Tarea 2: Alineamiento de imágenes usando Harris y descriptores ORB

EL7008 – Procesamiento Avanzado de Imágenes

Nombre:	Sebastián Parra
Profesor:	Javier Ruiz del Solar
Auxiliar:	Patricio Loncomilla
Ayudante:	Francisco Leiva
	Nicolás Cruz
Fecha:	5 de octubre de 2018

Introducción

El presente informe trata sobre las actividades realizadas en el marco de la segunda tarea del curso EL7008 – Procesamiento Avanzado de Imágenes, cuyo objetivo es lograr una familiarización con la implementación de herramientas de detección de puntos de interés y algoritmos de *matching*, en particular la utilización de filtros de Harris y descriptores ORB, para aplicaciones de procesamiento de imágenes utilizando una librería estándar para estos motivos (OpenCV en C++).

Para lograr estos objetivos, la tarea consistió en diseñar un sistema que permita alinear dos imágenes, obteniendo una suerte de vista "panorámica" de las dos imágenes fusionadas en sus zonas comunes. Sin embargo, estas imágenes no necesariamente calzarán perfectamente, y en ocasiones las zonas comunes se podrán encontrar rotadas y trasladadas. Para lograr esto, se pidió programar un filtro de Harris para la detección de esquinas, seguido de la utilización de descriptores ORB para hacer un *matching* entre las dos imágenes a fusionar. Finalmente, se solicitó utilizar funciones de OpenCV para encontrar una homografía que relacione ambas imágenes y proyectar la fusión de las dos figuras según la transformación encontrada.

A lo largo de este documento se presenta un breve marco teórico donde se explican las principales herramientas utilizadas para resolver el problema propuesto, seguido de una sección de desarrollo y resultados donde se explicará cómo se realizó la programación de cada paso del procedimiento pedido y se mostrarán los resultados de aplicar el algoritmo diseñado sobre 4 pares de imágenes entregadas por el equipo docente, indicando si el sistema fue exitoso y exponiendo observaciones en donde fuese necesario.

Marco Teórico

Para lograr comprender la programación de la solución de esta tarea, es necesario describir algunos conceptos, los cuales serán explicados en esta sección.

Filtro de Harris

El filtro de Harris es un detector de esquinas que corresponde a una mejora del filtro Moravec. Ambos se basan en observar en cuánto varían los pixeles de una región local de una imagen frente a pequeños desplazamientos, utilizando una fórmula conocida como la energía, cuya ecuación se puede apreciar en la ecuación 1. En esta fórmula, la energía corresponde a la suma de las diferencias cuadráticas de los pixeles de la imagen desplazada menos los pixeles de la imagen original, para todos los pixeles dentro de una ventana $w(x, y)$. Luego, un punto será una esquina si su energía es máxima para todas las direcciones de desplazamiento.

$$E(x, y; \Delta x, \Delta y) = \sum_{x,y} w(x, y)(I(x + \Delta x, y + \Delta y) - I(x, y))^2$$

Ecuación 1: Fórmula de la energía de una imagen en (x, y), considerando un desplazamiento $(\Delta x, \Delta y)$.

En el caso específico del detector Harris, las funciones utilizadas cambian levemente: La función ventana pasa a ser una ventana gaussiana, y el valor de pixel de cada imagen se calcula sobre una versión suavizada por un filtro gaussiano, quedando la fórmula de la ecuación 2.

$$E_{Harris}(x, y; \Delta x, \Delta y) = \sum_{x,y} N(x, y, \sigma_I)(L(x+\Delta x, y+\Delta y, \sigma_D) - L(x, y, \sigma_D))^2$$

$$L(x, y, \sigma_D) = I(x, y) * N(x, y, \sigma_D)$$

Ecuación 2: Fórmula de la energía de una imagen en un filtro de Harris.

Sin embargo, al usar esta formulación se puede obtener la expresión matricial de la energía mostrada en la ecuación 3, a partir de la cual se puede diagonalizar la matriz μ e igualar el procedimiento de maximizar la energía al de maximizar los valores propios λ_1, λ_2 de la matriz, lo cual a su vez es equivalente a maximizar el indicador *cornerness* presentado en la ecuación 4.

$$E_{Harris}(x, y; \Delta x, \Delta y) = (\Delta x, \Delta y)\mu(x, y)\begin{pmatrix}\Delta x \\ \Delta y\end{pmatrix}$$

$$\mu(x, y) = N(x_0 - x, y_0 - y, \sigma_I) * \begin{bmatrix}\frac{\partial L}{\partial x}\frac{\partial L}{\partial x} & \frac{\partial L}{\partial x}\frac{\partial L}{\partial y} \\ \frac{\partial L}{\partial x}\frac{\partial L}{\partial y} & \frac{\partial L}{\partial y}\frac{\partial L}{\partial y}\end{bmatrix}$$

Ecuación 3: Fórmula matricial de la energía de una imagen en un filtro de Harris.

$$cornerness(x, y) = det(\mu(x, y)) - \alpha Tr^2(\mu(x, y))$$

Ecuación 4: Indicador *cornerness* que resume el procedimiento de detección de esquinas.

De esta manera, un filtro de Harris es una operación que, a partir de una imagen, retorna una nueva imagen de igual tamaño, donde $I_{out}(x, y)$= *cornerness* (x, y). Posteriormente, la detección de esquinas se realizará mediante la detección de máximos locales en la imagen filtrada, fijando un umbral de detección.

