Comp Photography
Final Project

Sudeep Gupta

Fall 2017

sudeepgupta412@gmail.com

Team Mate: Venkatasatyanarayana kamisetti

Reflection Removal

Main goal of this team project is separating the reflection and transmission layers from images taken through window or glass. The main algorithm, is to model the ghosted reflection using a double-impulse convolution kernel, and automatically estimate the spatial separation and relative attenuation of the ghosted reflection components and using this to separate reflection and transmission layers.

The Goal of Your Project

Original project scope:

The project deals with separating reflection and transmission layers from the images taken through glass or window. In this project we model the ghosted reflection using double impulse convolution kernel and automatically estimate the spatial separation and relative attenuation of the ghosted reflection components. The code requires only a single input image. We demonstrate this using real world inputs as well as synthetic inputs.

What motivated you to do this project?

Many times we have experienced this situation where we were standing at a window and wanted to capture a moment in our camera, but couldn’t do it perfectly because reflection from the glass would come in our image. We wanted to implement a feature where we could remove the reflection and have the desired image. This motivated us for this project.

Scope Changes

- Did you run into issues that required you to change project scope from your proposal?

  No, we did not ran into issues that would require us to change the scope of the project.

  But we couldn’t run this on large images and multiple inputs because it was taking close to 20 hours just to run image of size 400x540 pixels.

Showcase

Input

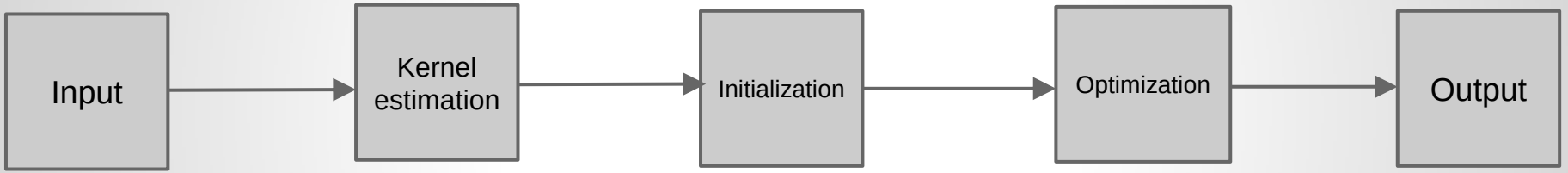
After Reflection Removal

Showcase

Reflection

Transmission

Project Pipeline



Project Pipeline

- **Kernel estimation** :

Ghosting convolution kernel is parameterized by spatial shift vector d_k [dx,dy] and attenuation factor c_k . We estimate both d_k and c_k in kernel_est() and est_attenuation() function and used them to get kernel k in get_k() function. The spatial shift vector d_k [dx,dy] is estimated by taking the 2-D autocorrelation on the Laplacian of the input image. The shifted copies of the reflection layer create a local maximum at d_k [dx,dy] on the autocorrelation map. To detect d_k [dx,dy] , we apply a local maximum filter in each 5-by-5 neighborhood.

The attenuation factor c_k is weighted sum of square roots of ratio of variances of different patches in the image.

Project Pipeline

- **Initialization :**

Our formation model for the observed image I is given in equation 3. (*all equation numbers are referenced from the paper). In this equation T is the transmission, R is the reflection, k is the ghosting kernel and n is additive i.i.d. Gaussian noise with variance σ^2 .

$$I = T + R \otimes k + n \qquad (3)$$

This leads to the formation of following cost function (given by equation 4) :

$$L(T, R) = \frac{1}{\sigma^2}\|I - T - R \otimes k\|_2^2 \qquad (4)$$

The above cost function is optimized using patch based prior based on Gaussian Mixture Models (GMM) as additional prior to regularize the inference.

$$\min_{T,R} \quad \frac{1}{\sigma^2}\|I - T - R \otimes k\|_2^2 - \sum_i \log(\mathrm{GMM}(P_i T)) \; - \sum_i \log(\mathrm{GMM}(P_i R)), \text{ s.t. } 0 \le T, R \le 1 \qquad (6)$$

Where PiT and PiR are patches on T and R components respectively.

Project Pipeline

- **Initialization :**

A good initialization is crucial in achieving better local minima. The paper suggested sparsity inducing (gradients and Laplacians filter $\{f_i\}$) based model with convex L1 prior penalty:

$$\min_{T,R} \quad \frac{1}{\sigma^2}\|I - T - R \otimes k\|_2^2 + \sum_j \|f_j \otimes T\|_1 + \sum_j \|f_j \otimes R\|_1 \qquad (8)$$

We have taken MATLAB code as reference provided on the website: *https://dilipkay.wordpress.com/reflection_ghosting* and rewrote the code in python for generation of initial estimation.

