Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Electrónica

Composición de Múltiples Imágenes de Microscopía

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

Fabián Alfaro Monge

Cartago, Junio de 2008

INSTITUTO TECNOLOGICO DE COSTA RICA

ESCUELA DE INGENIERIA ELECTRONICA

PROYECTO DE GRADUACIÓN

TRIBUNAL EVALUADOR

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Ing. Arys Carrasquilla Batista	Ing. Roberto Pereira Arroyo
Profesor lector	Profesor lector

Ing. Pablo Alvarado Moya

Profesor asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Cartago, Junio 2008

Declaratoria de autenticidad

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Cartago, 12/06/2008

Fabián Alfaro Monge

Céd: *3-0398-0012*

Resumen

El presente proyecto forma parte del desarrollo de una herramienta de visión por computadora que permita mejorar significativamente el tiempo de trabajo requerido para estudiar los nematodos. Estos son organismos microscópicos que viven en el suelo y gran cantidad de especies dañan los productos agrícolas pues se alimentan de estos. Los investigadores están tratando de perfeccionar el control de las plagas de nematodos por medio del control biológico, pero para esto requieren análisis de sus características y comportamiento en los cultivos.

Los nematólogos invierten gran cantidad de tiempo en analizar una muestra de nematodos pues el microscopio no les permite observar toda la escena en una captura. Entonces para estudiar un conjunto de individuos deben hacer el estudio por secciones, lo que es propenso a errores y requiere horas de trabajo.

En este proyecto se presenta una solución al problema planteado haciendo uso de las ventajas que brinda el procesamiento digital de imágenes. Obteniendo imágenes digitales de cada sección que se desea analizar, se puede implementar un algoritmo que permite crear una imagen completa de la escena. Para esto, se presenta un algoritmo capaz de realizar el registro de imágenes, es decir, ubicar dos imágenes en el mismo sistema de coordenadas por medio de una trasformación geométrica. Conociendo la transformada que se debe aplicar para alinear las imágenes, se presenta un algoritmo que permite obtener una imagen compuesta por las dos imágenes iniciales.

Palabras claves: nematodos, procesamiento digital de imágenes, registro de imágenes, composición de imágenes, mosaicing.

Abstract

This project forms part of the development of a vision tool that allows, via computer, to significantly improve the time required to study nematodes. Nematodes are microscopic organisms that live in the ground, and a great amount of this species damage agricultural products since they feed from them. Researchers are trying to enhance plague control of nematodes through biological control; however, this requires studying their characteristics and behavior in the crops.

Nematologists invest a great amount of time in analyzing a sample of nematodes due to the fact that the microscope does not allow to observe the complete scene in one shot. Therefore, to be able to study a group of individuals the work must be done in segments, which is error prone and requires a lot of work time.

What this project aims at is to significantly improve this current limitation when it comes to observing nematodes, by using the advantages that digital image processing provides. Through obtaining digital images of each section or segment that needs to be analyzed, an algorithm can be applied so that it creates a complete image of the scene. What is required for this specific task is the introduction of an algorithm that is capable of performing a registration of images that can locate two images in the same system of coordinates by means of a geometric transformation. Having knowledge of the geometric transformation that has to be applied to line up the images, an algorithm that stitches them up is used.

Keywords: nematodes, digital image process, image register, mosaicing

Dedicatoria

A mis padres Carlos Luis y María de los Ángeles por su apoyo incondicional durante toda la vida que me ha permitido llegar a finalizar esta carrera.

A mi novia Lucía por apoyarme en todo momento durante el desarrollo de este proyecto.

Agradecimiento

Al Dr. Ing. Pablo Alvarado Moya por permitirme desarrollar este proyecto y por toda la ayuda brindada durante la realización del mismo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1:Introducción

1.1 Entorno del proyecto

Los nematodos son organismos con un tamaño en el orden de los 200 micrómetros, imperceptibles a simple vista en su mayoría, de forma alargada cilíndrica. Algunos investigadores estiman que existen alrededor de un millón de especies de nematodos, a pesar de que hasta el momento la cantidad de especies descritas es del orden de las decenas de miles. No todos los nematodos son dañinos, hay gran cantidad que son beneficiosos para el suelo. Los nematodos que se alimentan de plantas lo hacen succionando el contenido de la célula, dañando principalmente las raíces, pues viven en el suelo [7].

Los cultivos en las zonas tropicales de América Latina se ven afectados por los nematodos, ya que viven mejor en suelos arenosos, con calor y riego abundante. Dentro de las plagas que dañan mayormente las raíces se encuentran los nematodos fitoparásitos. Estos afectan la producción agrícola de nuestro país, dañando alimentos como el plátano, las raíces y los tubérculos pues, además, el daño a la raíz permite que hongos y bacterias ingresen con mayor facilidad a la planta.

Para el control de los nematodos se utilizan diferentes nematicidas, pero estos aparte de no ser muy exitosos en la eliminación de estos organismos también afectan la calidad de los cultivos por destruir otros organismos beneficiosos y contaminar las aguas subterráneas, afectando la salud humana. Por lo tanto, se buscan nuevas formas de atacar los nematodos mediante el control biológico sin afectar el medio ambiente ni la salud humana.

La utilización de agentes naturales para controlar las plagas de nematodos fitoparásitos es entonces la mejor solución tanto desde el punto de vista ambiental como económico, pero involucra una investigación exhaustiva de los efectos a nivel biológico y ecológico del uso de estos microorganismos. El proyecto de investigación “Prospección, caracterización y evaluación de las relaciones de organismos benéficos para el control de nematodos patógenos en condiciones del trópico” que se

desarrolla en el Instituto Tecnológico de Costa Rica con la colaboración de otras instituciones pretende realizar un estudio de los organismos que puedan ser usados para el control biológico [6].

Al ser individuos tan pequeños, la única forma de estudiarlos es utilizando microscopios. Se van a obtener imágenes digitales con un microscopio y una cámara acoplados. Sin embargo, el tiempo que invierte un nematólogo en analizar una muestra de imagen del microscopio puede variar desde unos quince minutos hasta horas, por lo que dentro de los objetivos del proyecto mencionado se busca crear una herramienta de visión por computadora que sea capaz de reducir este tiempo a unos 5 minutos, facilitando el trabajo para el investigador y mejorando la eficiencia del proceso.

Para desarrollar la herramienta mencionada se tienen que elaborar varios módulos que comúnmente se utilizan en los sistemas de visión por computadora. En la figura 1.1 se muestra el diagrama general de estos sistemas [2].

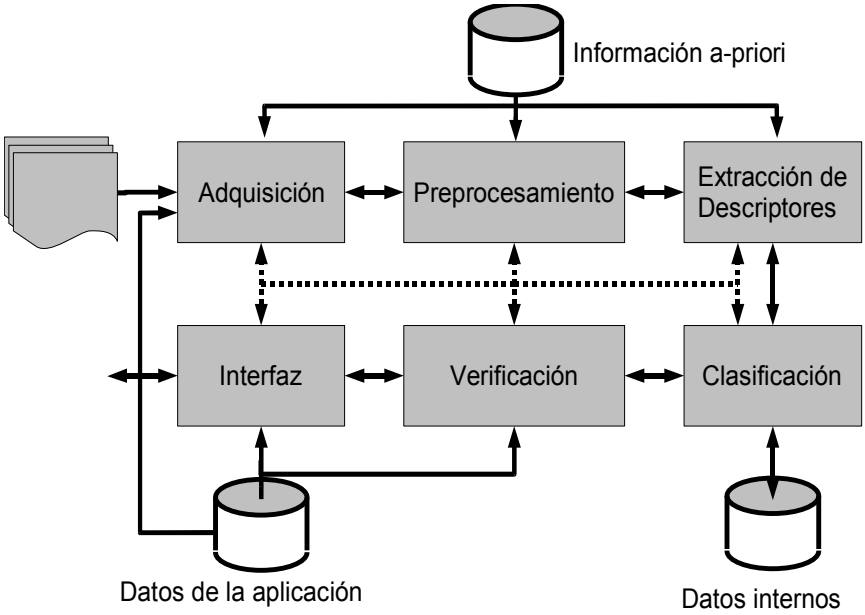


Figura 1.1 Estructura del sistema de visión por computadora

La etapa en la cual se realizará este proyecto es la de preproceso de las imágenes adquiridas con el microscopio.

1.2 Dificultad de analizar imágenes separadas

Para realizar el estudio de los nematodos, en la mayoría de los casos el grado de detalle obtenido si se toma solamente una imagen de toda la muestra no es suficiente, por lo que se hace necesario adquirir varias imágenes que en conjunto forman la escena completa. Para poder capturar varias imágenes de diferentes partes de la muestra de nematodos se mueve el portaobjetos en el que se encuentran.

El problema de hacerlo de esta manera es que también se complica el proceso de análisis para el nematólogo pues debe entonces hacerlo con una serie de imágenes por separado. De esta manera tendría que determinar la secuencia de las imágenes y tomando en cuenta el traslape y la orientación de las mismas, tratar de formar una completa, haciendo el trabajo muy tedioso y con altas posibilidades de cometer errores.

Por lo tanto, al digitalizar el proceso de adquisición de datos con el microscopio, el problema que se debe resolver es que se obtienen múltiples imágenes que complican el estudio para los nematólogos, quienes requieren una única imagen de toda la escena.

1.3 Composición de imágenes

La solución que se pretende dar al problema es crear una herramienta que sea capaz de formar una imagen compuesta a partir de las imágenes obtenidas con la cámara colocada en el microscopio. Esto se conoce como “mosaicing” o composición en el campo del procesamiento digital de imágenes y su nombre se debe a que se forma un cuadro compuesto a partir del montaje de varias imágenes relacionadas entre sí.

Como el movimiento de la cámara y del portaobjetos están sujetos al error humano, la orientación y la zona de traslape van a variar por lo que primero se deben alinear las imágenes automáticamente. Esta parte de la composición de imágenes es

lo que se conoce como registro de imágenes, el cual consiste en ubicar los datos en un mismo sistema de coordenadas, es decir, en el mismo plano, para que luego puedan ser unidos o comparados. En este caso la idea es obtener los parámetros necesarios para mover una imagen y alinearla con la otra. Para esto, existen varios algoritmos propuestos en artículos, como por ejemplo [5], [10], [13] y [15], por lo que se pretende implementar uno y comparar los resultados con otro existente.

Luego, se tiene que implementar un algoritmo que permita unir las dos imágenes, pues, por ser tomadas en momentos distintos, no pueden ser simplemente traslapadas ya que la zona de traslape puede no coincidir exactamente. Este algoritmo lo que permite es compensar las diferencias, de tal forma que la imagen final no parezca la simple unión de dos imágenes, sino una imagen completa con características de ambas.

1.4 Objetivos y estructura del trabajo

La meta de este proyecto es facilitar el estudio de los nematodos por medio de la obtención de imágenes compuestas a partir de varias imágenes de fragmentos de la muestra, lo que permitiría disminuir el tiempo de análisis y evitar que el investigador haga la unión de manera manual.

Esto se pretende lograr mediante el diseño e implementación de un algoritmo que sea capaz de realizar la composición de múltiples imágenes, formando una única imagen que mantenga las características de las imágenes iniciales.

La primera etapa consiste en la definición e implementación del algoritmo que lleva a cabo el registro de las imágenes para colocarlas en el mismo sistema de coordenadas en el plano de una de las dos imaǵenes.

De la misma manera, se elige el algoritmo que se utilizará como referencia para hacer comparaciones con el que se implementó. Una vez escogido, se comparan los resultados con cada algoritmo, para lo cual se debe definir de qué manera se hará la evaluación.

En la última etapa se implementa un algoritmo que construya una imagen

compuesta por las dos iniciales usando los resultados obtenidos con el registro de imágenes que permiten alinearlas. La idea con esto es que se logre la unión de la mejor forma posible, creando una imagen completa que no parezca formada a partir de dos diferentes.

En la figura se muestra un diagrama de bloques del proceso completo de composición de imágenes. En este proyecto se implementarán algoritmos para llevar a cabo las dos etapas finales (registro y unión de las imágenes) pues las primeras dos ya se realizaron.

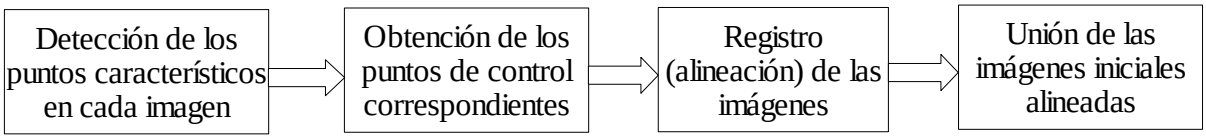


Figura 1.2 Diagrama de bloques general del sistema de composición de imágenes

En el Capítulo 2 se presentan los conceptos necesarios para comprender la solución propuesta, incluyendo la referencia a algunos métodos existentes para realizar el registro y el empate de imágenes. En el Capítulo 3 se presenta de manera detallada la explicación de la solución implementada. Una vez descritas las diferentes partes de la solución, en el Capítulo 4 se muestran y analizan los resultados obtenidos para cada algoritmo y la comparación entre los de registro de imágenes. Finalmente, en el Capítulo 5 se incluyen las conclusiones derivadas del análisis de la solución elaborada y las recomendaciones sugeridas para trabajos posteriores relacionados con este proyecto.

Capítulo 2:Marco teórico

En este capítulo se explican los principales temas que se deben conocer para poder comprender la solución presentada. Primero, se explica la teoría de las transformaciones geométricas, que en este caso son el resultado de los algoritmos para el registro de imágenes. Se describen algunos métodos de optimización de parámetros pues están relacionados directamente con los algoritmos implementados.

También se da una descripción detallada del registro y el empate de imágenes, como etapas de la composición de imágenes. Se incluyen referencias a trabajos anteriores encontrados sobre cada una de las etapas.

2.1 Transformaciones geométricas

Las trasformaciones geométricas se deben calcular y aplicar a las imágenes para poder ubicarlas en un mismo sistema de referencia y así lograr formar una sola imagen.

Existen varios tipos de trasformaciones que se pueden aplicar a una imagen dependiendo de las características que se deseen modificar. Entre éstas están la transformación euclideana, la de similitud, la afín y la proyectiva. Sin embargo, considerando las características de las imágenes que se utilizarán, solo se comentarán las primeras tres [12].

La más simple es la euclideana que cubre rotación y traslación de la imagen, sin alterar el tamaño ni los ángulos entre puntos de la imagen. Si se tiene un punto (x, y), el nuevo punto después de aplicar la transformada sería (x', y'). Si solamente se desea hacer una traslación h en x y k en y, esto se puede expresar utilizando vectores como:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} h \\ k \end{bmatrix} \tag{2.1}$$

Ahora, si se desea hacer una rotación de la imagen en un ángulo α, se procedería a utilizar una matriz de la siguiente manera:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.2}$$

Utilizando las ecuaciones (2.1) y (2.2) se puede obtener la ecuación para hacer la rotación y traslación al mismo tiempo:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} h \\ k \end{bmatrix} \tag{2.3}$$

Si también se desean hacer cambios de escala, se tendría que recurrir a la transformación de similitud, que introduce un factor de escala s que aparece en cada uno de los términos de la matriz que multiplica al vector de posición inicial:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s*\cos\alpha & -s*\sin\alpha \\ s*\sin\alpha & s*\cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} h \\ k \end{bmatrix} \tag{2.4}$$

Si además se desea estirar (empujar) en dirección a alguno de los ejes la imagen (conocido en inglés como “shear”), se estaría utilizando una transformación afín. Pero para lograr cualquiera de los dos efectos lo que se hace es multiplicar diferentes elementos de la matriz 2x2 mostrada en la ecuación (2.3) por constantes. Entonces, la transformada afín sí puede alterar el tamaño y los ángulos entre puntos de la imagen, pero las líneas continúan siendo líneas. La transformada afín es una generalización de la euclideana y la de similitud, estaría dada por:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} h \\ k \end{bmatrix} \tag{2.5}$$

2.2 Algoritmos para la optimización de parámetros

Cuando se tienen múltiples variables que dependen unas de otras y se quieren obtener los valores de cada una de ellas que hacen que se obtenga el menor error en determinado modelo matemático, se recurre a los algoritmos para la optimización de parámetros. Entonces, una vez que se ha definido una función de error en términos de los parámetros a optimizar, se debe definir un algoritmo que permita minimizar esta función.

Existen numerosos algoritmos para la optimización de parámetros con diferentes grados de complejidad, pero no se puede afirmar que un algoritmo sea mejor que otro pues dependiendo de las condiciones del problema puede variar la eficiencia de cada uno.

Como el error E es función de los parámetros, éstos se pueden agrupar en un vector w. Como se desconoce la relación entre E y w, pueden existir varios mínimos que cumplen:

$$\nabla E = 0 \tag{2.6}$$

donde ∇E corresponde al gradiente de E en el espacio definido por w. El mínimo para el que el valor de la función de error es el menor sería el mínimo global, mientras que los otros mínimos serían mínimos locales. También puede haber puntos de inflexión y máximos locales que cumplan (2.6). Cada vector w para el que se satisface esta condición se conoce como punto estacionario [4].

A causa de la no linealidad de la función de error, en la mayoría de los casos no es posible encontrar una expresión algebraica para un mínimo. Por esta razón, se definen algoritmos que realizan una búsqueda en el espacio que contiene todos los parámetros realizando una serie de iteraciones de la forma:

$$w^{(n+1)} = w^{(n)} + \Delta w^{(n)} \tag{2.7}$$

donde n se refiere a la iteración que se está corriendo. Cada algoritmo lo que hace es definir un valor diferente para la variación $\Delta w^{(n)}$ que se va a dar en w en cada iteración. Cuando la función de error es tal que no incrementa independientemente de la variación en w, se puede presentar un problema pues si se alcanza un mínimo local, no hay manera de salir de ahí pues esto requeriría de un incremento en el error. Por lo tanto la escogencia de los valores iniciales de los parámetros va a determinar en qué mínimo local va a converger el algoritmo.

Cuando se utilizan este tipo de algoritmos, se debe definir en qué condiciones se van a detener las iteraciones. Una de las posibilidades es detenerlo cuando se cumplen cierta cantidad de ciclos, sin embargo, en muchos casos es difícil saber con anticipación este número. Otra posibilidad es detenerlo luego de cierta cantidad de tiempo, pero esto también requeriría de una aproximación que no necesariamente va a producir los mejores resultados en todos los casos. También se puede hacer cuando el error es menor que cierto valor, lo que es problemático si este valor nunca es alcanzado. Otra opción es que se mida el cambio relativo del error, aunque en algunos casos esta variación puede no llegar al valor definido. Dependiendo del modelo, se podría hacer con una combinación de estos métodos.