Descriptor Oriented FAST and Rotated BRIEF(ORB)

ORB es un algoritmo eficiente para el cálculo de descriptores locales, el cual se basa en la utilización de versiones modificadas de los algoritmos FAST (para detección de puntos de interés) y BRIEF (para la obtención de os descriptores).

En cuanto al proceso de detección, el método FAST destaca por su bajo costo computacional, pero también por su incapacidad de ser flexible ante cambios de escala y orientación, por lo que ORB introduce la utilización de pirámides de multiresolución y el cálculo de centroides para obtener una versión más robusta del algoritmo frente a estas transformaciones. Por otro lado, para el proceso de obtención de descriptores, se utiliza el método BRIEF, pero como también se conoce que este método, a pesar de su gran velocidad de cómputo, presenta un mal desempeño ante rotaciones, se aprovecha la información de la orientación de los puntos obtenida en la sección anterior para orientar la matriz de puntos a ser comparados en la dirección del punto de interés, efectivamente obteniendo una versión "rotada" del algoritmo.

A partir de estas modificaciones, se pudo obtener un algoritmo competitivo con SIFT y SURF, haciendo un uso eficiente de recursos computacionales.

Transformación de Homografía

Una transformación de homografía relaciona la transformación entre dos planos mediante la ecuación:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Donde la matriz H es conocida como matriz de homografía. Es importante destacar que para que dos planos estén relacionados por una homografía, necesariamente deben estar observando el mismo plano, pero desde ángulos distintos, como es el caso del movimiento rotacional de una cámara al sacar una fotografía panorámica. De esta manera, si se tienen varias fotos que forman parte de un panorama, se puede encontrar una homografía que proyecta la vista de una imagen sobre el plano de la otra, de manera que se vean como una imagen continua, como se puede apreciar en el ejemplo de la figura 1.

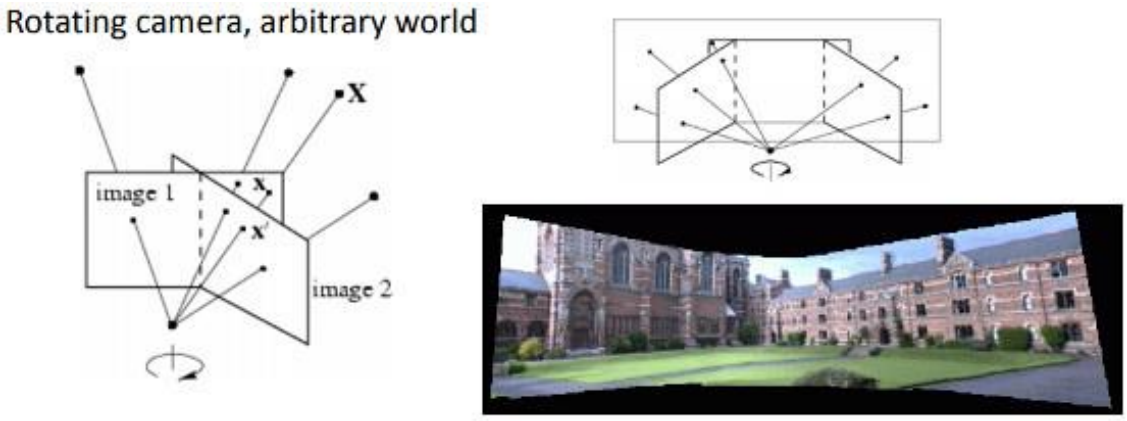

Figura 1: Transformación de homografía para obtener una vista panorámica a partir de una serie de imágenes.

RANSAC

RANSAC es un método iterativo que permite estimar los parámetros de algún modelo matemático caracterizado por presentar *inliers*, datos que se ajustan al modelo, y una cantidad no definida de *outliers*, datos que no se ajustan al modelo, que no deben ser considerados para la estimación. El algoritmo funciona se la siguiente forma:

1. Se muestrea un subconjunto pequeño de datos aleatoriamente.
2. Se ajusta el modelo al subconjunto obtenido y se obtienen sus parámetros (La idea es que el subconjunto sea del tamaño más pequeño suficiente para determinar los parámetros del modelo).
3. Revisar qué elementos del conjunto total de datos son consistentes con el modelo obtenido. Si un dato se desvía del modelo por sobre un cierto umbral, se considerará como un *outlier*.
4. Repetir los pasos anteriores hasta obtener un modelo con una cantidad suficiente de *inliers*.

Este algoritmo se puede aplicar para encontrar homografías, ya que a partir de una serie pequeña de puntos se puede estimar una transformación de homografía que relaciona una imagen con la otra, y luego utilizando alguna métrica de error (como el error cuadrático) se puede determinar si el resto de los puntos

se ajusta a esta transformación. Finalmente, el algoritmo seleccionaría una matriz de homografía adecuada una vez que una cantidad adecuada de puntos se ajuste a alguna transformación.

