
Alineamiento de imágenes usando puntos Harris y descriptores ORB

Profesor:

Javier Ruiz del Solar

Estudiante:

Giovanni Pais L.

Auxiliar:

Patricio Loncomilla

Ayudantes:

Nicolás Cruz

Francisco Leiva

Fecha:

04 de Noviembre de 2018

Índice

1. Introducción	1
2. Marco Teórico	2
2.1. Filtro de Harris	2
2.2. Descriptor ORB	2
2.3. Transformación de Homografía	3
2.4. RANSAC	4
3. Desarrollo	5
3.1. Implementación	5
3.1.1. Filtro de Harris	5
3.1.2. Obtención puntos de Harris	6
3.1.3. Descriptores ORB	7
3.1.4. Matches	8
3.1.5. Obtención transformada	9
3.1.6. Proyección y Fusión	9
3.2. Resultados	10
3.2.1. Imagen 1: acantilado	11
3.2.2. Imagen 2: casa	15
3.2.3. Imagen 3: auto	18
3.2.4. Imagen 4: cerro	21
4. Conclusiones	28

Índice de figuras

1.	Matriz de transformación homográfica	3
2.	RANSAC aplicado en detección de líneas	4
3.	Imágenes originales	11
4.	Imágenes luego de aplicar filtro de Harris	11
5.	Puntos de interés encontrados usando el filtro de Harris	12
6.	Matches encontrados usando los puntos de interés	12
7.	Matches seleccionados luego de filtrar los mejores	13
8.	Imágenes alineadas luego de proyectar usando transformación encontrada	14
9.	Imágenes originales	15
10.	Imágenes luego de aplicar filtro de Harris	15
11.	Puntos de interés encontrados usando el filtro de Harris	16
12.	Matches encontrados usando los puntos de interés	16
13.	Matches seleccionados luego de filtrar los mejores	17
14.	Imágenes alineadas luego de proyectar usando transformación encontrada	17
15.	Imágenes originales	18
16.	Imágenes luego de aplicar filtro de Harris	18
17.	Puntos de interés encontrados usando el filtro de Harris	19
18.	Matches encontrados usando los puntos de interés	19
19.	Matches seleccionados luego de filtrar los mejores	20
20.	Imágenes alineadas luego de proyectar usando transformación encontrada	20
21.	Imágenes originales	21
22.	Imágenes luego de aplicar filtro de Harris	22
23.	Puntos de interés encontrados usando el filtro de Harris	23
24.	Matches encontrados usando los puntos de interés	24
25.	Matches seleccionados luego de filtrar los mejores	25
26.	Imágenes alineadas luego de proyectar usando transformación encontrada	26

1. Introducción

El objetivo de la tarea es alinear 2 imágenes de un mismo escenario tomada de distintas ubicaciones. Para esto se debe aplicar filtro de Harris a ambas imágenes para encontrar puntos de interés (esquinas) y generar descriptores ORB. Tomando estos descriptores se busca match entre ambas imágenes y se seleccionan los mejores para encontrar la transformación asociada entre ambas imágenes. Finalmente al tener la transformación se proyecta la imagen derecha y se combinan ambas imágenes obteniendo las imágenes alineadas.

Para el desarrollo de la tarea se utiliza la librería OpenCV en el lenguaje C++, principalmente para cargar las imágenes, trabajar con ellas y mostrarlas. Se deben usar las herramientas de OpenCV para implementar el filtro de Harris, luego generar los descriptores y finalmente realizar la proyección. Todo esto se debe aplicar en 4 pares de imágenes, de las cuales se muestran resultados en este informe. La compilación se realiza a través de cmake usando una configuración de cmake similar a la primera tarea.

A continuación en este informe se encuentran las siguientes secciones:

1. Marco Teórico: Se explican brevemente los conceptos teóricos utilizados en la realización de la tarea, como son explicar el filtro de Harris, descriptores ORB, la transformación de homografía y describir el algoritmo RANSAC para inliers.

2. Desarrollo: Se explica la implementación del filtro de Harris y la función para obtener los puntos de interés para luego construir los descriptores. Luego explica la construcción de descriptores y como se hace match entre ellos. Finalmente se explica la obtención de la transformada asociada y como se fusionan las funciones. Se muestran imágenes de las distintas etapas del proceso para los 4 pares de imágenes, en torno a estos resultados se realiza un análisis de la alineación.

3. Conclusiones: Finalmente se concluye a partir del análisis general de los resultados, el efecto de algunos parámetros y como se cumple el objetivo de la tarea. También se concluyen sobre los conocimientos adquiridos y dificultades.

2. Marco Teórico

2.1. Filtro de Harris

El filtro de Harris corresponde a un filtro detector de esquinas, por esto una utilidad que tiene es generar los puntos de interés para construir descriptores locales. Este filtro se basa en el cálculo de los gradientes sobre la imagen original suavizada con filtros Gaussianos, luego de obtener los gradientes se construye una matriz de momentos. Las componentes de esta matriz se suavizan y se utiliza el determinante y la traza de la matriz para construir un puntaje de ser esquina (cornerness) que si supera un umbral se detecta como esquina.

A continuación se muestran algunas ecuaciones que resumen el algoritmo:

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

La matriz L corresponde a la imagen I suavizada, donde N corresponde al filtro Gaussiano.

$$\mu = N(x - x_0, y - y_0, \sigma_D) * \begin{pmatrix} \frac{dL}{dx} \frac{dL}{dx} & \frac{dL}{dx} \frac{dL}{dy} \\ \frac{dL}{dy} \frac{dL}{dx} & \frac{dL}{dy} \frac{dL}{dy} \end{pmatrix}$$

Luego se construye la matriz de momentos μ conformada por los gradientes de la matriz L en ambas direcciones. Los momentos obtenidos se suavizan usando otro filtro Gaussiano.

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

Finalmente se define el puntaje de ser esquina (cornerness) usando la ecuación mostrada previamente.

Para obtener los puntos de interés se buscan los puntos que son máximos locales de cornerness y se verifica si superan un umbral fijado para determinar si son esquinas o no.

2.2. Descriptor ORB

ORB (Oriented Fast and rotated BRIEF) son descriptores locales que tienen la característica de ser rápidos y robustos por lo que tiene ventajas en ciertas aplicaciones de visión computacional y robótica, estos usan FAST para hacer la detección de puntos de interés (keypoint) y utiliza

BRIEF(Binary Robust Independent Elementary Features) para la construcción de los descriptores locales usando los puntos de interés generados.