2.2.1 Aproximación cuadrática local

Hay métodos que se basan en la estimación de la función de error utilizando una aproximación cuadrática local. Se considera le expansión de Taylor de $E(w)$ alrededor de un punto $\hat{w}$

$$E(w) = E(\hat{w}) + (w - \hat{w})^T b + \frac{1}{2}(w - \hat{w})^T H(w - \hat{w}) \qquad (2.8)$$

donde b es el gradiente de E evaluado en $\hat{w}$ y la matriz Hessiana (“Hessian matrix”) está dada por:

$$(H)_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}|_{\hat{w}} \tag{2.9}$$

De la ecuación (2.8) se obtiene por derivación vectorial que la aproximación local para el gradiente es:

$$\nabla E = b + H(w - \hat{w}) \tag{2.10}$$

Para puntos cercanos a $\hat{w}$, estas ecuaciones permiten calcular una aproximación bastante cercana al error y su gradiente. En el caso en que se aplica (2.8) sobre un punto w' que corresponde a un mínimo, considerando que allí $b = 0$, se obtiene:

$$E(w) = E(w') + \frac{1}{2}(w - w')^T H(w - w') \tag{2.11}$$

con H evaluada en w'.

2.2.2 Descenso en el gradiente

Uno de los algoritmos más simples y utilizados es el del descenso en el gradiente (gradient descent o steepest descent). Se inicia con la asignación de valores iniciales para todos los parámetros, que corresponde a $w^{(0)}$. Luego, en cada iteración, se actualiza el valor de los parámetros de forma que se desplaza una corta distancia en la dirección de la mayor razón de disminución del error, es decir, en la dirección del gradiente negativo:

$$\Delta w^{(n)} = -\eta \nabla E|_{w^{(n)}} \tag{2.12}$$

El parámetro η es conocido como la razón de aprendizaje por la utilización en redes neuronales. Su valor debe ser definido tan pequeño como para que en cada iteración el error E disminuya, de manera que eventualmente se logre alcanzar el valor de w para el que la condición (2.6) se cumpla.

Como se mencionó anteriormente, uno de los problemas de este algoritmo es que puede permanecer en un mínimo local sin poder salir de él. Pero en ciertos casos se pueden inicializar los parámetros en valores conocidos que lo acercan al mínimo buscado, por lo que se evita este problema. También le puede tomar muchos pasos llegar al mínimo dependiendo del valor de η que se utilice. Por lo que se pueden hacer modificaciones en esta razón de cambio para aumentarla cierta cantidad cuando el error disminuye y disminuirla cuando el error aumenta.

2.2.3 Método de Newton

El método de Newton hace uso de la matriz Hessiana pues por (2.11) se tiene que el gradiente en un punto w, con $b=0$, está dado por:

$$g=\nabla E=H(w-w') \tag{2.13}$$

entonces el mínimo w' de la función de error satisface:

$$w'=w-H^{-1}g \tag{2.14}$$

El vector $-H^{-1}g$ es conocido como la dirección de Newton y, a diferencia del gradiente local, evaluado en cualquier w apunta directamente al mínimo de la función de error. Como se utiliza una aproximación, la matriz Hessiana debe ser calculada de nuevo en cada punto, lo que puede hacerlo muy lento de ejecutar dependiendo del problema. El algoritmo del descenso en el gradiente puede interpretarse como un caso particular del método de Newton, con la matriz Hessiana igual al producto de la matriz identidad por la razón de aprendizaje η .

2.2.4 Algoritmo de Levenberg-Marquardt

Este algoritmo está diseñado específicamente para minimizar un error de suma de cuadrados de la forma:

$$E=\frac{1}{2}\sum_{n}\left(\varepsilon^{n}\right)^{2}=\frac{1}{2}\|\varepsilon\|^{2} \tag{2.15}$$

donde ε^{n} es el error del parámetro n y ε es el vector con elementos ε^{n}. Suponiendo que se está en el punto w_{old} y se desea mover a w_{new}, si el desplazamiento w_{new} - w_{old} es pequeño entonces se puede hacer una expansión con series de Taylor de primer orden:

$$\varepsilon\left(w_{new}\right)=\varepsilon\left(w_{old}\right)+Z\left(w_{new}-w_{old}\right) \tag{2.16}$$

con Z una matriz que contiene los elementos

$$\left(Z_{ni}\right)\equiv\frac{\partial\varepsilon^{n}}{\partial w_{i}} \tag{2.17}$$

Con lo que se obtiene una nueva función de error:

$$E=\frac{1}{2}\|\varepsilon\left(w_{old}\right)+Z\left(w_{new}-w_{old}\right)\|^{2} \tag{2.18}$$

Si se minimiza el error con respecto al nuevo punto w_{new} se obtiene:

$$w_{new}=w_{old}-\left(Z^{T}Z\right)^{-1}Z^{T}\varepsilon\left(w_{old}\right) \tag{2.19}$$

Para esta función de error, los elementos de la matriz Hessiana son de la forma:

$$(H)_{ik}=\frac{\partial^2 E}{\partial w_i \partial w_k}=\sum_n \left(\frac{\partial \varepsilon^n}{\partial w_i}\frac{\partial \varepsilon^n}{\partial w_k}+\varepsilon^n \frac{\partial^2 \varepsilon^n}{\partial w_i \partial w_k}\right) \tag{2.20}$$

Despreciando el segundo término, se obtiene una aproximación de la matriz Hessiana

$$H=Z^T Z \tag{2.21}$$

El problema en este caso es que si el vector que se resta en (2.19) es relativamente grande, la aproximación que se presentó en (2.18) no sería válida. Esto se soluciona modificando la función de error:

$$E=\frac{1}{2}\|\varepsilon(w_{old})+Z(w_{new}-w_{old})\|^2+\lambda\|w_{new}-w_{old}\|^2 \tag{2.22}$$

donde λ determina el tamaño de la variación de w. Si se minimiza este nuevo error con respecto a w_{new} se tiene:

$$w_{new}=w_{old}-(Z^T Z+\lambda I)^{-1} Z^T \varepsilon(w_{old}) \tag{2.23}$$

donde I es la matriz identidad. Se debe asignar un valor inicial a λ y si el error aumenta entonces se aumenta λ y viceversa.

2.3 Registro de imágenes

Las imágenes obtenidas de la cámara colocada en el microscopio serán secciones de la muestra total, por lo que para que el nematólogo pueda hacer los

estudios que necesita, se debe obtener una sola imagen formada a partir de la unión de todas las secciones.

Para lograrlo, primero es necesario determinar el algoritmo que se va a utilizar para, dadas un par de imágenes, establecer la relación geométrica entre ellas. De esta manera, al aplicar la transformación obtenida a una de las imágenes, la zona de traslape entre ambas imágenes quedará alineada.

Existen en la literatura dos tendencias para llevar a cabo el registro de imágenes: basadas en puntos característicos de la imagen o basadas en la intensidad. La primera involucra necesariamente un paso previo de determinar los puntos en ambas imágenes que contienen características distintivas (orillas, esquinas, picos, etc.) para poder luego buscar los puntos correspondientes en cada imagen. Los puntos asociados entre las dos imágenes, conocidos como puntos de control, permiten calcular la transformación geométrica que alínea de mejor forma los puntos entre sí. La segunda tendencia consiste en alinear los pixeles de tal forma que las intensidades de éstos coincidan de la mejor forma posible. Ambas tendencias han sido bastante utilizadas pues cada una tiene sus ventajas, pero la primera ha tomado más popularidad entre los métodos de composición de imágenes [15].

Entre los algoritmos que trabajan con la intensidad, existen algunos que lo hacen en el dominio de la frecuencia utilizando propiedades de la transformada de Fourier [13] pero esto involucra gran cantidad de procesamiento pues se deben calcular varias transformadas en cada iteración. Además, la cantidad de iteraciones depende directamente del tamaño de las imágenes, lo cual hace un poco más lentos a este tipo de algoritmos. Sin embargo, una de las ventajas es que se puede lograr un mejor acople entre las imágenes pues se busca obtener la transformación que produce la menor diferencia de intensidad entre los pixeles.

En cuanto a las opciones para hacer el registro de las imágenes utilizando puntos de control característicos, uno de los métodos más simples es calculando la transformación que produzca la menor sumatoria de las diferencias de los cuadrados de las distancias entre los puntos correspondientes para las dos imágenes[15]. Esto involucra la obtención de los puntos de control, el cálculo de descriptores para cada punto de tal forma que puedan ser diferenciados dependiendo de su entorno y la

comparación de estos descriptores para asociar los puntos que se encuentran en las dos imágenes a la vez. El utilizar puntos de control para obtener la matriz de trasformación es mucho más veloz que hacerlo con los métodos de intensidad.

Otro algoritmo que utiliza puntos de control es conocido como RANSAC (Random Sample Consensus)[5]. Este algoritmo empieza a trabajar asumiendo que todos los puntos son incorrectos y realiza la primera estimación de la transformación con la cantidad de puntos suficiente para hacerlo. Luego determina, para la totalidad de los puntos, cuántos coinciden con el modelo obtenido dentro de un umbral de error definido. Si esta cantidad de puntos consistentes es mayor o igual que cierto valor, se considera que el modelo es el adecuado y finaliza el algoritmo. En caso contrario, se repite el proceso calculando de nuevo la transformación geométrica. Puede ser que al final no se encuentre un modelo que permita tener la cantidad de puntos consistentes necesaria, por lo que no se obtendría un resultado satisfactorio.

En [10] se presenta un algoritmo que puede tomar ventaja tanto del cálculo por intensidad como del uso de puntos de control. Este algoritmo define un modelo para cada uno de los métodos y con éstos una función de semejanza entre los dos que debe ser maximizada (“Exact Maximum Likelihood”). Entonces se calcula a la vez la transformación que logra la mejor alineación de los puntos correspondientes entre las imágenes, pero a la vez de manera que se disminuya la diferencia de intensidad entre puntos en la zona de traslape.

2.4 Unión de imágenes

El otro problema que se tiene que resolver es el de obtener una imagen completa a partir de las dos imágenes iniciales. Se debe definir un algoritmo tal que al hacer la composición de las imágenes, se mantengan las características de ambas imágenes pero a la vez la zona de transición sea lo menos notoria posible. Esto porque si simplemente se toma una parte de una imagen y el complemento de la otra en la sección de traslape, se podría distinguir la zona o la línea donde se unen.

Existen muchas formas de llevar a cabo esta unión, dependiendo de la

relación entre las imágenes y el grado de complejidad de éstas. Uno de los primeros métodos existentes conocido como *difuminado* consiste en trabajar en la zona de traslape de manera horizontal con cada línea de pixeles[11]. Simplemente se toman pixeles de la primera imagen hasta cierto punto en el cual se hace la transición a pixeles de la segunda imagen. El punto de transición se escoge de tal forma que es la parte del traslape donde la diferencia entre el pixel de una imagen y el correspondiente en la otra sea menor. Para hacer gradual la transición de pixeles de una imagen a la otra se utiliza una función lineal en cada fila de manera que el salto no es tan abrupto. Este método no es muy eficiente pues, por trabajar en una dirección, se hacen notar las líneas horizontales pues las transiciones para cada línea varían y las diferencias de brillo, contraste o pequeños errores en la alineación no son bien corregidos.

Otro método para lograr el empate de las imágenes consiste en la utilización de representaciones piramidales que se obtienen al definir varios niveles de las imágenes en diferentes frecuencias[1]. De esta manera se producen transiciones graduales de una imagen a la otra en bajas frecuencias, mientras que se mezclan los detalles finos en altas frecuencias.

El algoritmo GIST (Image Stitching in the Gradient Domain) realiza la unión de las imágenes basándose en el gradiente de cada una[9]. Se define una función de error en función de los gradientes la cual debe ser minimizada por medio de la optimización de la imagen compuesta. En cada iteración se actualiza la imagen compuesta, que sería el resultado del algoritmo, utilizando una función definida en términos del error obtenido.

Los anteriores métodos son presentados por los autores para trabajar con imágenes panorámicas por lo que funcionan en el plano horizontal, pero en el caso de este proyecto se tendría que hacer una modificación pues se pueden tener traslaciones verticales y rotaciones.

Existen algoritmos más complejos que logran no sólo hacer invisible la transición de una imagen a otra, sino que resuelven problemas tales como presencia de objetos en las dos imágenes en diferentes posiciones debido al movimiento de éstos. El problema es que éste tipo de fenómenos no son corregidos por los

anteriores métodos completamente, pues puede presentarse el mismo objeto dos veces en la imagen compuesta, en muchos casos apareciendo borroso una o dos veces dependiendo su ubicación en la zona de traslape. Métodos como el propuesto en [8] logran resolver estos problemas basados en la deformación y alineación de las estructuras, mejorando también las diferencias de intensidad.

Capítulo 3:Implementación de la composición de imágenes

La solución sugerida al problema de tener múltiples imágenes de una muestra de nematodos es entonces desarrollar una serie de algoritmos que permitan formar una imagen compuesta a partir de dos imágenes iniciales que comparten una zona de traslape. Este proceso se puede dividir en dos partes que son: la alineación de las imágenes y la unión de éstas para formar una imagen final.

Los dos métodos se implementan como parte de un biblioteca de algoritmos y estructuras de datos utilizadas en el procesamiento digital de imágenes llamada CVR-Lib (Computer Vision and Robotics Library)[14]. Todos los programas que contiene esta biblioteca son escritos en lenguaje C++, por lo que los algoritmos implementados son escritos en este lenguaje.

3.1 Algoritmo EML para el registro de imágenes

Para el registro de imágenes se escogió el algoritmo Exact Maximum Likelihood (EML) [10] pues se espera que al usar puntos de control el tiempo de ejecución sea bajo y al utilizar intensidad permita una mejor alineación de las imágenes.

La ventaja que existe en este caso para la utilización de puntos de control es que en la CVR-Lib ya existen algoritmos para obtener los puntos característicos de cada imagen y calcular los descriptores para estos puntos que permiten asociar puntos correspondientes entre las imágenes. Son tres algoritmos que basados en el método SURF (Speeded Up Robust Features) [3] determinan los puntos relevantes correspondientes entre dos imágenes dadas que pueden ser usados para alinearlas.

El algoritmo EML estima una transformación geométrica afín utilizando un método de optimización muy similar a Levenberg-Marquardt. Se define un modelo tanto para el cálculo por puntos de control como para intensidad. En el primer caso

se tiene que si $X_1(k)=\begin{bmatrix} x_1(k) \\ y_1(k) \end{bmatrix}$ y $X_0(k)=\begin{bmatrix} x_0(k) \\ y_0(k) \end{bmatrix}$ son las coordenadas teóricas de un punto de control k en las imágenes que se denominarán de referencia y distorsionada, respectivamente, se pueden relacionar con una transformada afín:

$$X_1(k)=RX_0(k)+t=\begin{bmatrix} 1+r_{11} & r_{12} \\ r_{21} & 1+r_{22} \end{bmatrix}\begin{bmatrix} x_0(k) \\ y_0(k) \end{bmatrix}+\begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{3.1}$$

donde los elementos diagonales de R se definen como la suma de la unidad más un parámetro r_{jj} para simplificación del algoritmo de optimización que se utiliza.

En este caso se trabaja con los cuatro términos de la matriz 2x2 por separado por ser una transformación afín, aunque en la mayoría de los casos de las imágenes de los nematodos se tendría la necesidad de usar solamente una transformación de similitud.

Si $X_A(k)=\begin{bmatrix} x_A(k) \\ y_A(k) \end{bmatrix}$ y $X_B(k)=\begin{bmatrix} x_B(k) \\ y_B(k) \end{bmatrix}$ son las coordenadas obtenidas experimentalmente de los puntos de control, en la imagen distorsionada y de referencia, respectivamente, entonces éstas pueden ser expresadas de la siguiente manera:

$$\begin{aligned} x_A(k)&=x_0(k)+n_1(k) \\ y_A(k)&=y_0(k)+n_2(k) \\ x_B(k)&=(1+r_{11})\,x_0(k)+r_{12}\,y_0(k)+t_x+n_3(k) \\ y_B(k)&=r_{21}\,x_0(k)+(1+r_{22})\,y_0(k)+t_y+n_4(k) \end{aligned} \tag{3.2}$$

donde los n_i son los valores de ruido gaussiano con media cero y varianza δ_1^2 .

El sistema de ecuaciones (3.2) puede ser escrito en forma de matriz de la siguiente manera:

$$x(k)=A(k)\,\eta+b(k)+n(k) \tag{3.3}$$

donde:

$$x(k)=\begin{bmatrix} x_A(k) \\ y_A(k) \\ x_B(k) \\ y_B(k) \end{bmatrix}, \qquad b(k)=\begin{bmatrix} x_0(k) \\ y_0(k) \\ x_0(k) \\ y_0(k) \end{bmatrix}, \qquad n(k)=\begin{bmatrix} n_1(k) \\ n_2(k) \\ n_3(k) \\ n_4(k) \end{bmatrix}, \qquad \eta=\begin{bmatrix} r_{11} \\ r_{12} \\ r_{21} \\ r_{22} \\ t_x \\ t_y \end{bmatrix}=\begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \\ \eta_4 \\ \eta_5 \\ \eta_6 \end{bmatrix} \quad \text{y}$$

$$A(k)=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ x_0(k) & y_0(k) & 0 & 0 & 1 & 0 \\ 0 & 0 & x_0(k) & y_0(k) & 0 & 1 \end{bmatrix}.$$

De la misma manera, se plantea un modelo con la intensidad de los pixeles. Dadas las coordenadas $u_0(i,j)=\begin{bmatrix} u_0(i) \\ v_0(j) \end{bmatrix}$ y $u_1(i,j)=\begin{bmatrix} u_1(i) \\ v_1(j) \end{bmatrix}$ de un pixel en la imagen distorsionada y de referencia, respectivamente, se pueden relacionar , como se hizo en (3.1) por medio de la transformación afín:

$$u_1(i,j)=R\,u_0(i,j)+t \tag{3.4}$$

Si los valores verdaderos de intensidad de los pixeles $u_1(i,j)$ y $u_0(i,j)$ son $f_1(u_1(i,j))$ y $f_0(u_0(i,j))$, respectivamente, el valor medido del pixel $u_1(i,j)$ puede ser expresado como:

$$f_A(u_1(i,j))=f_1(u_1(i,j))+n_5(i,j) \tag{3.5}$$

donde n_5 es un valor de ruido gaussiano con media cero y varianza δ_2^2 .

En el caso de las imágenes de los nematodos el ruido de medición no se puede considerar gaussiano pues los errores en las falsas correspondencias de los

puntos de control no se distribuyen de esta manera. Más adelante se discutirá más a fondo este problema.