Desarrollo y Resultados

Parte 1: Programación del filtro de Harris

La primera actividad de esta tarea consistió en completar la función *harrisFilter()* en el código proporcionado por el equipo docente, de manera que la función pudiese, a partir de una imagen de entrada, devolver su filtro de Harris. La programación de esta función se dividió en 6 pasos:

1. Se transforma la imagen de entrada a escala de grises y se suaviza utilizando un filtro gaussiano de 3x3 con $\sigma = 2$, mediante la función *GaussianBlur()* de OpenCV.
2. Se calculan por separado las derivadas en x e y de la imagen suavizada mediante un operador de Scharr, utilizando la función de OpenCV que lleva el mismo nombre, como se puede notar en la figura 2.

```
// Paso 2: Calcular derivadas en x e y
Mat ix, iy;
Scharr(input_gray, ix, CV_32FC1, 1, 0);
Scharr(input_gray, iy, CV_32FC1, 0, 1);
```

Figura 2: Cálculo de las derivadas en x e y de la matriz suavizada mediante operadores de Scharr.

3. Se calculan los momentos de la matriz μ mostrada en la ecuación 3 mediante operadores de multiplicación de matrices elemento-por-elemento, como se puede apreciar en la figura 3.

```
// Paso 3: Calcular ixx, ixy, iyy
Mat ixx, ixy, iyy;
ixx = ix.mul(ix);
ixy = ix.mul(iy);
iyy = iy.mul(iy);
```

Figura 3: Cálculo de los momentos de la matriz μ.

4. Se suavizan los momentos calculados en el paso anterior mediante la utilización de filtros gaussianos de 3x3 con $\sigma = 2/0{,}7$, según fue sugerido por el auxiliar.
5. Se calcula el indicador *cornerness* determinando el determinante y la traza de μ, y fijando $\alpha = 0.04$, como se puede notar en la figura 4.

```
// Paso 5: Calcular cornerness
Mat det, Tr;
det = ixx.mul(iyy) - ixy.mul(ixy);
Tr = (ixx + iyy);

harris = det - 0.04*(Tr.mul(Tr));
```

Figura 4: Cálculo del *cornerness* para cada punto de la imagen.

6. Finalmente, se normalizan los valores de la imagen de salida para que se encuentren dentro del rango 0-255 (que es el usado por OpenCV para visualizar imágenes). Esto se realizó utilizando la función *normalize()* de la librería.

Parte 2: Programación de la búsqueda de máximos en la imagen de Harris

Para esta parte se pidió completar la función *getHarrisPoints()* en el código proporcionado por el equipo docente, de manera que la función pudiese, a partir de la imagen de Harris, devolver un vector con todos los puntos de interés de la figura. La función consistió en los siguientes pasos:

1. Comenzando en el origen de la imagen, se dibujó una ventana de 3x3, alineando su esquina superior izquierda con el punto actual de la imagen. Luego, utilizando la función *mixMaxLoc()* de OpenCV se obtuvo el máximo local de la ventana y su ubicación. En la figura 5 se muestra la sección se código donde se implementó este paso, donde las variables (c, r) representan la columna y la fila de la iteración actual, maxVal representa el valor del máximo local, y maxLoc representa la ubicación del máximo local en la ventana.

```
// Paso 1: Obtener maximo local
minMaxLoc(harris(Rect(c, r, 3, 3)), nullptr, &maxVal, nullptr, &maxLoc);
```

Figura 5: Cálculo de máximos locales de la imagen usando *mixMaxLoc()*.

2. Se verifica si el máximo local se encuentra en el centro de la ventana. Esto se realiza para que no existan puntos repetidos en la salida, ya que existe intersección entre las ventanas. Si bien esta condición hace imposible el encontrar puntos de interés en los bordes, se decidió implementar de todos modos ya que se supuso que no se estaría perdiendo demasiada información, al estar la información más valiosa lejos de los bordes.
3. Se verifica que el valor del máximo local sea mayor a un umbral fijado. Este umbral se debe ir ajustando dependiendo de la imagen mostrada.
4. Si ambas condiciones anteriores se cumplen, agregar el máximo local al vector de puntos de interés.
5. Repetir este procedimiento hasta que el punto medio de la ventana haya recorrido todos los puntos de la imagen, salvo sus bordes.

Parte 3: Programación de la visualización de los puntos de Harris

Una vez programadas las funciones anteriores, se procedió a programar la función principal del programa. Las primeras operaciones que se realizaron, luego de importar el par de imágenes a fusionar, fueron calcular las imágenes de Harris de ambas figuras, guardarlas en un archivo, seguido de determinar sus vectores de puntos de interés y dibujarlos sobre las imágenes originales utilizando la función de OpenCV *drawKeyPoints()*.

En el extracto de código de la figura 6 se muestra cómo es que a partir de las imágenes de entrada *imleft* y *imright* se obtienen y guardan las imágenes de Harris *harrisleft* y *harrisright*, para luego calcular los vectores de puntos de interés *pointsleft* y *pointsright* y finalmente guardar en un archivo las visualizaciones de los puntos de interés en los archivos *impointsleft.jpg* y *impointsright.jpg*.

```
// Crear matrices de harris
Mat harrisleft, harrisright;
harrisleft = harrisFilter(imleft);
harrisright = harrisFilter(imright);
// Guardar las imagenes de harris
imwrite("../harrisleft.jpg", harrisleft); // Grabar imagen
imwrite("../harrisright.jpg", harrisright); // Grabar imagen

// Obtener vectores de puntos de interes
// Por defecto se recomienda dejar un umbral de 115, pero a veces puede ser necesario bajarlo
vector<KeyPoint> pointsleft = getHarrisPoints(harrisleft , 115);
vector<KeyPoint> pointsright = getHarrisPoints(harrisright, 115);

// Dibujar los puntos de interes
drawKeypoints(impointsleft, pointsleft, impointsleft);
drawKeypoints(impointsright, pointsright, impointsright);
// Guardar imagen con los puntos de interes dibujados
imwrite("../impointsleft.jpg", impointsleft); // Grabar imagen
imwrite("../impointsright.jpg", impointsright); // Grabar imagen
```

Figura 6: Primeras líneas de código de la función principal del archivo main.cpp.