Para la tarea se utilizarán estos descriptores pero la detección de puntos de interés se realizará utilizando filtro de Harris implementado usando OpenCV y explicado en el desarrollo del informe.

2.3. Transformación de Homografía

Las transformaciones de homografía corresponde a transformaciones geométricas que relacionan 2 imágenes, existen distintas transformaciones homográficas como son: traslación, rotación, simetría y escalamiento. Estas también pueden ser la combinación de las transformaciones anteriores.

En el reconocimiento de imágenes estas transformaciones pueden usarse para relacionar el contenido de imágenes, por ejemplo si tengo un objeto que quiero reconocer como una botella y luego se analiza la imagen de una mesa con objetos, se puede hallar la misma botella pero dado que la imagen se toma de otra posición y orientación la botella en la mesa podría estar relacionada la botella original con la botella de la mesa con una transformación homográfica.

Estas transformaciones se pueden caracterizar por una matriz de transformación, con lo que es posible hacer una proyección de una imagen usando la transformación homográfica caracterizada en la matriz.

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$\mathbf{x}' = \mathbf{Hx}$

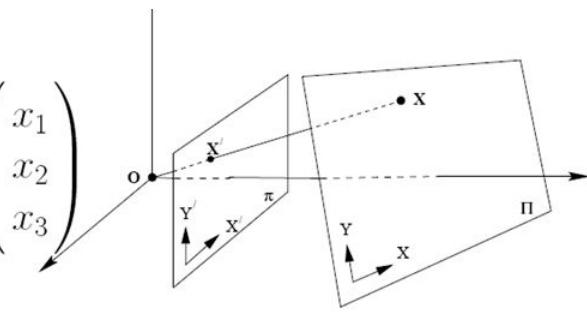


Figura 1: Matriz de transformación homográfica

En la Figura 1 se observa que el multiplicar la matriz H que caracteriza la transformación homográfica al multiplicar la imagen x , se obtiene la imagen proyectada x' .

2.4. RANSAC

El algoritmo RANSAC (Random sample consensus) se utiliza para verificar la pertenencia de muestras a un modelo utilizando la cantidad de inliers pertenecientes a modelos aleatorios. En el contexto de detección de calces se tienen muchos calces entre los descriptores locales de 2 imágenes, pero existen calces equivocados que se deben eliminar. Para esto el algoritmo toma una muestra mínima de puntos dependiendo del tipo de modelo a verificar, en este caso si se aplica a la verificación de calces se toman 3 calces mínimos (aleatorios) y construye un modelo a partir de estas muestras, en este caso el modelo corresponde a la matriz de transformación asociada a estos calces. Teniendo esta matriz de transformación se ve cuantos calces comparten también esta transformación (inliers) y cuantos no (outliers), a partir de la proporción de inliers y outliers se verifica si se supera cierto umbral de concenso, en caso se superarlo se acepta como una calce valido. En caso de no superar el umbral, se repite el proceso tomando una muestra aleatoria nueva.

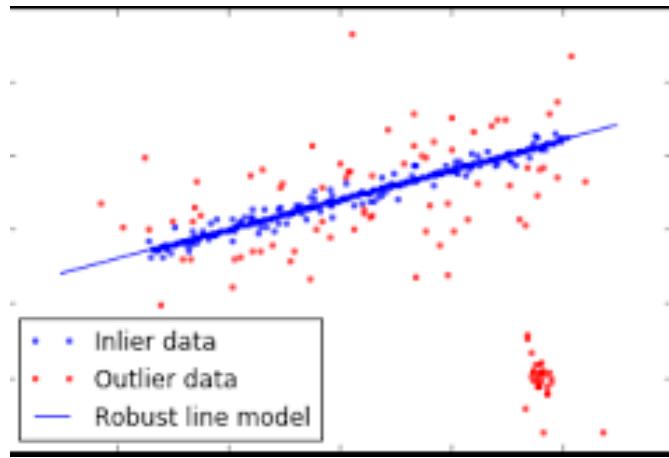


Figura 2: RANSAC aplicado en detección de lineas

En la Figura 2 se ejemplifica el algoritmo para el caso de detección de rectas, en este caso se quiere detectar rectas entre ese conjunto de puntos por lo que se toman 2 puntos al azar y se construye el modelo (ecc. de la recta). Se cuentan el número de inliers que corresponden a los puntos azules que pertenecen a la recta generada por los 2 puntos iniciales y el numero de outliers que no pertenecen a la recta, dependiendo de la proporción entre inliers y outliers se validará si hay una recta con ese modelo.

3. Desarrollo

3.1. Implementación

A continuación se muestra como se realizó la implementación de la alineación de imágenes, se explica por etapas y se añaden fragmentos importantes del código.

3.1.1. Filtro de Harris

La implementación del filtro de Harris se realizó en la función **harrisFilter** la cual recibe una imagen *input_raw* y retorna la imagen con el filtro aplicado *harris_n*.

Para implementar el filtro de Harris se pasa a escala de grises la imagen de entrada y se suavizan las imágenes usando filtro pasa-bajo Gaussiano para luego sacar los gradientes en dirección horizontal y vertical.

```
1 Mat input;
2 cvtColor(input_raw, input, CV_BGR2GRAY);
3 // 1) Suavizar la imagen de entrada
4 Mat input.blur;
5 GaussianBlur(input, input.blur, Size(3,3), 1);
6 // 2) Calcular derivadas ix e iy
7 Mat grad_x, grad_y;
8 Sobel( input.blur, grad_x, ddepth, 1, 0, 3);
9 Sobel( input.blur, grad_y, ddepth, 0, 1, 3);
```

Código 1: Conversión a escala de grises y suavización

```
1 //3) Calcular momentos ixx, ixy, iyy
2 Mat ixx = grad_x.mul(grad_x);
3 Mat ixy = grad_x.mul(grad_y);
4 Mat iyy = grad_y.mul(grad_y);
5 //4) Suavizar momentos ixx, ixy, iyy
6 GaussianBlur(ixx, ixx, Size(3,3), 1);
```

```
7 GaussianBlur(ixy,ixy,Size(3,3),1);  
8 GaussianBlur(iyy,iyy,Size(3,3),1);
```

