# Reflection Removal

## Motivation

Reflection produced by glass windows is a bothering issue for photographers. Sometimes we only want to keep the scene behind or in front of the window, but the glass window produce a reflection of unimportant things, which is called undesired reflections. Reflection removal is a solved problem in the past research. In this report, we implemented and discussed three approaches according to three papers.

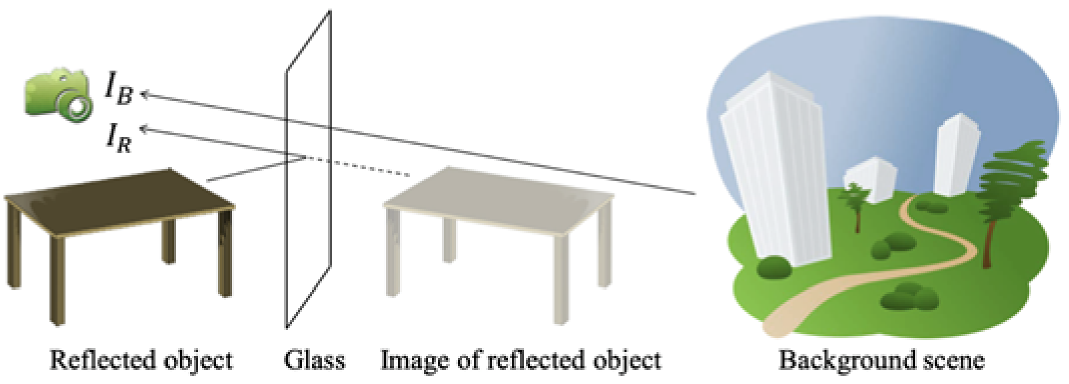
## Problem Definition

The figure shows the image formation model, in which the camera is taking a picture through the glass. The resulting image $I \in R^n$ will contain two layers, one is the reflected object, denoted by $I_R \in R^n$, the other is the background scene, denoted by $I_B \in R^n$. Our goal is to remove the reflected object part in the image.

Therefore, we have:

$$I = I_R + I_B$$

We can remove the reflected object if we get $I_R$ and $I_B$ seperatedly.



## Algorithm and Experiment result

### Method I. Smoothness Approach

This method is based on the following paper:

- Y. Li and M. S. Brown, "Single Image Layer Separation Using Relative Smoothness," 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 2752-2759, doi: 10.1109/CVPR.2014.346.
  Paper and source code is accessed on author's website.

#### *Formulation*

An image with reflection can be expressed as

$$I = L_1 + L_2 = L_B + L_R * h$$

$I$ is the input image, $L_1$ 、 $L_2$ is two seperated layer, $L_R$ is the Reflection layer and $L_B$ is the Background layer. Assume that desired layer is more in focus than the reflection layer, which is blurred, so $L_R$ is convolved with a Gaussian filter h.

This paper build a model based on supposing $L_2$ is smoother than $L_1$ and therefore large gradients are more likely to belong to $L_1$.On the other hand, $L_2$ represents the blurred reflection layer. To recognize smooth and sharp pattern, we use three derivative filter

$$f_1 = \begin{bmatrix} -1 & 1 \end{bmatrix} \quad f_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad f_3 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$f_1$,$f_2$ extract large gradient in image, and $f_3$ is a Laplacian kernel, which can find smoother parts. $L_1$ and $L_2$ is obtained by maximining the convolution $L_1 * f_1 + L_1 * f_2$ and $L_2 * f_3$. This method could get good results in 3~5 iterations.

### *Algorithm*

```
**Algorithm** Layer Separation using Relative Smoothness
**Input**: image $I$; smoothness weight $\lambda$; initial $\beta$; iterations number $i_{max}$; increasing rate $\eta$
**Initialization**: $L_1 \leftarrow I$; $\beta \leftarrow \beta_0$; $i \leftarrow 0$
    **while** $i < i_{max}$ **do**
        Update $g_i^j$ (*step 1)
        Compute $L_1$ (*step 2)
        Normalize $L_1$ (*step 3)
        $\beta = \eta * \beta$ (*step 4)
        $i \leftarrow i + 1$
    **end while**
    $L_2 = I - L_1$
    Intensify $L_1,\ L_2$
**Output:** The estimation of two layers $L_1$ and $L_2$
```

- Symbol Marks : Subscript or $_i$ represents the pixel grid index. Superscript or $^j$ represents the filter index.

#### *Step 1*

Current $L_1$ is convolved with filter $f_1$ and $f_2$ individually. An auxiliary variabes $g_i^j$ is used here. Each pixel convolution value would be saved into $g_i^j$ if the value exceeds the threshold $\frac{1}{\beta}$, otherwise $g_i^j$ is set to 0.

$$g_i^j = \begin{cases} F_i^j L_i, & (F_i^j L_i)^2 > \frac{1}{\beta} \\ 0 & otherwise. \end{cases}$$

The purpose of this step is to find sharp components in the image.

#### *Step 2*

Calculate new $L_1$ from current $g_i^j$. Based on the cost function of maximizing Convolution is:

$$\min_{L_1, g^j} \sum_i \left( \sum_{j=1,2} (\beta(F_i^j L_1 - g_i^j)^2 + \rho(g_i^j)) + \lambda(F_i^3 L_1 - F_i^3 I) \right)$$

$$\rho(x) = min\{\frac{x^2}{k}, 1\}$$

Rewrite the above cost function and apply 2D FFT, $L_1$ is calculted by the following formula:

$$L_1 = \mathcal{F}^{-1}(A)$$

$$A = \frac{\beta \sum_j (\mathcal{F}(F^j) * \mathcal{F}(g^j)) + \lambda \mathcal{F}(F^3) * \mathcal{F}(F^3)\mathcal{F}(I)}{\beta \sum_j (\mathcal{F}(F^j) * \mathcal{F}(F^j)) + \lambda \mathcal{F}(F^3) * \mathcal{F}(F^3) + \tau}$$

In this formula, only $g_i^j$ is not constant. Thus, Except $\mathcal{F}(g^j)$ is calculated each iteration, other 2D FFT is done at the initialization step.