Parte 4: Cálculo de descriptores ORB para ambas imágenes

Esta parte del código estuvo en su mayoría implementada por el equipo docente, y se trata de crear un objeto estimador tipo ORB en OpenCV, y luego llamar al método *compute()* que recibe una imagen de entrada y un vector de puntos de interés, la cual se encarga de calcular los descriptores necesarios para cada imágen. En la figura 7 se muestra el código implementado para esta parte, donde se siguieron las nomenclaturas utilizadas en la parte anterior.

```
// Crear descriptores ORB
Ptr<ORB> orb = ORB::create();
Mat descrleft, descrright;
orb->compute(imleft, pointsleft, descrleft);
orb->compute(imright, pointsright, descrright);
```

Figura 7: Cálculo de descriptores ORB para cada imagen de entrada por medio del método *compute*() de OpenCV.

Parte 5: Programación de los calces y su visualización

Para esta parte se utilizó un objeto *BFMatcher (Brute Force Matcher)* de OpenCV, el cual mediante el método *match()* permite, a partir de los descriptores obtenidos para ambas imágenes, guardar en un vector todos los calces realizados entre los puntos de interés. En la figura 8 se puede apreciar la implementación de esta parte, donde el vector de calces de guarda en la variable *matches*.

```
// Hacer matching
BFMatcher matcher(NORM_HAMMING);
vector<DMatch> matches;
matcher.match(descrleft, descrright, matches);
```

Figura 8: Cálculo de calces a partir de los descriptores obtenidos para ambas imágenes.

Parte 6: Estimación de la transformación de homografía

Una vez obtenido el vector de calces, se procedió a generas dos vectores que almacenen los puntos de interés utilizados para cada *match* entre las imágenes. Luego, a partir de estos vectores y la función *findHomography()* de OpenCV se estimó la transformación de homografía que relaciona ambas imágenes utilizando RANSAC para desechar *outliers* entre los candidatos a calces, guardando la matriz de transformación como variable para ser utilizada en los siguientes pasos.

Parte 7: Obtención de la imagen fusionada

Finalmente, para obtener la imagen fusionada, se comienza generando una imagen vacía de tamaño igual al doble del tamaño de las imágenes de entrada, para asegurarse que la fusión cabría dentro de la matriz. Luego, se utilizó la función *warpPerspective()* de OpenCV para proyectar la imagen de la derecha sobre en canvas vacío utilizando la matriz de homografía obtenida en la parte anterior. Posteriormente, se procedió a copiar la imagen de la izquierda sobre el canvas utilizando como referencia la esquina superior izquierda de éste. Luego, para terminar, se guardó este canvas en un archivo.

Resultados

Archivos left1.png y right1.png

En las figuras 9, 10, 11, 12, y 13 se pueden apreciar las figuras originales, las imágenes de Harris de cada figura, los puntos de interés de cada una, los calces entre las imágenes, y la fusión resultante, respectivamente.

Figura 9: Archivos originales left1.png y right1.png.

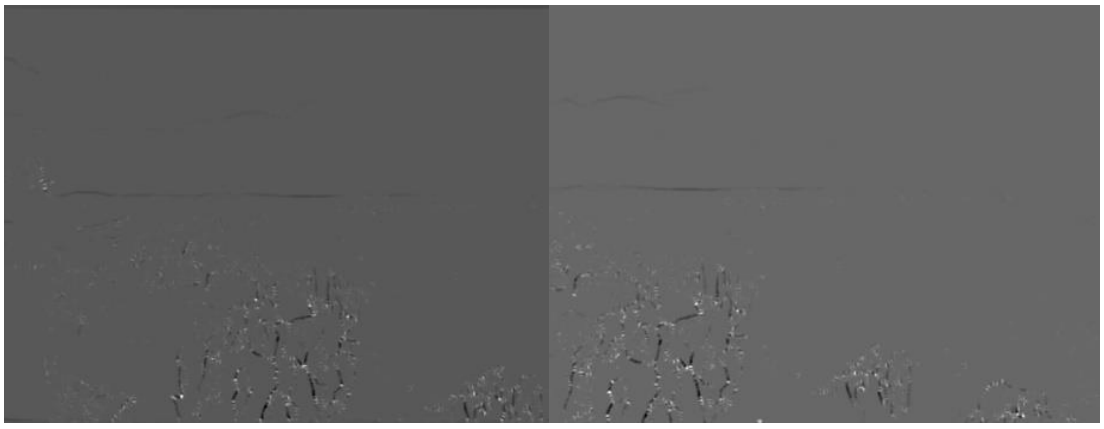

Figura 10: Imágenes de Harris de cada figura (izquierda y derecha, respectivamente).

