion fields to get dense motion fields (Section 3.3) by using bilinear interpolation, the interpolation used here must be the same bilinear interpolation. A possible improvement would be to get a smoother motion field by

using e.g. bicubic interpolation, but when tested, it significantly slowed the computation in Section 3.3, so was not used there, and therefore is not used here either. As an example, the i^{th} pixel of I_R^t is a weighted sum of the four pixels from I_R that surround the interpolated point (4 because bilinear interpolation on a regular grid considers the 4 corners surrounding a given back-projected point). So, the i^{th} row of $\boldsymbol{W}(V_R^t)$ has 4 nonzero elements, and element *ij* of $\boldsymbol{W}$ gives the contribution of pixel index *j* of the reference layer to pixel *i* of the index *t* layer. Only the 4 corner pixels around the back-projected location have nonzero values in the $\boldsymbol{W}$ matrix, and those values are exactly the bilinear interpolation weights. One caveat is that if a back-projected location would be outside the target image, it is excluded and that row has all 0's. Also, because each pixel has a single motion vector (not a motion vector for each individual color channel), I used the same warping matrix for all three color channels.

Since the ultimate purpose of the optimization is to get the best possible I_R and I_B estimates, we can temporarily ignore some of the other variables in the optimization problem of Eq. 3. Specifically, we can assume that the initial motion fields are approximately correct and treat them as constants by holding their values fixed. This significantly simplifies the optimization problem and allows us to move forward in solving it for I_R and I_B. If desired, the motion fields could be reconsidered after improving the estimates of I_R and I_B (see end of this section and also Section 5).

Even though all the terms are now understood, a complication with the above formulation (Eq. 3) is that there are products of the variables we are trying to solve for (the terms containing αI_R and αI_B are products and so are nonlinear). So, we can linearize the system of equations to simplify it. Filling in some gaps in the derivation of [8], I approximate the product $xy \approx x\hat{y} + \hat{x}y - \hat{x}\hat{y}$, where the hat terms are close in value to the true non-hat terms. Specifically, I use $\alpha I_R \approx \alpha \hat{I}_R + \hat{\alpha} I_R - \hat{\alpha}\hat{I}_R$ and same for the *B* subscript term (background). For this problem, the hat terms are just the values obtained from the previous iteration of the algorithm, so as long as the initial guess is not too bad, then the hat terms are close to the true values and the approximation is valid. In rewriting the objective function, there end up being a few terms that are constants (the gray highlighted terms below), so we can remove them since they won't affect where the cost is minimized:

$$C = \sum_t \left\| I^t - W_R^t\left[(1-\alpha)\hat{I}_R + (1-\hat{\alpha})I_R - (1-\hat{\alpha})\hat{I}_R\right] - W_B^t\left[\alpha\hat{I}_B + \hat{\alpha}I_B - \hat{\alpha}\hat{I}_B\right]\right\|$$

$$= \sum_t \left\| I^t - \left[W_R^t\hat{I}_R - \left(W_R^t\hat{I}_R\right)\alpha + \left((1-\hat{\alpha})W_R^t\right)I_R\right] - \left[\left(W_B^t\hat{I}_B\right)\alpha + \left(\hat{\alpha}W_B^t\right)I_B\right]\right\|$$

$$= \sum_t \left\| I^t - \left(W_B^t\hat{I}_B - W_R^t\hat{I}_R\right)\alpha - \left((1-\hat{\alpha})W_R^t\right)I_R - \left(\hat{\alpha}W_B^t\right)I_B\right\| \quad (4)$$

where I've used the shorthand notation $\boldsymbol{W}_*^t$ to mean $\boldsymbol{W}(V_*^t)$. Now define $\beta \equiv 1-\alpha$ and similarly, $\hat{\beta} \equiv 1-\hat{\alpha}$. Every term with a hat accent in Eq. 4 is just a constant obtained from the previous iteration, so the reformulated cost function can now be put in the form **Ax-b** as follows:

$$\begin{bmatrix} \hat{\beta}W_R^1 & \mathbf{0} & \mathbf{0} & \hat{\alpha}W_B^1 & \mathbf{0} & \mathbf{0} & (W_B^1\hat{I}_B^1 - W_R^1\hat{I}_R^1) \\ \mathbf{0} & \hat{\beta}W_R^1 & \mathbf{0} & \mathbf{0} & \hat{\alpha}W_B^1 & \mathbf{0} & (W_B^1\hat{I}_B^2 - W_R^1\hat{I}_R^2) \\ \mathbf{0} & \mathbf{0} & \hat{\beta}W_R^1 & \mathbf{0} & \mathbf{0} & \hat{\alpha}W_B^1 & (W_B^1\hat{I}_B^3 - W_R^1\hat{I}_R^3) \\ & \cdots & & & \cdots & & \cdots \\ \hat{\beta}W_R^N & \mathbf{0} & \mathbf{0} & \hat{\alpha}W_B^N & \mathbf{0} & \mathbf{0} & (W_B^N\hat{I}_B^1 - W_R^N\hat{I}_R^1) \\ \mathbf{0} & \hat{\beta}W_R^N & \mathbf{0} & \mathbf{0} & \hat{\alpha}W_B^N & \mathbf{0} & (W_B^N\hat{I}_B^2 - W_R^N\hat{I}_R^2) \\ \mathbf{0} & \mathbf{0} & \hat{\beta}W_R^N & \mathbf{0} & \mathbf{0} & \hat{\alpha}W_B^N & (W_B^N\hat{I}_B^3 - W_R^N\hat{I}_R^3) \end{bmatrix} \begin{bmatrix} I_R \\ I_R \\ I_R \\ I_B \\ I_B \\ I_B \\ \alpha \end{bmatrix} - \begin{bmatrix} I^1 \\ I^1 \\ I^1 \\ I^2 \\ I^2 \\ I^2 \\ \cdots \\ I^N \\ I^N \\ I^N \end{bmatrix} \quad (5)$$

where the superscripts denote the index of the image (1 to N, where N is the number of images in the series), and the coloring scheme emphasizes that there are 3 color channels and the warping matrix for each color channel is the same for a given image pair and reflection/background layer. Note that the vector **x** in Eq. 5 above contains the single constant α in addition to the two flattened image vectors I_R and I_B. This is because we don't know its value a priori, although I do assume it's constant across the whole image instead of being defined differently at each pixel since materials like glass and plastic reflect light approximately homogenously under most conditions [7]. I set it to .5 to start with, which I think is a reasonable approach. The vector **b** is the flattened vector of all 3 color channels of every index *t* image in the stack. Note that the **A** matrix is very big, but is extremely sparse, having at most 9 nonzero elements per row.

Now that the system of equations is in a convenient form, we need to decide what L^p-norm to minimize. Initially, I tried L2, enforcing the constraints of Eq. 3 by using MATLAB's built-in "lsqlin" function. However, unsurprisingly, the L2 norm proved not to be robust and gave poor results. Next, I decided to minimize the L1 norm as in Xue et al. [7] since it is more robust. I used Iteratively Reweighted Least Squares (IRLS), which can minimize cost functions of various L^p-norms, including p=1. It does this iteratively by solving the least squares problem and then updating the weights based on the residuals [5], and allows us to take advantage of the good initial estimates arduously obtained in Section 3.3.

To enforce the [0,1] upper and lower bounds on α, I_R, and I_B, I followed the discussion in example 4.7.6 of Boyd and Vandenberghe [10] and added a penalty constraint on each as follows:

$$\lambda_k[\min(0,x)^2 + \min(0,1-x)^2] \quad (6)$$

where x represents the three quantities α, I_R, and I_B. It is zero if the term is within [0,1], and large otherwise. For α, the penalty term is a scalar, whereas for I_R and I_B the penalty is a vector of length 3*Npixels since it applies to all 3 color channels of each pixel in the image. Initially, I decided to gradually ramp up the penalty rather than keep it fixed, as this has stability benefits according to [9]. So, λ_k represents the penalty constant at iteration k, and I started it at 10e1 and multiplied it by 10 after each iteration. Ultimately, however, I decided not to spend time on hyperparameter tuning to figure out the best λ_k ramp-up schedule, so I implemented a very large fixed cost by setting λ_k=100000 for every iteration k.