The numerator in the last formula means superposition of Original image $I$ and Sharper Layer obtained by $g_i^j$. The weight is controlled by *smoothness weight* $\lambda$, this value suggested by author is 50. $\tau$ is a small number(1e-6) to avoid devide by zero.

***Step 3***

Normalization of $L_1$ makes sure that each pixel value is in the range [0, 255] (or stricter $[0, I]$). After FFT & IFFT at step 2, $L_1$ values might be amplified. Without normaliztion, different color channel might perform different shift, and output figure color would be strange. Object function:

$$\min_t \sum_i m_i((L_1)_i + t - lb_i)^2 + \sum_i n_i((L_1)_i + t - ub_i)^2$$

$m_i(x)$、$n_i(x)$ is indicative function. $m_i(x)$、$n_i(x)$ is equal to 1 if x < 0, otherwise $m_i(x)$、$n_i(x) = 1$. Pixel values is bounded by $[lb_i, ub_i]$

Each color channel is normalized seperatedly. To implement this objective function, variable $t$ is added to all pixels, which means a linear shift. Solve the objective function with gradient descent to find $t$.

$$L_{new} = L_1 + t$$

$$dt = 2 * \sum_i (L_{new})_i * m_i((L_{new})_i - lb_i) + (L_{new})_i * m_i(ub_i - (L_{new})_i$$

Iteration stops if $dt$ is small enough or exceed 500 iterations. Ater linear $t$ shift, if there is any pixel value still out of the bound, The pixel values is clipped to $lb_i$ or $ub_i$

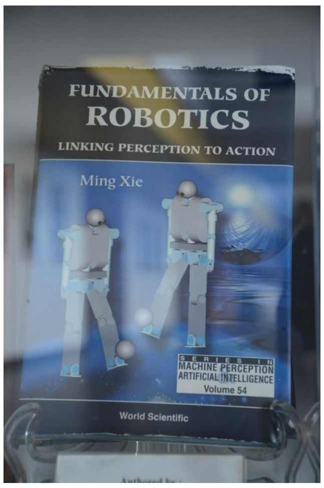
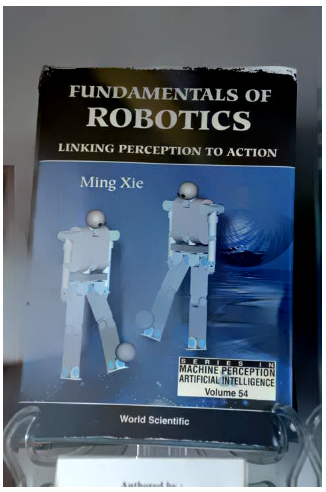
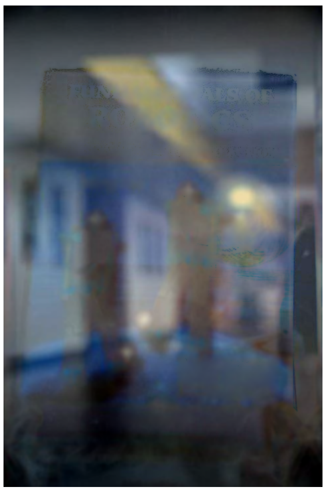
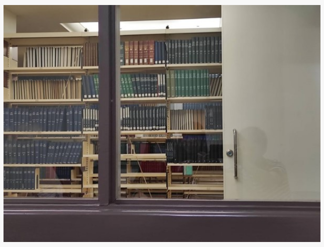
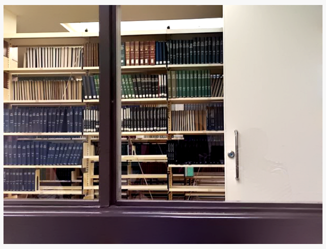
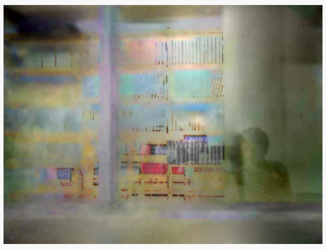
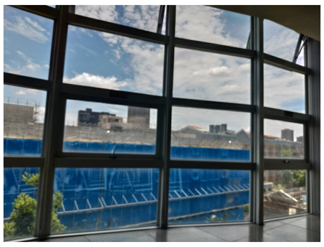
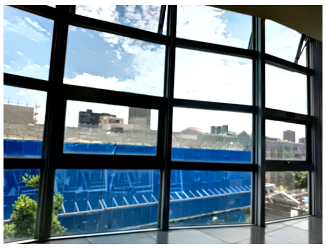
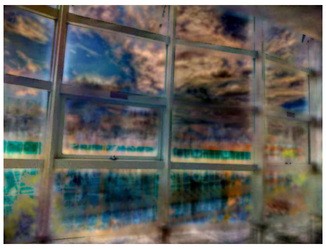
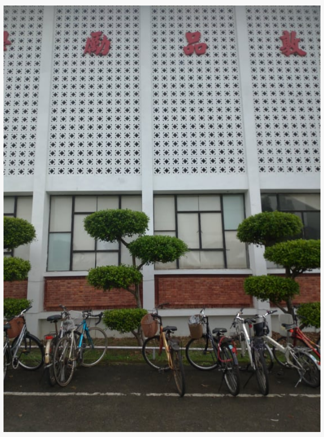
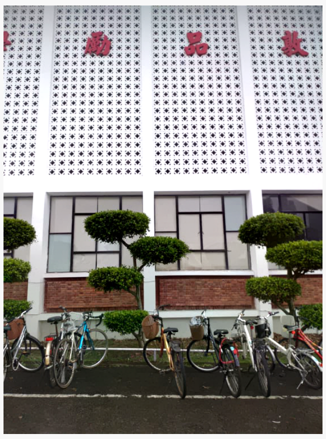
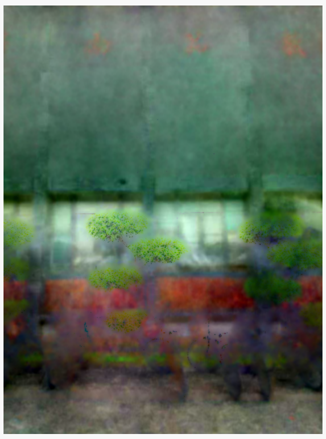
***Step 4***

$\beta$ increased every iteration to make method converge faster. $\beta$ and $rate$ are both decided by user. Paper suggestion is $\beta_0 = 20$, $rate = 2$.