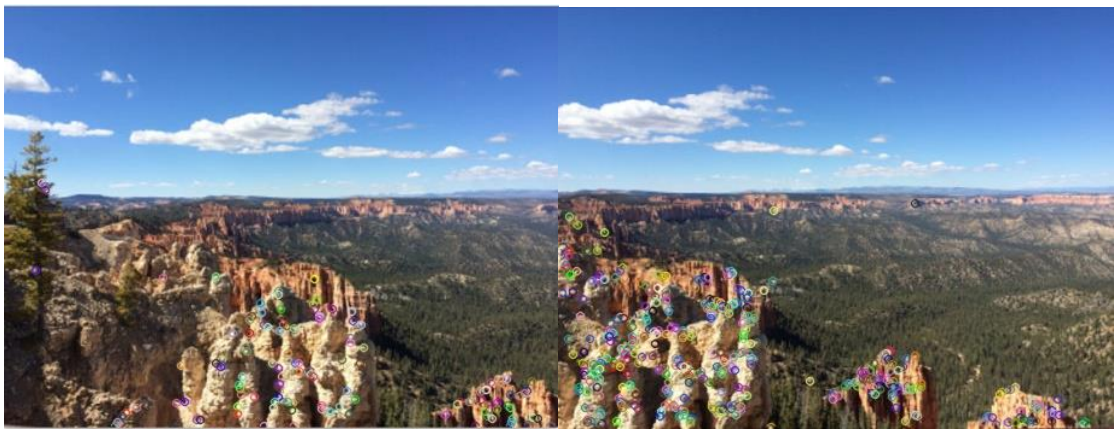

Figura 11: Puntos de interés detectados para ambas imágenes.

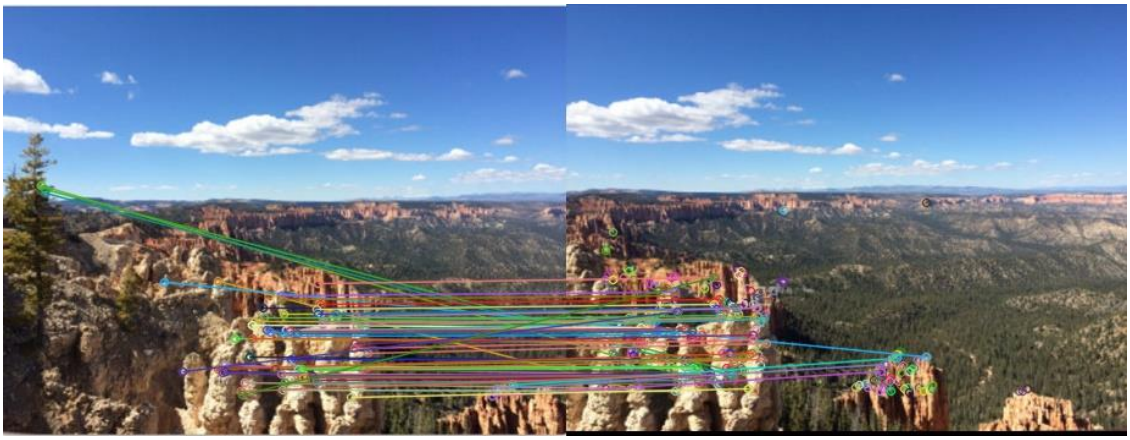

Figura 12: Calces obtenidos para los puntos de interés de la figura 10.

Figura 13: Imágen resultante de fusionar left1.png con right1.png.

Archivos left2.jpg y right2.jpg

En las figuras 14, 15, 16, 17, y 18 se pueden apreciar las figuras originales, las imágenes de Harris de cada figura, los puntos de interés de cada una, los calces entre las imágenes, y la fusión resultante, respectivamente.

Figura 14: Archivos originales left2.png y right2.png.

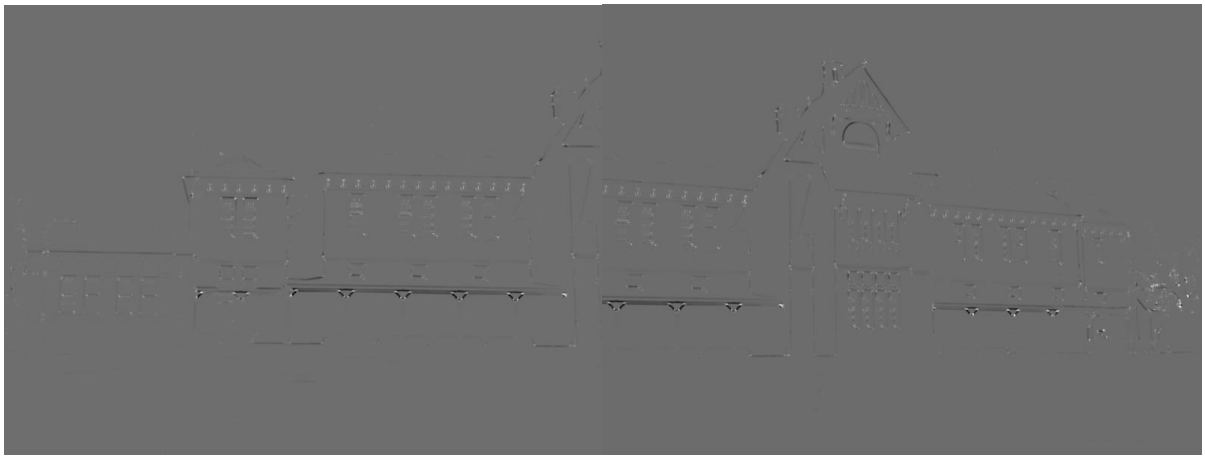

Figura 15: Imágenes de Harris de cada figura (izquierda y derecha, respectivamente).

