UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

MÉTODOS DE ANÁLISIS DE IMÁGENES APLICADOS AL PROCESAMIENTO DE DATOS DE UNA CÁMARA SO_2

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

FÍSICO

P R E S E N T A :

SALVADOR PEDRAZA ESPITIA

TUTOR

DR. ROBIN CAMPION

2017

CIUDAD UNIVERSITARIA, CDMX

1. Datos del alumno
Pedraza
Espitia
Salvador
5566999716
Universidad Nacional Autónoma de México
Facultad de Ciencias
Física
411041939

2. Datos del tutor
Dr.
Robin Andre Lucien Perceval Tristan Didie
Campion

3. Datos del sinodal 1
Dr.
Hugo
Delgado
Granados

4. Datos del sinodal 2
Dr.
Michel Alexandre
Grutter
de la Mora

5. Datos del sinodal 3
Dr.
Mathieu Christian Anne
Hautefeuille

6. Datos del sinodal 4
Dra.
Claudia Inés
Rivera
Cárdenas

7. Datos de la tesis
Métodos de análisis de imágenes aplicados al procesamiento
de datos de una cámara SO2
148 p
2017

A mi mamá y papá.

Agradecimientos

A los apoyos recibidos durante la licenciatura y en la tesis

A la Universidad Nacional Autónoma de México
A la Facultad de Ciencias por su espacio y sus profesores.
Al instituto de Geofísica.
A la beca PUMC, y a todos los que conocí a través de este programa.
A Hugo Delgado por su apoyo y consejos.
Al apoyo recibido de CONACYT por ser ayudante de investigador.
A mi director de tesis Robin Campion por su amistad, por compartir sus conocimientos y experiencia. Por su ayuda en la elaboración de este trabajo. Y por hacer mi estancia agradable en geofísica.

A mis sinodales

Hugo Delgado Granados
Michel Grutter de la Mora
Robin Campion
Mathieu Hautefeuille
Claudia Inés Rivera Cárdenas

A todos los que han contribuido a mi formación, que compartieron sus perspectivas, a los profesores que hicieron sus clases amenas, a los que eran exigentes y a los que no, de todos aprendí algo. A mis amigos por sus conversaciones, sus chistes y apoyo. Y a mi familia: mis padres Bernardo y María que me han educado, aconsejado y amado siempre. A mi hermana Hilda cuya alegría y risas siempre han sido contagiables. :)

A todos ellos, infinitas gracias.

Resumen

Los gases volcánicos son considerados como el motor de las erupciones porque controlan la intensidad de las erupciones volcánicas, esto hace que sea importante medirlos y monitorearlos. La información que se obtiene se utiliza para estudiar el comportamiento del volcán y el análisis de los datos puede incluso dar información relevante de comportamientos futuros.

En esta tesis se trabaja con los datos e imágenes obtenidos con una cámara SO_2, diseñada para medir distribuciones bidimensionales de irradiancia en el rango ultravioleta. Por medio de un filtro, la cámara selecciona un rango de longitudes de onda donde la absorción por moléculas de bióxido de azufre es prominente.

Desde distancias del orden de kilómetros, la cámara SO_2 mide intensidades de la radiación difusa del cielo y de la cantidad de radiación difusa que atraviesa la pluma volcánica.

El procesamiento de imágenes es necesario para determinar la distribución de concentraciones de SO_2. La cámara se vuelve un excelente instrumento de monitoreo porque tiene una resolución temporal de hasta 1 Hz y una resolución espacial del orden de metros.

Dentro de la pluma y en la atmósfera ocurren procesos de absorción y esparcimiento de la radiación por componentes incluyendo al bióxido de azufre. Disminuir el efecto de esparcimiento en la medición implica hacer aproximaciones a los modelos que describen estos fenómenos. También deben hacerse correcciones a efectos que se originan por los propios componentes de la cámara: filtro pasabandas, lente y CCD. Se utilizan dos filtros centrados en 310 y 330 nm, el primero detecta la absorción por moléculas de SO_2 al igual que efectos de esparcimiento por aerosoles y el segundo es sensible idealmente solo al esparcimiento por aerosoles, con este método puede aproximarse en primer orden la contribución debida a la absorción del SO_2. El estudio de la transferencia radiativa en la atmósfera y los principios físicos que implica

la medición con la cámara son importantes porque permiten saber las condiciones ideales en las mediciones, configuración del hardware de la cámara y correcciones que se aplicarán a las imágenes, con esto se procura obtener información confiable de las distribuciones de bióxido de azufre.

En este trabajo se escribió un programa en lenguaje Python, con el propósito de procesar las imágenes obtenidas por la cámara SO_2 y obtener distribuciones de bióxido de azufre, sus campos de velocidades asociados y su tasa de emisión. El programa contiene las consideraciones derivadas tanto del análisis al principio físico de medición como las correcciones implicadas en la transferencia de radiación y el hardware de la cámara. Considerando que los parámetros de corrección pueden cambiar en el tiempo, el programa se ha dotado con la capacidad de detectar las distintas regiones en una imagen: pluma, cielo y edificio volcánico, y de esta manera se recuperan parámetros actualizados continuamente.

El programa, de forma automatizada hace correcciones en función de la información contenida en cada imagen. El reconocimiento de la región del cielo se hizo para obtener una imagen extrapolada de la absorción en el cielo, y con ello aproximar la absorción generada por el cielo detrás de la pluma volcánica. Para lograrlo, se analizaron histogramas de imágenes creadas a partir de absorbancias, con las que se resaltan las regiones de mayor movimiento y absorbancia.

Las distribuciones de las densidades de columna se obtuvieron aplicando la ley de Beer Lambert Bouguer a las intensidades de los pixeles. La cámara puede obtener mediciones cada segundo. Esta característica combinada con el análisis a secuencias de imágenes, permitió el cálculo de campos de velocidades bidimensionales. Los campos de velocidades se obtienen con un algoritmo de flujo óptico que usa el método de Farneback y con una matriz de las distancias a los pixeles en la imagen.

Finalmente, para hacer el cálculo del flujo de masa de SO_2 se utilizó un algoritmo basado en el teorema de la divergencia, tomando como base el campo de velocidades obtenido con un par de imágenes consecutivas y la distribución correspondiente de densidades de columna.

El programa es puesto a prueba con imágenes de plumas de diferentes volcanes en el mundo: Popocatépetl, Ubinas, Masaya, Pacaya y Stromboli. Con los parámetros adecuados y la optimización del código se puede pensar en adaptar la cámara SO_2 y el software para hacer monitoreo desde una estación fija, con vista a un volcán y estudiar la dinámica de la pluma.

Abstract

Volcanic gases are considered the engine of the eruptions because they drive the eruption intensity, so it is important to measure and keep permanent observation to them. The measurements are needed to study volcanoes behavior and plume dynamics, data analysis also can give relevant information for future behaviors.

In this work data and images are provided by an SO_2 camera, designed to measure radiation and save it as 2D arrays of intensities. The scattered solar radiation in the atmosphere passes through the volcanic plume containing sulfur dioxide. Using two cameras and filters at 310 and 330 nm, the instrument selects a range of UV wavelengths where the absorption by sulfur dioxide molecules is strong and the second filter selects wavelengths where the absorption by SO_2 is negligible but aerosol scattering still remain. This instrumental set up provides a first order correction of the radiative extinction by the aerosols present in the plume. Camera features allow to contuct remote measurements to volcanic plumes up to ~10 kilometers and the ability to obtain up to one image per second.

In this thesis, a program was written in Python language for the purpose of processing images obtained from de SO_2 camera. The results are images showing the distribution of SO_2 columns amount, the field of velocities and rate of SO_2 emission.

The study of the radiative transference in the atmosphere is necessary to understand the cameras measurements, also allows knowing the best conditions for the measurements and configurations in the hardware. The program contains considerations derived from the analysis of physical principles measurement, the corrections in the transference of the radiation and effects originating from the camera components itself – bandpass filters, lens and CCD –. Considering that the parameters of correction can change within time, the program has been given the capacity to detect different regions

at each image: plume, sky and volcanic edifice. In this way, updated parameters are obtained. The purpose is to obtain trustworthy information from the distribution of SO_2

The result is an automatized program that makes corrections based on the information provided by each image. Sky recognition was made to extrapolate and get the background behind the plume. To achieve it, it was necessary to analyze histograms of images created using absorbances, in order to highlight signals of motion and absorption zones.