***Output***

Reflection layer $I_2$ is obtained by original input image $I$ minus $L_1$. Because both seperatered layer values are smaller than original one, intensify layers by multiply a value such that maximum in both layer are $I_{max}$.

## *Result*

| | Original | Backgroud | Reflection |
|---|---|---|---|
| 1 |  FUNDAMENTALS OF<br>ROBOTICS<br>LINKING PERCEPTION TO ACTION<br>Ming Xie<br>SERIES IN MACHINE PERCEPTION ARTIFICIAL INTELLIGENCE<br>Volume 54<br>World Scientific |  FUNDAMENTALS OF<br>ROBOTICS<br>LINKING PERCEPTION TO ACTION<br>Ming Xie<br>SERIES IN MACHINE PERCEPTION ARTIFICIAL INTELLIGENCE<br>Volume 54<br>World Scientific |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |

- Discussion
  1. **Sample0.jpg** : (Image from paper.) This Input image has obvious reflection of aisle. Most of the white walls are removed. Some yellow light is left, perhaps the yellow is siginificant, thus cannot remove it totally.
  2. **Sample1.jpg** : (Our photo.) Reflection in front of the door is removed and a little outline is left. Reflection in front of the bookshelves is not well recognized in reflection layer. Complicated background might be an issue to deal with.
  3. **Sample2.jpg** : (Our photo.) In this photo, undesired reflection is the desks and chairs in front of the blue background. However, the reflection layer is composed of Clouds and

building. Take a close look at this photo, the problem is that the photo is out of focus, so most of the components are blurred. Here bring up a problem : Smoothness approach method assume that reflection is blurred, this method can only apply to the photo is well focused. In addition, Clouds are flattened.

4. **Sample3.jpg** : (Our photo.) Tree reflection is well recognized in the reflection layer, but it is not totally removed in the background layer. Adjust the *smoothness weight* $\lambda$ might be a solution to this kind of problem.

## Method II. Motion Approach

This method is based on the following paper:

- XUE, Tianfan, et al. A computational approach for obstruction-free photography. ACM Transactions on Graphics (TOG), 2015, 34.4: 1-11.

### *Formulation*

In this method, the user should take a short image sequence rather than a single image. Because the background and the reflected object have different distance to the camera, we can separate them by analyzing the relative motion in the image sequence.

Due to the additional information of "motion", we modify the problem definition here:
$I^t = W(V_R^t)I_R + W(V_B^t)I_B$

- $I^t$ is the image at frame t.
- $V_B^t$ denotes the motion field for the background layer from reference frame $t_0$ to the frame $t$ . Similarly, $V_R^t$ is the motion field for the reflected-object layer.
- $W\left(V_B^t\right) \in R^{n \times n}$ is a warping matrix such that $W\left(V_B^t\right) I_B$ is the warping background component $I_B$ according to the motion field $V_B^t$

The goal is to recover the background layer $I_B$ and the reflected-object image $I_R$ for the reference frame $I^{t_0}$, from an input image sequence $\{I^t\}$ without knowing the motion fields $V_R^t$ and $V_B^t$.

We define an optimization problem with the objective function:

$$\underset{I_R, I_B, \{V_R^t\}, \{V_B^t\}}{min} \sum_t || I^t - W\left(V_R^t\right) I_R + W\left(V_B^t\right) I_B||$$

### *Algorithm*

The algorithm consists of two steps, initialization and iterative optimization.

- **initialization:**

1. Edge map
   First, extract the edge map for each frame using the Canny edge detector.
2. Sparse motion field
   Calculate the motion vectors on extracted edge pixels by solving a discrete Markov random field.

$$\min_V \sum_{x \in Edge\left(I^1\right)} NCC\left(I^1\left(x\right), I^2\left(x + V\left(x\right)\right)\right) + \sum_{x, x' \in Edge\left(I^1\right)\ and\ \left(x, x'\right) \in N} S\left(V\left(x\right), V\left(x'\right)\right)$$

$I^1$ and $I^2$ are two neighboring input images, $V$ is the motion field from image $I^1$ to $I^2$ that we want to estimate, $Edge\left(I^1\right)$ is the set of edge pixels in image $I^1$, and $N$ is the 4-connected pixel neighborhood.
The first term with the normalized cross correlation (NCC) between these two patches describes how well the patch located at position x in image $I^1$ matches the patch located at $x + V(x)$ in image $I^2$. The second term is the smoothness term that enforces neighboring edge pixels to have similar motion.

3. Classification
   After obtaining sparse motion field $V$ ,one should separate it into two layers by fitting two perspective transforms to the edge motion and assign each pixel to either the background layer or the object-reflected layer using RANSAC.
4. Dense motion field
   Compute the dense motion fields for both layers of sparse motion fields using visual surface interpolation.
5. warping to get initial decomposition

   First warp all the frames to the reference frame according to the background motion. We take the initial estimation of the background image to be the minimum intensity across the warped frames. (Because when warping according to the background, the background pattern should be roughly-aligned, while the reflected-object is moving.)

- **optimization**
  1. Use an alternating gradient descent method to solve the objective function. That is, alternate between the decomposition and motion estimation until convergence.
     Decomposition step: fix motion fields $V_R^t$ and $V_B^t$ , solve for $I_R$ and $I_B$

     Motion estimation step: fix $I_R$ and $I_B$,solve for the motion fields $V_R^t$ and $V_B^t$.
  2. Pseudo code:

```
Algorithm Motion Approach
Input: sequence of images $\{I^t\}_t$; initial guess of $I_R, I_B, \{V_R^t\}$ and $\{V_B^t\}$
  for $Scale\ s = 1$ to $n_s$ do
      $\{\hat{I}^t\} \leftarrow$ downsampling input image sequence $\{I^t\}$ to scale $s$
      $I_R, I_B, \{V_R^t\}, \{V_B^t\} \leftarrow$ downsampling/upsample $I_O, I_B$
      $\{V_R^t\}$, and $\{V_B^t\}$ to scale $s$
      for $i = 1$ to $n_s$ do
          $I_R, I_B \leftarrow Decompose(\{\hat{I}^t\}, \{V_R^t\}, \{V_B^t\})$
          $\{V_R^t\}, \{V_B^t\} \leftarrow EstimateMotion(\{\hat{I}^t\}, I_R, I_B)$
      end for
  end while
Output: $I_R, I_B, \{V_R^t\}$ and $\{V_B^t\}$
```