Código 2: Cálculo de momentos y suavización

```
1 //5) Calcular harris como: det(m) - 0.04*Tr(m)^2, con:  
2 Mat det = ixx.mul(iyy) - ixy.mul(ixy);  
3 Mat Tr = ixx + iyy;  
4 Mat harris = det - 0.04*Tr.mul(Tr);  
5 //Ademas se debe transformar la imagen de harris para que quede en el  
rango 0-255  
6 Mat harris_n;  
7 normalize(harris, harris_n, 0, 255, NORM_MINMAX, ddepth);  
8 harris_n.convertTo(harris_n, CV_8UC1);  
9 return harris_n;
```

Código 3: Cálculo de momentos y suavización

3.1.2. Obtención puntos de Harris

Para la obtención de los puntos de interés se implementa la función `getHarrisPoints` que recibe la imagen filtrada con Harris `harris` junto al threshold `val` que determinará si un punto corresponde o no a una esquina y retorna un vector de keypoints `points`. Dentro de la función se buscan máximos locales recorriendo la imagen con ventanas de 5x5 y encontrando el máximo de cada una, luego se verifica si el máximo encontrado es superior a `val`.

```
1 for (int i=0; i<windows_r;i++){  
2     for (int j=0; j<windows_c;j++){  
3         if(harris.at<char>(r*windows_r+i,c*windows_c+j)>h_max){  
4             h_max=harris.at<char>(r*windows_r+i,c*windows_c+j);  
5             r_max=r*windows_r+i;  
6             c_max=c*windows_c+j;  
7     }
```

```
8     }
9 }
10 // 2) Mayores que el umbral val
11 if(harris.at<char>(r_max,c_max)>(char)val){
12     KeyPoint KP=KeyPoint(c_max,r_max,1);
13     points.push_back(KP);
14 }
```

Código 4: Algoritmo que se ejecuta en cada ventana para encontrar máximo y ver si punto es una esquina

3.1.3. Descriptores ORB

Para la implementación de los descriptores ORB se crea el objeto *orb* de OpenCV, luego de esto se llama a la función **getHarrisPoints** para obtener los puntos que se usaran en ORB. Los threshold ingresados a la función son definidos en arreglos dado que para cada una de las 4 imágenes el umbral es distinto. Posterior a esto se dibujan los puntos en las imágenes usando la función **drawKeypoints** y se guardan usando **imwrite**. Para la obtención de los descriptores se utiliza el método **compute** del objeto *orb*.

```
1 std::string const s = ("ORB");
2 Ptr<ORB> orb = ORB::create(s);
3
4 int vall[4]={110,115,100,100};
5 int valr[4]={115,120,90,80};
6
7 vector<KeyPoint> pointsleft = getHarrisPoints(harrisleft, vall[idx]);
8 vector<KeyPoint> pointsright = getHarrisPoints(harrisright, valr[idx])
9 ;
10 Mat l_points,r_points;
11 drawKeypoints(imleft,pointsleft,l_points);
12 drawKeypoints(imright,pointsright,r_points);
```

```
13 //-----
14 imwrite(n+"l_points.jpg", l_points); // Grabar imagen
15 imwrite(n+"r_points.jpg", r_points); // Grabar imagen
16
17 Mat descrleft, descrright;
18 orb->compute(imleft, pointsleft, descrleft);
19 orb->compute(imright, pointsright, descrright);
```

Código 5: Construcción de descriptores ORB

3.1.4. Matches

Para construir los match se crea el objeto **BFMatcher** *matcher* usando la norma de Hamming como criterio, usando el método **match** se genera el vector *matches* con los matches. Se toman estos matches y se procede a dibujarlos en las imágenes usando la función **drawMatches**. Ahora se tienen los matches de los descriptores obtenidos, pero hay descriptores que estan malos por lo que se procede a filtrar los mejores. Para filtrarlos se ordenan estos, donde la función sort los ordena según que tan buenos son considerando la distancia entre ellos, luego se eliminan los peores del vectores *matches*. Para la tarea se dejó el 20 % de los mejores matches solamente y se vuelven a dibujar en las imágenes.

```
1 vector<DMatch> matches;
2 BFMatcher matcher(NORM_HAMMING);
3 matcher.match(descrleft, descrright, matches);
4 // (4) -----
5 Mat img_matches;
6 drawMatches(imleft, pointleft, imright, pointright, matches, img_matches);
7 imwrite(n+"matches.jpg", img_matches); // Grabar imagen
8 //Filtrar MATCHES
9 sort(matches.begin(), matches.end());
10 float K=0.2;
11 int KMatches = matches.size()*K;
```

```
12 matches.erase(matches.begin() + KMatches, matches.end());
13 drawMatches(imleft, pointsleft, imright, pointsright, matches, img_matches)
    ;
14 imwrite(n+"matchesF.jpg", img_matches);
```

Código 6: Obtención de matches y selección de los mejores

3.1.5. Obtención transformada

Para obtener la transformada asociada entre ambas imágenes se generan los pares de puntos de los matches seleccionados y luego pasando estos puntos a la función **findHomography** se obtiene la matriz de transformación H

```
1 vector<Point2f> points1, points2;
2 for (int i=0;i<matches.size();i++){
3     points1.push_back(Point2f(pointsleft[ matches[i].queryIdx ].pt));
4     points2.push_back(Point2f(pointsright[ matches[i].trainIdx ].pt));
5 }
6 // (6) -----
7 Mat H = findHomography( points2, points1, CV_RANSAC );
```

Código 7: Obtención de matriz de transformación H

3.1.6. Proyección y Fusión

Tomando la matriz obtenida previamente se procede a hacer la proyección de la imagen derecha a través de la función **warpPerspective** es importante destacar que las dimensiones de la imagen de salida debe ser suficiente para albergar ambas imágenes alineadas, dado que el traslape de las imágenes es distinto se definieron factores de permutación para los tamaños de cada imagen, así logrando que lo que sobra de la imagen en negro sea menor.