Images corrected were used to compute column density amounts, applying the Beer Lambert Bouguer law. The camera is able to obtain measurements each second, this feature combined with the analysis image sequences allows to compute field velocities. They are obtained using the Farneback optical flow algorithm and an array of distances between the camera and the plume.

Finally, the rate of SO_2 emission was calculated with an algorithm based on the divergence theorem, and take advantage of the velocities field obtained from a pair of consecutive images and their corresponding SO_2 distribution.

The program is tested with several images of different volcanos around the world: Popocatepetl, Ubinas, Masaya, Pacaya and Stromboli.

Optimizations on the code give the possibility to set up the camera and software to keep track of volcanic gases from a fixed viewing point.

Contenido

Prólogo

Aprender a extender nuestros sentidos con instrumentos aseguró la supervivencia de nuestra especie. Nuestros sentidos han sido la base para explorar, entender y transformar nuestro entorno. Han evolucionado y siguen haciéndolo pero el ritmo en que lo hacen ha sido superado por nuestra capacidad creativa y colectiva de acumular y ordenar conocimientos, eso nos ha permitido diseñar instrumentos de medición los cuales se han convertido en una extensión artificial de nuestros sentidos.

El desarrollo de un instrumento de medición es un ciclo en el cual se van mejorando sus características. El conocimiento al principio empírico fue usado para construir herramientas de medición, que ahora se han vuelto más sofisticados con el desarrollo de la electrónica. Los instrumentos actuales utilizan sensores electrónicos de todo tipo, haciendo mediciones con ellos se obtiene información que es analizada, y después da la posibilidad de pensar como aprovechar el nuevo conocimiento para buscar la eficiencia en nuevos instrumentos de medición con sensores de mayor sensibilidad, resolución, precisión y exactitud.

Los instrumentos de medición se diseñan para arrojar datos, y estos deben ser procesados. Cuando la cantidad de datos es grande se vuelve necesario extender nuestra capacidad de procesamiento con otro instrumento diseñado para este objetivo: una computadora, la cual recibe instrucciones de las operaciones que debe realizar por medio de un lenguaje de programación. El punto de procesar los datos de las mediciones es entender y obtener conclusiones del fenómeno observado.

Para sintetizar el conocimiento adquirido se cuenta con otro tipo de instrumentos, que resultan de la abstracción de las observaciones hechas al fenómeno medido, las construcciones matemáticas. Estas describen fenómenos físicos y se usan para generalizar y extender el entendimiento de nuestro entorno. Los modelos construidos son en ocasiones muy generales. Los encon-

tramos descritos de forma limpia, y hasta con rasgos de simetría y armonía. Lamentablemente la aplicación de estos modelos no es siempre directa, es importante reconocer que pueden haber términos difíciles de estimar, pues el mundo real es complejo, está compuesto de una inmensidad de variables, no tan fáciles de manejar porque unas dependen de otras y se complica definir hasta que punto se correlacionan. Para superar este problema se hacen aproximaciones, justificando las hipótesis, se diseña un instrumento para medir la parte que falta o pueden hacerse simulaciones computacionales del sistema. De esto se trata el trabajo científico. La realidad es compleja y si el modelo sigue dando la impresión de que no funciona, o que esta restringido a un dominio particular significa que el modelo no está completo. Esto es lo que da vida a la ciencia, son muchísimas cosas por hacer.

El trabajo científico hace que la ciencia se encuentre en continuo desarrollo. Los conocimientos que servirán para mejorar la comprensión de nuestro entorno se van construyendo y ordenando. La instrumentación en las mediciones, y el procesamiento de los datos es esencial para adquirir estos conocimientos. Justamente este trabajo muestra el procesamiento de datos de un instrumento que hace mediciones remotas, la camara SO_2. La información contenida en las imágenes de la cámara permite medir las distribuciones de SO_2 en gases volcánicos, esto sirve para rastrear el comportamiento de algunos volcanes, probablemente los datos puedan servir para desarrollar modelos y predecir eventos catastróficos.

La redacción de mi tesis, al igual que la licenciatura, me dio un panorama de la actividad científica y de la importancia de varios tipos de instrumentos en la recuperación de datos. Se hace evidente la importancia de entender su funcionamiento para obtener datos e información confiable que pueda corroborar o mejorar los modelos existentes y que la aplicación de los modelos y el procesamiento de datos requiere de los conocimientos previos que se han acumulado y sistematizado.

La instrumentación es un claro ejemplo de la aplicación directa de la ciencia porque ésta nos permite diseñar y mejorar los instrumentos. A su vez los instrumentos generan datos que complementan y ayudan a construir modelos matemáticos, que son la base de las teorías. Por tanto, la teoría y la práctica no dejarán de funcionar en conjunto.

En la redacción de esta tesis, mientras investigaba e integraba el marco teórico, haciendo pruebas modificando el código, haciendo robusta la programación, diseñando y optimizando los diagramas y gráficas que complementaron el texto, utilicé varias herramientas, las más relevantes fueron:

PYTHON, Inkscape, Sublime Text y LaTeX.

Internet me resultó de gran ayuda. Me permitió encontrar artículos, referencias, datos, diagramas para guiarme, fue una forma rápida de encontrar documentación en la programación, sugerencias en foros y otras comunidades. Hay que reconocer que hay mucha información que se encuentra enmarañada con información redundante y eso hace difícil recolectar eficientemente contenido de calidad. Me ocurrió que al encontrar una nueva referencia, la estructura y la forma en que se presentaba la información parecía mas entendible o que las convenciones usadas eran diferentes; querer adaptar mi redacción a estas nuevas fuentes me hizo borrar, modificar y replantear varias veces lo que escribía. Pienso que internet es una gran herramienta de investigación, pero es fácil empezar a divagar en otros temas afines al tema que se busca y leer información redundante o poco relevante para la investigación.

Este trabajo[1] es un reflejo de la experiencia y conocimientos adquiridos en la licenciatura. Es parte de mi interés por la programación, porque me parece que es una de las formas mas convenientes de visualizar y aplicar la teoría, para hacer cálculos y tareas específicas con eficiencia.

[1]Para una lectura mas cómoda de esta tesis hay dos versiones en formato PDF, una en tamaño original y otra adaptada al tamaño de un lector electrónico promedio, puedes solicitarlos vía correo electrónico: salva@ciencias.unam.mx. Las referencias a los capítulos, secciones, figuras y ecuaciones dentro del texto son 'hyperlinks', esto facilita su lectura.

Capítulo 1

Introducción

Los volcanes son un fenómeno intrigante y fascinante, son una manifestación de la energía almacenada bajo la superficie de la Tierra. Han tenido un gran impacto en la evolución del planeta y lo seguirán haciendo, cambiando directamente nuestro entorno, en algunos casos de forma drástica como ha sucedido en la historia de la Tierra. La motivación científica para estudiarlos, junto a la satisfacción de comprender acerca de la historia y procesos en nuestro planeta, es desarrollar modelos con el potencial de aproximar comportamientos futuros, dándonos la oportunidad de prevenir y disminuir el riesgo de desastres naturales.

Los modelos requieren datos y observaciones, los cuales son obtenidos por expertos en distintas áreas. Vulcanólogos, geólogos, químicos, sismólogos y mas científicos, colaboran para diseñar y utilizar instrumentos que permiten *medir* y *analizar* los datos, los cuales cuentan cada vez con mayor precisión y exactitud.

Actualmente los instrumentos más difundidos y viables para el estudio de volcanes son los que hacen uso de sensores remotos. Este trabajo presenta un instrumento que ha sido diseñado para hacer mediciones remotas de dióxido de azufre mezclado entre los gases emitidos por volcanes activos. Se trata de la cámara SO_2, cuyos pioneros Mori y Burton [1] mostraron que era una forma más barata y rápida para medir SO_2, comparada con tecnologías ampliamente usadas como DOAS, COSPEC y FTIR tanto desde la superficie como en plataformas satelitales. La cámara utiliza un filtro para seleccionar un rango de longitudes de onda, en la región ultravioleta, en el que la absorción por moléculas de dióxido de azufre es prominente.

