Computer Vision HW2 Read me

作者：105062509 / 羅右鈞

註：此 readme 原先在 dropbox paper 上編輯，直接至
https://paper.dropbox.com/doc/Computer-Vision-HW2-Read-me-AhQDyjusSgU1BZGFL7ZcT
就能看到更美的排版。

檔案架構

- program 資料夾包含各個 part 的程式碼以及執行結果。
- 每個 part 的主程式以 “part_<第幾大題第幾小題>.m” 來命名，例如，part 1-A 的主程式的檔案則命名為 “part_1a.m”。
- 大部分的 part 的主程式皆附其執行結果的暫存檔案 (.mat 檔案)，命名方式跟主程式相同 (只是後面多了 “_result”)，以供主程式做快速 demo 為用 (每個 part 的主程式內有 “QUICK_DEMO” 控制參數，其作用在下面會補充)。
- 每個檔案皆有詳細的註解。

如何觀看程式執行結果

- 例如想看 part 1-A 的程式執行結果，打開 “part_1a.m” 直接跑就行了，其他 part 也是一樣。
- 每個 part 的程式中有 “QUICK_DEMO” 控制參數，若設為 1，則會從 .mat 檔讀取之前就計算好的執行結果並顯示出來（為了在 demo 的時候省時）；若設為 0 (default)，則會完整跑完計算過程，需要稍微等候才會顯示執行結果。

獨立函式說明

此部分解釋獨立出來的 function 檔案

function P = computeCameraMatrix(point2D, point3D)

```
% compute camera projection matrix (3x4) using eigenvector approach
function P = computeCameraMatrix(point2D, point3D)
  % point2D and point3D should be same size
  % compose A matrix (Ap = 0)
  A = [];
  for i=1:size(point2D, 1)
    % image plane coord.
    x = point2D(i, 1);
    y = point2D(i, 2);
    % corresponding world coord.
    X = point3D(i, 1);
    Y = point3D(i, 2);
    Z = point3D(i, 3);
    A = [...
      A; ...
      X, Y, Z, 1, 0, 0, 0, 0, -x*X, -x*Y, -x*Z, -x; ...
      0, 0, 0, 0, X, Y, Z, 1, -y*X, -y*Y, -y*Z, -y  ...
    ];
  end
  % compute eigenvector corresponding smallest eigenvalue from A'A to get camera projection matrix
  ATA = A'*A;
  [SMeigvec, SMeigval] = eigs(ATA, 1, 'SM');
  P = [ ...
        SMeigvec(1), SMeigvec(2), SMeigvec(3), SMeigvec(4);   ...
        SMeigvec(5), SMeigvec(6), SMeigvec(7), SMeigvec(8);   ...
        SMeigvec(9), SMeigvec(10), SMeigvec(11), SMeigvec(12) ...
  ];
end
```

功能概述：計算從 3D 座標投影到 2D 座標的 camera projection matrix (3x4)

參數說明：

- point2D: 2D 座標
- point3D: 3D 座標

回傳變數：

- P: camera projection matrix，也就是 K[R|t]

做法：

$$\mathbf{x}_i = \mathrm{P}\mathbf{X}_i.$$

Each correspondence generates two equations

$$x_i = \frac{p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}} \qquad y_i = \frac{p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}}$$

Multiplying out gives equations linear in the matrix elements of P

$$x_i(p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}) = p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}$$
$$y_i(p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}) = p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}$$

These equations can be rearranged as

$$\begin{bmatrix} X & Y & Z & 1 & 0 & 0 & 0 & 0 & -xX & -xY & -xZ & -x \\ 0 & 0 & 0 & 0 & X & Y & Z & 1 & -yX & -yY & -yZ & -y \end{bmatrix} \mathbf{p} = \mathbf{0}$$

with $\mathbf{p} = (p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}, p_{23}, p_{24}, p_{31}, p_{32}, p_{33}, p_{34})^\top$ a 12-vector.

- 遵照第三章講義上的做法，建立 Ap = 0 後，我們將問題 format 成 argmin || Ap ||, subject to || p || = 1 的問題，p 其實就等於 eigenvector with smallest eigenvalue of transpose(A)*A (這是此題的解法)，另外也可以用 SVD 去分解 transpose(A)*A，求得 p = vector with smallest singular value，然後再將 p vector 重組成 P matrix function [Q, R] = QR(A)

```matlab
% QR decomposition.
% params A: 3x3 matrix
% return Q: 3x3 orthogonal matrix
% retrun R: 3x3 upper triangular matrix
function [Q, R] = QR(A)
  % column vectors from A
  v1 = A(:,1);
  v2 = A(:,2);
  v3 = A(:,3);
  % perform Gram-Schmidt process, starting from the first column
  u1 = v1;
  u2 = v2 - proj(u1, v2);
  u3 = v3 - proj(u1, v3) - proj(u2, v3);
  % compose Q matrix, each row is orthogonal unit vector
  Q = zeros(3,3);
  Q(:,1) = u1/norm(u1);
  Q(:,2) = u2/norm(u2);
  Q(:,3) = u3/norm(u3);
  % compute R matrix
  R = Q'*A;
end

% orthogonal projection for Gram-Schmidt process
% v projects onto u
function proj_uv = proj(u, v)
  proj_uv = (dot(u, v)/dot(u, u))*u;
end
```

