Informe Tarea No 2

Tatiana Lopez Guevara
Universidad Tecnológica de Pereira
zepolitat@utp.edu.co

***Resumen*—El presente documento explica los resultados obtenidos al remover las distorciones proyectiva y afín haciendo uso de las estructuras propias de la imágen, a diferencia del trabajo 1 en donde se daban explícitamente la correspondencia entre la imágen original y la distorsionada para 4 pares de puntos.**

***Index Terms*—Computer Vision**

I. Rectificación Afín

A diferencia de una transformación proyectiva, en una transformación afín las líneas paralelas permanecen paralelas, es decir, los puntos ideales que forman la línea al infinito L_∞ permanecen en el infinito. Resulta conveniente entonces aprovechar esta característica invariante para identificar una matrix de transformación H que vuelva a enviar la línea de desvanecimiento nuevamente al infinito, removiendo así la perspectiva [2].

La primera parte del trabajo consistió en identificar dentro de la imágen 2 conjuntos de pares de líneas (l_1, l_2) y (m_1, m_2) que se sabe que son paralelas en el mundo real. Luego, se obtuvo la línea de desvanecimiento vl mediante (1).

$$\begin{aligned} P &= l_1 \times l_2 \\ Q &= m_1 \times m_2 \\ vl &= P \times Q \end{aligned} \tag{1}$$

En la figura 1 se muestra la imágen con la distorción proyectiva.

Figura 1: Imágen con distorción proyectiva

La matriz H se obtiene entonces mediante

$$H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ vl_1 & vl_2 & vl_3 \end{pmatrix} \tag{2}$$

Figura 2: Imágen rectificada hasta afinidad. (a) Resultado obtenido en este trabajo. (b) Resultado obtenido con el toolbox de Matlab

Una vez conocida la matriz H, lo primero que se hizo fue hallar el tamaño de la imágen final. Para esto se aplicó la homografía inversa sobre las esquinas de la imágen cargada empleando (3).

$$x = H^{-1}x^{'} \tag{3}$$

Teniendo entonces el tamaño de la imágen, se procedió a encontrar para cada punto (x, y) de la imágen final, su equivalente $(x^{'}, y^{'})$. Cabe notar que a pesar de que (4) hace referencia a la transformación de un solo punto, en el desarrollo se usó la representación matricial en donde x es el conjunto de puntos de la forma $[x_1y_1 1; x_2y_2 1; ..x_Ny_N 1]'$ y por lo tanto x' contiene todos los puntos transformados de x;

$$x^{'} = Hx \tag{4}$$

El inconveniente al hallar los puntos $(x^{'}, y^{'})$ que se obtienen de (4), es que éstos pueden no ser enteros y por lo tanto se debió aplicar una técnica de interpolación (ver sección III).

Finalmente, al remover la perspectiva y aplicar la interpolación bilineal sobre la figura 1 se obtubo la figura 2.

II. Rectificación Métrica

La segunda parte del proyecto consistió en remover la transformación afín hasta una similaridad. Para esto se toma ventaja de la cónica C^*_∞ formada por los puntos circulares $I = (1, i, 0)^T$ y $J = (1, -i, 0)^T$. Esta cónica es invariante ante las trasnformaciones similares (5) y puede ser empleada para medir ángulos euclidianos en una proyectividad mediante (6) ya que dicha expresión es invariante ante transformaciones proyectivas [2].

$$C_\infty^{*\prime} = H_s C_\infty^* H_s^T = C_\infty^* \tag{5}$$

$$\cos\theta = \frac{l^T C_\infty^* m}{\sqrt{(l^T C_\infty^* l)(m^T C_\infty^* m)}} \tag{6}$$

Esta última propiedad (6) permite identificar que para un par de líneas ortogonales el $\cos 90$ es 0 y por lo tanto:

$$l^T c_\infty^* m = 0 \tag{7}$$

Dados 2 conjuntos de pares de líneas ortogonales y gracias a la ecuación (7), se puede encontrar C_∞^*, o más específicamente la matriz K resolviendo (8)

$$\begin{pmatrix} l'_1 & m'_1, l'_1 m'_2 + l'_2 m'_1 \\ n'_1 & p'_1, n'_1 p'_2 + n'_2 p'_1 \end{pmatrix} \begin{pmatrix} s_{11} \\ s_{12} \end{pmatrix} = - \begin{pmatrix} l'_2 m'_2 \\ n'_2 p'_2 \end{pmatrix} \tag{8}$$