La cámara obtiene mediciones de intensidad de radiación en su campo

de visión, esto permite obtener imágenes bidimensionales de la distribución de SO_2 en una pluma volcánica. Aunque para lograr esto de forma satisfactoria se requiere un procesamiento a las imágenes crudas obtenidas por la cámara. A su vez el procesamiento de las imágenes requiere de una revisión a los principios físicos implicados en su funcionamiento, asi como entender las fuentes de error y los cálculos necesarios para aproximar el contenido de SO_2.

La realización del presente trabajo es motivada por el interés de obtener mediciones mas eficientes al dióxido de azufre contenido en una pluma volcánica. Si se decide tomar un par de imágenes cada 10 segundos, la cantidad de pares de imágenes obtenidas en una hora es de 360. El procesamiento de cada par puede tomar de 1 a 3 minutos utilizando un procesamiento semiautomatizado que consista en seleccionar manualmente coordenadas de regiones en las imágenes. Automatizar la selección de estas regiones incrementaría la velocidad de cálculo, incluso podría adaptarse un procesador potente capaz de hacer los cálculos en tiempo real. Las regiones seleccionadas se usan como entrada de las funciones que hacen los cálculos.

La extracción de los datos como el flujo de SO_2 de las imágenes y su distribución espacial y temporal son útiles para estudiar la dinámica de la pluma y su relación con el comportamiento del flujo de magma u otros procesos en el interior del volcán.

La exsolución de gases en forma de burbujas, inicialmente disueltos en magma, almacenado en la profundidad del volcán, dan al magma exceso de presurización y flotabilidad que permite vencer la presión litostática y ascender a la superficie. Luego, durante la erupción, la expansión de los gases en las burbujas puede fragmentar el magma y arrojarlo a la atmósfera. El bióxido de azufre es uno de los componentes principales en los gases volcánicos, medir su tasa de emisión da excelente información de la desgasificación magmática a altas temperaturas, la tasa de alimentación en el magma y los procesos que afectan la transferencia de los gases desde la cámara magmática hasta la superficie. Además, de forma indirecta se pueden hacer inferencias sobre la profundidad a la cual los gases se separan del magma y sus posibles interacciones con el sistema hidrotermal del volcán.

La hipótesis es que usando las imágenes adquiridas por la cámara SO_2, las cuales contienen información de la radiación difusa en la atmósfera y la cantidad de radiación absorbida por moléculas de SO_2, se pueden diseñar algoritmos que calculen umbrales necesarios para rastrear las distintas regiones de la imagen: cielo, volcán y pluma volcánica, y con ello hacer más

eficiente la obtención de las distribuciones de SO_2.

Automatizar la detección de las regiones en la imagen aumenta la eficiencia en el procesamiento de imágenes de la cámara, las distribuciones de SO_2 en consecuencia adquieren mayor precisión. La variación de las condiciones de medición como el cambio de radiación difusa con el ángulo solar cenital o el contenido de aerosoles en el tiempo, hacen que sea necesario calcular nuevos factores de corrección, y por lo tanto es mejor que los cálculos sean realizados obteniendo parámetros actualizados.

El objetivo es escribir un programa en lenguaje Python que pueda identificar la pluma volcánica y las coordenadas de regiones necesarias para extraer los parámetros de entrada en las funciones que hacen los cálculos de la distribución de densidad de columna de SO_2.

Para dar contexto a este trabajo en la Sección 1.1 se estudia y deriva la ecuación de la transferencia radiativa, esto es importante porque una solución particular es la ley de Beer–Lambert–Bouguer, la cual es esencial para entender el principio físico de medición en la cámara. En la Sección 1.2 se aborda la química atmósferica, la contribución de los gases volcánicos a la atmósfera, su relevancia y otras técnicas usadas para monitorearlos además de la cámara SO_2.

La cámara SO_2 ha evolucionado en la última década, con cada aportación se mejoran las debilidades de las versiones anteriores. Tanto el diseño como el desarrollo teórico son importantes para obtener un instrumento confiable. Entre los trabajos teóricos está el de Kern et al. [2], en el que discute los principios físicos de medición. Por otra parte Lübcke et al. hace hincapié en los métodos viables de calibración, así como un tratamiento para aproximar el esparcimiento por aerosoles que afecta las mediciones [3]. Métodos para hacer correcciones a las imágenes y acotar errores debidos al esparcimiento de la luz entre la pluma volcánica y la cámara son descritos por Campion et al. [4]. La precisión de la cámara depende del éxito en la corrección de varios efectos y errores sistemáticos en las mediciones, que se analizan en estos trabajos. Basada en estas aportaciones, una revisión exhaustiva a los fundamentos físicos en las mediciones, descripción de las características y diseño de la cámara se encuentran en el Capítulo 2.

Las imágenes adquiridas por una cámara SO_2 contienen información sobre la distribución de los gases. Las imágenes a procesar son arreglos de números de 512×512 con valores enteros de 0 a $2^{16} - 1$, de esta manera se muestra evidente la necesidad de hacer los cálculos y análisis con una computadora. Programar los algoritmos para hacer los cálculos requiere co-

nocimientos de un lenguaje de programación. Para el desarrollo de este trabajo se utilizó PYTHON, un lenguaje de programación con amplia difusión y de código abierto que cuenta con excelentes bibliotecas para cálculo científico y manipulación de imágenes. Una breve descripción del lenguaje PYTHON así como los módulos y funciones mas relevantes para el desarrollo de este trabajo aparece en el Apéndice A.

Las imágenes tienen que llevar un preprocesamiento que incluye rotaciones, eliminación de pixeles sin información y en general correcciones fijas determinadas por la configuración de la cámara al adquirir los datos. Los cálculos de las correcciones se traducen a algoritmos en el lenguaje de programación PYTHON. En el Capítulo 3 se describen los algoritmos programados para procesar las imágenes. También se incluyen las ideas utilizadas para automatizar la detección de umbrales que dan la capacidad al programa de decidir qué datos pueden ser usados como entrada a los algoritmos, en función de la información contenida implícitamente en cada imagen. Esta característica permite que el software y la cámara se puedan adaptar para instalarse en estaciones fijas de monitoreo, aumentando la rapidez para cuantificar el dióxido de azufre sin intervención humana.

Se procesaron datos de los volcanes Popocatépetl, Stromboli, Pacaya, Ubinas y Masaya. Los resultados, y salidas del programa pueden revisarse en el Capítulo 4.

Finalmente la conclusión de esta tesis se presenta en el Capítulo 5, así como algunas recomendaciones de los problemas que se deben superar a futuro.

Se ha incluido en el Apéndice B, el código para el programa principal descrito en el desarrollo de la presente tesis, así como el código utilizado para generar imágenes de calibración usadas por el programa principal.

1.1. Transferencia Radiativa en la Atmósfera

La atmósfera terrestre es una capa gaseosa que funciona como mediador de la transferencia de energía del Sol a la superficie terrestre, también transporta energía entre distintas regiones del planeta manteniendo el equilibrio termodinámico, tales transferencias son un factor determinante para el clima.

El flujo radiativo solar que llega a la Tierra es en promedio 341 W m^{-2}. Más del 80% de la energía que absorbe la atmósfera llega a través de la

transferencia radiativa. La transferencia de energía en la atmósfera es gobernada por las longitudes de onda corta emitidas por el Sol y longitudes de onda larga emitidas por la superficie de la Tierra. La atmósfera pierde su energía con la emisión de radiación, en el rango de longitudes de onda larga, a la superficie terrestre y al espacio [5].

Los procesos que ocurren en la interacción de la radiación con la materia, que son la absorción, emisión y esparcimiento se discutirán en este capítulo. Se deducirá una expresión general para la transferencia de radiación en la atmósfera. Aunque veremos que la emisión es despreciable en el rango de longitudes de onda corta.