Project Pipeline

- **Optimization :** In this step, we used Gaussian Mixture models (GMM) priors to capture covariance structure pixel dependencies over patches of size 8X8 for image reconstructions. We used pre-trained zero mean GMM model with 200 mixture components of patch size 8x8. We formulated the cost function:

$$\min_{T,R,z_T,z_R} \frac{1}{\sigma^2}\|I - T - R \otimes k\|_2^2 + \frac{\beta}{2}\sum_i \left(\|P_i T - z_T^i\|^2 + \|P_i R - z_R^i\| - \sum_i \log(\mathrm{GMM}(z_T^i)) - \sum_i \log(\mathrm{GMM}(z_R^i))\right)$$

$$\text{s.t. } 0 \le T, R \le 1$$

In the above equation P_i is the linear operator that extracts i^{th} patch from T and R.
z_T^i and z_R^i are auxiliary variables for each patch $P_i T$ and $P_i R$.

We optimize the cost function using Limited Memory BFGS to obtain the output image.
In case of coloured images, this process is done separately on each channel.

Demonstration: Result Sets (1)

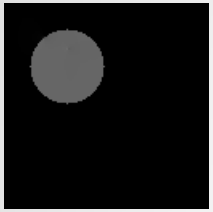
Input

Transmitted

Reflected

This is the image taken from the authors, It shows the perfect application of the algorithm.
The original image was provided in raw format and we clipped image so colors are slightly off.

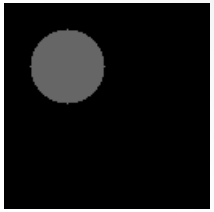
Demonstration: Result Sets (2)



Given transmission component

Given reflection component

Generated Synthetic Input



Output transmission component

Output reflection component

This is synthetically generate image using parameters dx = 15, dy = 7 and c = 0.5.
This shows best possible output using ghosting cues as demonstrated in the paper.

Demonstration: Result Sets (3)

Input

Transmitted

Reflected

This is the image taken from our own camera. This is the view from our office which shows reflection of the inside office space in the Input image.
The algorithm failed to separate out Transmitted and Reflected components. This might be due to insufficient information about ghosting cues.

Project Development

- We started this project by reading the research paper “Reflection Removal using Ghosting Cues”. The paper was not sufficient to give understanding of various concepts described in it. We had to do multiple readings to understand the paper. Also we had to refer the reference papers of “Reflection Removal using Ghosting Cues” to get the concepts.
- It took us two weeks to code the project completely. We started implementing the algorithm in python. The main issue we faced was the long duration of code run.
- We started with estimating the kernel. To get the kernel, had to estimate dk using 2-D autocorrelation. Earlier we used scipy correlate2D for autocorrelation which was taking a lot of time to compute. After a lot of search, we found a faster way to do that using fftconvolve. The paper suggested discarding local maxima but it was very difficult to narrow down on thresholds to suit the different input images. Also the suggested process failed to generate good results on images with repetitive patterns.
- In the initialization part, the details given in the paper were nearly insignificant and we had to struggle a lot to understand but all in vain. Finally, we had to refer to the MATLAB implementation provided by the authors for this to get some understanding. Even now, we are not very confident about some steps used in this part of the code. The code we used in this section is taken from the filenames irls_grad.m and grad_irls.m in the MATLAB implementation.

Project Development (cont'd)

- We have taken the pre-trained GMM model from paper - "From Learning Models of Natural Image Patches to Whole Image Restoration." The description given in the paper is clear enough to understand and start coding. But because of exponential nature of the space requirements, we faced issues with computer memory. So most of the code was written using sparse matrices.
- The running time of the code is very impractical for image of size 400x540 pixels. It took nearly 20 hours to finish processing. We had to rely on low resolution images and this became very tedious to debug the code. We started debugging using our synthetic image to avoid unwanted parameters.
- If we were to do this project again, we would implement Sparse Blind separation with motions (SPBS-M) algorithms because it gives better results compared to ghosting cues in removing reflections as mentioned in the paper "Reflection Removal Algorithms" by Matthew Hu.