Si N_1 y N_2 son la cantidad de puntos de control y de medición de intensidad, respectivamente, con los pixeles (i,j) en la zona de traslape, entonces utilizando (3.3) y (3.5) se define la función de densidad condicional:

$$L=\frac{1}{\sqrt{2\pi}^{N_1+N_2}\delta_1^{N_1}\delta_2^{N_2}}L_1L_2 \tag{3.6}$$

donde:

$$L_1=\exp\left\{\frac{-1}{2\delta_1^2}\Gamma_1\right\} \quad , \quad L_2=\exp\left\{\frac{-1}{2\delta_2^2}\Gamma_2\right\}$$

$$\Gamma_1=\sum_{k=1}^{N_1}\left[x(k)-A(k)\eta-b(k)\right]^T\left[x(k)-A(k)\eta-b(k)\right]$$

$$\Gamma_2=\sum_{(i,j)}^{N_2}\left[f_A(u_1(i,j))-f_0(u_0(i,j))\right]^2$$

Los parámetros desconocidos de la función anterior en el caso de L_1 son los seis coeficientes de la transformación afín η, las verdaderas coordenadas $\xi(k)=\begin{bmatrix}x_0(k) & y_0(k)\end{bmatrix}^T$ y la varianza δ_1^2 del ruido para los puntos de control. Por otro lado, en L_2 se desconoce el vector η, así como la varianza δ_2^2 del ruido para las mediciones de intensidad. Entonces la cantidad de parámetros que se deben optimizar es variable pues ξ depende de la cantidad de puntos de control.

La ecuación (3.6) representa la función de semejanza que se debería maximizar utilizando un método de optimización para lograr el registro de las imágenes. Sin pérdida de generalidad, se ignora el término constante en (3.6) para la optimización y se define el logaritmo negativo de la función de semejanza:

$$\Lambda=-\log L=\Lambda_1+\Lambda_2 \tag{3.7}$$

donde:

$$\Lambda_1 = N_1 \ln(\delta_1) + \frac{1}{2\delta_1^2} \Gamma_1$$

$$\Lambda_2 = N_2 \ln(\delta_2) + \frac{1}{2\delta_2^2} \Gamma_2$$

De los parámetros mencionados anteriormente, las dos varianzas son independientes entre sí y se pueden calcular directamente al minimizar la función (3.7) con respecto a cada una. Entonces en cada iteración se calculan usando los últimos valores estimados de los otros parámetros como se muestra a continuación:

$$\delta_1^2 = \frac{1}{N_1} \sum_{k=1}^{N_1} \left[x(k) - \hat{A}(k)\hat{\eta} - \hat{b}(k) \right]^T \left[x(k) - \hat{A}(k)\hat{\eta} - \hat{b}(k) \right] \tag{3.8}$$

$$\delta_2^2 = \frac{1}{N_2} \sum_{(i,j)}^{N_2} \left\| f_A(\hat{t} + \hat{R}\, u_0(i,j)) - f_0(u_0(i,j)) \right\|^2 \tag{3.9}$$

Para obtener los otros dos parámetros, se tiene que dejar uno fijo para calcular el otro y luego el primero es actualizado haciendo uso del valor del segundo. Se continúan haciendo iteraciones hasta que el algoritmo converge.

Teniendo el valor estimado de los parámetros de la transformada, se pueden calcular las coordenadas verdaderas de los puntos de control en la imagen distorsionada con la siguiente ecuación:

$$\xi(k) = \left[I_2 + \hat{R}^T \hat{R} \right]^{-1} \left[X_A(k) + \hat{R}^T (X_B(k) - \hat{t}) \right] \tag{3.10}$$

donde I_2 es una matrix identidad de 2x2.

Con la estimación de estas coordenadas, se puede actualizar el vector con los parámetros de la transformación afín, para lo cual se hace uso de un método de optimización muy similar a Levenberg-Marquardt pero a la que los autores agregan

una variable α que proponen se disminuya cada cierta cantidad de iteraciones. La función con la que se calcula el nuevo valor de este parámetro es:

$$\hat{\eta}^{(t+1)}=\hat{\eta}^{(t)}-\alpha(\nabla^2\Lambda+\lambda I)^{-1}\nabla\Lambda \tag{3.11}$$

La derivada de Λ con respecto a η se puede calcular usando (3.7), esto es:

$$\nabla\Lambda=\frac{\partial\Lambda}{\partial\eta}=\frac{\partial\Lambda_1}{\partial\eta}+\frac{\partial\Lambda_2}{\partial\eta} \tag{3.12}$$

donde:

$$\frac{\partial\Lambda_1}{\partial\eta}=\frac{1}{\hat{\delta}_1^2}\sum_{k=1}^{N_1}\hat{A}^T(k)\left[x(k)-\hat{A}(k)\eta_1-\hat{b}(k)\right]$$

$$\frac{\partial\Lambda_2}{\partial\eta}=\frac{1}{\hat{\delta}_2^2}\sum_{(i,j)}^{N_2}\left[f_A(u_1(i,j))-f_0(u_0(i,j))\right]\frac{\partial f_A(u_1(i,j))}{\partial\eta_m}$$

La matriz Hessiana viene dada por:

$$\nabla^2\Lambda=\nabla^2\Lambda_1+\nabla^2\Lambda_2 \tag{3.13}$$

donde:

$$\frac{\partial^2\Lambda_1}{\partial\eta_l\partial\eta_m}=\frac{1}{\hat{\delta}_1^2}\sum_{k=1}^{N_1}\hat{A}^T(k)\hat{A}(k)$$

$$\frac{\partial^2\Lambda_2}{\partial\eta_l\partial\eta_m}=\frac{1}{\hat{\delta}_2^2}\sum_{(i,j)}^{N_2}\frac{\partial f_A(u_1(i,j))}{\partial\eta_l}\frac{\partial f_A(u_1(i,j))}{\partial\eta_m}$$

Como se puede notar, tanto en (3.12) como en (3.13) se hace necesario calcular la derivada parcial de la intensidad medida en un pixel con respecto a cada uno de los parámetros del vector η que dependen de la posición horizontal o

vertical del pixel. Por regla de la cadena, para el valor de la traslación en x como parámetro, por ejemplo, se tiene:

$$\frac{\partial f_A(u_1(i,j))}{\partial \eta_m} = \frac{\partial f_A(u_1(i,j))}{\partial u_1(i)} \frac{\partial u_1(i)}{\partial t_x} \tag{3.14}$$

En este caso, de (3.4) se tiene que $\partial u_1(i)/\partial t_x = 1$. Para los parámetros de la matriz 2x2 de la transformada, este valor es la posición del pixel horizontal o vertical. Para calcular $\partial f_A(u_1(i,j))/\partial u_1(i)$ y $\partial f_A(u_1(i,j))/\partial v_1(j)$ se utiliza una interpolación de convolución cúbica con el siguiente kernel:

$$z(s) = \begin{cases} \frac{3}{2}|s|^3 - \frac{5}{2}|s|^2 + 1 & 0 < |s| < 1 \\ -\frac{1}{2}|s|^3 + \frac{5}{2}|s|^2 - 4|s| + 1 & 1 < |s| < 2 \\ 0 & 2 < |s| \end{cases} \tag{3.15}$$

Ahora, suponiendo que el punto $u_1(i,j)$ se encuentra dentro de la subdivisión rectangular $[U(i), U(i+1)] \times [V(j), V(j+1)]$ en la imagen que se desea alinear, con $U(i)$ y $V(j)$ enteros. El valor de la intensidad sería:

$$f_A(u_1(i,j)) = \sum_{l=-1}^{2} \sum_{m=-1}^{2} c_{i+l,\, j+m}\, z\left(\frac{u_1(i) - U(i+l)}{h_x}\right) z\left(\frac{v_1(j) - V(j+m)}{h_y}\right) \tag{3.16}$$

donde h_x y h_y definen el rango de valores válidos en los ejes horizontal y vertical, respectivamente, y $c_{i,j} = f_A(U(i), V(j))$ es el valor de intensidad en ese vértice de la subdivisión. Así, $\partial f_A(u_1(i,j))/\partial u_1(i)$ y $\partial f_A(u_1(i,j))/\partial v_1(j)$ se calcularon a partir de (3.16), de manera que dependen del valor de la derivada del kernel también.

En resumen, el algoritmo EML para el registro de imágenes presentado en [10]

viene dado por los siguientes pasos:

1. Se define el error aceptable para que el algoritmo converja y un tiempo máximo en caso de no converger.
2. Se inicializan los valores de las varianzas y los elementos del vector η para poder iniciar las iteraciones.
3. Se estiman las verdaderas coordenadas de los puntos de control en la imagen que se desea alinear con la de referencia usando (3.10).
4. Se calculan los nuevos valores de los elementos de la trasformación afín con la ecuación (3.11), para lo que se deben haber calculado anteriormente el gradiente y la matriz Hessiana definidos en (3.12) y (3.13).
5. Se calculan las varianzas usando los valores obtenidos en los dos puntos anteriores. Se disminuye α luego de pocas iteraciones y λ se ajusta dependiendo del error obtenido.
6. Si el tiempo es igual o mayor al tiempo máximo definido o la distancia entre los últimos dos valores de η es menor que el error aceptado, se detiene el algoritmo. En caso contrario, se itera a partir del paso 3.

Para los puntos de traslape que se utilizan para hacer los cálculos con las ecuaciones de intensidad, se utilizaron los mismos puntos de control pues todos ellos deben estar en ambas imágenes si han sido bien obtenidos. Esto se hizo así porque no se puede determinar con anticipación cual será la zona de traslape, que es más bien parte de lo que se debe obtener.

Como se indica en el punto 2 del algoritmo, se deben dar valores iniciales a las varianzas y a la trasformación afín. Primero, con los puntos de control dados, se calculan los parámetros iniciales de la transformación estimando los valores que producen el menor error de los cuadrados de las distancias para todos los puntos. Con estos valores, se aproximan las varianzas iniciales.

3.2 RANSAC para la comparación y mejora del algoritmo EML

Como se mencionó anteriormente, el algoritmo EML presenta un problema porque se asume que el ruido para los puntos es gaussiano, lo cual en muchos casos no es cierto pues hay uno o varios puntos que insertan ruido en una dirección sin que haya otro punto que lo compense. Esto implica que con un solo punto que se encuentre mal asociado en la otra imagen se tome en cuenta en el momento de correr el algoritmo, se van a obtener resultados muy diferentes a los deseados.

Por lo tanto, se propone utilizar el algoritmo RANSAC como complemento del EML, para que no se tomen en cuenta todos los puntos de control dados, sino solamente los que, dentro de un rango de error establecido, se pueden relacionar usando una misma transformación geométrica. Esta transformación sería la que RANSAC calcularía, sin embargo, en este caso lo que interesa son los puntos de control consistentes que determina el algoritmo, con el fin de utilizarlos como puntos de control para el algoritmo EML. La idea es que esto permita resolver en la mayoría de los casos el problema de asumir que el ruido es gaussiano, pues se reduce considerablemente al descartar todos los puntos que están fuera del rango determinado por la transformación calculada.

3.3 Unión de imágenes utilizando el gradiente

Para lograr la unión de las imágenes se encontraron varios métodos, como se discutió en el marco teórico. Se decidió utilizar el método GIST (“Gradient Image Stitching”) que trabaja en el dominio del gradiente [9] por las ventajas que tiene con respecto al de difuminado [11] y el de representaciones piramidales [1], que aparte de tener más de veinte años de haber sido propuestos, presentan muchos problemas cuando las imágenes no están bien alineadas pues se hacen muy notorios los contornos repetidos y el de difuminado tiende a producir líneas en la dirección (horizontal o vertical) en que se recorren las imágenes.

De antemano se conoce que el algoritmo GIST no resuelve el problema de

que uno o varios objetos se muevan en el intervalo de tiempo entre la captura de las dos imágenes, por lo que al hacer la unión de éstas, se pueden apreciar como repetidos en la imagen compuesta. Sin embargo, la solución de este problema escapa a los objetivos planteados para este proyecto.

El objetivo del algoritmo GIST es obtener una imagen compuesta al minimizar una función de error que representa las diferencias entre las derivadas de la imagen a estimar y las derivadas de las imágenes de entrada. Se definen las dos imágenes alineadas I_1 e I_2 que se quieren unir, τ_1 y τ_2 las regiones exclusivas de éstas, respectivamente, ω la región de traslape y W una máscara del tamaño de la imagen final con valores entre 0 y 1, inclusive, en cada pixel.

Con esto, la imagen resultante I sería con la que se obtiene el menor error E calculado de la siguiente manera:

$$E\left(\hat{I}, I_1, I_2, W\right) = d\left(\nabla \hat{I}, \nabla I_1, \tau_1 \cup \omega, W\right) + d\left(\nabla \hat{I}, \nabla I_2, \tau_2 \cup \omega, U - W\right) \tag{3.17}$$

donde U es una imagen uniforme (todos sus valores son 1) del tamaño de W y la función $d\left(J_1, J_2, \phi, W\right)$ es la distancia entre J_1 y J_2 en la región ϕ que se calcula de la siguiente manera:

$$d\left(J_1, J_2, \phi, W\right) = \sum_{q \in \phi} W(q)\left[J_1(q) - J_2(q)\right]^2 \tag{3.18}$$

Para calcular el gradiente de cada una de las imágenes se calcula la derivada en un punto (x, y) en función de los valores de los pixeles adyacentes:

$$\begin{aligned} \frac{\partial I(x, y)}{\partial x} &= I(x+1, y) - I(x-1, y) \\ \frac{\partial I(x, y)}{\partial y} &= I(x, y+1) - I(x, y-1) \end{aligned} \tag{3.19}$$

De esta forma, la ecuación (3.18) se puede reescribir como:

$$d\left(\nabla \hat{I}, \nabla I_k, \phi, W\right) = \sum_{(x,y)\in\phi} W(x,y)\left[\left(\frac{\partial \hat{I}}{\partial x} - \frac{\partial I_k}{\partial x}\right)^2 + \left(\frac{\partial \hat{I}}{\partial y} - \frac{\partial I_k}{\partial y}\right)^2\right] \tag{3.20}$$

Con esto definido, se determina el método de optimización que se seguirá para calcular la imagen compuesta. Como la cantidad de parámetros es igual a la cantidad total de pixeles en la imagen final, no se consideró conveniente calcular la matriz Hessiana por el gran tamaño que puede llegar a tener, que haría muy lento el proceso. Por lo tanto, el método escogido fue el del descenso en el gradiente, para lo cual se debe calcular el gradiente del error, y luego actualizar el valor de la imagen $\hat{I}$ de la siguiente manera:

$$\hat{I}_{new} = \hat{I}_{old} - \eta \nabla E \tag{3.21}$$

El gradiente del error se calculó con base en la evaluación de (3.20) con las dos imágenes de entrada en (3.17) y calculando la derivada para un pixel cualquiera. La expresión completa obtenida es muy extensa, por lo que se encuentra en el Apéndice A.1, acá se presenta una versión simplificada de la misma para un punto (x,y):

$$\frac{\partial E}{\partial \hat{I}(x,y)} = 4\hat{I}(x,y) - \hat{I}(x-2,y) - \hat{I}(x+2,y) - \hat{I}(x,y-2) - \hat{I}(x,y+2) + K \tag{3.22}$$

donde K es una constante que depende únicamente de las dos imágenes de entrada y de la máscara W por lo que se puede calcular desde antes de iniciar las iteraciones y almacenar los valores en una matriz.

Uno de los principales inconvenientes que presenta este algoritmo es que la máscara W se calcula en una dimensión, pues es planteado para resolver el problema con imágenes panorámicas que solamente se traslapan horizontalmente.

La idea de la máscara es que los pixeles que solamente pertenecen a la primera imagen sean 1, los que pertenecen a la zona de traslape disminuyan gradualmente hasta llegar a 0 en los pixeles de la segunda imagen. Por lo tanto, para calcular esta máscara de manera que se pueda aplicar a rotación y traslación vertical y horizontal, se hace el cálculo de una máscara recorriendo las filas y otra recorriendo las columnas, que después se combinan para obtener la máscara W con la siguiente función:

$$W(x,y)=1-[1-h(x,y)][1-v(x,y)] \tag{3.23}$$

donde $h(x,y)$ y $v(x,y)$ representan los valores respectivos de las máscaras calculadas horizontal y verticalmente.

El método de optimización del descenso en el gradiente puede llegar a ser muy lento, sin embargo, si el valor inicial de la imagen que se quiere estimar es muy cercano al valor final que se desea obtener, el algoritmo tendería a disminuir el error en cada iteración, acercándose rápidamente a la imagen compuesta con el menor error. Por lo tanto, el valor inicial de la imagen se define pixel por pixel tomando el valor de la primera imagen en los casos en que la máscara presenta un valor mayor o igual a 0.5 y de la segunda imagen en caso contrario.

En resumen, el algoritmo utilizado para llevar a cabo la unión de las dos imágenes se puede reducir a los siguientes pasos:

1. Se alinean las imágenes de entrada utilizando la transformación dada y a la vez se calcula el tamaño que va a tener la imagen compuesta.
2. Se crea una máscara del tamaño de la imagen final con la cual se va a determinar si un pixel pertenece a la primera imagen, a la segunda, a ambas o a ninguna.
3. Se calcula la máscara de pesos W con valores entre 1 y 0, inclusive, que también tiene el mismo tamaño de la imagen final y en la zona de traslape disminuye de 1 hasta 0 al pasar de la primera imagen a la segunda.
4. Se calcula la matriz de constantes K del tamaño de la imagen final que

depende únicamente de las imágenes de entrada y de la máscara W.

5. Se le da el valor inicial a la imagen compuesta que se debe optimizar, tomando pixeles de las dos imágenes de entrada.
6. Se calcula el gradiente del error utilizando (3.22).
7. Se actualiza el valor de la imagen resultante según (3.21).
8. Si se cumplió la cantidad de iteraciones definidas, se detiene el algoritmo. En caso contrario, se vuelve al punto 6 para repetir los últimos dos pasos.

Capítulo 4:Análisis de Resultados

Una vez descrita e implementada la solución, se presentan los resultados obtenidos y un análisis de los mismos tanto para el registro de imágenes como para la unión de éstas. En el caso del registro de imágenes se presenta una comparación de los resultados obtenidos con los diferentes algoritmos para varios tipos de transformaciones con imágenes de nematodos. Para la unión de las imágenes se presentan los resultados de aplicar el algoritmo a imágenes con diferente brillo, contraste y con ruido.

4.1 Registro de imágenes

Se define un nuevo algoritmo que calcula los parámetros con el método EML (“Exact maximum Likelihood”) a partir de los puntos definidos por el algoritmo RANSAC (“Random Sample Consensus”). Entonces, para la comparación que se pretende hacer de los algoritmos para el registro de las imágenes, se tienen tres: el algoritmo propuesto que combina EML y RANSAC, el cálculo solamente con el algoritmo EML y otro que nada más usa RANSAC. De esta manera, se tiene que decidir cuál de los algoritmos se recomienda usar para alinear imágenes de nematodos que contienen una zona de traslape entre ellas.

Para poder compararlos entre sí, lo que se hace en cada prueba es definir una transformación teórica entre dos imágenes, con valores definidos para los parámetros de traslación, ángulo de rotación y escala, y después comparar los valores obtenidos. Para hacer una evaluación equitativa, se usa RANSAC inicialmente para obtener los puntos de control correctos, como un filtro, con una transformación de similaridad pues es la más compleja que se obtendría con el microscopio. Luego, se calculan RANSAC y EML para obtener una transformada afín, pues el algoritmo EML está definido solamente para este tipo de transformación, lo cual permitiría trabajar con imágenes más complejas en otras aplicaciones. En el

caso de RANSAC, el tipo de transformación es variable y se define antes de correr el algoritmo.