1.1.1. Propagación de la radiación electromagnética

Por su naturaleza, la descripción matemática de la radiación en tres dimensiones se debe hacer considerando su dependencia en la longitud de onda y en la dirección. La energía radiante atravesando una superficie con normal unitaria en la dirección n tiene contribuciones de todas las direcciones, y es caracterizado por un vector unitario $\hat{\Omega}$, ver Figura 1.1. Si la radiación llega a lo largo de una sola dirección, como lo hace la radiación solar, es llamada radiación de haces paralelos.

La intensidad monocromática o radiancia está dada por $\vec{I}_\lambda = I_\lambda \hat{\Omega}$. Esto representa la razón de energía de longitud de onda λ a $\lambda + \mathrm{d}\lambda$ que fluye por unidad de área a través de un ángulo sólido $\mathrm{d}\Omega$ en la dirección $\vec{\Omega}$.

$\vec{I}_\lambda(\vec{x}, \hat{\Omega})$ es una función de la posición (x_1, x_2, x_3) y de la dirección, descrita por los cosenos directores $(\cos\Omega_1, \cos\Omega_2, \cos\Omega_3)$ o en coordenadas esféricas por el ángulo cenital θ y azimutal ϕ. Las dimensiones de $\vec{I}_\lambda$ son potencia/(área $\cdot$ longitud de onda $\cdot$ estereorradián).

La radiación electromagnética interactúa con la materia fundamentalmente de tres formas: procesos de absorción, esparcimiento y emisión. Un lápiz[1] de un haz de radiación ocupando el ángulo sólido $\mathrm{d}\Omega$ es atenuado por la proporción en la absorción y densidad características del medio por el cual pasa. La energía también puede ser esparcida fuera del ángulo $\mathrm{d}\Omega$ y de igual manera atenuar el flujo de energía en ese ángulo. También puede existir energía emitida dentro del ángulo sólido o energía esparcida hacia

[1] En óptica se utiliza el término 'lápiz' para referirse a la construcción geométrica usada para describir una porción de haz de radiación electromagnética, con la forma de un cono o cilindro delgado.

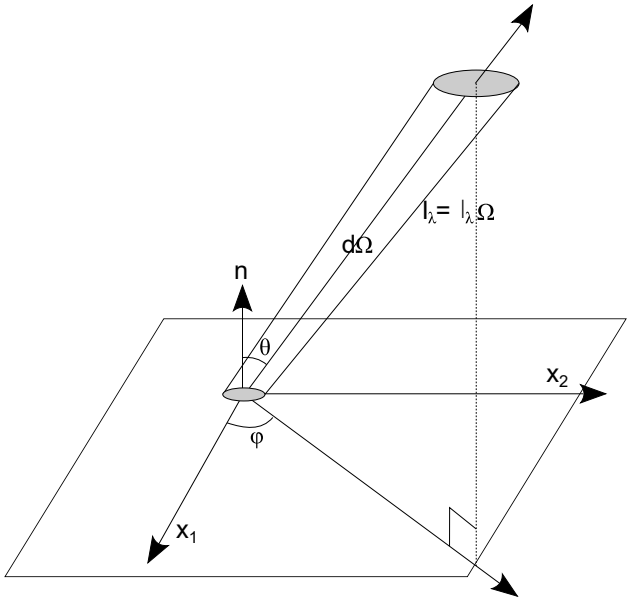


Figura 1.1: Haz de radiación en la dirección $\hat{\Omega}$ representado por el ángulo sólido $\mathrm{d}\Omega$ atravesando una superficie con normal unitaria $\hat{n}$.

adentro del ángulo sólido intensificando el flujo de energía a través del haz de radiación. Estas interacciones son modeladas por leyes que describen la transferencia de radiación a través de la materia. La suma de absorción + esparcimiento contribuyen a la extinción de la energía.

1.1.2. Extinción

Suponiendo un volumen pequeño $\mathrm{d}V$ con una concentración de partículas definida como la razón entre el número de partículas $\mathrm{d}N$ y el volumen $\mathrm{d}V$ y por conveniencia sea el volumen $\mathrm{d}V$ un bloque de grosor $\mathrm{d}s$ y área $\mathrm{d}A$, si se hace pasar sobre él un haz con energía $I_\lambda \,\mathrm{d}A\,\mathrm{d}t\,\mathrm{d}\lambda\,\mathrm{d}\Omega$, el haz interactúa con las partículas ópticamente activas y ocurren procesos de absorción y/o esparcimiento. La energía que emerge del otro lado se reduce a $(I_\lambda - \mathrm{d}I_\lambda)\,\mathrm{d}A\,\mathrm{d}t\,\mathrm{d}\lambda\,\mathrm{d}\Omega$, entonces se dice que el haz ha experimentado extinción [6].

Experimentalmente se encuentra que el grado de atenuación depende linealmente tanto de la intensidad incidente como de la cantidad de materia ópticamente activa, a lo largo de la dirección del haz. La cantidad de materia

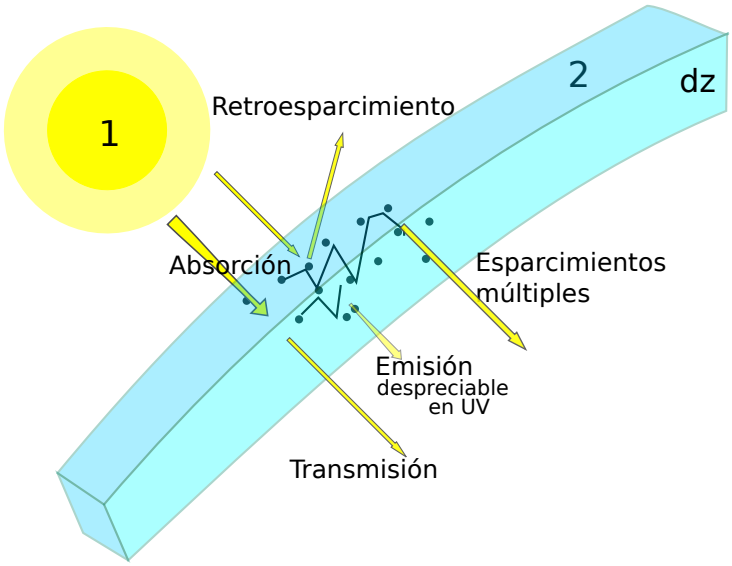


Figura 1.2: Fenómenos que ocurren en la transferencia radiativa. Los procesos fundamentales en la propagación de la radiación solar (1) en la atmósfera (2) son la absorción, emisión y esparcimiento.

a su vez depende del grosor del bloque ds

$$\mathrm{d}I_\lambda \propto -I_\lambda \mathrm{d}s \tag{1.1}$$

La Ecuación (1.1) indica que existe una constante de proporcionalidad, la cual es conocida como coeficiente de extinción o atenuación. Las tres formas que se encuentran en la literatura son: coeficiente de extinción [m^{-1}], coeficiente de extinción de masa [$m^2 \cdot \mathrm{kg}^{-1}$] y sección transversal de extinción [m^2], respectivamente dados por

$$\beta(\lambda) \equiv -\frac{\mathrm{d}I_\lambda}{I_\lambda \,\mathrm{d}s} = \rho k^m(\lambda) = N k(\lambda) \tag{1.2}$$

$$k^m(\lambda) \equiv -\frac{\mathrm{d}I_\lambda}{I_\lambda \rho \,\mathrm{d}s} = -\frac{\mathrm{d}I_\lambda}{I_\lambda \,\mathrm{d}\mathscr{M}} \tag{1.3}$$

$$k(\lambda) \equiv -\frac{\mathrm{d}I_\lambda}{I_\lambda N \,\mathrm{d}s} = -\frac{\mathrm{d}I_\lambda}{I_\lambda \,\mathrm{d}\mathscr{N}} \tag{1.4}$$