***Implementation***

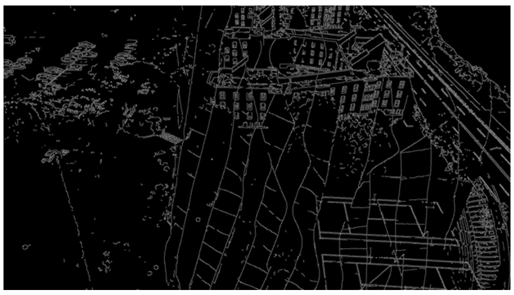
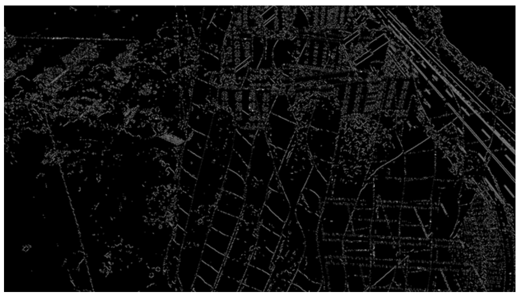
To implement the above algorithm, I have modified some of the algorithm. Below are the step-by-step introduction of my implementation

1. Input sequence:
   Below 6 frames are the continuous sequence images taken through the window by a person wearing plaid shirt. One can see the reflected image of plaid shirt and outside scenery.

|  |  |  |
|---|---|---|
| | | |
| | | |

Our goal is to remove the reflected image.

2. Edge map
   Below are the results from `cv2.Canny()` and my canny edge detection implementation from hw2.

| **opencv canny edge** | **my canny edge implementation** |
|---|---|
|  |  |

   I found that OpenCV's performance is much better than mine, which have lot of uncontinuous segments. However, this step is fundamental because all the following steps depend on the information of edge map. Therefore, I get rid of my own implementaiton and choose the OpenCV's.

3. Extract sparse motion field
   Instead of using the original solution of solving Markov random field, I use “Lucas-Kanade Optical Flow algorithm” to calculate the motion field.

   The Lucas–Kanade method assumes that the displacement of the image contents between two nearby image frames is small and approximately constant within a neighborhood of the point p. Thus, the optical flow equation can be assumed to hold for all pixels within a window(assume $5 \times 5$) centered at p. Namely, the local image flow (velocity) vector $(u, v)$ must satisfy:

$$\begin{aligned} I_x(p_1)u + I_y(p_1)v &= -I_t(p_1) \\ I_x(p_2)u + I_y(p_2)v &= -I_t(p_2) \\ &\vdots \\ I_x(p_{25})u + I_y(p_{25})v &= -I_t(p_{25}) \end{aligned}$$

- $p_1, p_2, ... p_{25}$ are the pixels inside the window
- $I_x(p_i) = \frac{\partial I}{\partial x}$is the partial derivatives of the image I with respect to position x
- $I_y(p_i) = \frac{\partial I}{\partial y}$is the partial derivatives of the image I with respect to position y
- $I_t(p_i) = \frac{\partial I}{\partial t}$is the partial derivatives of the image I with respect to time t (frame)

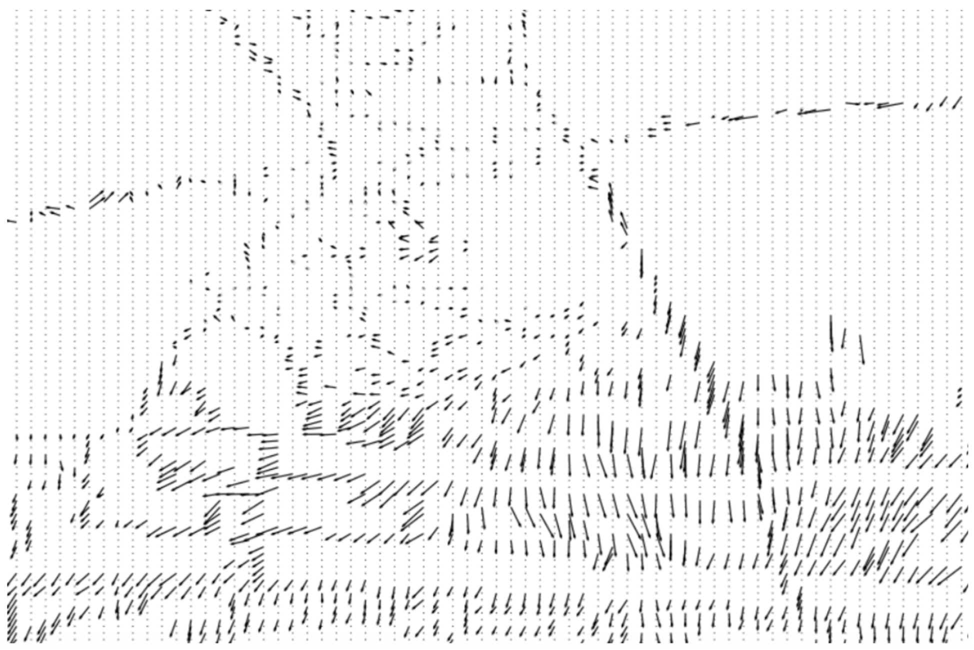
   Write the equations in matrix form $Av = b$, we have:

$$A = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \quad b = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ \vdots \\ -I_t(p_{25}) \end{bmatrix}$$

We should solve the equation by least squares approximation.
We know that $\hat{x} = argmin_x ||Ax - b||^2$ is equivalent to solving $A^\top A\hat{x} = A^\top b$. That is, $\hat{x} = (A^\top A)^{-1} A^\top b$.