En este caso se presentan los resultados de aplicar los tres algoritmos: solamente EML; la combinación que usa RANSAC para filtrar los puntos de control y EML para estimar la transformación afín, que se llamará RANSAC&EML; y el que usa RANSAC como filtro y se ejecuta luego para calcular la transformación afín.

En el primer caso las imágenes presentan entre sí una relación de traslación en el eje x solamente. Las dos imágenes de entrada se presentan en la figura 4.1.En la figura 4.2 se muestran los puntos de control obtenidos en cada imagen y las correspondencias entre éstos, encontradas con el algoritmo SURF [3]. Como se puede notar, hay gran cantidad de puntos que no se asociaron correctamente, pues no se encuentran en la zona de traslape.

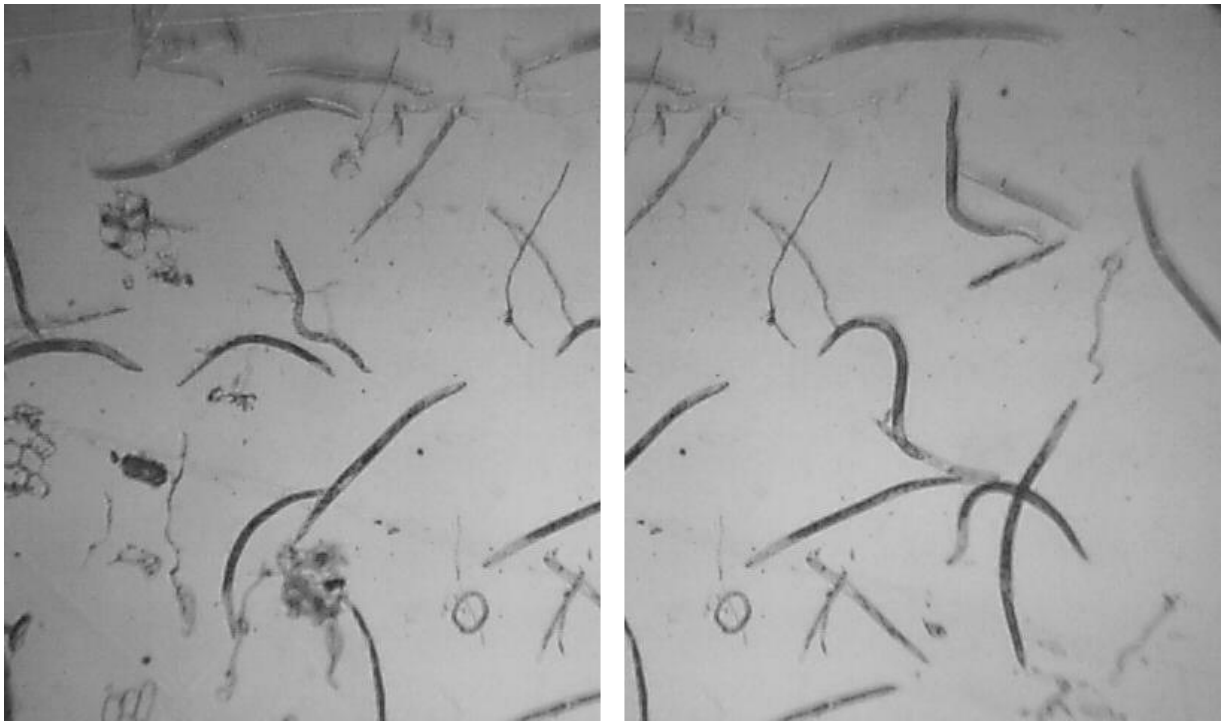


Figura 4.1 Imágenes de entrada que solamente presentan traslación en el plano horizontal

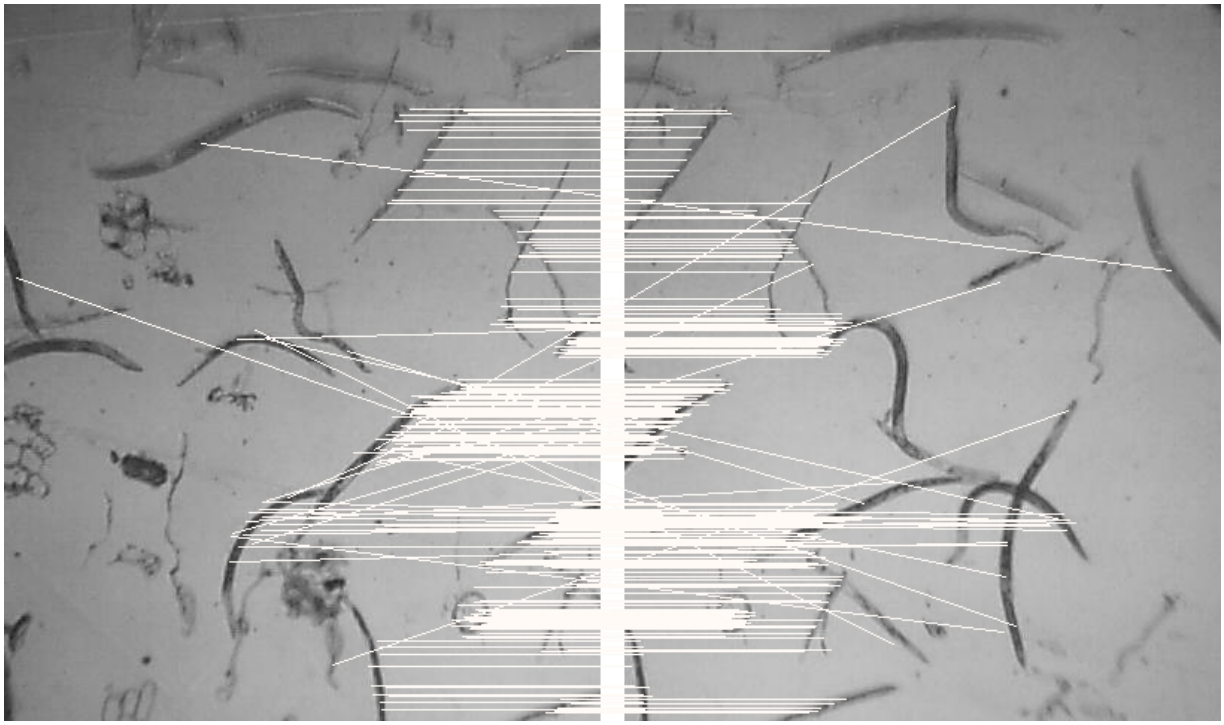


Figura 4.2 Correspondencias entre los puntos de control obtenidos para las imágenes de la figura 4.1

Se puede llevar a cabo un cambio en el parámetro de la clase que realiza la asociación de los puntos de control de manera que sea más estricta en la escogencia de los puntos que se encuentran en ambas imágenes. Esto se muestra en la figura 4.3, donde todos parecen corresponder adecuadamente.

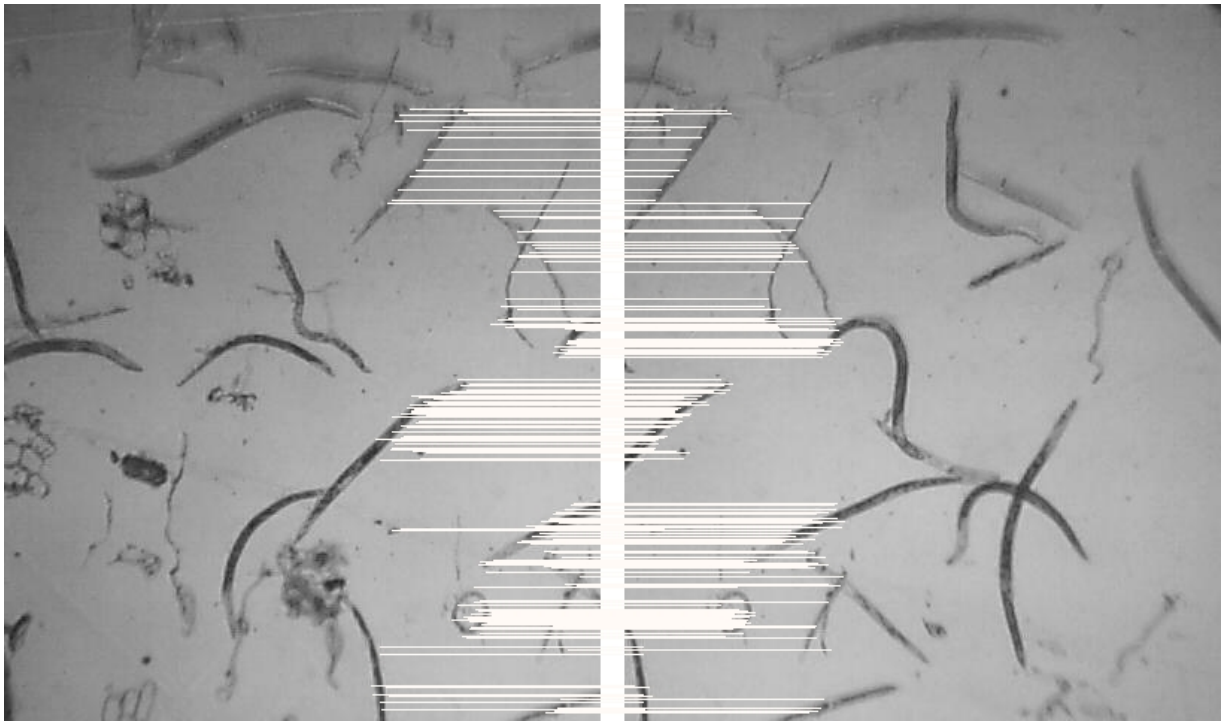


Figura 4.3 Correspondencias obtenidas al limitar la escogencia de puntos mostrada en la figura 4.2

A continuación se muestran en la tabla los resultados de aplicar los algoritmos EML, RANSAC y RANSAC con EML con puntos de control señalados en la figura 4.2. En la tabla 4.2 se muestran los obtenidos para los puntos de control mejorados.

Tabla 4.1 Resultados para el registro de las imágenes en la figura 4.2

		EML		RANSAC		RANSAC&EML	
	Teórico	Experimental	Diferencia	Experimental	Diferencia	Experimental	Diferencia
Traslación en x	-240	90.019	330.019	-239.996	0.004	-239.996	0.004
Traslación en y	0	28.3259	28.3259	-0.0072416	0.0072416	-0.0069612	0.0069612
Ángulo (rad)	0	-3.13273	3.13273	0.000021	0.000021	0.000021	0.000021
Escala	1	0.485423	0.514577	1	0	1	0

Tabla 4.2 Resultados para el registro de las imágenes en la figura 4.3

		EML		RANSAC		RANSAC&EML	
	Teórico	Experimental	Diferencia	Experimental	Diferencia	Experimental	Diferencia
Traslación en x	-240	-240.013	0.013	-240.013	0.013	-240.013	0.013
Traslación en y	0	0.0013621	0.0013621	0.0011882	0.0011882	0.0013621	0.0013621
Ángulo (rad)	0	0.0000039	0.0000039	0.0000041	0.0000041	0.0000039	0.0000039
Escala	1	1.00002	0.00002	1.00002	0.00002	1.00002	0.00002

En la tabla 4.1 se observa que los resultados obtenidos para RANSAC y RANSAC&EML son iguales en traslación en el eje x, ángulo de rotación y escala. Solamente se diferencian en la traslación en el eje y, donde RANSAC&EML presenta una diferencia de 0.0002804 pixeles menos que RANSAC, que no se puede considerar significativa.

El principal problema se presenta con EML, como era esperado, pues los cuatro parámetros resultantes se desvían completamente de los valores esperados. Como se mencionó anteriormente, esto se debe a que el algoritmo asume que el ruido en la medición de los puntos de control correspondientes es una distribución de Gauss. Esto no es correcto pues este ruido no presenta este tipo de distribución ya que las diferencias en la medición de cada pareja de puntos es independiente y de ninguna manera se garantiza que el ruido tenga esta forma.

En la tabla 4.2 se muestran los resultados obtenidos al aplicar un parámetro de escogencia de puntos correspondientes más estricto, de tal forma que no hay ningún error en estos puntos asociados. Con esto se logra disminuir el ruido aproximadamente a cero y el hecho de que tenga que ser gaussiano puede ser ignorado. Los parámetros obtenidos para EML y RANSAC&EML son iguales entre sí y muy cercanos a los teóricos, lo que confirma la mejoría lograda. Para RANSAC se logró mejorar la traslación en el eje y pero se empeoró el ángulo de rotación con respecto a los otros dos métodos, aunque las diferencias son de 0.0002 y 0.0000002, respectivamente, por lo que no tienen mucha relevancia.

También se puede notar que el uso de RANSAC como filtro de los puntos de control para obtener los que son consistentes permite mejorar notablemente el rendimiento del algoritmo EML. Esta manera de hacerlo es mejor que variar el umbral

para determinar cuáles puntos entra cada imagen son correspondientes, pues dependiendo del par de imágenes, se tendría que estar cambiando este valor porque algunas veces puede no ser suficiente para tener la cantidad de puntos de control necesaria para estimar la transformación, y en otras ocasiones puede ser muy alto, produciendo puntos que no corresponden realmente.

Se comprueba que el algoritmo EML ejecutado independientemente no es adecuado para cumplir con los objetivos que se buscan con la aplicación de que sea automática porque se tendría que hacer el ajuste de los parámetros de la clase que asocia los puntos de control para cada par de imágenes de entrada. Además, dependiendo del ruido, se puede dar el caso en que se encuentren muy pocos puntos, uno o varios de ellos sean incorrectos. Lo ideal sería que estos parámetros sean definidos en una primera fase de configuración y luego se ejecute la herramienta completa de procesamiento de todas las imágenes de los nematodos.

Ahora se continúa con la comparación de los algoritmos RANSAC y RANSAC&EML para diferentes tipos de transformaciones geométricas. Para aumentar la complejidad de la alineación de las imágenes se busca ahora obtener los parámetros de una transformación que solamente involucra desplazamiento en los dos ejes.

En la figura 4.4 se muestran las dos imágenes de entrada utilizadas para aplicar los algoritmos RANSAC y RANSAC&EML. De nuevo, son partes de la misma imagen pues esto permite tener mayor control sobre las pruebas, ya que se conoce con certeza la transformación que se debe hacer para alinearlas correctamente. Se disminuyó el contraste y aumentó el brillo en la primera imagen para observar el efecto en los algoritmos.

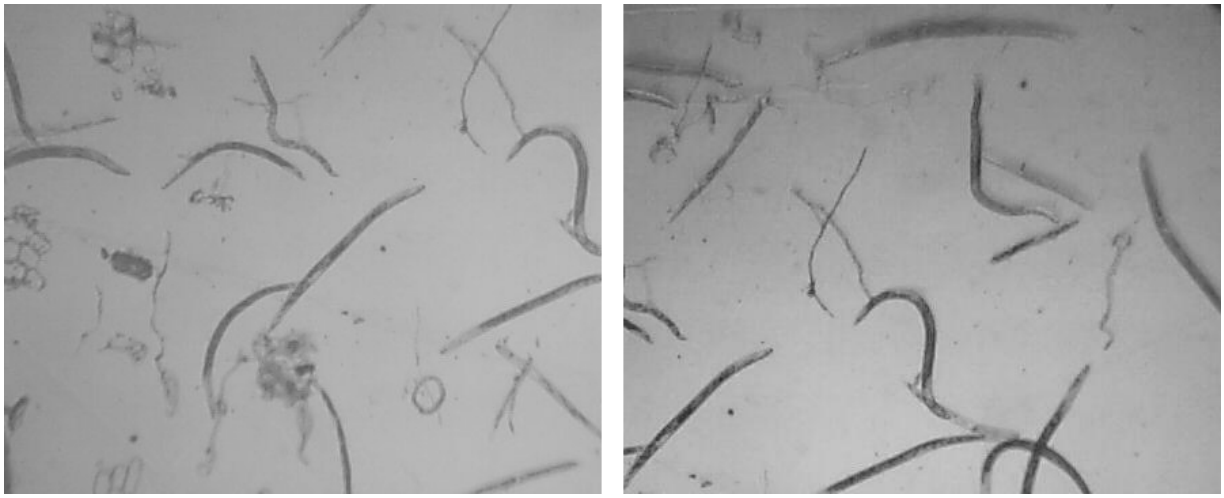


Figura 4.4 Imágenes de entrada que presentan traslación en los dos ejes y diferencias de brillo y contraste

Los resultados obtenidos para estas pruebas se encuentran en las tablas 4.3 y 4.4, siendo la primera con las imágenes originales y la segunda con el cambio en el brillo y contraste en una de ellas. Las diferencias entre las dos tablas se pueden considerar insignificantes, por lo que los cambios en brillo y contraste no afectan considerablemente el registro de las imágenes.

En la tabla 4.4 las diferencias entre los dos algoritmos son despreciables. Para la traslación en el eje x y la escala los valores son iguales. Para la traslación en el eje y, la diferencia con respecto al valor teórico al usar RANSAC es 0.006 más que cuando se usa RANSAC&EML. Para el ángulo de rotación, la diferencia es mayor en el caso de RANSAC&EML pero solamente por 0.000005 radianes. Por lo tanto, para traslación en las dos direcciones y diferencias en el brillo y contraste, los algoritmos calculan una transformación prácticamente igual a la teórica.

Tabla 4.3 Resultados para el registro de las imágenes en la figura 4.4 sin cambios en brillo y contraste

		RANSAC		RANSAC&EML	
	Teórico	Experimental	Diferencia	Experimental	Diferencia
Traslación en x	-200	-199.993	0.007	-199.993	0.007
Traslación en y	120	120	0	120	0
Ángulo (rad)	0	0.000023	0.000023	0.000023	0.000023
Escala	1	1.00001	0.00001	1.00001	0.00001

Tabla 4.4 Resultados para el registro de las imágenes en la figura 4.4

		RANSAC		RANSAC&EML	
	Teórico	Experimental	Diferencia	Experimental	Diferencia
Traslación en x	-200	-199.993	0.007	-199.993	0.007
Traslación en y	120	119.984	0.016	119.99	0.01
Ángulo (rad)	0	0.000020	0.000020	0.000025	0.000025
Escala	1	1.00004	0.00004	1.00004	0.00004

Para el caso de la rotación con traslación, las imágenes utilizadas se muestran en la figura 4.5. Lo que se hizo fue rotar 30 grados (0.523599 radianes) la imagen original completa y luego tomar una sección rectangular de ésta.