Finally, the optimization can be performed as follows. Using the linearized objective function from Eq. 5 with an L1-norm and the added constraints of Eq. 6, IRLS is performed for several iterations to get a refined estimate of the background and reflection layers of the reference image, I_R and I_B.

An interesting different approach used in Xue et al. is to use an alternating method where after getting updated I_R, I_B, and α estimates, those values are held fixed and the motion fields are refined. This alternating process could be repeated several times, refining all estimated terms to get an improved final solution. However, I didn't implement this two tier approach due to time and complexity constraints, but the single round of IRLS on its own moderately improved the initial results. The results in Section 4 show a final reflection layer with significant detail that is practically invisible in the original input series.

Another potential improvement could come in the form of additional constraints. Xue et al. imposed such constraints as sparsity on image gradients, sparsity on motion field vectors, and an edge ownership prior (following the logic that an edge usually does not belong to both the background and reflection).

I also thought of other potentially useful constraints to impose but ultimately did not implement them. For instance, vector calculus properties of the motion fields are constrained by the physics of how images can transform when the background and reflection layers are assumed to be static. Some possible constraints that exploit this are mentioned in Section 5 when discussing future improvements.

4. EXPERIMENTAL RESULTS

The final results are encouraging and show the potential of this approach to recover a reflection layer from a series of input images to provide useful insights.

The image series presented in this paper are a combination of my own images acquired in an office environment with other image series acquired by Xue et al. I used a Samsung Galaxy S4 for my photographs.

The computing environment used was a moderately high end Windows 7 PC. Unfortunately, due to high computational loads and the prototype, non-optimized code written in MATLAB, a typical series of images took over 40 minutes to process and sometimes had memory management difficulties. This computing time could be drastically reduced by improving the code structure, using a lower level language, and using different hardware.

The top row of Figure 4 shows a reference input image on the left (the 3rd of a series of 5 images), and the final recovered reflection output by the algorithm on the right. The bottom row is the same but for a different input image series. The algorithm performs reasonably well on the top series of images. This is probably for a few reasons. First, the images are of decent quality (e.g. not blurry). Second, the motion between sequential images is almost purely translational (confirmed by the camera matrix that RANSAC estimated) which may make it easier to separate the motion fields of the background and reflection. Also, there didn't seem to be any saturated regions or specular reflection (difficulties with these scenarios are discussed soon). The algorithm is more challenged by the series shown in the bottom row. There was some blur in some of the images of the series since I acquired this sequence indoors with less lighting, leading to inaccuracies in the edge flow motion fields, causing the alignment between sequential images to suffer, leading to reduced quality results. Also, changes in illumination from specular reflection on the potato chip bags confused NCC somewhat, causing difficulties for edge flow. Overall, though, the algorithm did help somewhat to extract the reflection layer. In particular, the gray and red striped shirt, the hand, and some tables become very clear.

This implementation does not perform nearly as well as Xue et al. In the initial edge flow step, due to resource constraints, I had to randomly sample a small fraction of all edge pixels, leading to sparser motion fields that required more interpolation, causing alignment to degrade somewhat. Also significant is the fact that my optimization does not recursively alternate between solving for the decomposed layers and then the motion fields as in Xue et al. Instead, my optimization only refines the estimates for the decomposed layers.

5. CONCLUSION

I demonstrated a MATLAB pipeline for ingesting a series of images taken at slightly different viewpoints and outputting separated background and reflection images for a reference input image. The algorithm takes advantage of motion differences of the background and reflection layers to get good initial estimates of the decomposition, and then solves a nonlinear optimization problem to refine those estimates.