Figura 16: Puntos de interés detectados para ambas imágenes.

Figura 17: Calces obtenidos para los puntos de interés de la figura 16.

Figura 18: Imágen resultante de fusionar left2.jpg con right2.jpg.

Archivos left3.jpg y right3.jpg

En las figuras 19, 20, 21, 22, y 23 se pueden apreciar las figuras originales, las imágenes de Harris de cada figura, los puntos de interés de cada una, los calces entre las imágenes, y la fusión resultante, respectivamente.

Figura 19: Archivos originales left3.png y right3.png.

Figura 20: Imágenes de Harris de cada figura (izquierda y derecha, respectivamente).

Figura 21: Puntos de interés detectados para ambas imágenes.

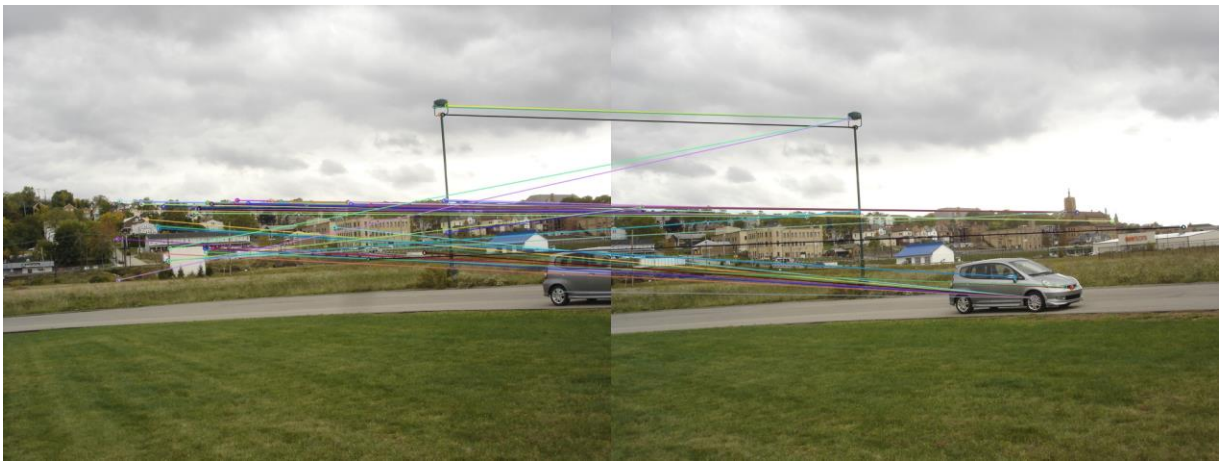

Figura 22: Calces obtenidos para los puntos de interés de la figura 21.

Figura 23: Imágen resultante de fusionar left3.jpg con right3.jpg.

Archivos left4.jpg y right4.jpg

En las figuras 24, 25, 26, 27, y 28 se pueden apreciar las figuras originales, las imágenes de Harris de cada figura, los puntos de interés de cada una, los calces entre las imágenes, y la fusión resultante, respectivamente.

Figura 24: Archivos originales left4.png y right4.png.

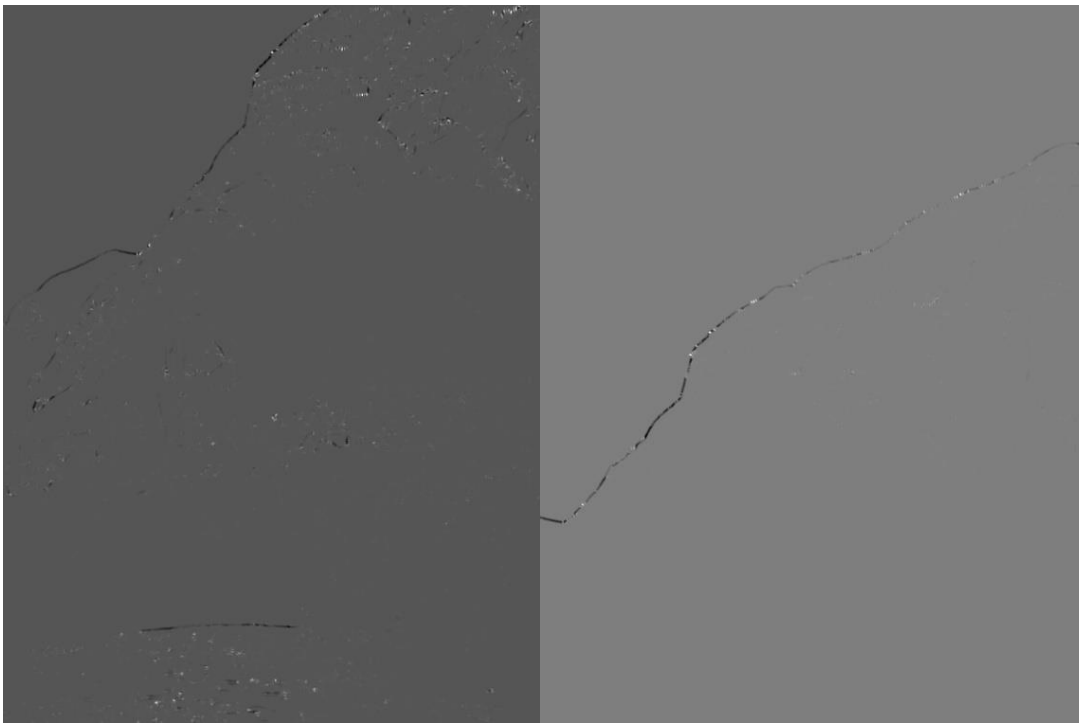

Figura 25: Imágenes de Harris de cada figura (izquierda y derecha, respectivamente).