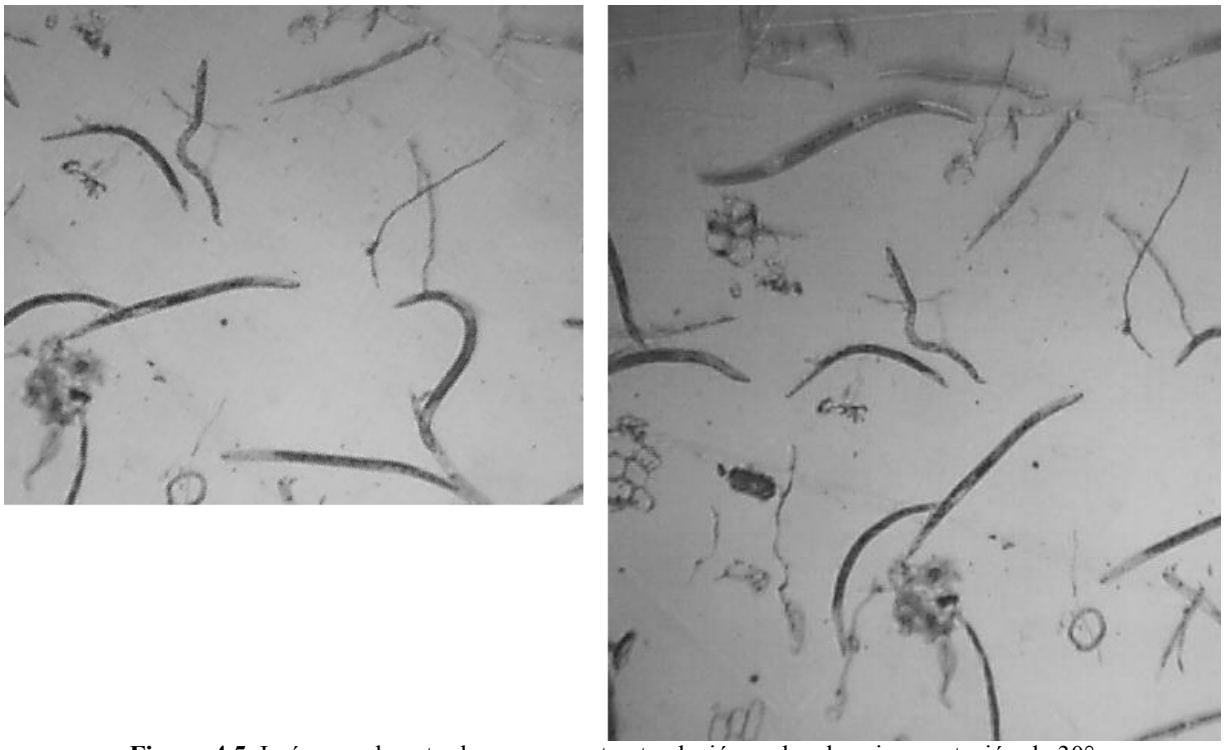


Figura 4.5 Imágenes de entrada que presentan traslación en los dos ejes y rotación de 30°

En este caso, el algoritmo de RANSAC no podía calcular suficientes puntos de control consistentes (“inliers”) para obtener la transformación afín si la entrada eran los puntos obtenidos con el filtro que usa RANSAC con transformación de similitud, a pesar de que se variaron los parámetros de esa clase de múltiples maneras. Lo que

se tuvo que hacer fue aplicar RANSAC de una vez con transformación afín, pues de esta manera si era posible estimar los parámetros. En el caso de RANSAC&EML se continuó haciendo de la misma manera. La tabla 4.5 contiene los valores estimados por ambos algoritmos y las diferencias con los teóricos.

En esta tabla se puede notar que sí hay importantes diferencias en los valores estimados de los cuatro parámetros para RANSAC. Mientras que RANSAC&EML presenta mucho mejores resultados pues continúa con diferencias insignificantes con respecto a los valores teóricos. Entonces cuando hay traslación y rotación, principalmente por esta última, es más confiable RANSAC&EML que puramente RANSAC.

Tabla 4.5 Resultados para el registro de las imágenes en la figura 4.5

		RANSAC		RANSAC&EML	
	Teórico	Experimental	Diferencia	Experimental	Diferencia
Traslación en x	51.3997	35.3648	16.0349	51.4515	0.0518
Traslación en y	192.027	183.335	8.692	191.97	0.057
Ángulo (rad)	-0.523599	-0.421718	0.101881	-0.524076	.000477
Escala	1	1.10238	0.10238	1.00096	0.00096

La siguiente prueba efectuada es muy parecida a la anterior, ya que los parámetros de la transformación son los mismos pero se incluye ruido blanco gaussiano en una de las imágenes, lo que podría dificultar el registro de las imágenes. Las dos imágenes utilizadas se muestran en la figura 4.6.

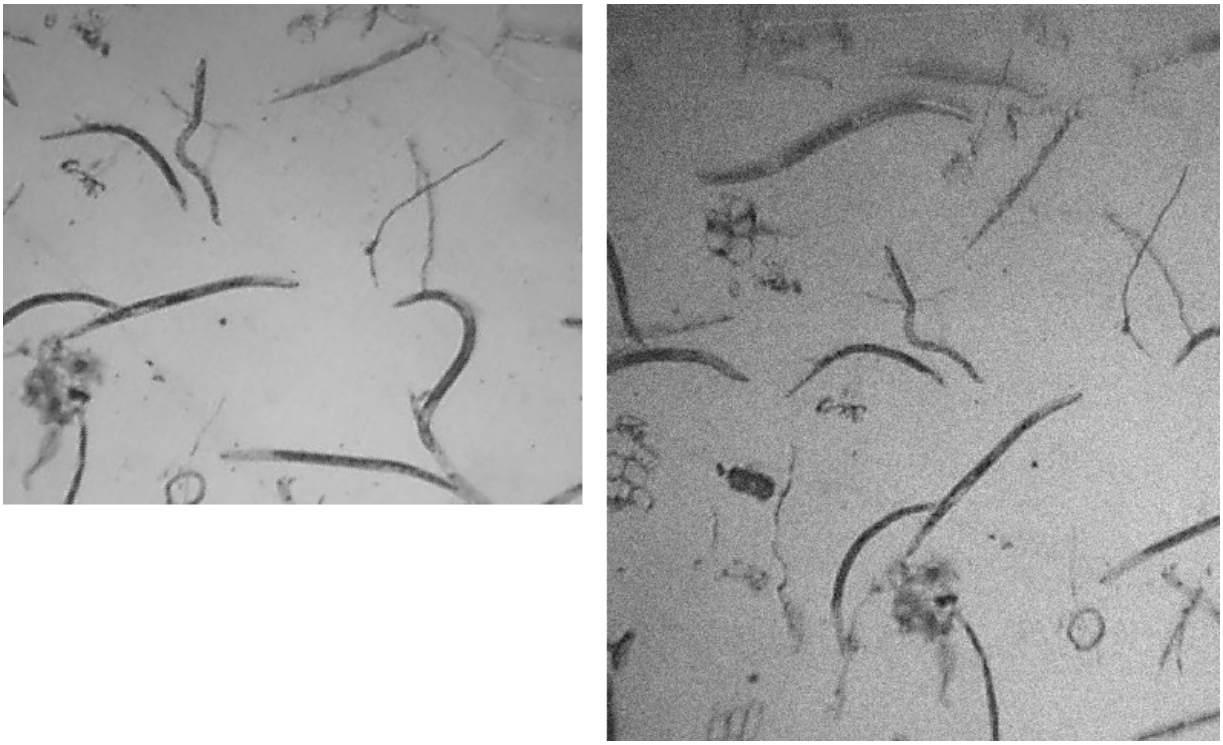


Figura 4.6 Imágenes de entrada con traslación, rotación de 30° y ruido blanco gaussiano en una de ellas

Por ser este caso más complejo que el anterior, también se presentaron problemas con RANSAC para estimar la trasformación afín, pues dependiendo de la cantidad de puntos de control, en algunas ocasiones no podía calcularla. En cambio RANSAC&EML no presentó problemas con la cantidad de puntos de control. Los datos adquiridos para estas imágenes se encuentran en la siguiente tabla.

Tabla 4.6 Resultados para el registro de las imágenes en la figura 4.6

		RANSAC		RANSAC&EML	
	Teórico	Experimental	Diferencia	Experimental	Diferencia
Traslación en x	51.3997	63.3145	11.9148	51.2903	0.1094
Traslación en y	192.027	182.627	9.4	192.034	0.007
Ángulo (rad)	-0.523599	-0.985476	0.461877	-0.524268	0.000669
Escala	1	0.592896	0.407104	1.00038	0.00038

De nuevo, los valores obtenidos para RANSAC son completamente diferentes a los teóricos. Sobre todo el factor de escala y el ángulo de rotación pues las diferencias obtenidas en relación con los valores esperados indican claramente que las imágenes no se encuentran alineadas. En cambio, para el algoritmo RANSAC&EML se siguen presentando variaciones muy pequeñas con respecto a los parámetros teóricos, lo que indica que también funciona cuando se presenta cierta cantidad de ruido entre las imágenes.

La última prueba que se llevó a cabo fue para evaluar el factor de escala como otro parámetro que puede variar en el caso de las imágenes de los nematodos. Entonces se redujo por un factor de 0.7 la imagen que contiene rotación y traslación para evaluar si el algoritmo RANSAC&EML puede lidiar con cambios de escala. Cabe aclara que si esta imagen fue reducida al multiplicar sus dimensiones por 0.7, entonces en el momento de alinearla con la otra, el algoritmo debe multiplicar por el inverso, es decir, 1/0.7 = 1.42857. La figura 4.7 muestra las dos imágenes que se deben alinear, siendo la de la izquierda la que fue reducida.

En la tabla 4.7 se muestran los valores obtenidos para los dos algoritmos. Para RANSAC al agregar el factor de escala aumentó la diferencia entre los valores calculados y los teóricos. En cambio para el algoritmo propuesto RANSAC&EML el aumento en las diferencias de los parámetros calculados con respecto a los valores esperados fue mínimo. Sin embargo, por el grado de complicación existente en la transformación teórica de las imágenes, que incluye traslación en los dos ejes, rotación y un factor de escala, es esperado que los valores sean los que muestran un mayor error.

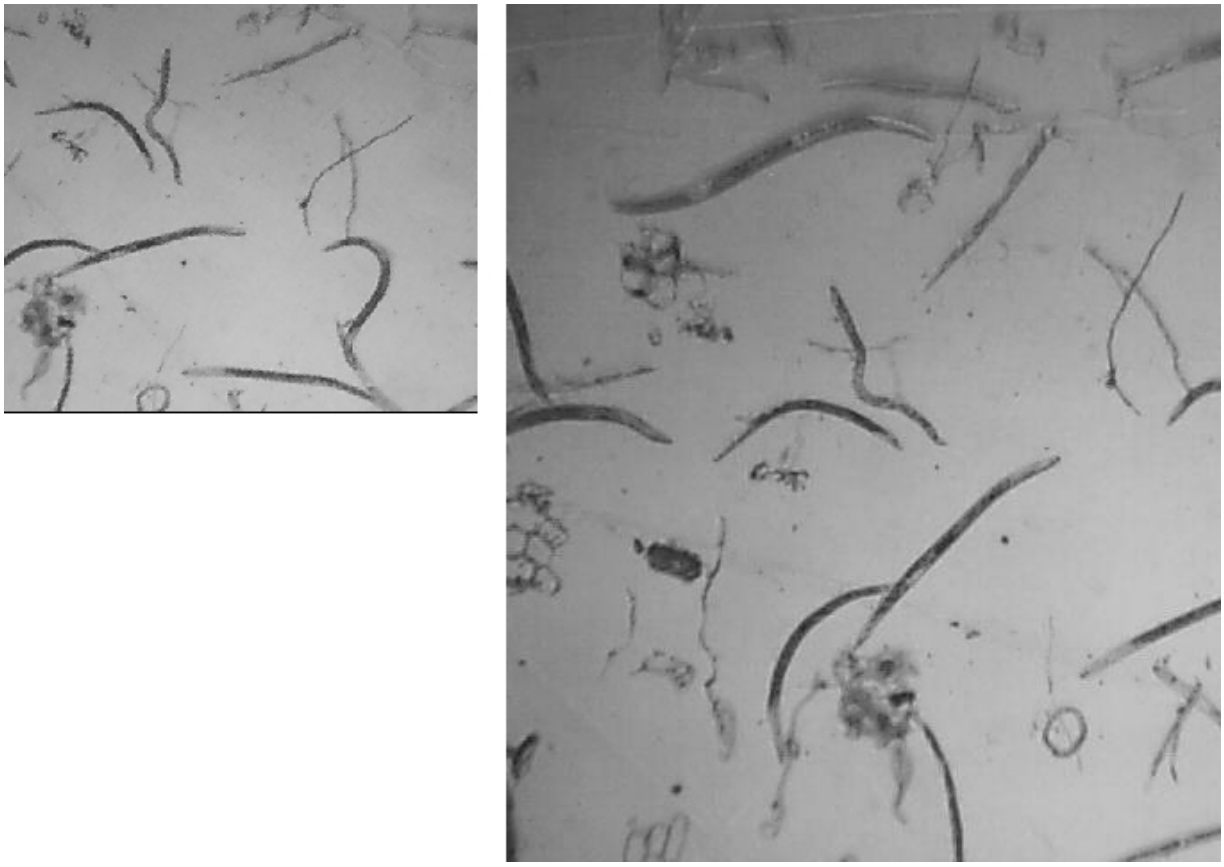


Figura 4.7 Imágenes de entrada con traslación, rotación de 30° y un factor de escala

Tabla 4.7 Resultados para el registro de las imágenes en la figura 4.7

		RANSAC		RANSAC&EML	
	Teórico	Experimental	Diferencia	Experimental	Diferencia
Traslación en x	51.3997	611.107	559.7073	50.2362	1.1635
Traslación en y	192.027	2035.17	1843.143	193.159	1.132
Ángulo (rad)	-0.523599	-1.19791	0.674311	-0.524714	0.001115
Escala	1.42857	7.58433	6.15576	1.42568	0.00289

Con los resultados anteriores se puede afirmar que la utilización del algoritmo RANSAC para obtener los puntos de control más consistentes y luego la aplicación del algoritmo EML, es el método que produce la mejor transformada afín que puede ser utilizada para varias aplicaciones, entre ellas la presentada en este proyecto para mejorar el proceso de análisis de los nematodos.

4.2 Unión de imágenes

Se presentan los resultados de aplicar el algoritmo descrito en imágenes con diferentes características, desde el caso más simple en que se dividió una imagen manualmente en dos y se unieron las partes con este método, hasta el caso en que se usaron imágenes reales traslapadas.

Para crear la imagen compuesta por las dos iniciales, se tiene que realizar el registro de las imágenes en primera instancia para alinearlas, por lo que los resultados en parte dependen de este proceso. Sin embargo, como parte de las ventajas del algoritmo GIST con las variaciones descritas, se espera que si hay algunas diferencias en la alineación de las imágenes, éstas puedan ser corregidas adecuadamente.

Como se realizó con el registro de imágenes, se dividió una imagen en dos de forma manual y se volvió a unir utilizando el algoritmo propuesto, las imágenes de entrada son las mismas mostradas anteriormente en la figura 4.1.

La imagen resultante de la unión de estas dos se muestra en la figura 4.9. Pero antes, en la figura 4.8 se presenta imagen de partida al tomar pixeles directamente de las imágenes de entrada antes de empezar las iteraciones. Como se puede notar, las diferencias entre ambas imágenes son imperceptibles pues la alineación fue muy precisa, entonces en un inicio parece no importar de que imagen se toman los pixeles. La zona de traslape se logra formar sin que se distinga la zona de traslape entre las dos y conservando las características de ambas. Este caso es uno de los más simples para el algoritmo pues son imágenes ideales y la zona de traslape es igual a la que se forma cuando se unen dos fotografías panorámicas, que es el objetivo del artículo que presenta GIST.

Figura 4.8 Imagen de partida la unión de las dos mostradas en la figura 4.1

Figura 4.9 Imagen final compuesta por la unión de las dos mostradas en la figura 4.1

Ahora, con las modificaciones realizadas al calcular la máscara que permite que se dé una unión gradual entre las imágenes, se esperaba formar una imagen compuesta de dos que presentan traslape tanto en el eje x como en el eje y. En la figura 4.11 se presenta la unión de las dos imágenes que se mostraron en la figura 4.4 que además del traslape cuenta con diferencias de brillo e intensidad. En la figura 4.10 se muestra la imagen inicial antes de empezar las iteraciones del algoritmo de unión.

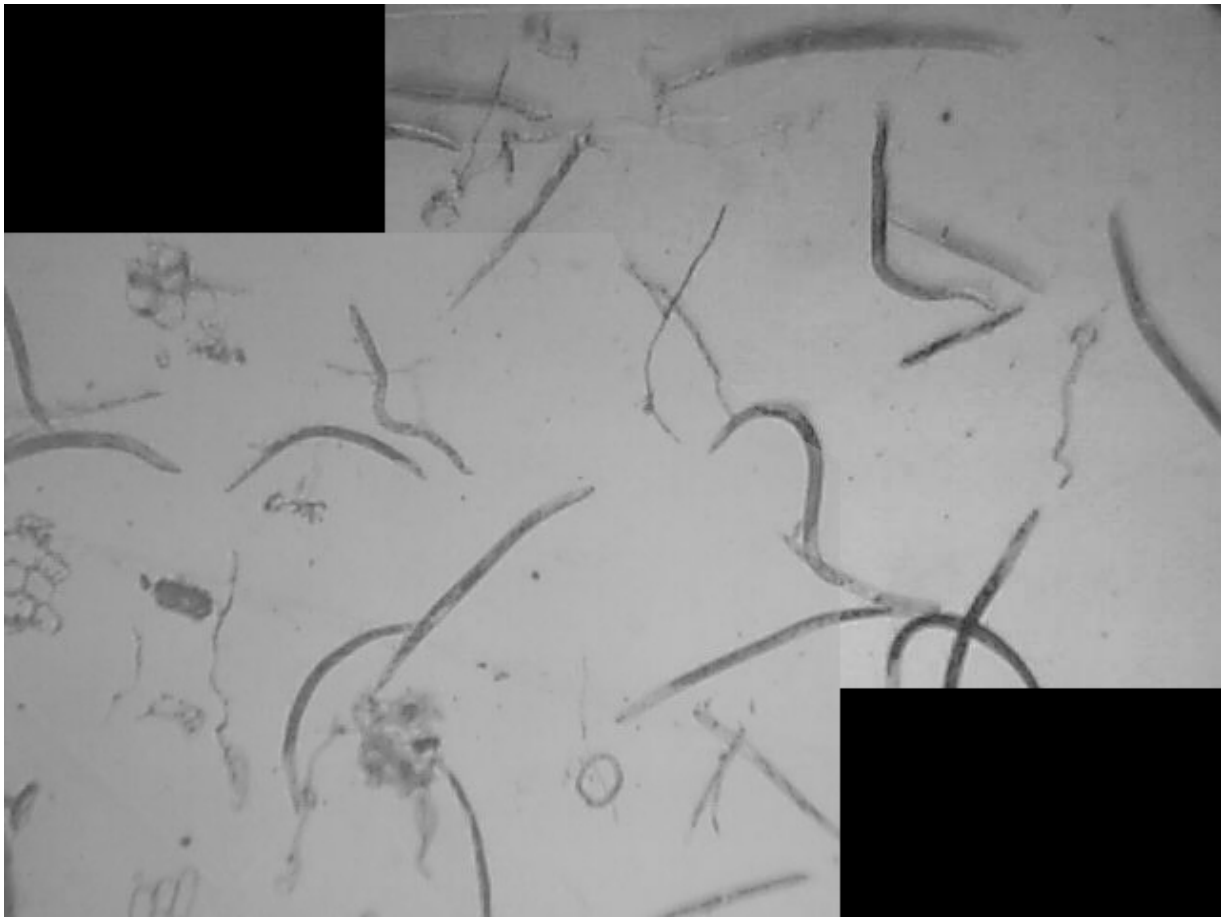


Figura 4.10 Imagen departida para la unión de las imágenes con diferente brillo y contraste de la figura 4.4

En la imagen de partida se pueden distinguir parte de las orillas de las zona de traslape y los pixeles que se tomaron inicialmente de una imagen y de la otra. En este caso se puede notar ligeramente lo que logra el algoritmo de unión.