功能概述：QR Decomposition using Gram-Schmidt process

參數說明：

- A: 3x3 matrix，用來分解 camera projection matrix

回傳變數：

- Q: 3x3 orthogonal matrix
- R: 3x3 upper triangular matrix

做法：

- 根據維基百科上的 Gram Schmidt process 的做法，先產生 Q

We define the projection operator by

$$\mathrm{proj}_{\mathbf{u}}(\mathbf{v}) = \frac{\langle \mathbf{v}, \mathbf{u} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle} \mathbf{u},$$

where $\langle \mathbf{v}, \mathbf{u} \rangle$ denotes the inner product of the vectors **v** and **u**. This operator projects the vector **v** orthogonally onto the line spanned by vector **u**. If **u** = **0**, we define $\mathrm{proj}_{\mathbf{0}}(\mathbf{v}) := \mathbf{0}$. i.e., the projection map $\mathrm{proj}_{\mathbf{0}}$ is the zero map, sending every vector to the zero vector.

The Gram–Schmidt process then works as follows:

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{v}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\
\mathbf{u}_2 &= \mathbf{v}_2 - \mathrm{proj}_{\mathbf{u}_1}(\mathbf{v}_2), & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\
\mathbf{u}_3 &= \mathbf{v}_3 - \mathrm{proj}_{\mathbf{u}_1}(\mathbf{v}_3) - \mathrm{proj}_{\mathbf{u}_2}(\mathbf{v}_3), & \mathbf{e}_3 &= \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|} \\
\mathbf{u}_4 &= \mathbf{v}_4 - \mathrm{proj}_{\mathbf{u}_1}(\mathbf{v}_4) - \mathrm{proj}_{\mathbf{u}_2}(\mathbf{v}_4) - \mathrm{proj}_{\mathbf{u}_3}(\mathbf{v}_4), & \mathbf{e}_4 &= \frac{\mathbf{u}_4}{\|\mathbf{u}_4\|} \\
&\ \ \vdots & &\ \ \vdots \\
\mathbf{u}_k &= \mathbf{v}_k - \sum_{j=1}^{k-1} \mathrm{proj}_{\mathbf{u}_j}(\mathbf{v}_k), & \mathbf{e}_k &= \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}.
\end{aligned}$$

- 有了 Q 之後，從 A = QR 求得 R 其實就等於 transpose(Q) * A function H = computeHomography(left, right)

```
% compute projective transformation matrix (3x3) using eigenvector approach
% such that left(homogeneous) = H * right(homogeneous)
function H = computeHomography(left, right)
  % point2D and point3D should be same size
  % compose A matrix (Ap = 0)
  A = [];
  for i=1:size(left, 1)
    x_p = left(i, 1);
    y_p = left(i, 2);
    % corresponding points.
    x = right(i, 1);
    y = right(i, 2);
    A = [...
      A; ...
      x, y, 1, 0, 0, 0, -x_p*x, -x_p*y, -x_p; ...
      0, 0, 0, x, y, 1, -y_p*x, -y_p*y, -y_p, ...
    ];
  end
  % compute eigenvector corresponding smallest eigenvalue from A'A to get camera projection matrix
  ATA = A'*A;
  [SMeigvec, SMeigval] = eigs(ATA, 1, 'SM');
  H = [ ...
        SMeigvec(1), SMeigvec(2), SMeigvec(3);  ...
        SMeigvec(4), SMeigvec(5), SMeigvec(6);  ...
        SMeigvec(7), SMeigvec(8), SMeigvec(9) ...
  ];
end
```

功能概述：計算從 2D 座標投影到另一個 2D 座標的 projective transformation matrix (3x3)

參數說明：

- left: 2D 座標
- right: 2D 座標

回傳變數：

- H: projective transformation matrix (3x3)，也就是 homography

做法：

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\begin{aligned} x'_i &= \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ y'_i &= \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \end{aligned}$$