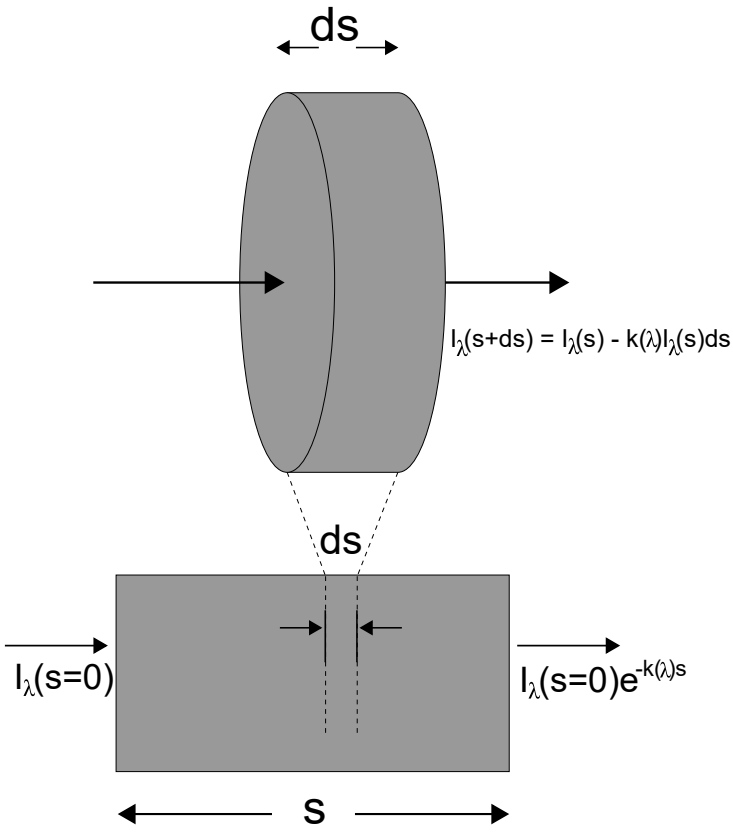


Figura 1.3: Intensidad pasando por un bloque delgado de grosor ds experimenta extinción proporcional a la longitud de camino ds (arriba). Intensidad pasa por un bloque de longitud finita s y experimenta extinción exponencial (abajo).

La Ecuación (1.4) es interpretada como el área efectiva de la partícula que se presenta al haz incidente. El resultado es que una fracción es absorbida y otra desviada en otras direcciones. ρ es la densidad de masa y N la concentración (número de partículas o moléculas por volumen).

La ley de extinción también es llamada ley de Beer Lambert Bouguer, aunque para llegar a su forma más conocida, es necesario definir la profundidad óptica[2]. Esto se hace obteniendo la integración sobre una distancia finita a lo largo del haz. Sea s la longitud total de camino óptico recto a

[2]En la literatura suele nombrarse también como opacidad a lo largo del camino óptico, extinción de camino óptico o densidad óptica.

través de un medio, y sea $s' \leq s$ un camino intermedio. Denotando la intensidad entrando al medio en $s' = 0$ como $I_\lambda(s' = 0, \hat{\Omega})$, se busca la intensidad $I_\lambda(s' = s, \hat{\Omega})$ donde $\hat{\Omega}$ es la dirección de propagación del haz.

Integrando de $s = 0$ a $s' = s$ en las Ecuaciones (1.2) – (1.4) se obtiene

$$\tau(\lambda) \equiv -\ln\left[\frac{I_\lambda(s' = s, \hat{\Omega})}{I_\lambda(s' = 0, \hat{\Omega})}\right] \tag{1.5}$$

donde

$$\tau(\lambda) \equiv \int_0^s \mathrm{d}s' k(\lambda) \equiv \int_0^s \mathrm{d}s' k^m(\lambda)\rho \equiv \int_0^s \mathrm{d}s' N k(\lambda) \tag{1.6}$$

τ es la profundidad óptica, es adimensional y es una medida del número de partículas ópticamente activas a lo largo del haz. Partiendo de la Ecuación (1.5) para resolver $I_\lambda(s' = s, \hat{\Omega})$ se obtiene

$$I_\lambda(s, \hat{\Omega}) = I_\lambda(0, \hat{\Omega}) e^{-\tau(\lambda)} = I_\lambda(0, \hat{\Omega}) e^{-\int_0^s \mathrm{d}s' N k(\lambda)} \tag{1.7}$$

Significa que la intensidad decae exponencialmente (ver Fig. 1.3) con el camino óptico a lo largo del haz. Si no hay extinción la exponencial es 1 y entonces la intensidad se mantiene constante.

La extinción es una suma de dos procesos: absorción y esparcimiento [7], entonces en el caso de la sección transversal de extinción k, puede ser vista como

$$k = k_\alpha + k_\sigma \tag{1.8}$$

donde los sumandos son la sección eficaz de absorción y de esparcimiento respectivamente.

Se definen el coeficiente de extinción β, el coeficiente de absorción α y el coeficiente de esparcimiento σ como

$$\beta = Nk = \rho k^m \tag{1.9}$$

$$\alpha = N k_\alpha = \rho k_\alpha^m \tag{1.10}$$

$$\sigma = N k_\sigma = \rho k_\sigma^m \tag{1.11}$$

La extinción de camino óptico, también llamada profundidad óptica, es una mezcla de efectos de absorción y esparcimiento $\tau = \tau_\alpha + \tau_\sigma$.

1.1.3. Absorción

Anteriormente se manejó la extinción como la suma de absorción y esparcimiento, aún queda por explicar los procesos físicos que estos conllevan. La absorción ocurre cuando la energía de la radiación es usada por una molécula para experimentar una transición cuántica a un nivel de energía mas alto. La energía transferida en una transición está cuantizada en múltiplos de $h\nu = hc/\lambda$, así el cambio en la energía en una transición del estado E_n al estado E_{n+1} está dado por

$$E_{n+1} - E_n = \frac{hc}{\lambda} \tag{1.12}$$

Las energías típicas de transición [8] son descritas a continuación

- Las transiciones electrónicas del orden de 1 eV, entre un estado exitado electrónico y el estado base corresponden al visible o cerca de la región UV.
- Transiciones vibracionales del orden de 0.1 eV para transiciones entre el estado excitado vibracional y el estado base se encuentran el rango infrarojo IR.
- Transiciones rotacionales del orden de $10^{-3} - 10^{-2}$ eV corresponden a transiciones entre el estado rotacional excitado y el estado base, están en el rango de las microondas.

En un espectro de absorción las líneas que aparecen en la región UV corresponden a transiciones de los tres tipos. Las transiciones cuánticas son discretas y dependen de la estructura de la molécula, esto hace que el espectro de absorción tenga un patrón único dependiendo de las moléculas analizadas.

1.1.4. Emisión

Para mantener el equilibrio térmico una sustancia que absorbe energía radiante también debe emitirla. Tal como la absorción, la emisión de energía en un haz de radiación es proporcional a la cantidad de partículas. La base para describir la emisión térmica es la teoría de la radiación de cuerpo negro.

Un cuerpo negro es una idealización que corresponde a la energía emitida por una cavidad aislada en equilibrio térmico, tal radiación es caracterizada por

- La radiación es determinada únicamente por la temperatura del emisor.
- Para una temperatura dada, la energía radiante emitida es la máxima posible en todas las longitudes de onda.
- La radiación es isotrópica.

En un cuerpo negro la radiación en todas las longitudes de onda es completamente absorbida, y también es una fuente de emisión perfecta.

Ley de Planck

Planck postuló que la energía E es cuantizada y puede experimentar únicamente transiciones discretas que satisfacen

$$\Delta E = \Delta n \cdot h\nu \tag{1.13}$$

donde n es un número entero, h es la constante de Planck y $\nu = c/\lambda$ es la frecuencia de radiación electromagnética emitida o absorbida para lograr la energía de transición ΔE. Por tanto, la radiación emitida o absorbida por moléculas individuales es cuantizada en fotones que llevan energía en múltiplos enteros de $h\nu$. Planck obtuvo con este principio, la relación entre el espectro de intensidad B_λ emitido por una población de moléculas y su temperatura absoluta.

$$B_\lambda(T) = \frac{2hc^2}{\lambda^5(e^{\frac{hc}{\lambda KT}} - 1)} \tag{1.14}$$

Donde K es la constante de Boltzmann, el espectro de un cuerpo negro 1.4 crece con la temperatura. Tiene un único máximo en λ_m. A longitudes de onda mas cortas que λ_m, B_λ decrece rápidamente, en contraste a longitudes de onda mas largas que λ_m, B_λ es una banda ancha que decrece lentamente.