Donde

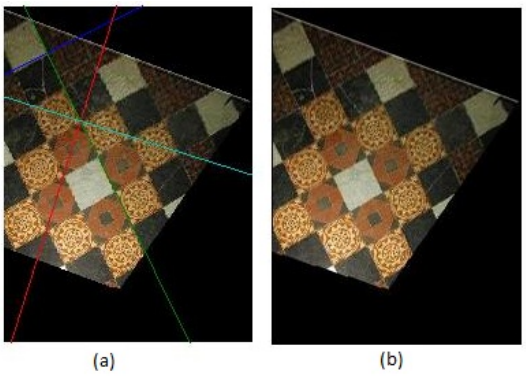
$$S = \begin{pmatrix} s_{11} & s_{12} \\ s_{12} & s_{22} \end{pmatrix} = \begin{pmatrix} KK^T & \vec{0} \\ \vec{0}^T & 0 \end{pmatrix} \tag{9}$$

Es decir que encontrando S y aplicando la descomposición de Cholesky, se puede obtener la matriz K que define a la matriz H_A y que remueve la afinidad.

$$H_A = \begin{pmatrix} K & \vec{0} \\ \vec{0}^T & 1 \end{pmatrix} \tag{10}$$

De forma similar a la parte 1, se realizó el cálculo de la matrix de transformación H a partir de los 8 puntos dados. Luego, ésta se usó para aplicar la distorción proyetiva sobre cada punto de la imágen planar mediante (4) . El resultado obtenido se puede ver en la figura 3.

Figura 3: Imágen rectificada hasta similaridad. (a) Resultado obtenido en este trabajo. (b) Resultado obtenido con el toolbox de Matlab



En las figuras 4 5 y 6 se pueden ver los resultados al aplicar las 2 partes sobre una fotografía de un cuadro.

Figura 4: Imágen original.

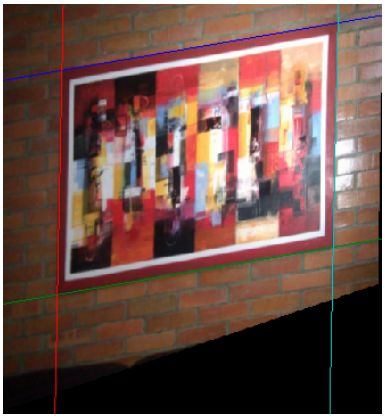


Figura 5: Imágen con corrección proyectiva hasta afinidad.

III. Métodos de Interpolación

A diferencia de los métodos de interpolación Nearest Neighbor (1 pixel más cercano) e interpolación bilinear (2x2 pixeles más cercanos), la interpolación bicúbica toma en cuenta los 4x4 pixeles más cercanos, dando mayor peso en el cálculo a los que se encuetran más cerca.

Esta técnica consiste en aproximar un valor mediante un polinomio de tercer grado (11).

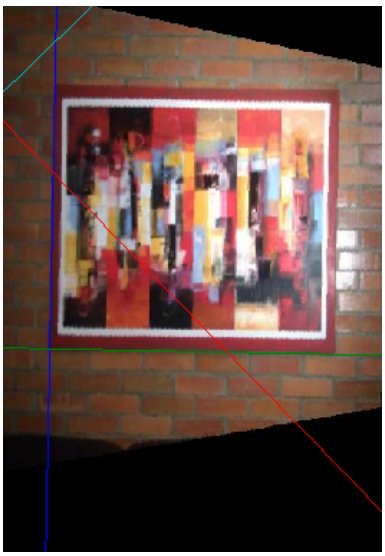


Figura 6: Imágen con corrección afín hasta similaridad.

$$p(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j \tag{11}$$

Como se describe en [3], el problema consiste en identificar los 16 coeficientes a_{ij} (a partir de los 16 puntos conocidos) del sistema descrito por (12).

$$\vec{\alpha} = A^{-1}\vec{P} \tag{12}$$

Donde

$$\vec{\alpha} = (a_{00}, a_{10}, a_{20}, a_{30}, a_{01}, a_{11}, a_{21}, a_{31} \\ a_{02}, a_{12}, a_{22}, a_{32}, a_{03}, a_{13}, a_{23}, a_{33})^T \tag{13}$$

.