$$\begin{aligned} x'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned}$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x'_i x_i & -x'_i y_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_i x_i & -y'_i y_i & -y'_i \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- 遵照第三章講義上的做法，建立 Ah = 0 後，我們將問題 format 成 argmin || Ah ||, subject to || h || = 1 的問題，h 其實就等於 eigenvector with smallest eigenvalue of transpose(A)*A，然後再將 h vector 重組成 H matrix function img_new = backwardWrap(img, img_new, projected_points, target_points)

```
% use bilinear interpolation to get right image region's new pixel
% value in the position of left image region.
function img_new = backwardWrap(img, img_new, projected_points, target_points)
    for i=1:size(target_points,1)
        i % just print index to show progress
        x = projected_points(i,1);
        y = projected_points(i,2);
        target_x = target_points(i,1);
        target_y = target_points(i,2);
        % calculate weights
        w = ceil(x) - floor(x);
        h = ceil(y) - floor(y);
        left_weight = (ceil(x) - x)/w;
        right_weight = (x - floor(x))/w;
        up_weight = (ceil(y) - y)/h;
        bottom_weight = (y - floor(y))/h;
        % interpolate color. color = img(y,x) <=> y=row, x=col
        top_left_color = img(floor(y), floor(x), :);
        top_right_color = img(floor(y), ceil(x), :);
        bottom_left_color = img(ceil(y), floor(x), :);
        bottom_right_color = img(ceil(y), ceil(x), :);
        up_weighted_color = left_weight*top_left_color + right_weight*top_right_color;
        bottom_weighted_color = left_weight*bottom_left_color + right_weight*bottom_right_color;
        target_color = up_weight*up_weighted_color + bottom_weight*bottom_weighted_color;
        % assign color to target pixel
        img_new(target_y, target_x, :) = target_color;
    end
end
```

功能概述：用 bilinear interpolation 做 backward wrapping，算出 pixel 在目標位置的顏色

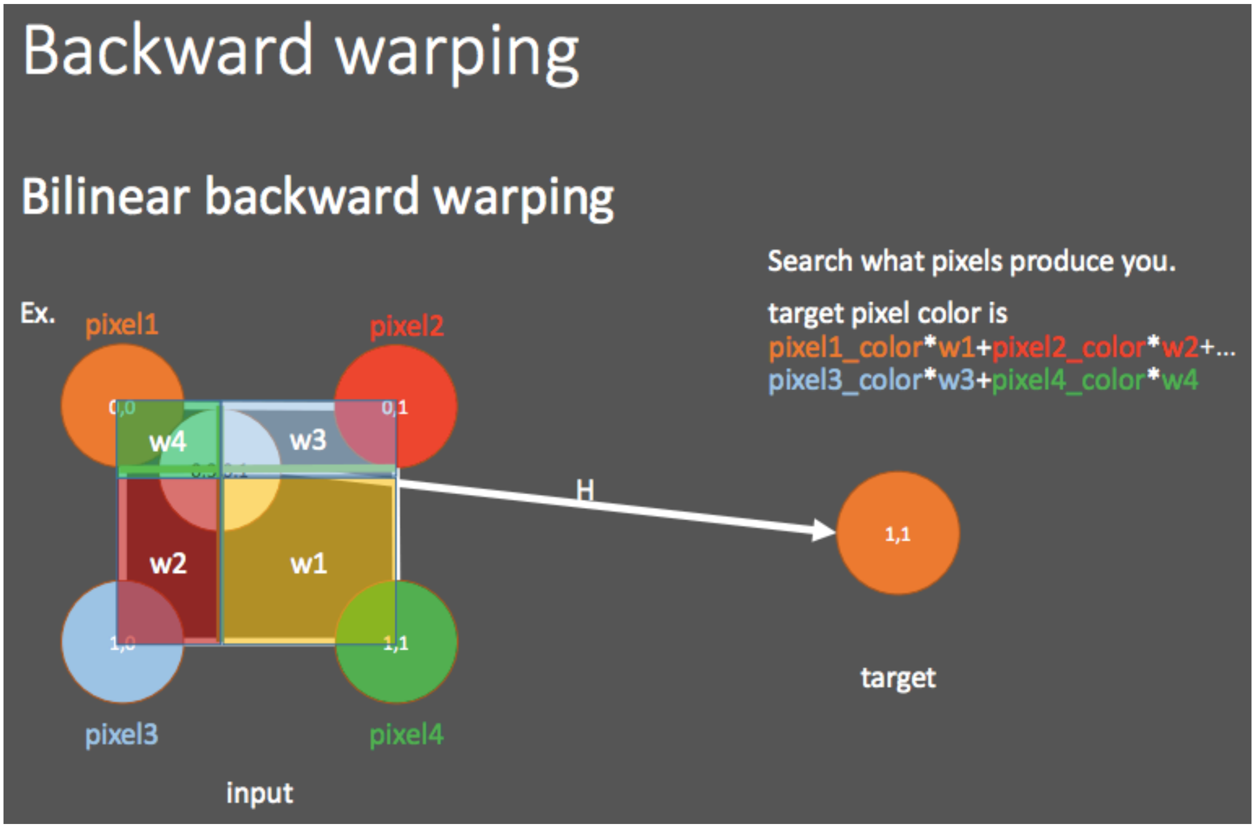
參數說明：

- img: 原圖
- img_new: 要編輯的圖
- projected_points: 經過 homography 投影的點
- target_points: 要內插出顏色的點

回傳變數：

- img_new: 結果圖

做法：



- 遵照 wrapping supplement 講義所示，找出 projected point 附近四個點的顏色，用 bilinear interpolation 的公式內差出 target point 的顏色