To obtain $I_x(p_i), I_y(p_i), I_t(p_i)$, I use the pixel domain convolution by $\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \otimes I_t$, $\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \otimes I_t$, and $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \otimes I_t - \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix} \otimes I_{t-1}$

I draw some vector to visualize result of this step and choose the best window size. It seemed to work well at window_size = image_size * 0.01. Below are a clip from the result. However, at the edge of the image, some pixels are out of the image boundary at next frame, causing the error in calculation. But this is o.k. because step 6 of warping will eliminate those vector automatically for out of boundary mapping.



4. Classfication

   In this step, we should separate sparse motion field V into two layers. In the original solution in the paper, we should do this by fitting two perspective transforms to the edge motion and assign each pixel to either the background layer or the object-reflected layer using RANSAC. However, instead of writing a RANSAC model, I take advantage of OpenCV library function call `cv2.findHomography(srcPoints=source_points, dstPoints=target_points, method=cv2.RANSAC, ransacReprojThreshold=0.1)` to meet the requirement.

   The function finds and returns the perspective transformation H between the source and the destination planes: $S_i \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \quad H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$

   So that the back-projection error

$$\sum_i (x'_i - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}})^2 + (y'_i - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}})^2$$

is minimized.

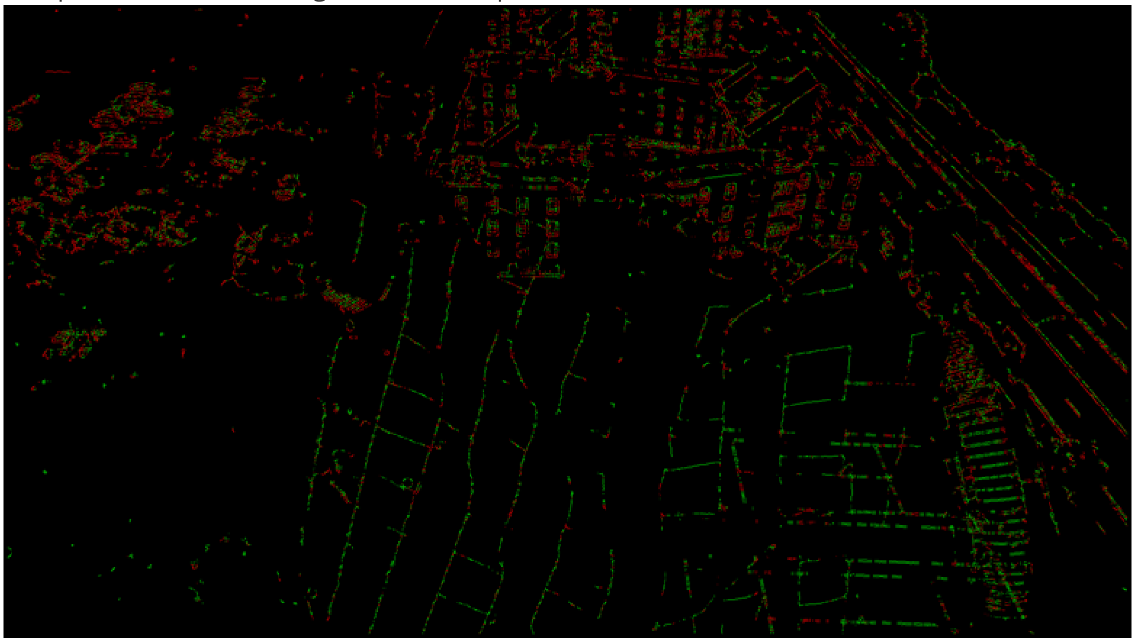
If not all of the point pairs ($srcPoints_i, dstPoints_i$) fit the rigid perspective transformation (that is, there are some outliers), this initial estimate will be poor. Therefore, the function use some methods (RANSAC, LmeDS or RHO) to try many different random subsets of the corresponding point pairs (of four pairs each, collinear pairs are discarded), estimate the homography matrix using this subset and a simple least-squares algorithm, and then compute the quality of the computed homography (which is the number of inliers for RANSAC). The best subset is then used to produce the initial estimate of the homography matrix and the mask of inliers/outliers.

To use cv2.findHomography() to separate background motion field and reflection motion field, I treat the points in edge map as first planes, the new points moved according to their corresponding motion vectors as the second planes. The function will find the perspective transformation between two points and use RANSAC to consider only inlier points. The output will not only gives the transformation matrix, but also a mask to tell which of those points are inliers. By the mask, I can separate the original points into inlier points and outlier points.

To separate the original sparse motion points into background motion points and reflection motion points, I call the function twice:

- (1) input the original motion points, treat the output inlier points as background motion points. ( I assume that background motion is the more dominant.)
- (2) input the oulier points from the output of the first call, treat the output inlier points as reflection motion points.

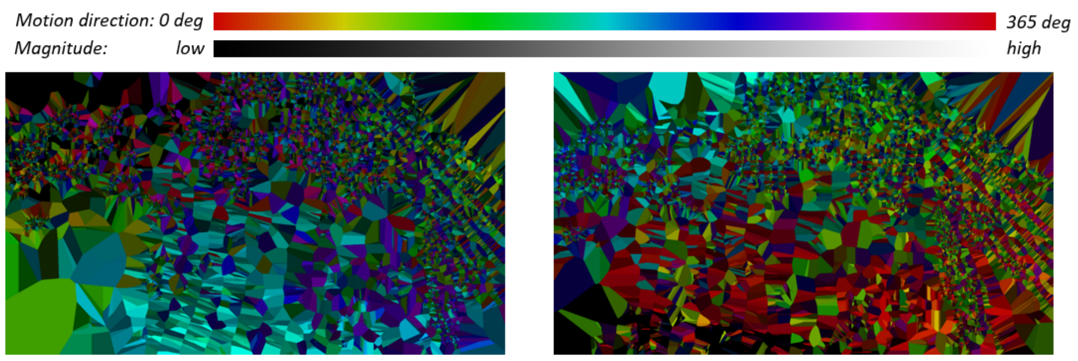
Below is a visulization of the seperation of motion field.
Red points stand for background; Green points stand for reflection.



One can observe that most of the background points and reflection points were separeted successfully. However, some points weren't seperated very well at the region that background feature and reflection feature overlapping. I think this is due to the assumption of identical velocity in the small windows in calculation of Lucas-Kanada optical flow.