$$A^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 \\ -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 \\ 9 & -9 & -9 & 9 & 6 & 3 & -6 & -3 & 6 & -6 & 3 & -3 & 4 & 2 & 2 & 1 \\ -6 & 6 & 6 & -6 & -3 & -3 & 3 & 3 & -4 & 4 & -2 & 2 & -2 & -2 & -1 & -1 \\ 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ -6 & 6 & 6 & -6 & -4 & -2 & 4 & 2 & -3 & 3 & -3 & 3 & -2 & -1 & -2 & -1 \\ 4 & -4 & -4 & 4 & 2 & 2 & -2 & -2 & 2 & -2 & 2 & -2 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{14}$$

$$\vec{P} = (P_{11}, P_{21}, P_{12}, P_{22}, P'_{x11}, P'_{x21}, P'_{x12}, P'_{x22} \\ P'_{y11}, P'_{y21}, P'_{y12}, P'_{y22}, P'_{xy11}, P'_{xy21}, P'_{xy12}, P'_{xy22})^T \tag{15}$$

Donde $P_{i,j}$ representa el valor del pixel en la posición (i,j) de la matriz de 4x4 cuyos valores se conocen. $P'_{xi,j}$ es la derivada con respecto a X en (i,j) y $P'_{xyi,j}$ es la derivada cruzada con respecto tanto a X como a Y.

Los valores de las derivadas en estos puntos son aproximados mediante la técnica de diferencias finitas (16)(17) y (18).

$$P'_{xi,j} = \frac{1}{2}(P_{i+1,j} - P_{i-1,j}) \tag{16}$$

$$P'_{yi,j} = \frac{1}{2}(P_{i,j+1} - P_{i,j-1}) \tag{17}$$

$$P'_{xyi,j} = \frac{1}{4}(P_{i+1,j+1} - P_{i+1,j-1} - P_{i-1,j+1} + P_{i-1,j-1}) \tag{18}$$

En [1] se muestra el valor de cada coeficiente en términos únicamente de los puntos P_{ij}.

T. Interpolación	Tarea	Matlab
Nearest Neighbors	0.012355	0.012411
Bilineal	0.014050	0.017320
Bicúbica	0.128893	0.034081

Cuadro I: Comparación Tiempos ToolBox Matlab para Rectificación Afín en segundos

T. Interpolación	Tarea	Matlab
Nearest Neighbors	0.014405	0.028832
Bilineal	0.039678	0.036699
Bicúbica	0.200090	0.054263

Cuadro II: Comparación Tiempos ToolBox Matlab para Rectificación Métrica en segundos

IV. Experimentos

IV-A. Conjuntos

Se realizaron diferentes ejecuciones para ambas partes con diferentes conjuntos de pares de líneas tanto paralelas como ortogonales obteniendo la misma transformación. Sin embargo se observó una variación en cuanto al tamaño de la imágen generada para la rectificación afín.

IV-B. Matlab

Se hizo uso de las funciones maketform y imtransform de Matlab con las matrices Hp y Ha obtenidas mediante (2) y (10) respectivamente. Estas funciones fueron comparadas con $createTransfImg$ realizada en este trabajo.

```
tic
[fImg, fVecImg] = createTransfImg(fSize,
        iSize, iPos, iVecImg,'bicubic');
toc

...

tic
T=maketform('projective',Hp');
P2=imtransform(iImg,T,'bicubic');
toc
```

```
tic
[fImg, fVecImg] = createTransfImg(fSize,
        iSize, iPos, iVecImg, 'bicubic');
toc

...

tic
T=maketform('affine',HaInv');
P2=imtransform(iImg,T,'bicubic');
toc
```

En la tablas I y II se muestran los tiempos obtenidos en el código desarrollado en este trabajo para la rectificación Afín y Métrica respectivamente, versus el tiempo que toma la herramienta de matlab para realizar el mismo procedimiento.

La comparación gráfica de los resultados se puede ver en las partes (a) y (b) de las figuras 2 y 3.