Ley de desplazamiento de Wien

La longitud de onda λ_m a la que se obtiene la máxima intensidad se obtiene derivando la Ecuación (1.14)

$$\lambda_m = \frac{2897}{T}(\mu\text{m}) \tag{1.15}$$

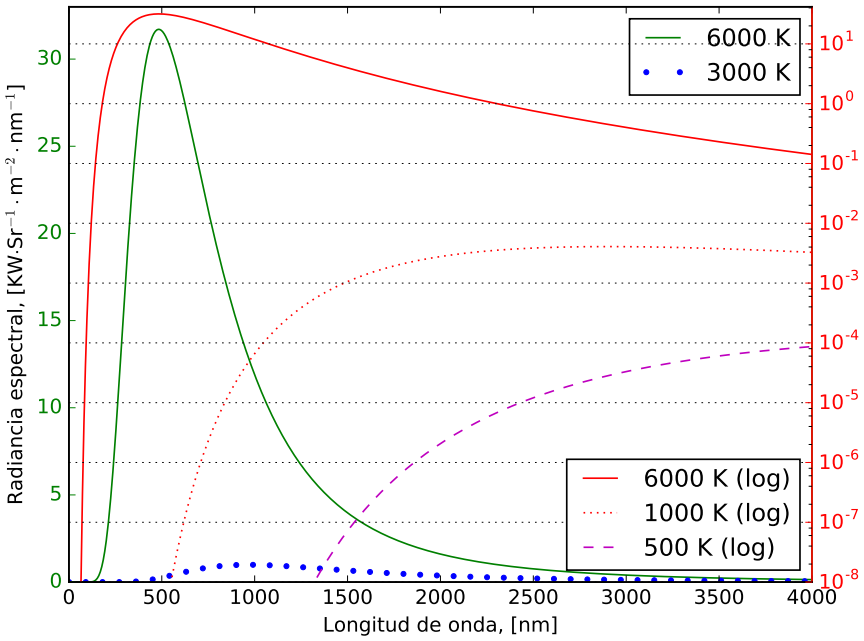


Figura 1.4: Espectros de emisión $B(\lambda, T)$ de cuatro cuerpos negros a temperaturas de T = 6000, 3000 1000 y 500 K. Para visualizar los espectros de 6000 y 500 K en una misma gráfica es útil la escala logarítmica en el eje vertical (lado derecho). El espectro para 6000 K está representado en escala logarítmica (línea continua roja) y en escala lineal 'normal' (línea continua verde). La emisión en la región UV a temperaturas menores a 500 K es despreciable.

el valor de λ_m decrece aumentando la temperatura del emisor. Esta característica del espectro de emisión permite inferir la temperatura del cuerpo negro analizando su espectro emitido.

Ley de Stefan-Boltzmann

Integrando (1.14) sobre el espectro electromagnético se obtiene que el flujo de radiación de un cuerpo negro es proporcional a $\sigma_{SB}T^4$, con σ_{SB} la constante de Stefan-Boltzmann.

Ley de Kirchoff

Un haz de radiación en un medio que se comporta como cuerpo negro es decrito por las leyes descritas anteriormente, pero muestras reales absorben y emiten radiación en proporciones menores a la de un cuerpo negro, estas pueden ser llamadas en su lugar, cuerpos grises. La eficiencia de emisión y absorción de un cuerpo gris son cuantificadas por la emisividad ϵ definida como la razón entre la intensidad de emisión y $B_\lambda(T)$, y la absortividad a definida como la razón entre la intensidad absorbida y $B_\lambda(T)$.

Para un cuerpo gris en equilibrio, el flujo de energía que emite $\epsilon\sigma_{SB}T^4$ debe ser igual al flujo de energía que absorbe $a\sigma_{SB}T^4$, esto se cumple para cualquier longitud de onda. La igualdad de $\epsilon_\lambda = a_\lambda$ es la ley de Kirchoff. Esta ley se cumple en estados de equilibrio termodinámico locales, en las que la temperatura es uniforme y la radiación es isotrópica en un volumen pequeño. Las condiciones se satisfacen cuanto las energías de transición son dominadas por las colisiones moleculares. Gases a alturas menores de 60 km se mantienen con presiones mayores a 0.01 hPa y la ley de Kirchoff sigue siendo válida, pero para alturas mayores, los estados de equilibrio térmico ya no ocurren porque el intervalo entre colisiones es corto comparado con el tiempo en que se establece un estado excitado asociado con la absorción y emisión, entonces la ley de Kirchoff ya no se cumple.

La fracción de energía emitida en un haz de radiación a lo largo de una distancia ds es

$$\frac{\mathrm{d}I_\lambda}{B_\lambda} = \mathrm{d}\epsilon_\lambda = \mathrm{d}a_\lambda = Nk\,\mathrm{d}s \tag{1.16}$$

Por la ley de Lambert

$$\mathrm{d}I_\lambda = NkJ_\lambda\,\mathrm{d}s \tag{1.17}$$

donde $J_\lambda = B_\lambda(T)$ define la función de fuente monocromática en la ausencia de esparcimiento, y en estados de equilibrio térmico local [5].

La temperatura promedio de la atmósfera es menor a 500 K, así para longitudes de onda corta, en particular el espectro visible y UV, la emisión es despreciable. Ver Figura 1.4.

1.1.5. Esparcimiento

La energía de un conjunto de moléculas esta dada por su energía cinética, asociada a la temperatura, a sus movimientos de rotación y traslación y a su energía electrónica. Estas formas de energía pueden ser excitadas con

la absorción de fotones que después es liberada de distintas maneras. Una transformación común se da cuando la energía interna es convertida a energía traslacional por medio de colisiones, esto se le conoce como termalización (el proceso inverso es la emisión térmica). La otra forma es cuando la energía excitada es reemitida por las moléculas en longitudes de onda y direcciones distintas de la dirección de la radiación inicial, este último es el proceso de esparcimiento.

El esparcimiento redirige la radiación de una dirección hacia otras direcciones. O bien los fotones de distintas direcciones llegan a un haz de radiación determinado.

Si los fotones son introducidos a un haz de radiación después de un solo encuentro con un partícula se le llama esparcimiento simple, por el contrario varios encuentros con partículas ocasiona esparcimiento múltiple.

El albedo para un esparcimiento simple es

$$\omega_\lambda = \frac{k_\sigma(\lambda)}{k(\lambda)} \tag{1.18}$$

donde k y k_σ fueron definidos por la Ecuación (1.8), la Ecuación (1.18) representa la fracción de radiación perdida con el esparcimiento hacia afuera del haz de radiación.

Entonces

$$1 - \omega_\lambda = \frac{k_\alpha(\lambda)}{k(\lambda)} \tag{1.19}$$

representa la fracción perdida a través de extinción, concretamente absorción en el haz de radiación.

La direccionalidad de la componente de esparcimiento es descrita por la función de fase $P_\lambda(\hat{\Omega}, \hat{\Omega}')$. La función de fase representa la fracción de radiación esparcida por una partícula de la dirección $\hat{\Omega}'$ en la dirección $\hat{\Omega}$. Si la función de fase es normalizada con

$$\frac{1}{4\pi} \int_{4\pi} P_\lambda(\hat{\Omega}, \hat{\Omega}') \, \mathrm{d}\Omega' = 1 \tag{1.20}$$

entonces,

$$\omega_\lambda \frac{P_\lambda(\hat{\Omega}, \hat{\Omega}')}{4\pi} \, \mathrm{d}\Omega' \tag{1.21}$$

la Ecuación (1.21) es la fracción de radiación perdida por extinción de un haz en dirección $\hat{\Omega}'$ que es esparcida a otro haz en dirección $\hat{\Omega}$.

Entonces la contribución del esparcimiento a la función de fuente es

$$J_{\sigma,\lambda} = \frac{\omega_\lambda}{4\pi} \int_{4\pi} I_\lambda(\hat{\Omega}') P_\lambda(\hat{\Omega}, \hat{\Omega}') \, \mathrm{d}\Omega' \tag{1.22}$$

Combinando las contribuciones de la emisión y el esparcimiento se obtiene la función de fuente [5] completa

$$J_\lambda = (1-\omega_\lambda) B_\lambda(T) + \frac{\omega_\lambda}{4\pi} \int_{4\pi} I_\lambda(\hat{\Omega}') P_\lambda(\hat{\Omega}, \hat{\Omega}') \, \mathrm{d}\Omega' \tag{1.23}$$

Uno de los objetivos en este trabajo es que el programa pueda disminuir el efecto de esparcimiento de fotones que ocurre entre la pluma y el campo de visión de la cámara, este efecto es conocido como dilución de luz [4], este efecto se debe mayormente a una combinación de esparcimiento Rayleigh y Mie que se revisará a continuación.