5. Dense motion field

To get the dense flow fields from sparse edge flows for both background and reflection motion, I use `scipy.interpolate.griddata()` to interpolate for both y-axis motion vector and x-axis motion vector.
Below is the visualization of the result. I plot the motion direction with color and motion magnitude as brightness.



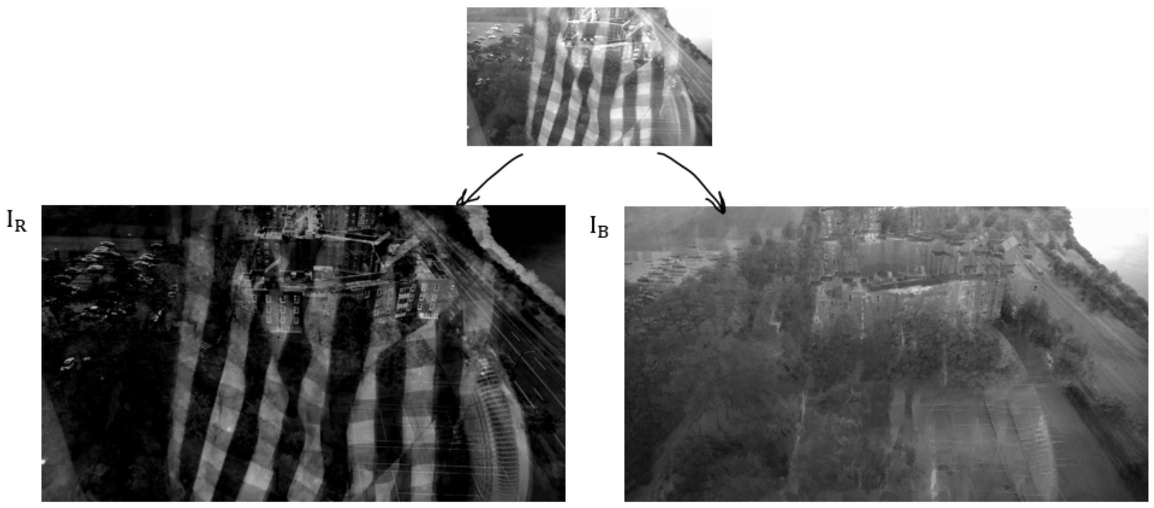
*dense motion field of reflection* *dense motion field of background*

One can observe that the major difference in the two visualized result are the motion direction.

6. Warping

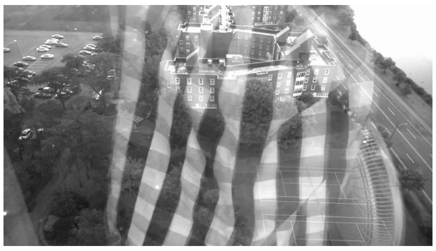
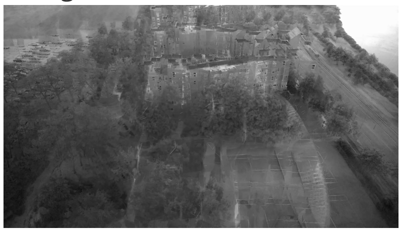
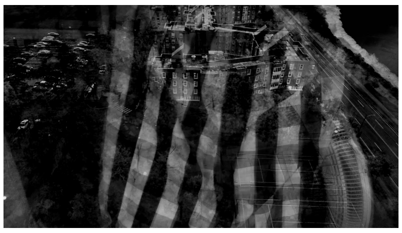
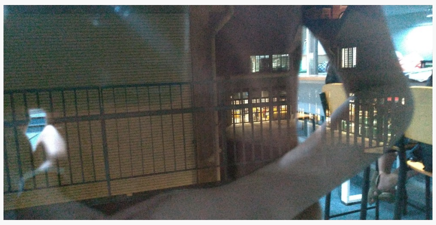
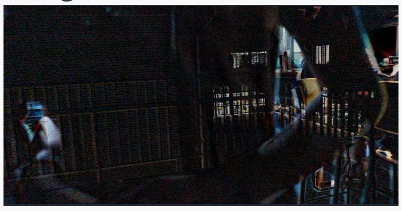
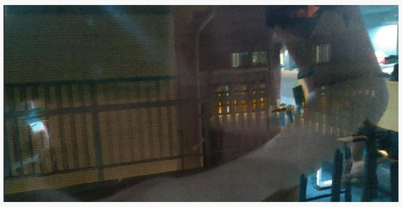
   Warping to the reference frame using the dense flow field. We take the initial estimation of the background image to be the minimum intensity across the warped frames.

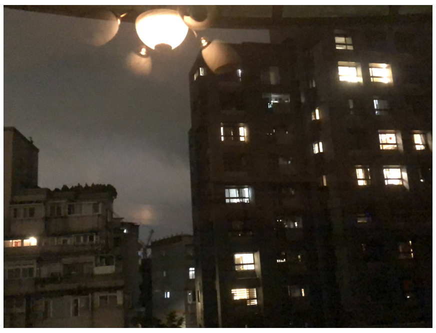
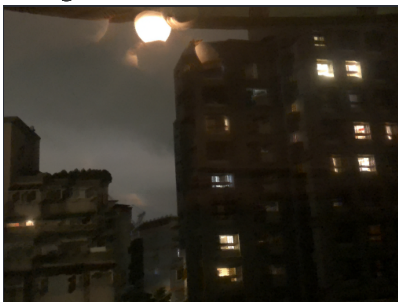
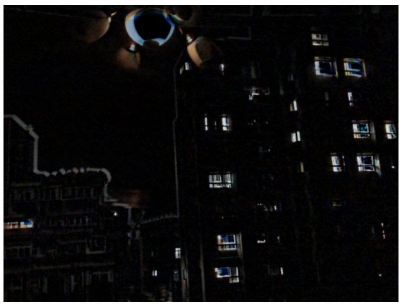
7. Result



The reflected image and background image are seperated well, except some contour of the house. Maybe this can be optimized by solving objective function using gradient descent method. However, I eliminate that step in my term project due to its difficulty of implementation.