Teniendo la imagen derecha proyectada se procede a copiar esta imagen proyectada y la imagen izquierda original en la imagen alineada *imfused*, esta es la imagen final con las imágenes alineadas y se guarda como resultado

```
1 // (8) -----
2 Mat r_warp;
3 warpPerspective(imright, r_warp, H, Size(imleft.cols*wfactor[idx],
imleft.rows*hfactor[idx]));
4
5 Mat imfused(r_warp.size(), CV_8UC3);
6 Mat imfused_left = imfused(Rect(0, 0, imleft.cols, imleft.rows));
7 Mat imfused_right = imfused(Rect(0, 0, r_warp.cols, r_warp.rows));
8 r_warp.copyTo(imfused_right);
9 imleft.copyTo(imfused_left);
10 imwrite(n+"alineadas.jpg", imfused); // Grabar imagen
```

Código 8: Proyección de imagen derecha y union de imagenes en imfused

3.2. Resultados

A continuación se muestran los resultados obtenidos para los 4 pares de imágenes, se muestran imágenes de distintas etapas del proceso.

3.2.1. Imagen 1: acantilado



Figura 3: Imágenes originales

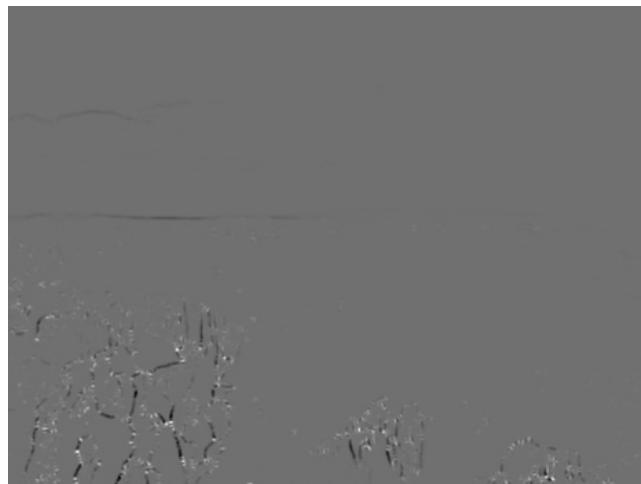


Figura 4: Imágenes luego de aplicar filtro de Harris

Se observa en la imagen filtrada que la zona rocosa y el horizonte se distinguen del resto.

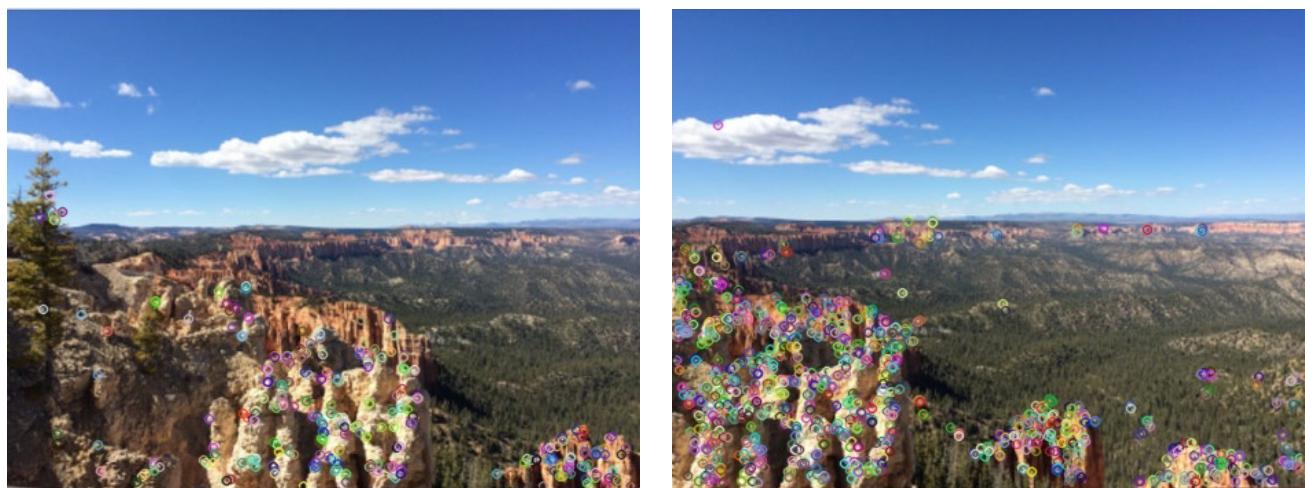


Figura 5: Puntos de interés encontrados usando el filtro de Harris

Se observa que los puntos de interés se concentran en la zona rocosa dado que posee mucha textura, esto se observa en ambas imágenes por lo que los matches estarán concentrados en esa área.



Figura 6: Matches encontrados usando los puntos de interés



Figura 7: Matches seleccionados luego de filtrar los mejores

Se observa una gran cantidad de matches en la zona rocosa como se estimaba dada la gran concentración de puntos de interés en ambas imágenes, antes de filtrar se observan también matches en el horizonte y distintos matches cruzado erroneos. Luego de seleccionar el 20 % de mejores matches se observa que todos están en la zona rocosa.



Figura 8: Imágenes alineadas luego de proyectar usando transformación encontrada

En la imagen alineada se observa que se logra exitosamente la alineación entre ambas, incluso si la segunda imagen estaba tomada de más atrás añadiendo parte de la imagen más abajo y completando parte de la imagen panorámica. El hecho de que los bordes de la imagen proyectada no sean cuadrados tiene relación con que la transformación no es muy simple, presentando inclinación y zoom.

3.2.2. Imagen 2: casa



Figura 9: Imágenes originales

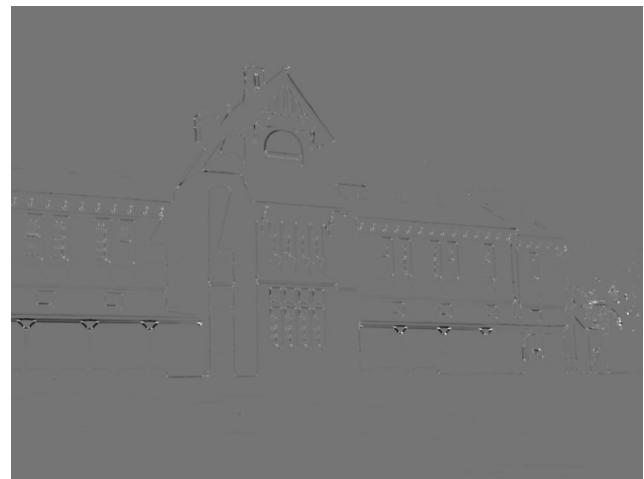
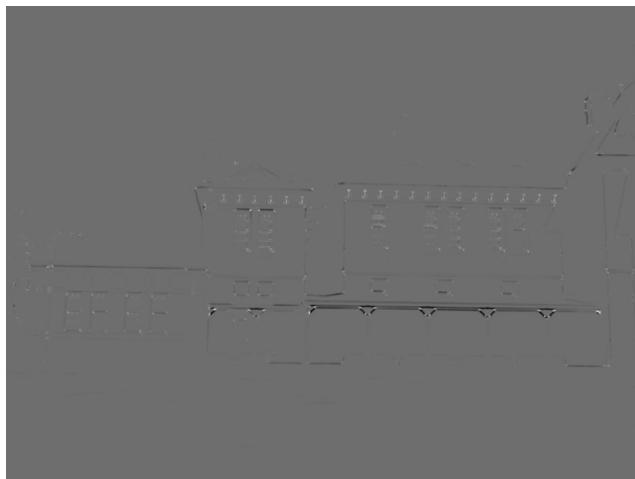