There are several issues and assumptions that future

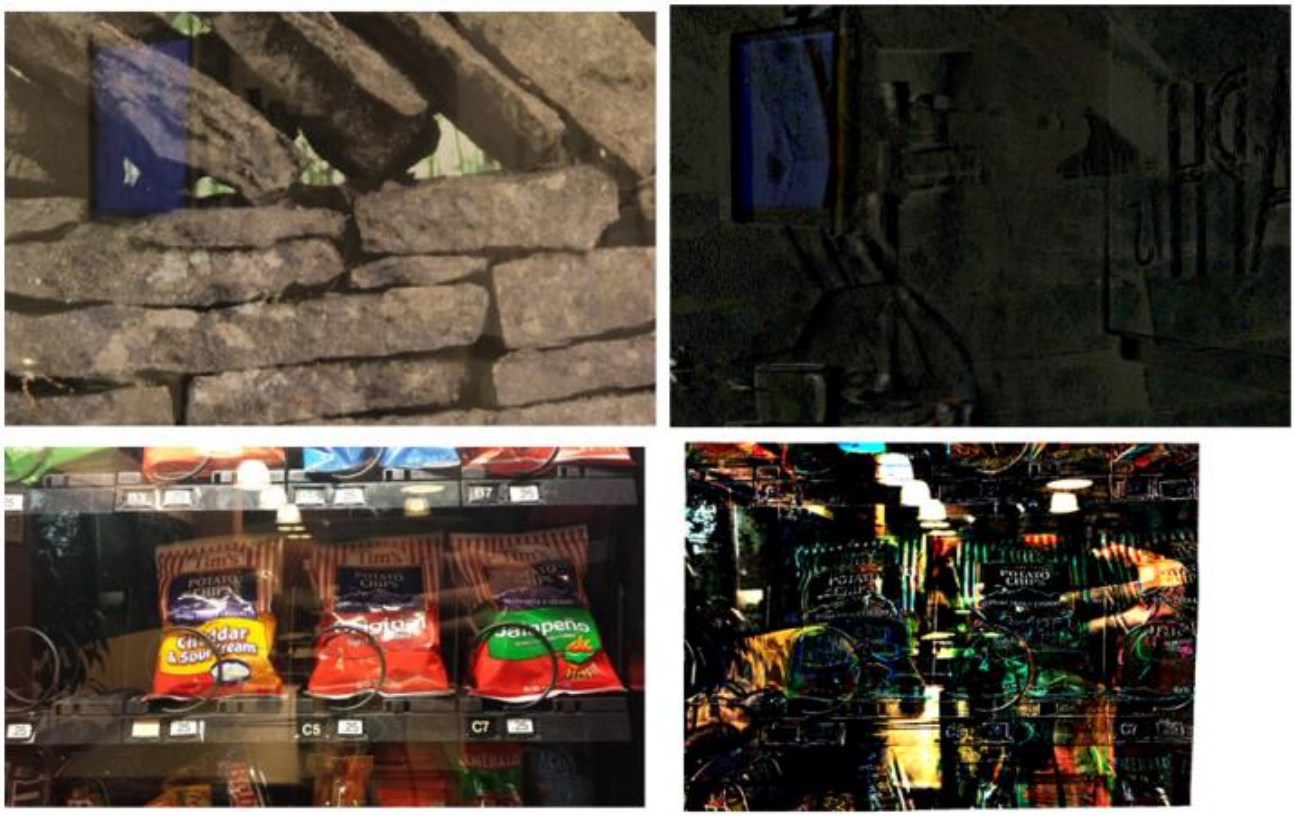

Figure 4. *Left*: Example original reference input images. *Right*: The reflection layers recovered after running through the algorithm. Note that the lower right image has been contrast adjusted for visibility. Note the overall good performance on the top image, and difficulties with the bottom image. However, visibility of some objects (hand, tables, plant, windows) are improved.

work could address. First, there are aspects of the image acquisition process that could be improved by being made more robust. In particular, if the background and reflection layers are not static but instead contain moving elements like cars or people or trees swaying noticeably in strong wind, then sequential images have motion fields that contain small scale local irregularities, e.g. due to a flexible tree branch being bent significantly differently across images. This worsens the alignment of the image stack and degrades the quality of the initial layer decomposition since it depends on pixel-wise operations and requires that pixels be reasonably well aligned. Also, specular reflection can sometimes be hard to deal with. Since specular reflection depends on the viewing orientation, when the camera is moved across the scene the pattern of specular reflection can sometimes change drastically. This also leads to local irregularities in the warping motion fields. Finally, saturated image regions pose problems because if a pixel region is clipped at a maximum value, you cannot recover the additive combination of two layers that produced it. Therefore, very bright lights may pose problems, and saturation should be avoided.