**Test sample and the result:**

| **Input** | **Output background / reflection** | **comment** |
|---|---|---|
|  source: the author | background  reflection  | The result is good because the black-white plaid shirt acts as a good edge information. |
|  source: 20210504 at multi-purpose classroom building | background  reflection  | Most of the light outside of the windows are classified correctly. However, because the unfocused reflection image is too blurred to act correctly as an edge. Therefore, some pixel near the arm isn't classified correctly. |

| Input | Output background / reflection | comment |
|---|---|---|
|  source: 20210605 outside of my house | background  reflection  | The result isn't very well because the movement of sequence is too subtle to cover every "background pixel" that is blocked by the lightbulb. (the lightbulb becomes "thinner" but not disappear in the background image, which is limited to the small movement.) The pixels inside of the lightbulb isn't classified correctly as well because that area lack of edges information |

## Method III. User-assisted Separation of Reflections with Sparse Prior

This is based on the following two papers:

- One was published in the *European Conference on Computer Vision (ECCV), May 2004*. User Assisted Separation of Reflections from a Single Image Using a Sparsity Prior. The codes, images and some supplementary files that I implemented was modified from the enclosure of this paper. In the following this version is referred to as *ECCV2004*.
- The author published another version with similar content was published in *IEEE Trans. Pattern Analysis and Machine Intelligence, Sep 2007*. User Assisted Separation of Reflections from a Single Image Using a Sparsity Prior. Unfortunately there is no enclosure package with this paper. In the following this version is referred to as *IEEE2007*.

These two papers and packages can be derived from the author's home page: Anat Levin.

## *Algorithm*

The method is based on the fact that the gradient of a natural image is sparse, in the sense that the distribution peak at zero and have heavy tails, so we can model the distribution by some sparse prior. In the following we will use the sum of two Laplace distribtions to do this.

Assume we are given an input image $I$ with two sets of image locations $S_R$, $S_B$, such that the gradients in location $S_R$ belong to one layer and gradients in location $S_B$ belong to the other layer. We then wish to find the two layers $I_R$, $I_B$ such that:

1. Sum of two layers form the input image $I = I_R + I_B$.
2. Gradients of $I_R$ at locations in $S_R$ agree with the gradients of $I$, and similalrly the gradients of $I_B$ at locations in $S_B$ agree with the gradients of $I$.

We assume the reflection and the background are independent , and derivatives of all pixels are independent. Under these constraints, we aim to maximize to probability of the layers:

$$\mathbb{P}(I_R, I_B) = \mathbb{P}(I_R)P(I_B) = \prod_{i,k} \mathbb{P}(f_{j,k} \cdot I_R)\mathbb{P}(f_{j,k} \cdot I_B)$$

where $f_{i,k} \cdot I$ is the convolution of the k'th filter with image $I$ centered at pixel $i$, and the filters include the first and second order derivatives, as its selection is of our interests.

As mentioned, we use the assumption of the sparse prior, mixture of Laplacian distribution, and take logarithm to approximate $\mathbb{P}(I)$ by

$$\log \mathbb{P}(I) = \sum_{i,k} \rho(f_{i,k} \cdot I), \quad \rho(x) = \log(\frac{\pi_1}{2s_1} e^{-\frac{|x|}{s_1}} + \frac{\pi_2}{2s_2} e^{-\frac{|x|}{s_2}})$$

Note that the probability function $\mathbb{P}(x)$ is a mixture of density function of Laplacian distributions. The parameters are set to be $s_1 = 0.01, s_2 = 0.05, \pi_1 = 0.4, \pi_2 = 0.6$, which are learned to perform well in experiments.

This is equivalent to minimize the function

$$J(I_R, I_B) = \sum_{i,k} \rho(f_{i,k} \cdot I_R) + \rho(f_{i,k} \cdot I_B)$$

constrained by the above two requirements.

To solve this, turn $I_B = I - I_R$ and write it into an unconstrained minimization; that is, we hope to find the layer $I_R$ that minimizes the cost

$$\begin{aligned} J_2(I_R) = &\sum_{i,k} \rho(f_{i,k} \cdot I_R) + \rho(f_{i,k} \cdot (I - I_R)) \\ &+ \lambda \sum_{i \in S_R, k} \rho(f_{i,k} \cdot I_R - f_{i,k} \cdot I) \\ &+ \lambda \sum_{i \in S_B, k} \rho(f_{i,k} \cdot I_R) \end{aligned}$$

For simplicity we can write this into the form

$$J_3(v) = \sum_j \rho(A_j v - b_j)$$

where $A_j$ is the j$^{\text{th}}$ row of the matrix $A$ corresponding to all filters $f$ we use, $v$ is the vector of image $I_R$ written in one-dimension, and $b = (b_j)^T$ is some one-dimension vector representing the deriviatives.

The papers *ECCV2004* and *IEEE2007* have their main content almost the same. The significant difference is that they use different ways to minimize the cost. *ECCV2004* uses the method of *Expectation-maximization(EM)* to solve the linear programming problem, while *IEEE2007* uses the method of *Iterative reweighted least squares optimization(IRLS)*.

## *Expectation-maximization(EM)*

This is done by iteratively find expectation (*E-step*) and perform minimization(*M-step*)

- *E-step*: Calculate two weights $w_1, w_2$ for each j$^{th}$ row of $A$ such that

$$w_1(j) \propto \frac{\pi_1}{s_1} e^{\frac{A_j v - b_j}{s_1}}, \quad w_2(j) \propto \frac{\pi_2}{s_2} e^{\frac{A_j v - b_j}{s_2}}$$

The proportion constant is set to make $w_1(j) + w_2(j) = 1, \quad \forall j$.

- *M-step*: Perform minimization by assigning

$$v^* \leftarrow arg \min_v ||DAv - Db||$$

  where $D$ is the diagonal matrix with elements $D_{j,j} = w_1(j)/s_1 + w_2(j)/s_2$

Iteratively perform *E-step* and *M-step* should minimize the cost function. In *ECCV2004* the author suggest about 15 iterations.