Esparcimiento Rayleigh

La luz blanca (policromática) proveniente del Sol se esparce en la atmósfera y de acuerdo al esparcimiento Rayleigh predominan las longitudes de onda corta que abarcan el color azul del espectro electromagnético visible. El efecto se presenta para partículas con tamaño menor a 10^{-7} m, los fotones incidiendo sobre partículas con estas dimensiones, interaccionan con los dipolos eléctricos de las moléculas haciéndolos oscilar. La oscilación de los dipolos reemite radiación con la misma frecuencia que la radiación incidente.

La radiación reemitida fue estudiada por Lord Rayleigh, determinando que la intensidad es inversamente proporcional a la cuarta potencia de la longitud de onda incidente y depende fuertemente del diámetro de las partículas [9],

$$I = I_0 \frac{9\pi^2}{2} \left(\frac{\epsilon - 1}{\epsilon + 2} \right)^2 \left(1 + \cos^2\theta \right) \frac{nV^2}{\lambda^4 r^2} \tag{1.24}$$

donde I e I_0 son las irradiancias esparcida e incidente (no polarizada), ϵ es la permitividad dieléctrica de la partícula relativa al medio que lo rodea, θ la dirección de propagación debida al esparcimiento, n es el número de partículas esparcida, V es el volumen de la partícula, λ es la longitud de onda de la radiación incidente y r es la distancia entre las partículas esparcidas y el detector (observador). La función de fase[3] $P(\theta)$ para el esparcimiento Rayleigh sin considerar la anisotropía de la polarizabilidad [8] va como $3/4 \cdot (1+\cos^2\theta)$.

[3] Considerando θ el ángulo entre $\hat{\Omega}$ y $\hat{\Omega}'$.

Para un haz delgado, los fotones que sean esparcidos fuera del haz tienen baja probabilidad de regresar al haz y llegar al detector, con este argumento, el esparcimiento es tratado como absorción al asociar un coeficiente de esparcimiento σ (proporcional a la sección transversal k_σ).

El coeficiente de esparcimiento para los aerosoles que presentan esparcimiento elástico Rayleigh es aproximado por

$$\sigma_{\text{Ray}} = \sigma_{R,0} \cdot \lambda^{-4} \tag{1.25}$$

donde $\sigma_{R,0}$ es una constante. En la región del infrarrojo térmico el esparcimiento Rayleigh es despreciable debido al término λ^{-4}.

Esparcimiento Mie

A diferencia del esparcimiento Rayleigh, el esparcimiento Mie tiene una menor dependencia con la longitud de onda, y se presenta mayormente para partículas mas grandes que una décima de micrómetro. En las nubes compuestas por gotas de agua, la luz solar policromática es esparcida con muy ligera alteración en sus longitudes de onda.

La teoría es generalmente compleja, pero al igual que en el esparcimiento Rayleigh, el esparcimiento Mie se simplifica con un coeficiente de esparcimiento, haciendo uso de la aproximación de haces delgados (Narrow Beam).

El coeficiente de esparcimiento [3] para una longitud de onda dada puede ser estimado como

$$\sigma_{\text{Mie}} = \sigma_{M,0} \cdot \lambda^{-\gamma} \tag{1.26}$$

donde γ es el exponente de Ångström, este exponente es inversamente proporcional al tamaño de las partículas de aerosol. El exponente γ tiene valores entre 0.5 y 2.5, un valor promedio 1.3 es reportado en [8].

Una función de fase analítica comúnmente introducida para reducir cálculos, es la parametrización de Henyey–Greenstein

$$P(\cos\theta) = \frac{1-g^2}{4\pi(1+g^2-2g\cos\theta)^{3/2}}, \tag{1.27}$$

la cual depende del parámetro asimétrico g, que es definido como un promedio: $g =< \cos\theta >$. Valores típicos de g para aerosoles en la tropósfera se encuentran entre 0.6 y 0.7 [8].

Los coeficientes de extinción para gotas esféricas líquidas de agua con un rango promedio de 10 a 20 μm de diámetro son del orden de 10^{-2}m^{-1}, mientras que coeficientes típicos de esparcimiento Rayleigh y Mie de moléculas

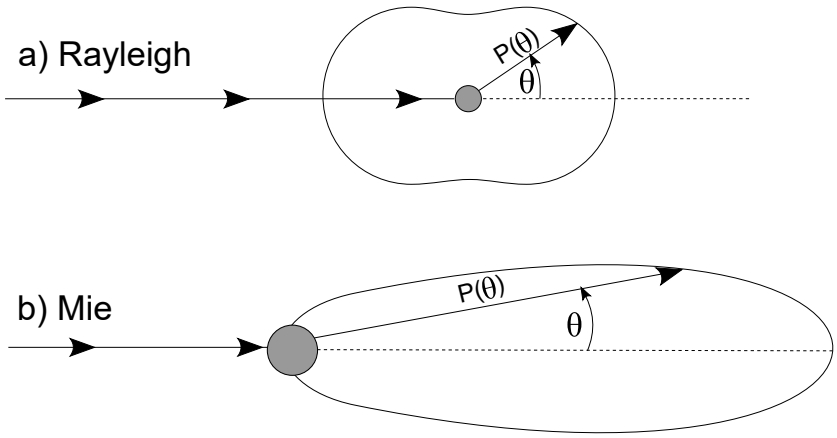


Figura 1.5: Distribución angular del esparcimiento debido a partículas con tamaños a) menor y b) mayor a la longitud de onda incidente, las funciones de fase P están en función del ángulo de esparcimiento θ. El diagrama b) implica que la energía redirigida por partículas grandes es dominada por esparcimiento en la dirección de la radiación incidente.

del aire tienen un orden de magnitud de $10^{-5}\mathrm{m}^{-1}$ en longitudes de onda de 300 a 500 nm [8, 10].

En la Figura 1.5 se muestran características en las que difieren el esparcimiento Mie y Rayleigh. El esparcimiento Rayleigh es dominante en partículas con tamaño menor a la longitud de onda incidente y en longitudes de onda cortas (dependencia con λ^{-4}). El esparcimiento de Mie tiene menor dependencia con la longitud de onda y se presenta para partículas con un tamaño mayor a la longitud de onda incidente (aerosoles y gotas de agua con tamaño del orden de 10^{-7}m).

1.1.6. Ecuación de la transferencia radiativa

La proporción en la cual la intensidad dentro de un haz de radiación cambia en la dirección del camino óptico s, está dada por la energía que entra + la que sale

$$\frac{\mathrm{d}I_\lambda}{Nk_\lambda \mathrm{d}s} = -I_\lambda + J_\lambda \tag{1.28}$$

La Ecuación (1.28) es la ecuación general de la transferencia radiativa.

Adaptando la ecuación general de la transferencia radiativa a una atmósfera, y mostrando explícitamente las contribuciones de absorción, emisión y

esparcimiento revisados anteriormente, para los distintos componentes i de la atmósfera: aerosoles y moléculas resulta

$$\begin{aligned}\frac{\mathrm{d}I_\lambda}{ds} = &-I_0\left(\sum_i k_{\alpha,i}N_i + \beta_\alpha\right)\\ &+(1-\omega_\lambda)B_\lambda(T)\left(\sum_i k_{\alpha,i}N_i + \beta_\alpha\right)\\ &+\left(\sum_i k_{\sigma,i}N_i + \beta_\sigma\right)\frac{\omega_\lambda}{4\pi}\int_{4\pi} I_\lambda(\hat{\Omega}')P_\lambda(\hat{\Omega},\hat{\Omega}')\,\mathrm{d}\Omega'\end{aligned} \tag{1.29}$$

donde se han usado los coeficientes de absorción β_α y esparcimiento β_σ para los aerosoles, así como las secciones transversales de absorción k_α y esparcimiento k_σ de las moléculas. El primer término es la fracción de la radiancia absorbida, donde contribuyen cada componente i de la atmósfera, el segundo término es debido a la emisión, $(1-w)$ es la fracción de energía que fue absorbida y que será reemitida de acuerdo al espectro de radiancia de Planck, el tercer término es la fracción de energía esparcida en todas las direcciones dictadas por las funciones de fase P [5].