Figura 10: Imágenes luego de aplicar filtro de Harris

Se observa que la el edificio presenta una gran cantidad de esquinas dado que tiene ventanas, diferencias en techos y muros.



Figura 11: Puntos de interés encontrados usando el filtro de Harris

Se obtienen puntos de interés concentrados en las esquinas de las ventanas y el cambio de muro con el techo que tiene franjas.



Figura 12: Matches encontrados usando los puntos de interés

Dado que el edificio tiene patrones similares de construcción se tienen muchos matches cruzados y erroneos.



Figura 13: Matches seleccionados luego de filtrar los mejores

Filtrando los matches se obtienen una gran cantidad de matches correctos concentrados en las ventanas.



Figura 14: Imágenes alineadas luego de proyectar usando transformación encontrada

La imagen resultante muestra el edificio completo, este no tiene marcada la interfaz en donde se juntan las imágenes y no se muestran bordes muy irregulares dado que la transformación entre ambas imágenes no es muy compleja, las imágenes parecen estar a una distancia similar con una traslación y pequeña rotación.

3.2.3. Imagen 3: auto



Figura 15: Imágenes originales

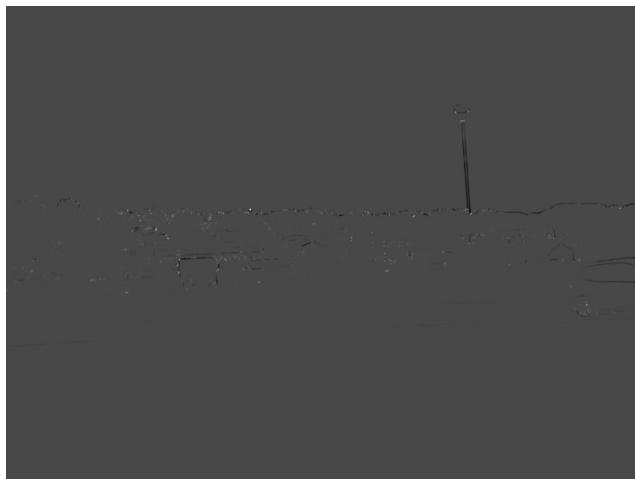


Figura 16: Imágenes luego de aplicar filtro de Harris

Se observan esquinas en el fondo de la imagen principalmente, dado que el auto está incompleto y el resto del terreno no tiene mucha textura.



Figura 17: Puntos de interés encontrados usando el filtro de Harris

La mayor concentración de puntos de interés están en el fondo donde se ve la ciudad, algunos en el auto y foco.

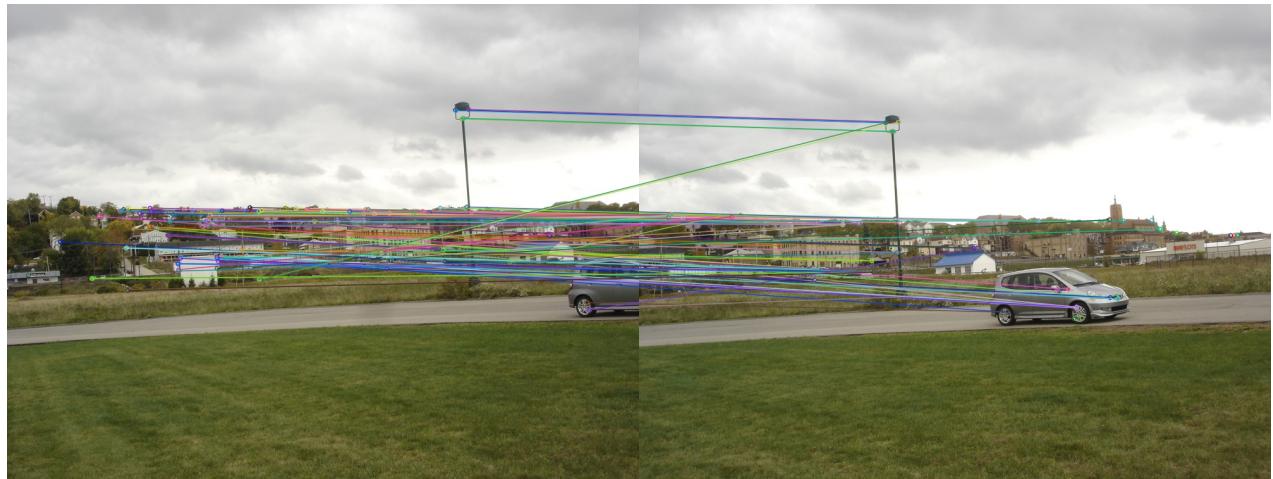


Figura 18: Matches encontrados usando los puntos de interés

Se observan pocos matches en general comparado con las otras imágenes, esto es dado el umbral que se fijó para esta imagen. Se puede observar que hay matches malos relacionados con el auto.



Figura 19: Matches seleccionados luego de filtrar los mejores

Luego del filtrado de matches se mantienen solo los matches del fondo de la imagen dado que se mantiene más en la traslación de la imagen al estar más lejos.



Figura 20: Imágenes alineadas luego de proyectar usando transformación encontrada

Se construye la imagen alineada logrando juntar ambas imágenes en el sector donde está el auto donde se termina la primera imagen. Se ve que la imagen tiene un cambio notorio de la iluminación,

esto se puede deber a que la rotación de la cámara al tomar la foto obtiene una mayor captura de luz debido a la orientación del sol.