Se puede comprobar la eficiencia del algoritmo en la imagen final pues es difícil distinguir las diferencias de brillo y contraste ya que GIST logra hacer gradual la transición de una imagen a la otra.

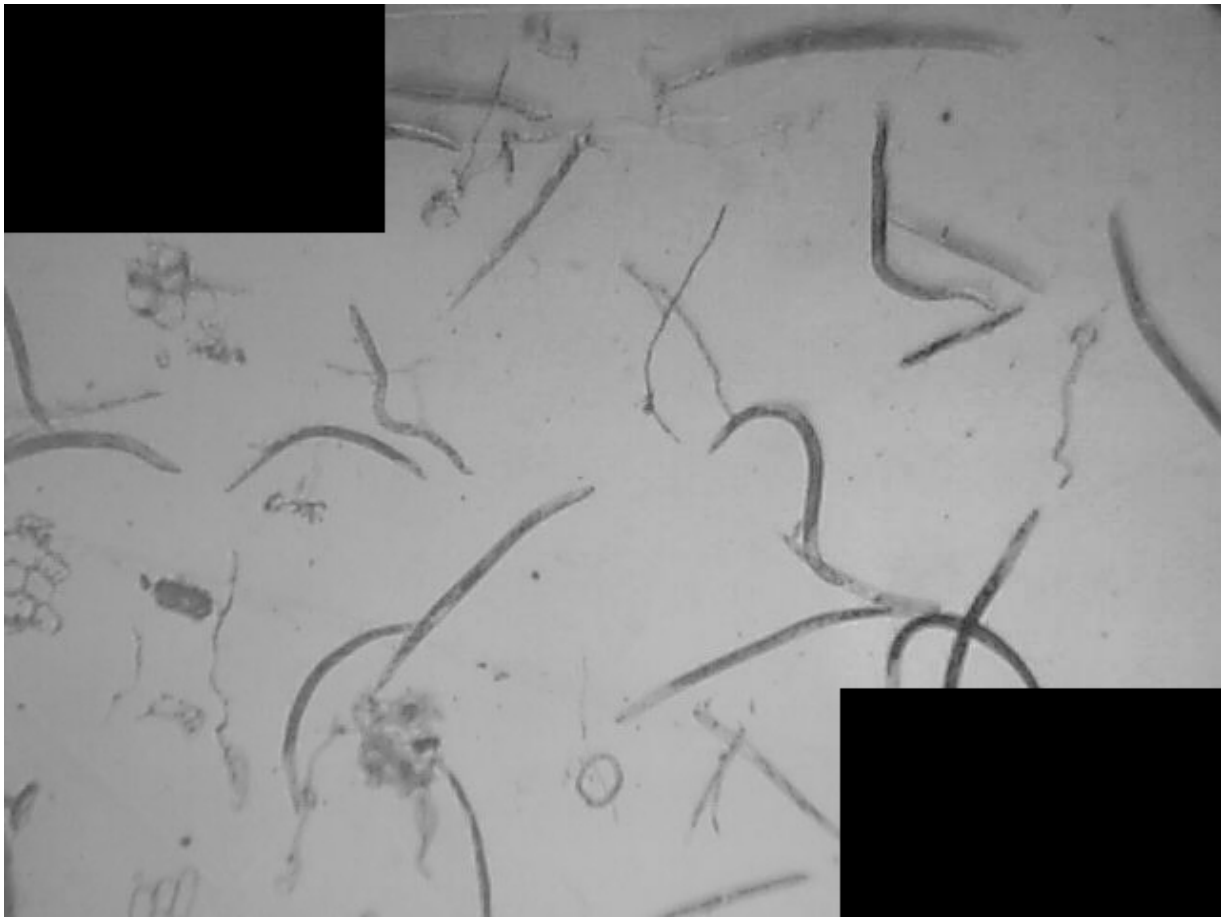


Figura 4.11 Imagen resultante de la unión de las imágenes con diferente brillo y contraste de la figura 4.4

También debe ser posible unir imágenes en las cuales existe una rotación entre ellas, como es el caso de las mostradas en la figura 4.5. Al igual que en los demás casos, el algoritmo recibe como entradas las dos imágenes y la transformación geométrica que las alinea y la salida sería la imagen compuesta por las dos entrantes. En la figura 4.12 se muestra la imagen de partida y en la 4.13 el resultado final obtenido cuando la matriz incluye una rotación y traslaciones en los dos ejes.

Figura 4.12 Imagen de partida cuanto existe rotación y traslación entre las imágenes entrantes

En este caso las imágenes de entrada son tomadas de la misma y no presentan modificaciones, por lo que los resultados dependen más de la precisión del algoritmo de registro de las imágenes, tal como se dio en el primer caso (figuras 4.8 y 4.9). La diferencia entre la imagen de partida y la final es imperceptible.

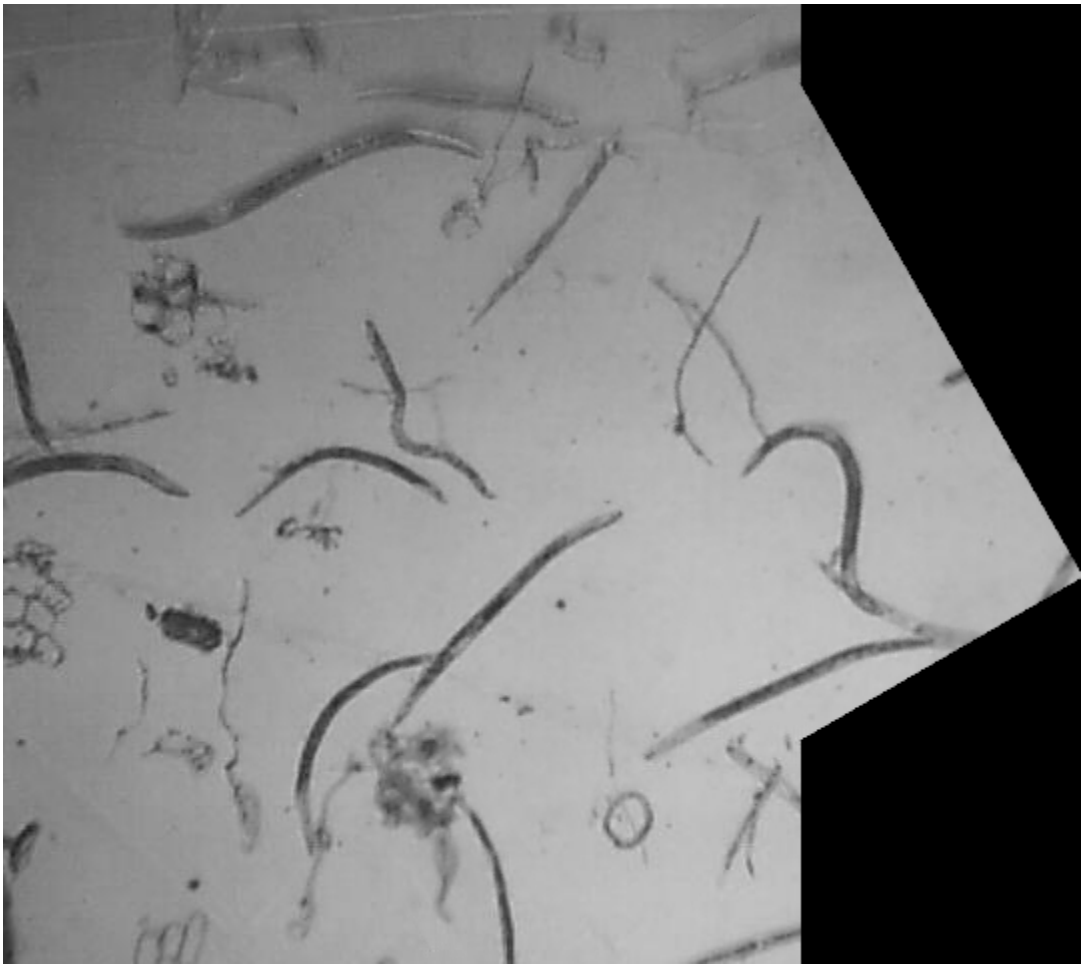


Figura 4.13 Imagen resultante cuanto existe rotación y traslación entre las imágenes entrantes

Como se puede observar en la figura 4.13 no se distinguen los bordes de cada imagen que delimitan la zona de traslape. También se logran unir las imágenes sin perder características de ninguna de las dos.

Ahora, uno de los casos más complicados es cuando se tiene ruido, como en las imágenes de la figura 4.6 y es algo que siempre se va a presentar, tal vez no tan notorio como el mostrado en esa figura pero siempre existe. Por lo que se debe lograr el mismo objetivo de no perder características de cada una de las imágenes porque no se puede saber si alguna es más importante que otra, y que la transición sea invisible. En la figura 4.14 se muestra la imagen inicial y en la 4.15 se presenta el resultado de aplicar el algoritmo propuesto basado en GIST.

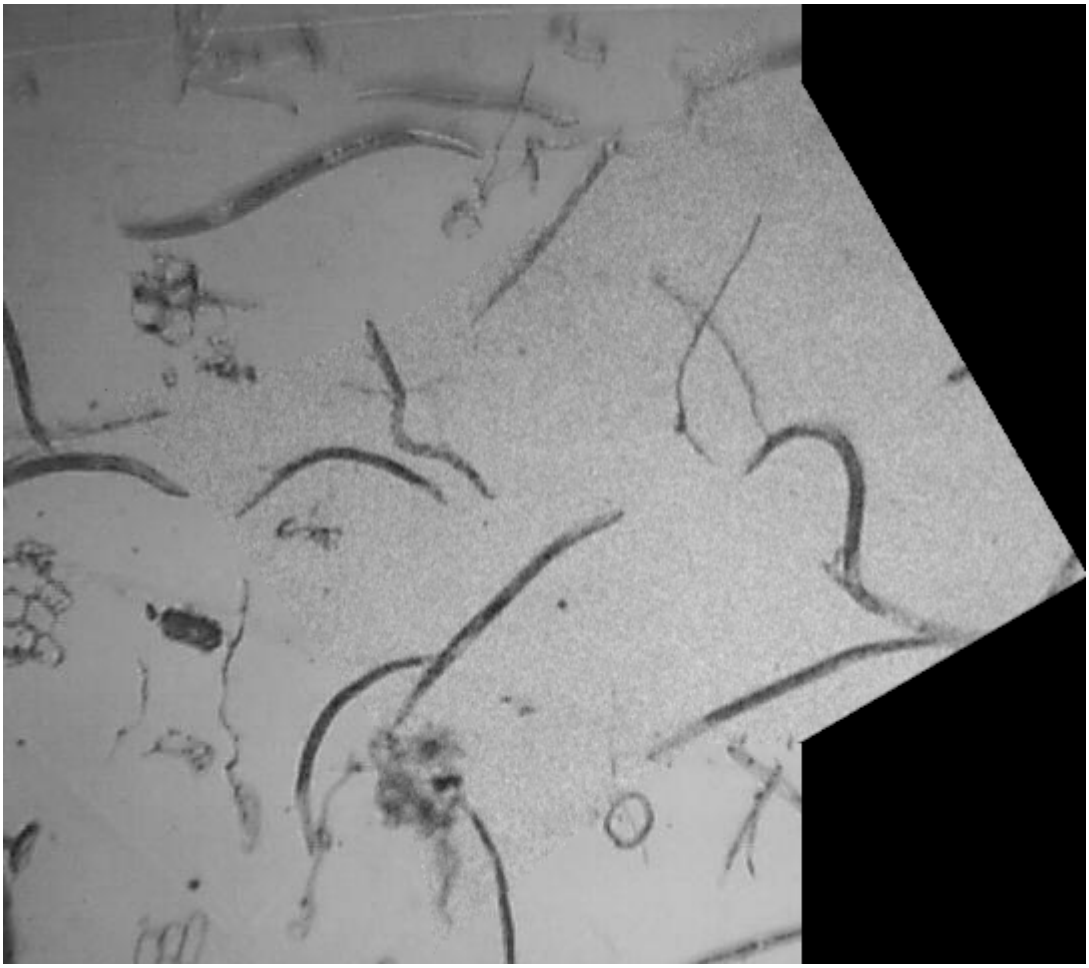


Figura 4.14 Imagen inicial antes de correr el algoritmo de unión cuando una entrada presenta traslación, rotación y ruido

En este caso es uno de los que más fácilmente se aprecian las diferencias entre la imagen de partida que solamente toma pixeles de las dos imágenes de entrada y la imagen final después de correr el algoritmo de unión. Claramente se distinguen las diferencias en la zona de traslape.

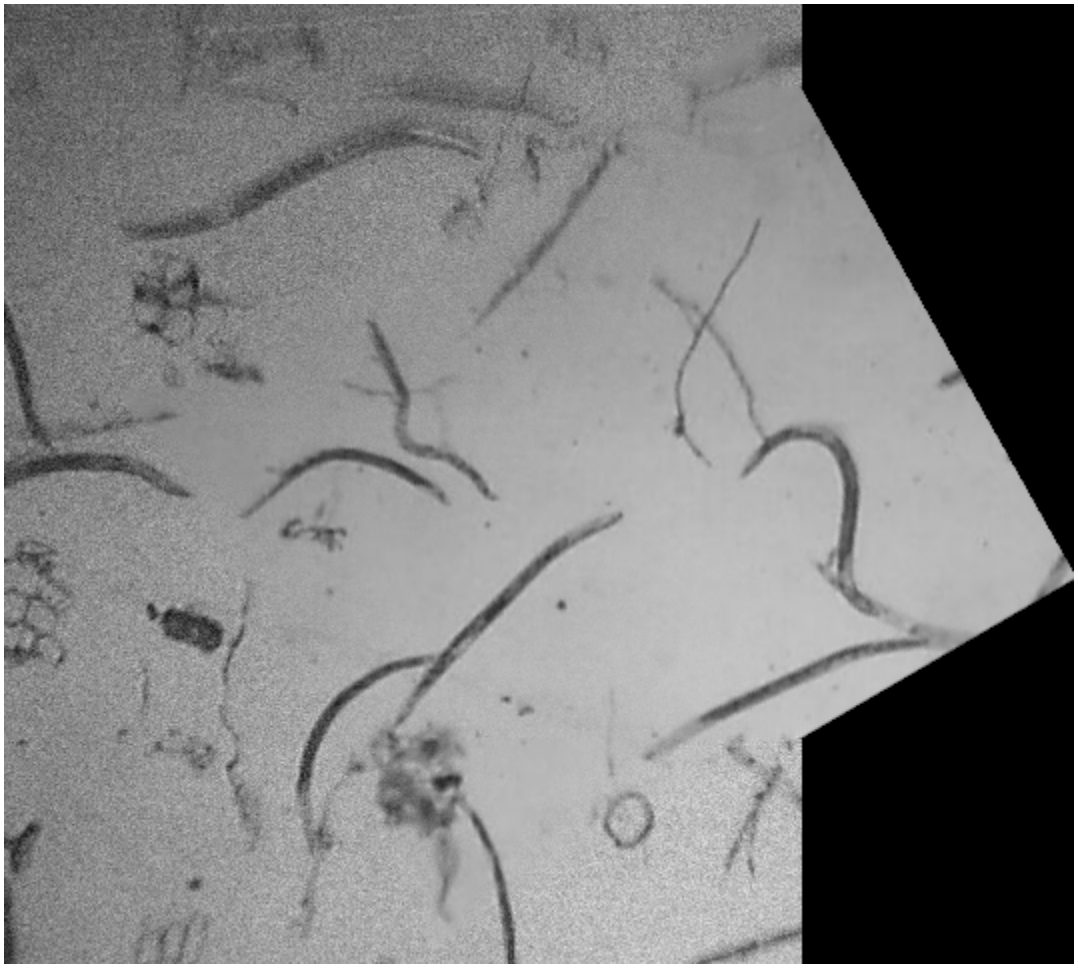


Figura 4.15 Imagen compuesta por una con traslación y rotación unida a otra con ruido

Se puede apreciar que se logra el objetivo exitosamente en esta imagen pues no se elimina el ruido porque la idea es que las características de cada una se mantengan y a la vez se puede ver que la transición entre las imágenes es gradual.

Por último, se lograron obtener dos imágenes de nematodos tomadas por separado que tienen una zona de traslape considerable y se muestran en la figura 4.16. Se realizó la unión de las mismas, lo cual se presenta en la figura 4.17. Las imágenes no tienen muchas diferencias, la mayoría son detalles, pero se demuestra de esta manera que el algoritmo funciona bien para imágenes no muy complejas. Esto pues en este caso pueden darse imágenes en que el nemátodo se desplace mientras se toman dos fotografías, por lo que se necesitaría un algoritmo más robusto.