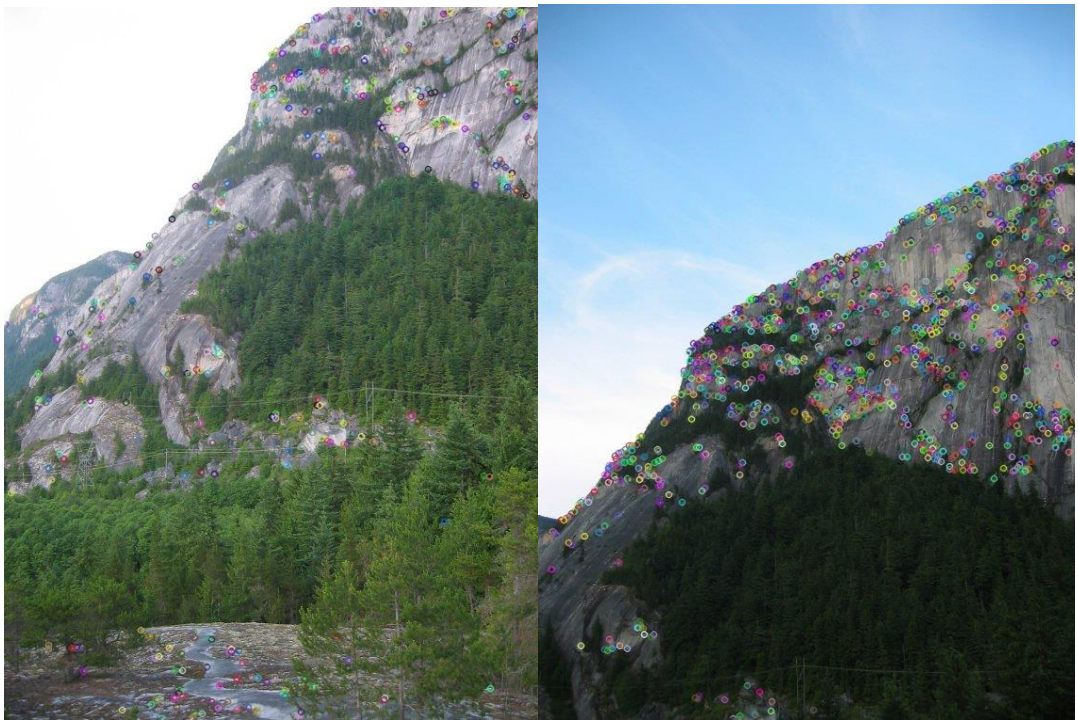

Figura 26: Puntos de interés detectados para ambas imágenes.

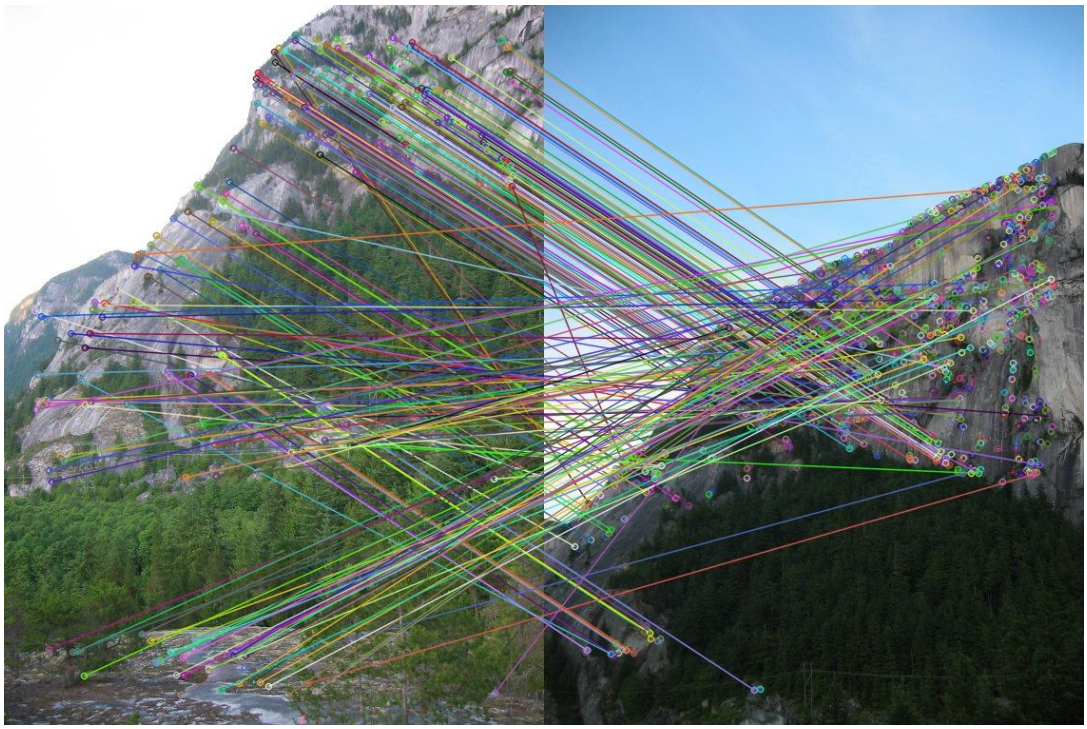

Figura 27: Calces obtenidos para los puntos de interés de la figura 26.