Computation: Code Functional Description

Kernel_est():

```
#Laplasian Filter
I_in=I_in.astype(np.float32)
if len(I_in.shape) == 3:
    I_in = cv2.cvtColor(I_in, cv2.COLOR_RGB2GRAY)

Laplacian = np.array([[0., -1., 0], [-1., 4., -1.],[0, -1., 0]])

resp = cv2.filter2D(I_in, -1, Laplacian, borderType=cv2.BORDER_DEFAULT)
```

```
#Auto correlation and finding the maxima
auto_corr = sp.signal.fftconvolve(resp, resp[::-1, ::-1])
max_1 = ordfilt2(auto_corr, 24, 5)
max_2 = ordfilt2(auto_corr, 23, 5)

(x,y) = auto_corr.shape
auto_corr[(x/2) - 4 : (x/2) + 4, (y/2) - 4 : (y/2)+4]=0

c = np.ones(max_1.shape)
c[(max_1 - max_2)<=70] = 0
c[auto_corr != max_1] = 0
candidates = np.where(c == 1)

if candidates[0].size >=2:
    idx =  np.argmax(auto_corr[candidates])
    dy = candidates[0][idx] - x//2
    dx = candidates[1][idx] - y//2
```

```
def est_attenuation(I_in, dx, dy):
    num_features = 200

    I = (I_in * 255).astype(np.uint8)
    feat_detector = cv2.ORB(nfeatures=num_features)
    image_1_kp, image_1_desc = feat_detector.detectAndCompute(I, None)

    cns = np.zeros((num_features,2), dtype=np.uint64)
    count = 0
    for i,kp in enumerate(image_1_kp):
        cns[i,:] = kp.pt
        count = count + 1

    hw = 18
    score = np.zeros((len(cns),1))
    atten = score * 0
    w = score * 0

    #attenuation
    for i in range(0,len(cns)):

        p1 = get_patch(I, int(cns[i,1]), int(cns[i,0]), hw)
        p2 = get_patch(I, int(cns[i,1] + dx), int(cns[i,0] + dy), hw)

        if p1 is not None and p2 is not None:
            p1 = p1 - np.mean(p1)
            p2 = p2 - np.mean(p2)
            score[i] = np.sum(p1 * p2)/np.sum(p1 ** 2)**0.5/np.sum(p2 ** 2)**0.5
            atten[i] = (np.max(p2)-np.min(p2))/(np.max(p1)-np.min(p1))
            if (atten[i] < 1) and (atten[i] > 0):
                w[i] = np.exp(-1* score[i]/(2*(0.2**2)))

    c = sum(w * atten)/sum(w)

    #print c
    return c
```

Computation: Code Functional Description

Main optimization Algorithm:

Half Quadratic Splitting was used to do the optimization. This involves fixing the auxiliary variable and computing the required value and fixing this new value to get optimized value of the auxiliary variable.

```
for i in range(25):
    print 'Optimizine %d iter...\n'%i
    x0 = np.hstack([I_t_i.ravel(order='F'),I_r_i.ravel(order='F')]).reshape(-1,1)
    sum_piT_zi = merge_two_patches(est_t, est_r, h, w, psize);
    sum_zi_2 = np.linalg.norm(est_t.ravel(order='F'))**2 + np.linalg.norm(est_r.ravel(order='F'))**2;
    z = (lbda * A.T.tocsr().dot(img.ravel(order='F'))).reshape(-1,1) +  (beta * sum_piT_zi);
    def calc_func_value_and_gradient(x,*args):

        f = lbda * np.linalg.norm(A.dot(x) - img.ravel(order='F'))**2
            + beta*(sum(x.reshape(-1,1)*mask*x.reshape(-1,1)
             - 2 * x.reshape(-1,1)* sum_piT_zi.ravel(order='F').reshape(-1,1)) + sum_zi_2)
        g = 2*(lbda * (A.T.tocsr().dot(A.dot(x))).reshape(-1,1) + beta*(mask*x.reshape(-1,1)) - z)
        return f,g

    (out,f,d) = sp.optimize.fmin_l_bfgs_b(calc_func_value_and_gradient,
                    x0,args=(A,mask),bounds=[(0,1) for i in range(h*w*2)],m=50,factr=1e4,pgtol=1e-8)

    out = out.reshape(h,w,2,order='F')
    I_t_i = out[:,:,0]
    I_r_i = out[:,:,1]

    #Restore patches using the prior
    est_t = im2patches(I_t_i, psize);
    est_r = im2patches(I_r_i, psize);
    noiseSD=(1/beta)**0.5;
```

#f = optimization function

#g = derivative calculation

#(out,f,d) = LBFGS optimization

Computation: Code Functional Description

Libraries and Models used:

- L-BFGS : https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html
- ORB features : https://docs.opencv.org/3.3.0/db/d95/classcv_1_1ORB.html
- GMM models : https://people.csail.mit.edu/danielzoran/EPLLICCVCameraReady.pdf
- IRLS code : https://dilipkay.wordpress.com/reflection_ghosting

Resources

- Reflection Removal using Ghosting Cues:
https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Shih_Reflection_Removal_Using_2015_CVPR_paper.pdf

- From Learning Models of Natural Image Patches to Whole Image Restoration:
https://people.csail.mit.edu/danielzoran/EPLLICCVCameraReady.pdf
- Reflection Removal Algorithms: http://stanford.edu/class/ee367/Winter2016/Hu_Report.pdf
- For vectorization of python code suggestions were taken from stackoverflow.

Teamwork

- If you worked in a Team, describe your original division of labor, and how that worked out.

With our original division of labor, I was supposed to work on separation model while he was to focus on ghosting model. But during execution, our roles got exchanged due to our interests.

- Do you feel that the actual division of labor to develop your project, code, and report was equitable?
  - For most part of the project we worked together to develop the code.
  - Both of us equally participated in the project and helped each other debug the code and fix issues.
  - His knowledge and experience helped me go through this smoothly.

Appendix: Your Code

Code Language: Python

List of code files:

- reflection_removal.py
- grad_irls.py
- kernel_est.py

Credits or Thanks

- We would like to thank the authors of the original paper YiChang Shih, Dilip Krishnan, Fredo Durand and William T. Freeman
- We would like to thank all the contributors of stackoverflow which helped us in vectorising our code.