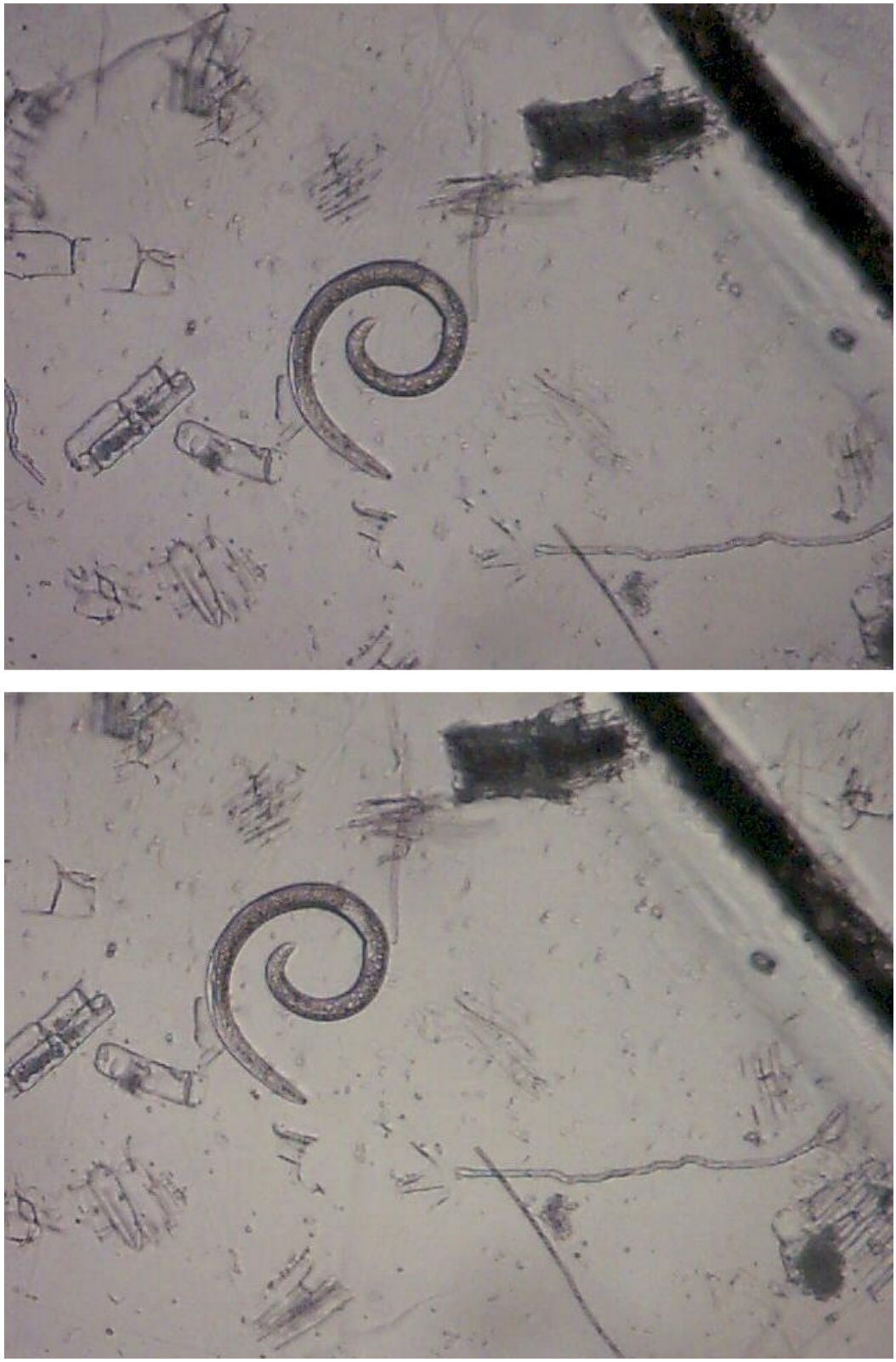


Figura 4.16 Dos imágenes reales de un nemátodo tomadas en diferentes instantes

Figura 4.17 Imagen resultante de la unión de las dos mostradas en la figura 4.16

Como se puede ver, la imagen muestra la aplicación funcionando con los dos algoritmos implementados, uno para el registro de imágenes y otro para la unión de éstas. Los resultados son los esperados pues se obtiene una imagen alineada que mantiene las características que aportan las dos entrantes y a la vez hace una transición gradual entre las mismas, pues en este caso particular existe una diferencia de intensidad entre las dos imágenes.

Capítulo 5:Conclusiones y recomendaciones

5.1 Conclusiones

El registro de imágenes realizado se basó en la implementación de un algoritmo llamado Exact Maximum Likelihood (EML) que consiste en la obtención de la trasformación afín utilizando puntos de control e intensidad. El algoritmo EML funciona correctamente solo si se cuenta con puntos de control que tienen ruido gaussiano. La obtención de puntos de control automáticamente no permite garantizar que el ruido o error entre los puntos de control correspondientes presente este tipo de distribución.

Se comprobó que los resultados de aplicar EML tomando todos los puntos obtenidos por el algoritmo de asociación de puntos característicos pueden llegar a ser completamente diferentes a los parámetros de la matriz de transformación esperada.

Se propuso utilizar el algoritmo RANSAC que permite obtener los puntos de control consistentes, es decir, solamente los puntos que cumplen, dentro de un margen de error, con la misma transformación de similitud. Se utilizó la transformación de similitud para obtener estos puntos pues en el caso de las imágenes de los nematodos, los movimientos que se pueden dar solamente incluyen traslación, rotación y cambio de escala.

Entonces, se implementó un algoritmo de registro de imágenes que combina RANSAC como filtro de los puntos de control y EML como algoritmo de cálculo de los parámetros de la transformada afín. Se llamó a este algoritmo RANSAC&EML.

Se escogió el método RANSAC para calcular la transformada afín como algoritmo de comparación de registro de imágenes. Para hacer la comparación de manera equitativa, este método también usó los puntos de control filtrados.

Para pares de imágenes en las que solamente se presenta traslación, RANSAC y RANSAC&EML obtienen transformaciones afines casi idénticas a la esperada.

Se comprobó que para transformaciones que incluyen rotación, el algoritmo RANSAC&EML obtiene los parámetros más cercanos a los esperados.

Cuando se introduce ruido a alguna de las imágenes o se agrega un factor de escala, se reduce el desempeño de RANSAC pero con RANSAC&EML se siguen obteniendo valores muy similares a los teóricos.

Por lo tanto, después de realizar las comparaciones se determinó que EML sin el filtro de los puntos de control tiene un desempeño muy bajo. RANSAC para obtener la transformada afín tiene un desempeño bajo en los casos de rotación con un factor de escala o en imágenes con ruido. RANSAC&EML es el que presentó mejor desempeño ante las diferentes comparaciones que se llevaron a cabo. Este algoritmo fue entonces el que se usó para determinar la matriz de trasformación que permite alinear las imágenes.

El algoritmo de unión de las imágenes se definió e implementó una vez que el algoritmo RANSAC&EML se había comprobado que era eficiente. Los resultados de este algoritmo se midieron en términos de que no se perdieran características de cada imagen y que la transición de una imagen a otra fuera invisible.

Se comprobó que el algoritmo de unión de imágenes funciona para imágenes con ruido gaussiano blanco y cuando se presenta una variación en la intensidad. En las pruebas hechas, la zona de traslape obtenida es una transición gradual de tal manera que no se distingue donde empieza una imagen y termina la otra.

Como última prueba se realizó todo el proceso de composición con dos imágenes de nematodos reales que se traslapaban y presentaban una traslación en los dos ejes, que sería la mayoría de casos que se presentarían para esta aplicación específica. El brillo en una imagen era mayor que en otra, pero el algoritmo completo logró alinear las imágenes perfectamente y se unieron creando una imagen resultante en la que el cambio de brillo es gradual, por lo que no se distingue qué sección pertenece a cuál imagen, cumpliendo el objetivo buscado.

5.2 Recomendaciones

Una de las recomendaciones más importantes para mejorar el desempeño del algoritmo propuesto es llevar a cabo un proyecto de investigación en el que se desarrolle un algoritmo más complejo de unión de imágenes pues en este momento existen maneras de eliminar objetos repetidos. Esto se debe a que hay especies de nematodos que se mueven muy rápido, por lo que en la siguiente imagen el nematodo puede aparecer de nuevo en otra posición, de manera que si se aplica el método implementado en este proyecto, podría aparecer dos veces en la imagen final. También se puede mejorar la imagen implementando un algoritmo que le dé un brillo uniforme.

Además, para el registro de imágenes, se pueden hacer modificaciones al algoritmo EML para que permita calcular los parámetros de una transformación de similitud, pues el que se utilizó solamente lo puede hacer para transformaciones afines. Esto implicaría volver a plantear el algoritmo para el caso de la transformación de similitud específicamente. Incluso se podría intentar implementar una variación de EML que pueda hacerlo para una mayor cantidad de transformaciones y el usuario defina el tipo que va a utilizar.

Bibliografía

[1] Adelson, E. H. et al. “Pyramid Methods in Image Processing”. RCA Engineer. v. 29 (6): 33-41, nov.-dic., 1984.

[2] Alvarado, Pablo. “Segmentation of color images for interactive 3D object retrieval”. Tesis doctoral, RWTH-Aachen, 2004

[3] Bay, H; Tuytelaars, T y Van Gool, L. “SURF: Speeded Up Robust Features”. European Conference on Computer Vision. May., 2006.

[4] Bishop, Christopher. Neural Networks for Pattern Recognition. Estados Unidos: Oxford University Press, 1996.

[5] Fischlet, M. y Bolles, R. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. Communications of the ACM. v.24 (6): 381-395, jun., 1981.

[6] Guzmán, Tomás et al. “Prospección, caracterización y evaluación de las relaciones de organismos benéficos para el control de nematodos patógenos en condiciones del trópico”. Proyecto de Investigación. Instituto Tecnológico de Costa Rica. Mayo, 2005.

[7] InfoJardin.com. *Mailxmail.com (cursos gratis)* [en línea]: *Los problemas de las plantas: Las plagas – nematodos*. <http://www.mailxmail.com/curso/vida/plantas_problemas/capitulo10.htm> [Consulta: 30 abr. 2007].

[8] Jia, Jiaya y Tang, Chi-Keung. “Image Stitching Using Structure Deformation”. IEEE Transactions on Pattern Analysis and Machine Intelligence. v.30 (4): 617-631, abr., 2008.

[9] Levin, Anat et al. “Seamless Image Stitching in the Gradient Domain”. European Conference on Computer Vision. v.4: 377-389, may., 2004.

[10] Li, W. y Leung, H. “ A Maximum Likelihood Approach for Image Registration Using Control Point and Intensity”. IEEE Transactions on Image Processing. v.13 (8): 1115-1127, ago., 2004.

[11] Milgram, David. “Computer Methods for Creating Photomosaics”. IEEE Transactions on Computers. v.24: 1113-1119, nov., 1975.

[12] MTU Department of Computer Science. *Geometric Transformations* [en línea]. <http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/geometry/geo-tran.html> [Consulta: 18 may. 2008].

[13] Reddy, B. S. y Chatterji, B. N. “An FFT-based technique for translation, rotation, and scale invariant image registration”. IEEE Transactions on Image Processing. v.5 (8): 1266-1271, ago., 1996.

[14] Sourceforge.net. *C++ Computer Vision & Robotics Library* [en línea]. <http://sourceforge.net/projects/cvrlib/> [Consulta: 17 may. 2008].

[15] Szeliski, Richard. “Image Alignment and Stitching: A Tutorial”. Microsoft Research Technical Report MSR-TR-2004-92, dic., 2006.

Apéndices

A.1 Ecuación del gradiente del error

La ecuación obtenida para el gradiente del error y la constante K se muestran a continuación:

$$\begin{aligned}\frac{\partial E}{\partial \hat{I}(x,y)} = {} & \hat{I}(x,y)[U(x-1,y)+U(x+1,y)+U(x,y-1)+U(x,y+1)] \\ & -\hat{I}(x-2,y)\,U(x-1,y)-\hat{I}(x+2,y)\,U(x+1,y) \\ & -\hat{I}(x,y-2)\,U(x,y-1)-\hat{I}(x,y+2)\,U(x,y+1) \\ & +K\end{aligned}$$

$$\begin{aligned}K= {} & \\ & W(x-1,y)[I_1(x-2,y)-I_1(x,y)]+[U(x-1,y)-W(x-1,y)][I_2(x-2,y)-I_2(x,y)] \\ & W(x+1,y)[I_1(x+2,y)-I_1(x,y)]+[U(x+1,y)-W(x+1,y)][I_2(x+2,y)-I_2(x,y)] \\ & W(x,y-1)[I_1(x,y-2)-I_1(x,y)]+[U(x,y-1)-W(x,y-1)][I_2(x,y-2)-I_2(x,y)] \\ & W(x,y+1)[I_1(x,y+2)-I_1(x,y)]+[U(x,y+1)-W(x,y+1)][I_2(x,y+2)-I_2(x,y)]\end{aligned}$$

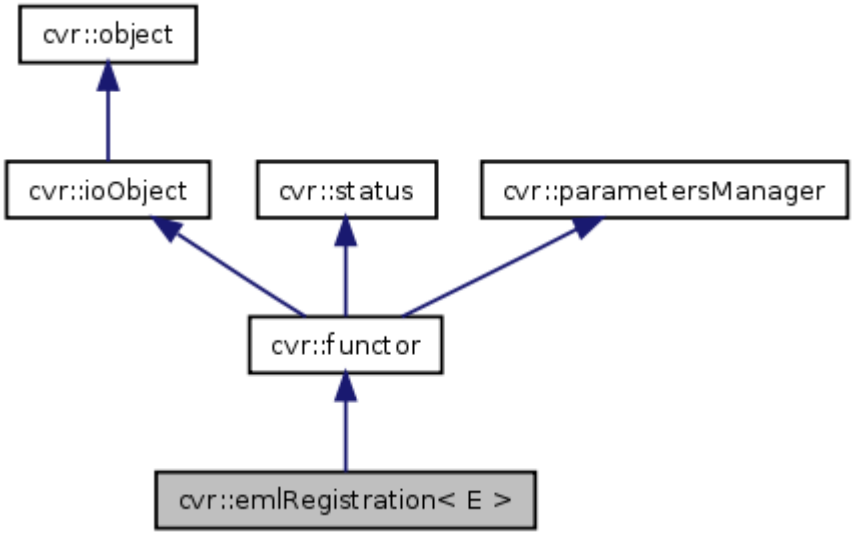
Como se puede notar K solamente depende de variables cuyos valores se obtienen antes de empezar las iteraciones y que no van a cambiar, por lo que se puede calcular inicialmente y sólo una vez.

A.2 Clase implementada para el registro de imágenes en la CVR-Lib

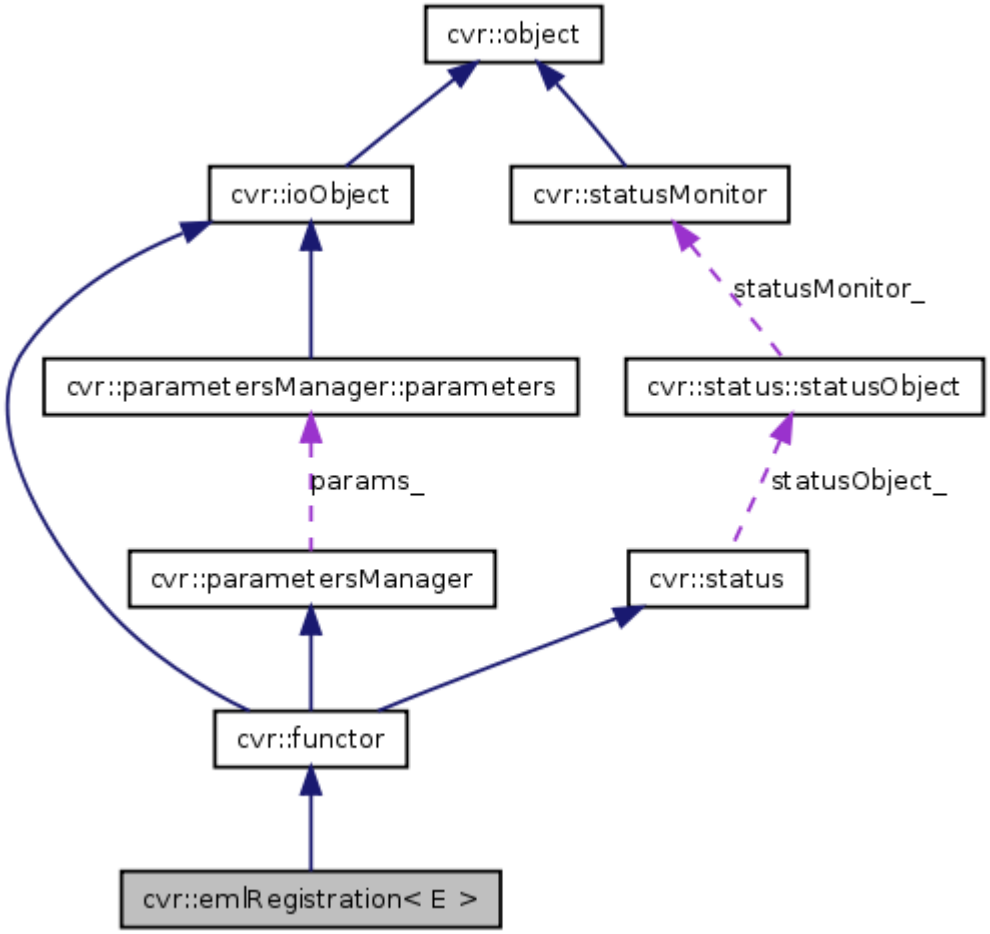
cvr::emlRegistration< E > Class Template Reference

`#include <cvrEmlRegistration.h>`

Inheritance diagram for cvr::emlRegistration< E >:



Collaboration diagram for cvr::emlRegistration< E >:



Detailed Description

template<class E> class cvr::emlRegistration< E >

Class **emlRegistration**.

This class computes an Exact Maximum Likelihood registration method for **image** alignment as proposed in:

Winston Li and Henry Leung. A Maximum Likelihood Approach for Image Registration Using Control Point and Intensity. IEEE Transactions on Image Processing, vol. 13, no. 8, pp. 1115-1126, Aug. 2004.

See also:

emlRegistration::parameters.

Public Member Functions

emlRegistration ()
emlRegistration (const **parameters** &par)
emlRegistration (const **emlRegistration**< E > &other)
virtual **~emlRegistration** ()
bool **apply** (std::vector< **point**< E > > &setA, std::vector< **point**< E > > &setB, **channel** &chnlA, **channel** &chnlB, **vector**< E > &eta) const
emlRegistration< E > & **copy** (const **emlRegistration**< E > &other)
emlRegistration< E > & **operator=** (const **emlRegistration**< E > &other)
virtual const std::string & **name** () const
virtual **emlRegistration**< E > * **clone** () const
virtual **emlRegistration**< E > * **newInstance** () const
const **parameters** & **getParameters** () const

Classes

class **parameters**
*The **parameters** for the class **emlRegistration**.*

Constructor & Destructor Documentation

template<class E> cvr::emlRegistration< E >::emlRegistration ()

Default constructor.

template<class E> cvr::emlRegistration< E >::emlRegistration (const parameters & *par*)

Construct a **functor** using the given **parameters**.

template<class E> cvr::emlRegistration< E >::emlRegistration (const emlRegistration< E > & *other*)

Copy constructor.

Parameters:
other the **object** to be copied

template<class E> virtual cvr::emlRegistration< E >::~emlRegistration () `[virtual]`

Destructor.

Member Function Documentation

template<class E> bool cvr::emlRegistration< E >::apply (std::vector< point< E > > & *setA*, std::vector< point< E > > & *setB*, channel & *chnlA*, channel & *chnlB*, vector< E > & *eta*) const

Operates on the given argument.