3.2.4. Imagen 4: cerro

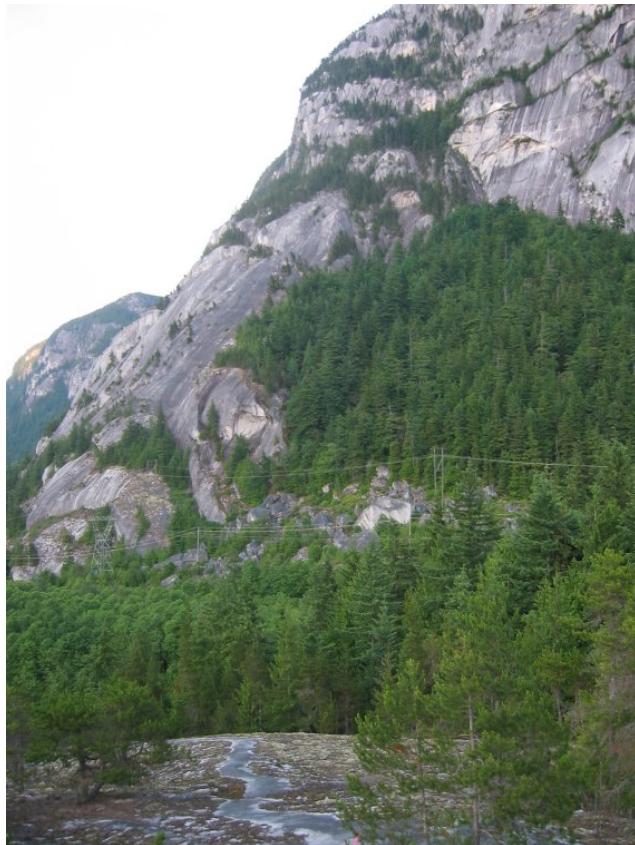


Figura 21: Imágenes originales

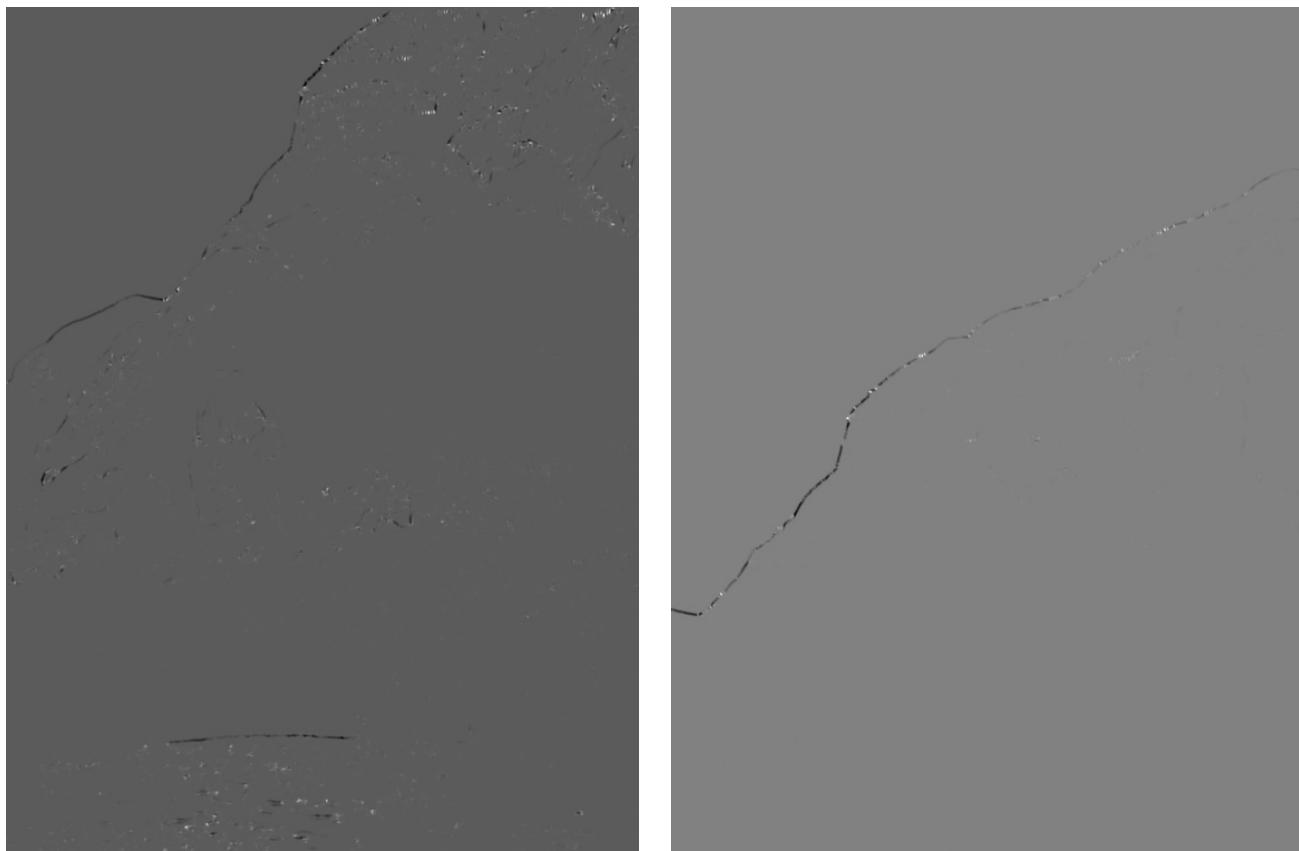


Figura 22: Imágenes luego de aplicar filtro de Harris

Se observa que los contornos del cerro al ser rocosos tienen muchas esquinas.