Second, in the initial decomposition of the image into a background and reflection layer, Xue et al. used a greedy RANSAC to sequentially fit a transformation to the background edge pixel inliers, and then a second transformation for the reflection layer. As discussed in Section 3, my idea of using RANSAC to fit the transformations in parallel should help the quality of the initial decomposition at the expense of increased computational complexity.

Third, the choice of appropriate constraints is of significant importance in any optimization problem. In addition to those constraints used in Xue et al., I think a potentially helpful set of constraints would view the motion field in terms of a general vector field and incorporate characteristics of the divergence and curl. Because the user is told how to move their camera across the scene (roughly horizontally over a few centimeters – inches), the vector field should have certain properties. For instance, unless there is a large zoom between images (which there wouldn't be if the user is following instructions) then there shouldn't be any large radial divergence. In other words, there shouldn't be source or sink points where the vectors diverge or converge radially. Similarly, even though there will likely be some degree of image-wide rotation due to the user tilting the camera, there shouldn't be small regions with large curl. Certain properties as these could be incorporated to smooth out the motion field and make it more realistic, thus improving the final result.

Also in the optimizations step, some possible improvements to IRLS stability are to figure out a good schedule for ramping up the penalty (lambda), and also using a similar adjustment on the value of p: you could

also ramp down the value of p in the p-norm value. For example, if using the L1-norm, you could start at p=2, then gradually decrement it toward 1 in small steps. According to [9], this is often helpful.

Finally, the likely biggest aid to performance would be to carry out the alternating optimization routine of Xue et al. where the motion fields and layer decompositions are alternately refined (instead of the approach here of just refining the layer decomposition a single time).

One last aspect that could be addressed is to accommodate reflections from non-planar reflecting elements. In the results presented here, the reflective element was a planar surface like a window or layer of plastic. More complex surfaces, like the curved side window of a car, also reflect light, but as the reflection moves across the surface the reflection is deformed as the local contour changes the geometry of the reflected light.

Code and sample data:
https://github.com/wasd12345/FinalProject

ACKNOWLEDGEMENTS:
Thanks to Tianfan Xue of MIT CSAIL for his willingness to answer multiple questions via email.

6. REFERENCES

[1] A. K. Agrawal, R. Raskar, S. K. Nayar, and Y. Li. Removing Photography Artifacts using Gradient Projection and Flash Exposure Sampling. ToG, 24(3):828–835, 2005.
[2] M. Brown, Y. Li. Exploiting Reflection Change for Automatic Reflection Removal, 2013.
[3] N. Kong et al. A Physically-Based Approach to Reflection Separation: From Physical Modeling to Constrained Optimization. IEEE Transactions on Software Engineering 36(2), 2013.
[4] Y. Schechner, N. Kiryati, and R. Basri. Separation of Transparent Layers using Focus. IJCV, 39(1):25–39, 2000.
[5] Stanford Exploration Project. Iteratively Reweighted Least Squares (IRLS), 2004. http://sepwww.stanford.edu/data/media/public/docs/sep115/jun1/paper_html/node2.html
[6] R. Szelisky et al. Layer Extraction from Multiple Images Containing Reflections and Transparency, 2000.
[7] T. Xue et al. A Computational Approach for Obstruction-Free Photography, 2015.
[8] T. Xue. Technical Report for Obstruction-Free Photography, 2015.
[9] C. Sidney Burrus. Iteratively Reweighted Least Squares. OpenStax-CNX module: m45285.
[10] S. Boyd and L. Vandenberghe. Convex Optimization, Seventh Printing with Corrections, 2009. Cambridge University Press.