Parameters:

setA locations of the control points in the first **image** to be used.
setB locations of the control points in the second **image** to be used.
chnlA first **image channel** used to get the points' intensity values.
chnlB second **image channel** used to get the points' intensity values.
eta resulting transformation **parameters** to register de images.

Returns:

true if apply successful or false otherwise.

template<class E> emlRegistration<E>& cvr::emlRegistration< E >::copy (const emlRegistration< E > & *other*)

Copy data of "other" **functor**.

Parameters:

other the **functor** to be copied

Returns:

a reference to this **functor object**

template<class E> emlRegistration<E>& cvr::emlRegistration< E >::operator= (const emlRegistration< E > & *other*)

Alias for copy member.

Parameters:

other the **functor** to be copied

Returns:

a reference to this **functor object**

template<class E> virtual const std::string& cvr::emlRegistration< E >::name () const `[virtual]`

Returns the complete name of the **functor** class.

Implements **cvr::functor**.

template<class E> virtual emlRegistration<E>* cvr::emlRegistration< E >::clone () const `[virtual]`

Returns a pointer to a clone of this **functor**.

Implements **cvr::functor** .

template<class E> virtual emlRegistration<E>* cvr::emlRegistration< E >::newInstance () const [virtual]

Returns a pointer to a new instance of this **functor**.

Implements **cvr::functor** .

template<class E> const parameters& cvr::emlRegistration< E >::getParameters () const

Returns used **parameters**.

Reimplemented from **cvr::parametersManager** .

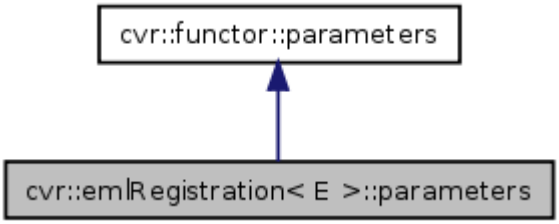
The documentation for this class was generated from the following file:

cvrEmlRegistration.h

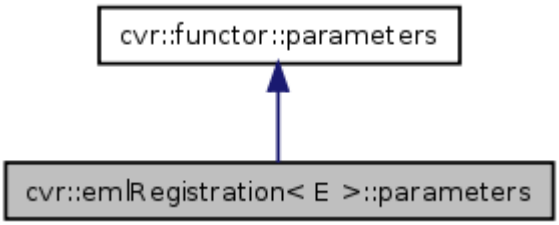
cvr::emlRegistration< E >::parameters Class Reference

`#include <cvrEmlRegistration.h>`

Inheritance diagram for cvr::emlRegistration< E >::parameters:



Collaboration diagram for cvr::emlRegistration< E >::parameters:



Detailed Description

template<class E> class cvr::emlRegistration< E >::parameters

The **parameters** for the class **emlRegistration**.

Public Member Functions

parameters ()
parameters (const **parameters** &other)
~parameters ()
parameters & **copy** (const **parameters** &other)
parameters & **operator=** (const **parameters** &other)
virtual const std::string & **name** () const
virtual **parameters** * **clone** () const
virtual **parameters** * **newInstance** () const
virtual bool **write** (**ioHandler** &handler, const bool complete=true) const
virtual bool **read** (**ioHandler** &handler, const bool complete=true)

Public Attributes

E **threshold**
int **maximumTime**
E **stepLambda**
E **stepAlpha**
int **iterationsAlpha**
int **hx**
int **hy**
int **windowSizeX**
int **windowSizeY**
bool **onlyCP**

Constructor & Destructor Documentation

template<class E> cvr::emlRegistration< E >::parameters::parameters ()

Default constructor.

Reimplemented from **cvr::functor::parameters** .

template<class E> cvr::emlRegistration< E >::parameters::parameters (const parameters & *other*)

Copy constructor.

Parameters:

other the **parameters object** to be copied

template<class E> cvr::emlRegistration< E >::parameters::~parameters () [virtual]

Destructor.

Reimplemented from **cvr::functor::parameters**

Member Function Documentation

template<class E> parameters& cvr::emlRegistration< E >::parameters::copy (const parameters & *other*)

Copy the contents of a **parameters object**.

Parameters:

other the **parameters object** to be copied

Returns:

a reference to this **parameters object**

template<class E> parameters& cvr::emlRegistration< E >::parameters::operator= (const parameters & *other*)

Copy the contents of a **parameters object**.

Parameters:

other the **parameters object** to be copied

Returns:

a reference to this **parameters object**

template<class E> virtual const std::string& cvr::emlRegistration< E >::parameters::name () const `[virtual]`

Returns the complete name of the **parameters** class.

Implements **cvr::functor::parameters**.

template<class E> virtual parameters* cvr::emlRegistration< E >::parameters::clone () const `[virtual]`

Returns a pointer to a clone of the **parameters**.

Implements **cvr::functor::parameters**.

template<class E> virtual parameters* cvr::emlRegistration< E >::parameters::newInstance () const `[virtual]`

Returns a pointer to a new instance of the **parameters**.

Implements **cvr::functor::parameters**.

template<class E> virtual bool cvr::emlRegistration< E >::parameters::write (ioHandler & *handler*, const bool *complete* = `true`) const `[virtual]`

Write the **parameters** in the given **ioHandler**.

Parameters:

handler the **ioHandler** to be used
complete if true (the default) the enclosing begin/end will be also written, otherwise only the data block will be written.

Returns:

true if write was successful

template<class E> virtual bool cvr::emlRegistration< E >::parameters::read (ioHandler & *handler*, const bool *complete* = `true`) `[virtual]`

Read the **parameters** from the given **ioHandler**.

Parameters:

handler the **ioHandler** to be used
complete if true (the default) the enclosing begin/end will be also written, otherwise only the data block will be written.

Returns:

true if write was successful

Member Data Documentation

template<class E> E cvr::emlRegistration< E >::parameters::threshold

Error threshold.

If the difference between the last two calculated affine transformation **parameters** vectors is less than the threshold, then the algorithm stops.

Default value: 0.02

template<class E> int cvr::emlRegistration< E >::parameters::maximumTime

Maximum iteration time.

The maximum time that the algorithm will be running in ms. If at the end of an iteration the time is equal or more than the maximum iteration time, then the algorithm stops.

Default value: 600

template<class E> E cvr::emlRegistration< E >::parameters::stepLambda

Increase/decrease step for lambda.

The variation of lambda that is applied in every iteration, depending on the reduction of the threshold. If the threshold is reduced, then lambda is decreased by the step; else, lambda is

increased by the same value.

Default value: 0

template<class E> E cvr::emlRegistration< E >::parameters::stepAlpha

Decrease step for alpha.

The decreasing value applied to alpha after an ammount of iterations specified by the parameter iterationsAlpha.

Default value: 0

template<class E> int cvr::emlRegistration< E >::parameters::iterationsAlpha

Number of iterations before alpha is decreased.

Default value: 2

template<class E> int cvr::emlRegistration< E >::parameters::hx

Weight of the kernel of intensity in the x axis.

Default value: 1

template<class E> int cvr::emlRegistration< E >::parameters::hy

Weight of the kernel of intensity in the y axis.

Default value: 1

template<class E> int cvr::emlRegistration< E >::parameters::windowSizeX

Size of the window of the kernel in the x axis.

Default value: 2

template<class E> int cvr::emlRegistration< E >::parameters::windowSizeY

Size of the window of the kernel in the y axis.

Default value: 2

template<class E> bool cvr::emlRegistration< E >::parameters::onlyCP

Use only Control Points.

When false it uses both Control Points and Intensity.

Default value: false

The documentation for this class was generated from the following file:

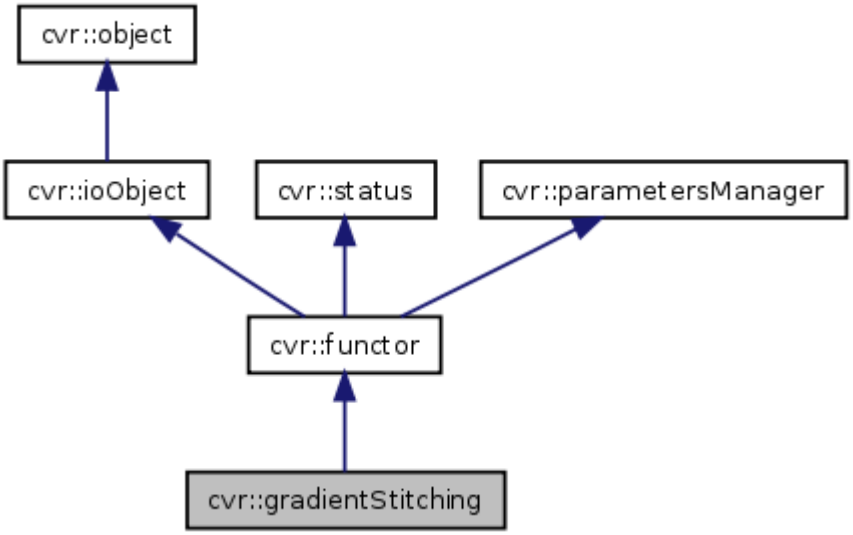
cvrEmlRegistration.h

A.2 Clase implementada para la unión de imágenes en la CVR-Lib

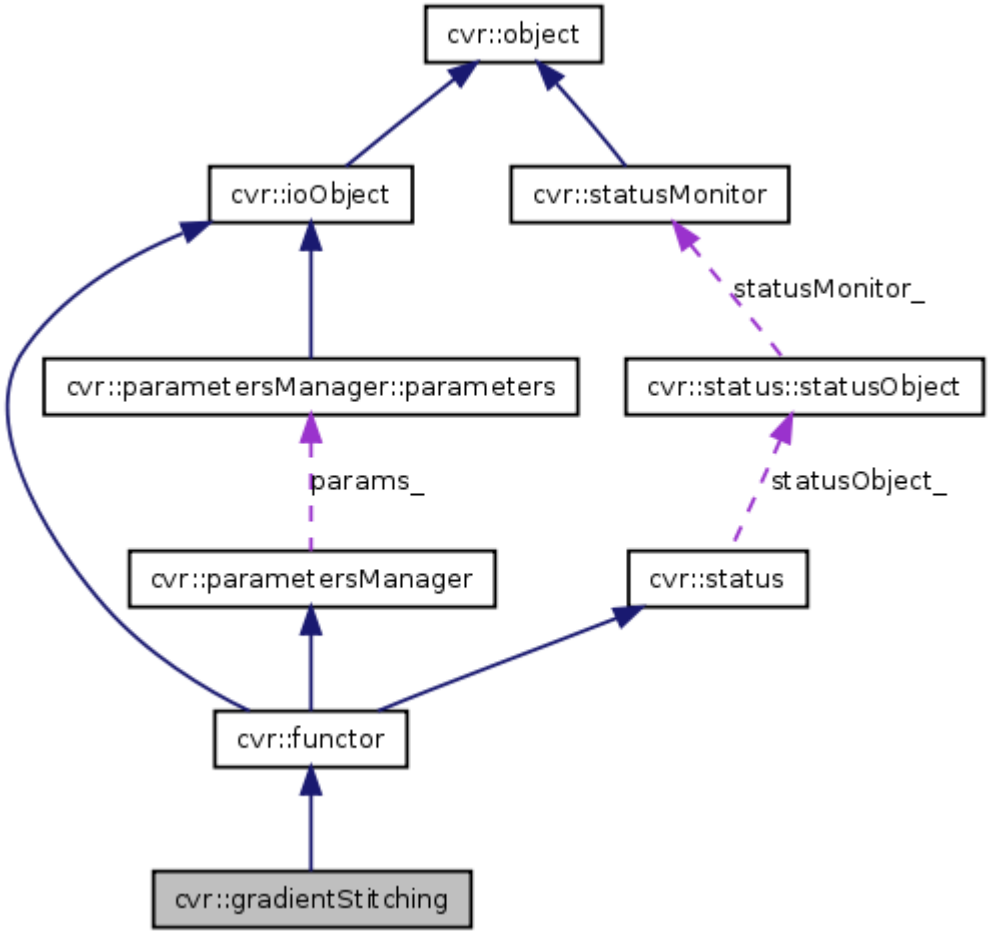
cvr::gradientStitching Class Reference

`#include <cvrGradientStitching.h>`

Inheritance diagram for cvr::gradientStitching:



Collaboration diagram for cvr::gradientStitching:



Detailed Description

Class **gradientStitching**.

This class computes the **image** stitching of two given images based on the Gradient-domain Image Stitching(GIST) algorithm proposed in:

A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless Image Stitching in the Gradient Domain. Proc. European Conf. Computer Vision, May 2004.

See also:

gradientStitching::parameters.

Public Member Functions

gradientStitching ()
gradientStitching (const **parameters** &par)
gradientStitching (const **gradientStitching** &other)
virtual **~gradientStitching** ()

bool **apply** (**channel** &chnl1, **channel** &chnl2, **fmatrix** &mtrans, **channel** &I, **viewer2D** &viewI) const
gradientStitching & **copy** (const **gradientStitching** &other)
gradientStitching & **operator=** (const **gradientStitching** &other)
virtual const std::string & **name** () const
virtual **gradientStitching** * **clone** () const
virtual **gradientStitching** * **newInstance** () const
const **parameters** & **getParameters** () const

Classes

class **parameters**
*The **parameters** for the class **gradientStitching**.*

Constructor & Destructor Documentation

cvr::gradientStitching::gradientStitching ()

Default constructor.

cvr::gradientStitching::gradientStitching (const parameters & *par*)

Construct a **functor** using the given **parameters**.

cvr::gradientStitching::gradientStitching (const gradientStitching & *other*)

Copy constructor.

Parameters:
other the **object** to be copied

virtual cvr::gradientStitching::~gradientStitching () `[virtual]`

Destructor.

Member Function Documentation

bool cvr::gradientStitching::apply (channel & *chnl1*, channel & *chnl2*, fmatrix & *mtrans*, channel & *I*) const

Operates on the given argument.

Parameters:
chnl1 **channel** for the **image** that needs to be tranformed.
chnl2 **channel** for the reference **image**.
mtrans **matrix** that contains the transformation to be applied.
I **channel** with the result of stitching chnl1 and chnl2.

Returns:
true if apply successful or false otherwise.

gradientStitching& cvr::gradientStitching::copy (const gradientStitching & *other*)

Copy data of "other" **functor**.

Parameters:

other the **functor** to be copied

Returns:

a reference to this **functor object**

gradientStitching& cvr::gradientStitching::operator= (const gradientStitching & *other*)

Alias for copy member.

Parameters:

other the **functor** to be copied

Returns:

a reference to this **functor object**

virtual const std::string& cvr::gradientStitching::name () const `[virtual]`

Returns the complete name of the **functor** class.

Implements **cvr::functor**.

virtual gradientStitching* cvr::gradientStitching::clone () const `[virtual]`

Returns a pointer to a clone of this **functor**.

Implements **cvr::functor**.

virtual gradientStitching* cvr::gradientStitching::newInstance () const `[virtual]`

Returns a pointer to a new instance of this **functor**.

Implements **cvr::functor**.

const parameters& cvr::gradientStitching::getParameters () const

Returns used **parameters**.

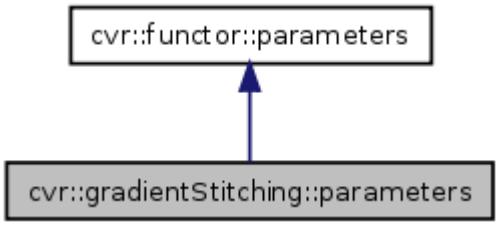
Reimplemented from **cvr::parametersManager**.

The documentation for this class was generated from the following file:
cvrGradientStitching.h

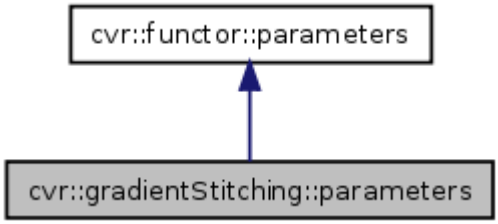
cvr::gradientStitching::parameters Class Reference

`#include <cvrGradientStitching.h>`

Inheritance diagram for cvr::gradientStitching::parameters:



Collaboration diagram for cvr::gradientStitching::parameters:



Detailed Description

The **parameters** for the class **gradientStitching**.

Public Member Functions

parameters ()
parameters (const **parameters** &other)
~parameters ()
parameters & **copy** (const **parameters** &other)
parameters & **operator=** (const **parameters** &other)
virtual const std::string & **name** () const
virtual **parameters** * **clone** () const
virtual **parameters** * **newInstance** () const
virtual bool **write** (**ioHandler** &handler, const bool complete=true) const
virtual bool **read** (**ioHandler** &handler, const bool complete=true)

Public Attributes

float **rate**

Constructor & Destructor Documentation

cvr::gradientStitching::parameters::parameters ()

Default constructor.

Reimplemented from **cvr::functor::parameters**.

cvr::gradientStitching::parameters::parameters (const parameters & *other*)

Copy constructor.

Parameters:

other the **parameters object** to be copied

cvr::gradientStitching::parameters::~parameters () `[virtual]`

Destructor.

Reimplemented from **cvr::functor::parameters**.

Member Function Documentation

parameters& cvr::gradientStitching::parameters::copy (const parameters & *other*)

Copy the contents of a **parameters object**.

Parameters:

other the **parameters object** to be copied

Returns:

a reference to this **parameters object**

parameters& cvr::gradientStitching::parameters::operator= (const parameters & *other*)

Copy the contents of a **parameters object**.

Parameters:

other the **parameters object** to be copied

Returns:

a reference to this **parameters object**

virtual const std::string& cvr::gradientStitching::parameters::name () const [virtual]

Returns the complete name of the **parameters** class.

Implements **cvr::functor::parameters**.

virtual parameters* cvr::gradientStitching::parameters::clone () const [virtual]

Returns a pointer to a clone of the **parameters**.

Implements **cvr::functor::parameters**.

virtual parameters* cvr::gradientStitching::parameters::newInstance () const [virtual]

Returns a pointer to a new instance of the **parameters**.

Implements **cvr::functor::parameters**.

virtual bool cvr::gradientStitching::parameters::write (ioHandler & *handler*, const bool *complete* = `true`) const [virtual]

Write the **parameters** in the given **ioHandler**.

Parameters:

handler the **ioHandler** to be used
complete if true (the default) the enclosing begin/end will be also written, otherwise only the data block will be written.

Returns:

true if write was successful

virtual bool cvr::gradientStitching::parameters::read (ioHandler & *handler*, const bool *complete* = `true`) [virtual]

Read the **parameters** from the given **ioHandler**.

Parameters:

handler the **ioHandler** to be used
complete if true (the default) the enclosing begin/end will be also written, otherwise only the data block will be written.

Returns:

true if write was successful

Member Data Documentation

float cvr::gradientStitching::parameters::rate

Learning rate.

The algorithm used to obtain the mosaic is gradient descent. The gradient is calculated for each iteration and the **image** updated subtracting the product of the rate and the gradient at each pixel;

The documentation for this class was generated from the following file:

cvrGradientStitching.h