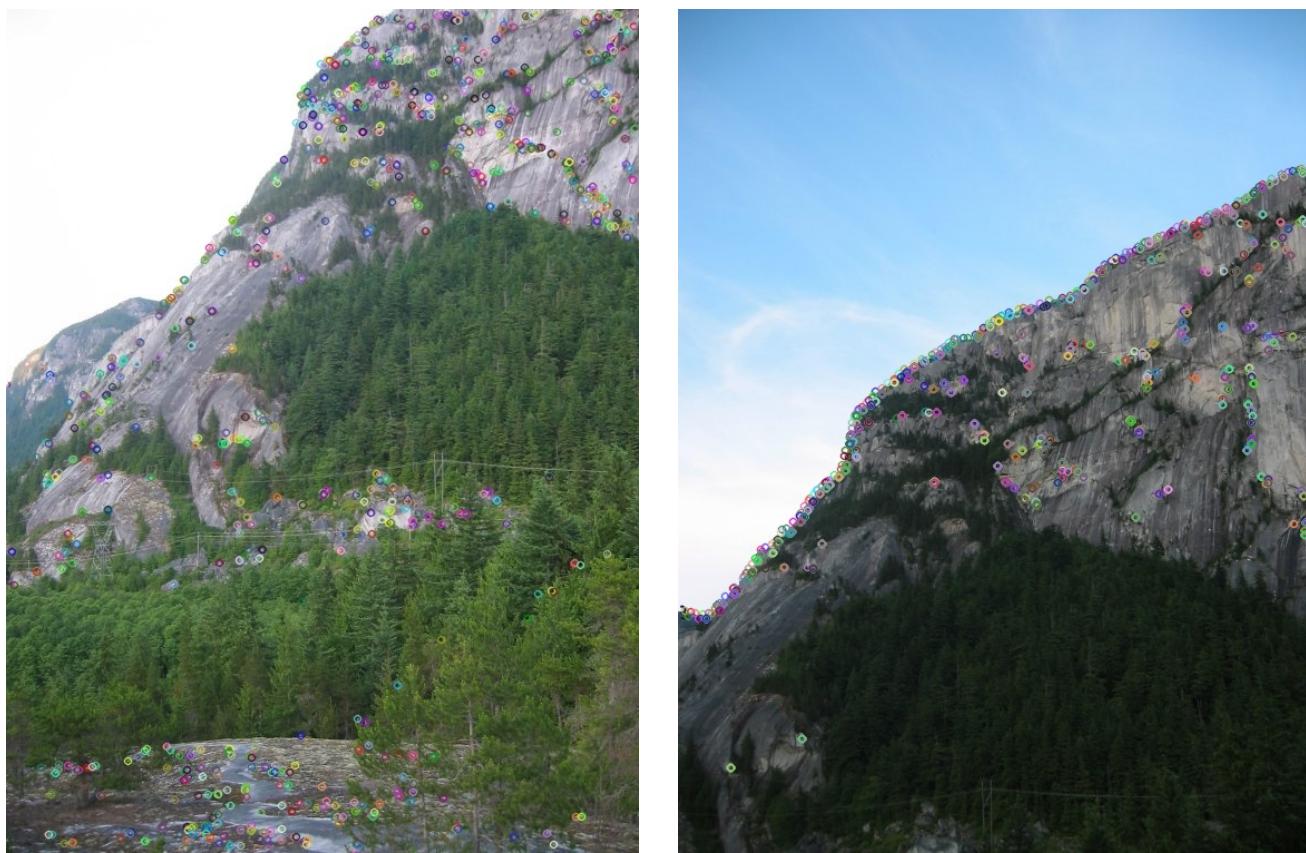


Figura 23: Puntos de interés encontrados usando el filtro de Harris

La mayor concentración de puntos de interés están en los bordes del cerros y los quiebres que tiene este.

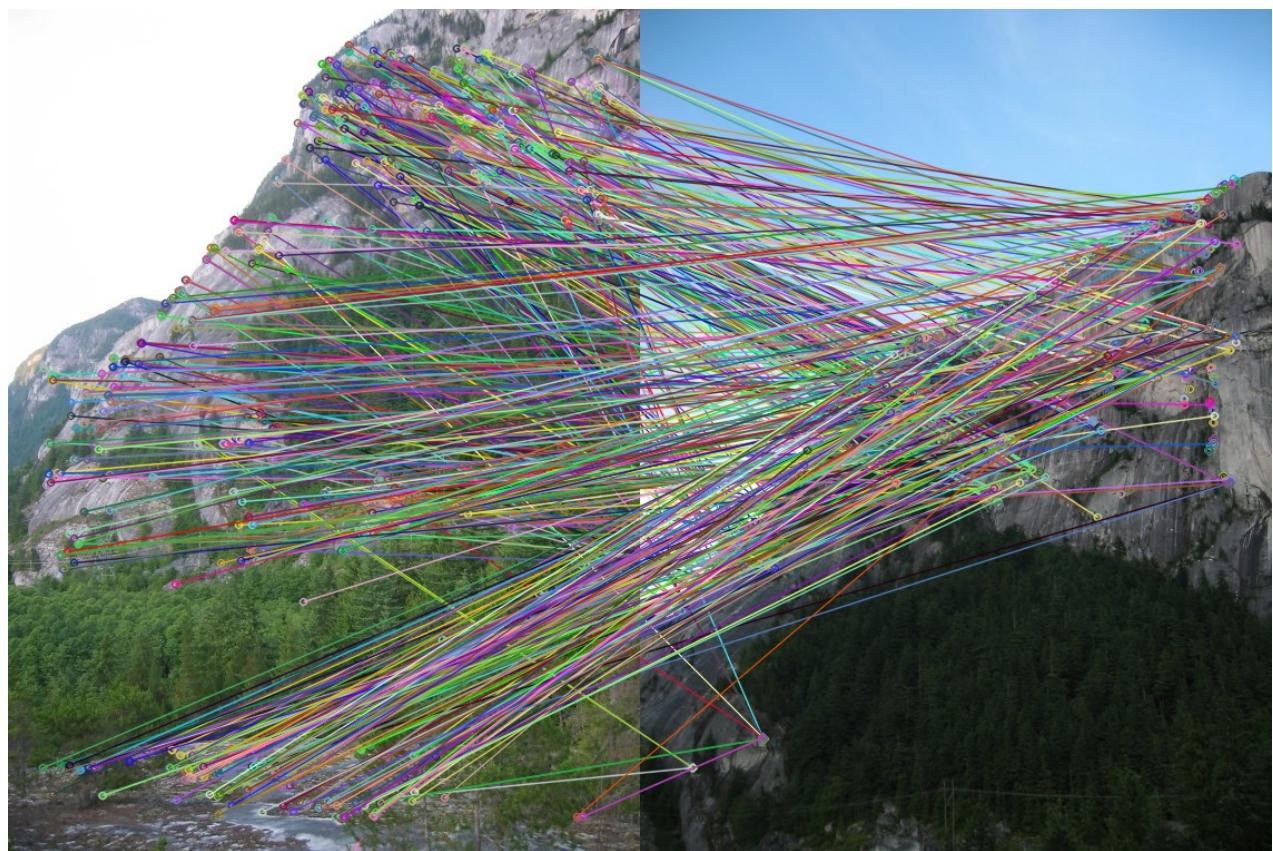


Figura 24: Matches encontrados usando los puntos de interés

Se puede observar una gran cantidad de matches en todas direcciones, lo que indica que una gran cantidad son matches erroneos.