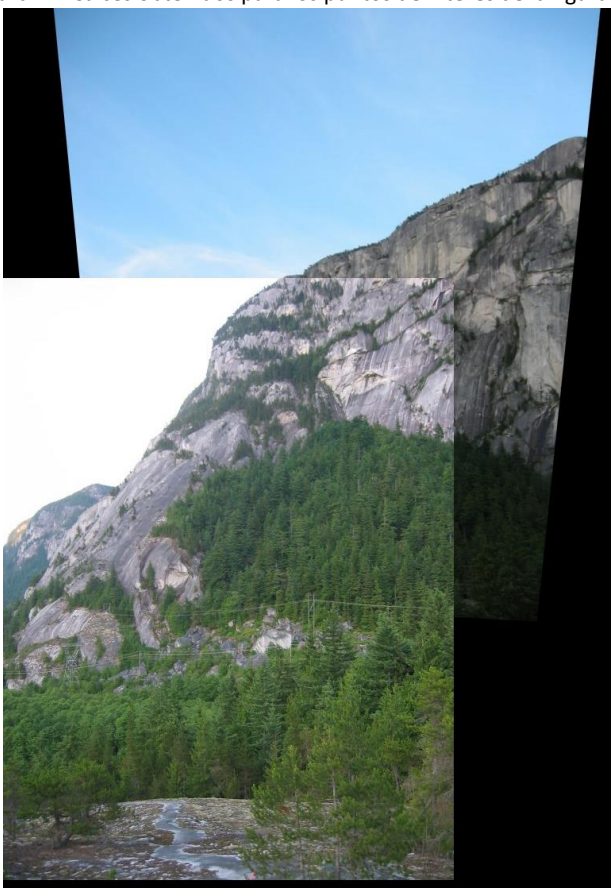

Figura 23: Imágen resultante de fusionar left4.jpg con right4.jpg.

Análisis de los Resultados

Como se puede observar en las imágenes, el sistema fue exitoso en fusionar cada par de imágenes al conseguir una transformación de homografía adecuada. Sin embargo, se deben notar un par de detalles: En primer lugar, para obtener los resultados mostrados, se tuvo que ir modificando el umbral de la función *getHarrisPoints()* en función de la imagen de entrada, puesto que habían ejemplos con esquinas más marcadas que otras. Sin embargo, a pesar de esto, la magnitud de la oscilación del umbral no fue muy grande, encontrándose todos los valores escogidos dentro del rango 100-120, con 115 siendo el más utilizado. Por otro lado, también es importante notar que para casos donde la fusión de daba en un sentido vertical, como ocurrió con las imágenes left4.jpg y right4.jpg, la imagen proyectada por la transformación de homografía solía quedar fuera del canvas (se le asignaban coordenadas negativas). Debido a esto, fue necesario incorporar manualmente una traslación en el eje y a la matriz de transformación para estos ejemplos, pero es importante notar que esta solución no sería factible para aplicaciones reales donde se quiera procesar muchas imágenes en poco tiempo, por lo que sería interesante buscar métodos que puedan detectar tanto el umbral como la magnitud de la traslación en Y de manera automática.

Conclusiones

En el presente trabajo se aplicaron correctamente las herramientas de filtros de Harris, descriptores ORB, y transformaciones de homografía para resolver el problema de alineamiento de imágenes panorámicas, pudiendo visualizar cómo es que cada técnica aporta en resolver una parte aislada del problema.

A partir de las actividades realizadas en esta tarea, se pudo apreciar cómo es el procedimiento de implementación de algoritmos de detección de puntos de interés, computación de descriptores locales, y generación de calces, a partir del ejemplo específico de la fusión de imágenes mediante transformaciones de homología. Además, esta experiencia también permitió estudiar con mayor cercanía cuáles son las variables que un programador puede manipular para mejorar sus resultados obtenidos con estas herramientas, cuyo correcto manejo dependerá fuertemente del tipo de datos con el que se trabaje, y no existen respuestas universalmente correctas.

En cuanto a las dificultades encontradas, se destaca principalmente la utilización de C++, que si bien es un lenguaje que permite una rápida ejecución de los algoritmos, posee una gran desventaja al no poder expresar claramente la razón de un error en un código, problema que se hace más grave a medida que el código se hace más grande.

Finalmente, como se mencionó en la sección de análisis, los resultados corresponden con los teóricamente esperados, ya que el sistema logra encontrar una transformación de homografía que permite la correcta fusión de las dos imágenes, sin embargo presenta limitaciones en cuanto a la necesidad de fijar ciertos parámetros manualmente dependiendo del tipo de imagen que se introduce al algoritmo, por lo que los resultados podrían mejorarse si se implementara un método que permita lograr un ajuste automático de estos parámetros según el tipo de imagen que se introduzca, además de tratar de manera más precisa el tema del tamaño del canvas de la imagen fusionada, para evitar la aparición de zonas negras muy grandes.