## *Iterative reweighted least squares optimization(IRLS)*

The IRLS algorithm proceeds as below:

- *Initialization*: Set initial state $\psi_j^0 = 1$ where $j$ is indexed from 1 to size of image.
- *Repeat*:
  1. Let $\bar{A} = \sum_j A_j^T \psi_j^{t-1} A_j$ and $\bar{b} = \sum_j A_j^T \psi_j^{t-1} v_j$. Let $x^t$ to be the solution of $\bar{A}x = \bar{b}$.
  2. Set $u_j = A_j x^t - b_j$, and $\psi_j^t(u_j) = \frac{1}{|u_j|} \frac{d\rho(u_j)}{d_u}$. Recall that $\rho(x) = \log(\frac{\pi_1}{2s_1} e^{-\frac{|x|}{s_1}} + \frac{\pi_2}{2s_2} e^{-\frac{|x|}{s_2}})$ . Compute its derivative and plug in; that is

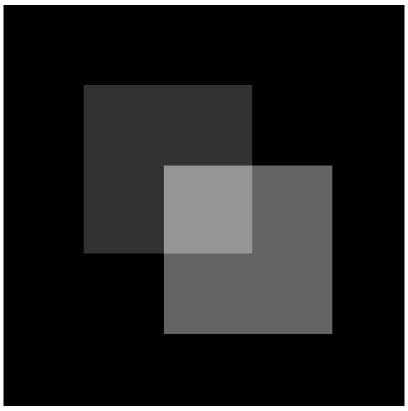
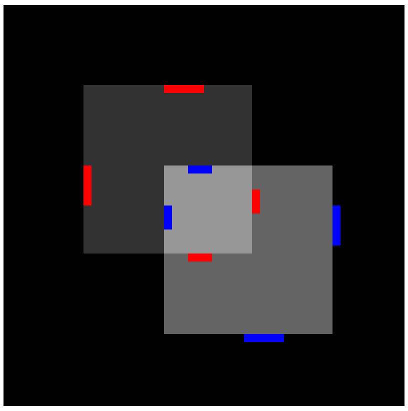
$$\psi_j^t(u_j) = \frac{1}{|u_j|} \frac{\frac{\pi_1}{2s_1^2} e^{-\frac{|u_j|}{s_1}} + \frac{\pi_2}{2s_2^2} e^{-\frac{|u_j|}{s_2}}}{\frac{\pi_1}{2s_1} e^{-\frac{|u_j|}{s_1}} + \frac{\pi_2}{2s_2} e^{-\frac{|u_j|}{s_2}}}$$

The number of repeat times is fixed to 10 that is suggested by *IEEE2007*.

## *Experiment result*

As mentioned above, the method of *EM* fail to separate the reflection. It outputs a black image and the original image. This result is derived from the given MATLAB code of the author, and the python code I which imitate from the package show the same output.

On the otherhand, the method of *IRLS* requires very large overhead of memory; e.g. $300 \times 300$ will crash on CSIE Work Station. To test this I made a test image with size $50 \times 50$.

| test image | test image with edge |
|---|---|
|  |  |

As above shows, the left is our test image, which is composed of two squares with different intensities. The right image was adds the edges we mark; while the red represents the edges in reflection and the blue represents the that in the real image.

By performing *IRLS* method with 10 iterations, the following shows the middle 9 iteration results.

Iteration 1

Iteration 2

Iteration 3

Iteration 4

Iteration 5

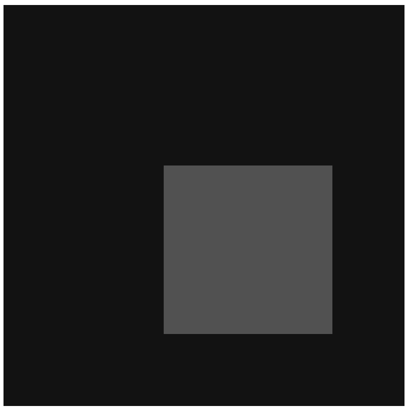
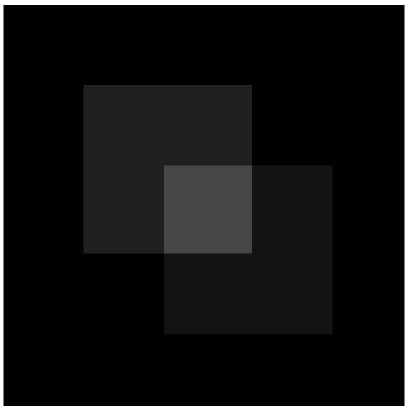
Iteration 6

Iteration 7

Iteration 8

Iteration 9

And the eventual result is shown below.

| test image | test image with edge |
|---|---|
|  |  |

We see the program can separate the reflection to some extent. It is worthy to note that the program seem not to converge as iteration time increases; as a result, this method is not really friendly.

## Conclusion

### Method I. Smoothness Approach

The first method applys to single static image. It provides good results with fast convergence and efficient computation. Performance is good while the desired layer is in focused and background is simple, on the contrary, photo with complicated background or out of focus should be improved.

### Method II. Motion Approach

The second method takes advantage of the slight movement in an image sequence, so user needs to take several images in a  sequence. Recently, many smartphones are equipped with continuous capture modes, so it isn't a problem. However, the method might fail when number of frames are too small or the image contains few edges or with many noise. Most of the time, it performs well in general condition.

### Method III. User-assisted Separation of Reflections with Sparse Prior

The third method requires the user to mark certain edges for the reflection layer and the background layer. It fails when number of marked edges is not big enough, and it causes a lot of memory overhead. To sum all, this method can do some work to some certain extent, but in most cases it is not very suitable to use this method.

## Reference

1. Y. Li and M. S. Brown, "Single Image Layer Separation Using Relative Smoothness," 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 2752-2759, doi: 10.1109/CVPR.2014.346.
2. XUE, Tianfan, et al. A computational approach for obstruction-free photography. ACM Transactions on Graphics (TOG), 2015, 34.4: 1-11.
3. LEVIN, Anat; WEISS, Yair. User assisted separation of reflections from a single image using a sparsity prior. In: European Conference on Computer Vision. Springer, Berlin, Heidelberg, 2004. p. 602-613.
4. LEVIN, Anat; WEISS, Yair. User assisted separation of reflections from a single image using a sparsity prior. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2007, 29.9: 1647-1654.