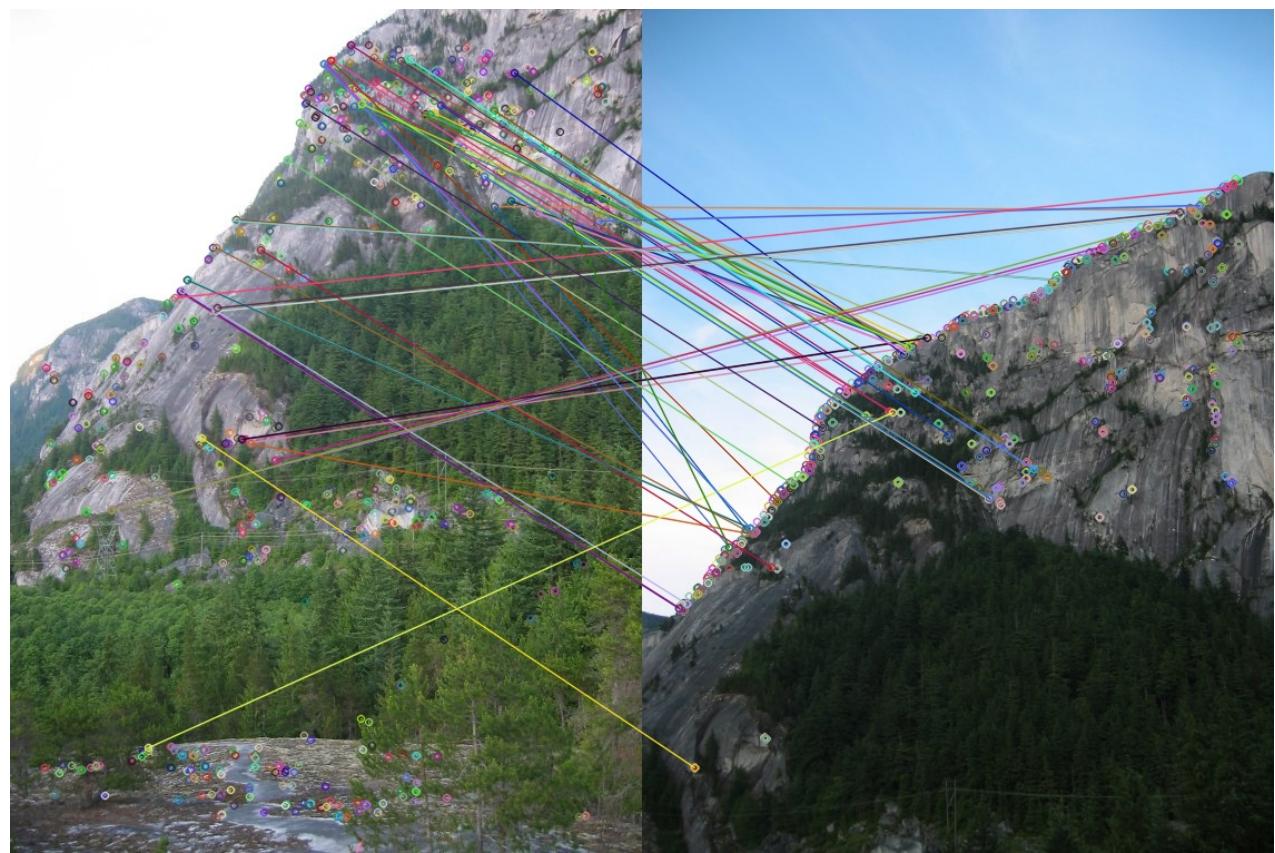


Figura 25: Matches seleccionados luego de filtrar los mejores

Luego de filtrar los matches se siguen observando algunos matches erroneos cruzados, esto debido a que las proporciones de matches buenos es baja y a pesar de filtrar el 20 % mejor siguen habiendo algunos erroneos. Esto se puede arreglar moviendo el threshold de los puntos de Harris y que porcentaje de los mejores se seleccionan.



Figura 26: Imágenes alineadas luego de proyectar usando transformación encontrada

Finalmente se observa que las imágenes se logran alinear pero se distingue la interfaz entre imágenes debido a que la segunda imagen es más oscura posiblemente dada la orientación del sol y la sombra. Por otro lado este par de imágenes presenta una diferencia de escala a diferencia de las otras y se puede ver que el algoritmo sigue funcionando. Otro punto a destacar es que una parte de la segunda imagen no se ve debido a que se usó la segunda imagen como referencia $(0,0)$ y la

otra parte de la segunda imagen está más arriba de la imagen izquierda, esto podría arreglarse desplazando ambas imágenes hacia abajo al momento de agregarlas a imfused.

4. Conclusiones

Se logró cumplir con el objetivo de la tarea, se lograron alinear las imágenes izquierdas y derechas encontrando la función homográfica entre ambas imágenes. La función se logró obtener al tener los mejores matches entre los descriptores locales ORB, con la variación que los puntos de interés utilizados fueron obtenidos a través del filtro de Harris.

Se obtuvieron como resultados las 4 imágenes alineadas, mostrando que para distintas transformaciones como rotación, traslación y escala este algoritmo funciona aceptablemente. En algunos casos se obtuvo una alineación casi imperceptible como el caso de la casa y en otras se nota la diferencia de luz entre las imágenes, lo que podría arreglarse modificando la distribución de contraste.

Por otro lado se logró aprender sobre el filtro de Harris, dado que se tuvo que implementar a mano usando las funciones de OpenCV, con esto entendiendo mejor los distintos pasos que este posee. El uso de descriptores locales ORB y como hacer matches usando OpenCV, dominando mejor la esta librería y aprendiendo nuevas herramientas. Las principales dificultades fueron el desconocimiento de la muchas funciones utilizadas para la tarea, esto se solucionó a través de investigación y leyendo la documentación de OpenCV.