V. Descripción del Código

Para el desarrollo del proyecto se construyeron los siguientes archivos fuente:

V-A. Scripts de Invocación

- **hw2ToAffinity.m**
  Archivo principal de la parte 1 de la tarea. A partir de 4 puntos que definen los 2 conjuntos de pares de líneas paralelas calcula la línea de desvanecimiento mediante (1) y a partir de ésta crea una homografía que remueve la proyectividad.
- **hw2ToAffinity.m**
  Archivo principal de la parte 2 de la tarea. A partir de 4 puntos que definen los 2 conjuntos de pares de líneas ortogonales resuelve el sistema dado por (8), y mediante descomposición de Cholesky recupera la matriz K para crear una homografía que remueve la afinidad hasta una similaridad.

V-B. Funciones para Interpolación

- **nearestN.m**
  Realiza una interpolación mediante la técnica del vecino más cercano de una imágen en representación vectorial de tamaño $[(W * H), 3]$ y de un conjunto de índices que representan coordenadas en punto flotante. El valor del pixel equivalente lo obtiene mediante: $round(x), round(y)$
- **bilineal.m**
  Realiza una interpolación bilineal de una imágen en representación vectorial de tamaño $[(W * H), 3]$ y de un conjunto de índices que representan coordenadas en punto flotante. Las coordenadas de las 4 esquinas sobre las que va a interpolar, se obtienen de: $floor(x), floor(y), floor(x) + 1, floor(y) + 1$
- **bicubic.m**
  Realiza una interpolación bicúbica de una imágen en representación vectorial de tamaño $[(W * H), 3]$ y de un conjunto de índices que representan coordenadas en punto flotante. El valor del pixel se obtiene mediante (11).

V-C. Funciones Utilitarias

- **zimread.m**
  Función utilitaria que carga una imágen y retorna su representación como un vector 2D de $W * H$ filas y 3 columnas de color. Por la agilidad de su operación, esta representación es la usada principalmente para realizar las operaciones sobre la imágen. Adicionalmente, retorna una matrix tridimensional de H filas, W columnas y 3 canales de color RGB principalmente para el despliegue en pantalla.
- **createTransfImg.m**
  Crea una imágen $[W]x[H]x[3]$ a partir de los puntos transformados por alguna homografía y que están almacenados en un vector de $[W * H][3]$ aplicando una interpolación dada. La técnica de *NearestNeighbor* es aplicada por defecto si no se especifica otra técnica.
- **findKMat.m**
  Resuelve el sistema dado por (9) a partir del conjunto de puntos que describen los 2 conjuntos de líneas ortogonales y retorna la matriz K.
- **findVL.m**
  Resuelve el sistema dado por (1) a partir del conjunto de puntos que describen los 2 conjuntos de líneas paralelass y retorna la línea de desvanecimiento.
- **transformCorner.m**
  Calcula las esquinas de una imágen dado el ancho y el alto y aplica una homografía sobre estas. Esta función invoca a transformX.
- **transformImg.m**
  Genera la combinación de puntos (x, y) de una imágen y aplica una homografía invocando a transformX.m.
- **transformX.m**
  Aplica una homografía a una matrix de puntos de la forma $[x_1x_2..x_N; y_1y_2..y_N]$ y la vuelve homogénea.

VI. Conclusiones

- Haciendo uso de la línea de desvanecimiento que se obtiene a partir de dos conjuntos de pares de líneas paralelas es posible remover la distorción afín enviando la línea al infinito que fue mapeada a un punto de la imágen nuevamente al infinito.
- A partir de dos conjuntos de pares de líneas ortogonales es posible obtener un sistema que permite obtener una matriz Ha que remueva la distorción afín.
- Al hacer la última componente de la línea de desvanecimiento igual a 1 (homogeneizar) antes de obtener la matriz de transformación H, se evitan los problemas de escalamiento de la imágen resultante.
- La interpolación bicúbica obtiene una representación más acertada de la imágen debido a que usa la información de (4x4) pixeles teniendo en cuenta los cambios (derivadas) de éstos por lo que brinda una mejor respuesta cuando hay cambios bruscos de un pixel a otro.
- Los resultados obtenidos mostraron que la interpolación bilineal también ofrecía buenos resultados en menor tiempo que la bicúbica, ya que sólo requiere de un promedio ponderado a partir de (2x2) puntos.

Referencias

[1] Paul Breeuwsma. Cubic interpolation. http://www.paulinternet.nl/?page=bicubic. [Online; accessed Apr-2013].

[2] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*, volume 2. Cambridge Univ Press, 2000.

[3] Wikipedia. Bicubic interpolation. http://en.wikipedia.org/wiki/Bicubic_interpolation. [Online; accessed Apr-2013].