La solución a la Ecuación (1.28) si se conocen k_λ, N y J_λ como función de la distancia s se obtiene introduciendo la extinción de camino óptico χ_λ

$$\chi_\lambda(s) = \int_{s_0}^{s} N(s')k_\lambda(s')\,\mathrm{d}s' \tag{1.30}$$

donde s_0 es el punto inicial del camino óptico, entonces la Ecuación (1.28) se reescribe

$$\frac{\mathrm{d}I(\lambda)}{\mathrm{d}\chi(\lambda)} + I(\lambda) = J_\lambda \tag{1.31}$$

después de multiplicar por el factor integrante $\exp(\chi_\lambda)$ la Ecuación (1.31), se integra para obtener

$$I(\lambda)\exp(\chi) = \int J_\lambda \exp(\chi')\,\mathrm{d}\chi' + \text{constante} \tag{1.32}$$

si la radiancia espectral es $I_\lambda(0)$ en el punto s_0 entonces

$$I_\lambda(s) = \int_0^\chi J_\lambda \exp(-(\chi-\chi'))\,\mathrm{d}\chi' + I_\lambda(0)\exp(-\chi(s)) \tag{1.33}$$

de este resultado se nota que en la ausencia de emisión ($J_\lambda = 0$) la radiancia espectral decae exponencialmente y esta es la ley de Beer Lambert Bouguer.

Ley de Beer Lambert Bouguer en la atmósfera

Un haz de luz que pasa por un volumen dado de la atmósfera es disminuido por tres razones físicas

- La concentración. Número de partículas que la absorben en su trayectoria.
- Las distancias (distancia del camino óptico) que debe atravesar en la muestra.
- La sección eficaz de extinción, es un valor asociado a la probabilidad de que un fotón de cierta amplitud de onda, sea absorbida por las partículas en la muestra.

Una atmósfera contiene una mezcla de componentes x, lo que convierte a la Ecuación (1.6) en

$$\tau = \int_{s_0}^{s_1} \sum_x N_x k_x(\lambda)\,\mathrm{d}s \tag{1.34}$$

Si el medio es homogeneo i.e. k y N son constantes, y se tiene una longitud finita s

$$\tau = \sum \tau_x(\lambda) = \sum k_x(\lambda) N_x (s_1 - s_0) \tag{1.35}$$

La ley de Beer-Lambert-Bouguer (1.7) se escribe como

$$I_\lambda(s, \hat{\Omega}) = I_\lambda(0, \hat{\Omega}) e^{-m \sum \tau_x(\lambda)} \tag{1.36}$$

En la Ecuación (1.36) cada τ_x representa una profundidad óptica, cuyo subíndice x sirve para identificar la fuente de absorción y/o esparcimiento. τ_x es para un camino vertical, el factor de masa de aire m es una corrección que considera el camino oblicuo de la luz y esparcimiento múltiple. Para una atmósfera plano paralela, o rayos solares atravesando la atmósfera de formas cercanamente perpendiculares, una aproximación simple se obtiene con $m = \sec(\theta)$ donde θ es el ángulo cenital del camino óptico.

La ley de Beer–Lambert–Bouguer es comúnmente descrita en términos de la absorbancia A en lugar de la profundidad óptica τ, estas definiciones se han utilizado dependiendo de la comodidad (y afinidad por la notación en Química o Física) por usar unas u otras unidades en sus mediciones [11]. La

profundidad óptica, como se definió arriba, para un gas con distintos componentes i está dado por

$$\tau = \ln \frac{I_\lambda^0}{I_\lambda} = \sum_i k_{\alpha,i} N_i s \tag{1.37}$$

y la absorbancia

$$A = \log_{10} \frac{I_\lambda^0}{I_\lambda} = \sum_i \hat{\epsilon}_i C_i s \tag{1.38}$$

donde $\hat{\epsilon} \rightarrow [\mathrm{dm^3 mol^{-1} cm^{-1}}]$ es el coeficiente de absorción molar, $C \rightarrow [\mathrm{mol\ dm^{-3}}]$ es la concentración molar y $s \rightarrow [\mathrm{cm}]$ la distancia.

La conversión entre coeficientes se hace considerando

$$C = \frac{N}{N_A} \tag{1.39}$$

$$\hat{\epsilon} = \frac{N_A}{\ln 10} k_\alpha \tag{1.40}$$

con N_A el número de Avogadro, por tanto la relación entre la profundidad óptica y la absorbancia es $A = \tau / \ln 10$.

1.2. Química Atmosférica y Gases Volcánicos

1.2.1. La atmósfera

La atmósfera es una capa que divide la superficie terrestre del espacio, esencial para la vida terrestre. La atmósfera es responsable de la redistribución de agua y calor, de protegernos de radiación intensa, elevar la temperatura de la superficie terrestre y proveer oxígeno. El interés por medir los procesos físicos y químicos ha crecido debido a que la actividad antropogénica la esta cambiando de forma acelerada y esto nos afecta directamente. Se verá mas adelante que también hay fuentes de origen no antropogénico, como las erupciones volcánicas, las cuales han influido en la evolución y composición de los gases en la atmósfera.

Química atmosférica

El aire en la atmósfera está compuesto principalmente por nitrógeno N_2 y oxígeno O_2, otros constituyentes se encuentran en menor proporción. Ver

Tabla 1.1 El peso molecular de los componentes del aire es en promedio de 28.97, lo que hace que en condiciones estándar de temperatura y presión (T=273K y p=1013h Pa) 22.4 L de aire sea equivalente a 22 g.

La composición de la atmósfera es controlada por los elementos provenientes del interior de la Tierra y de la vida en su superficie. La fotosíntesis produce oxígeno, y compuestos a base de nitrógeno son metabolizados por organismos los cuales regresan a la atmósfera como N_2. El gas argón es producido por decaimiento radioactivo de un isótopo del potasio proveniente de la corteza terrestre [12]. El vapor de agua es resultado principalmente de la evaporación de agua en los océanos. La proporción de algunos constituyentes (especialmente los que son física y químicamente activos) son variables en el espacio y en tiempo, tales constituyentes incluidos el agua, bióxido de carbono y ozono están relacionados con las actividades antropogénicas.

La composición química actual (Ver Tabla 1.1) ha sido producto de la evolución, la cantidad de nitrógeno, oxígeno y vapor de agua ha tenido contribuciones de las primeras erupciones volcánicas que ocurrieron con gran intensidad, definiendo la composición de nuestra atmósfera e hidrosfera [13].

El interés por conocer algunos gases que están en la atmósfera se remonta a Christian Schönbein quien midió la concentración de ozono en la atmósfera. Las primeras técnicas espectroscópicas fueron introducidas por Kirchhoff y Bunsen.

Tabla 1.1: Principales gases que componen una atmósfera seca, libre de contaminantes [8].

Gas	Fórmula química	Proporción por Vol [%]
Nitrógeno	N_2	78.08
Oxígeno	O_2	20.95
Argón	Ar	0.93
Dióxido de carbono	CO_2	0.037
Neón	Ne	0.0018
Helio	He	0.00052
Metano	CH_4	0.00017
Kriptón	Kr	0.00011
Xenón	Xe	0.00009
Hidrógeno	H_2	0.00005
Dinitrógeno óxido	N_2O	0.00003

En la actualidad los instrumentos de medición remota mas difundidos para el estudio de la química atmosférica son el método de espectroscopia de absorción óptica diferencial (DOAS - Differential Optical Absorption Spectroscopy), la espectrometría infrarroja con transformada de Fourier (FTIR - Fourier Transform InfraRed Spectrometry) y la obtención de imágenes satelitales. Otro instrumento que ha dejado de usarse es el espectrómetro de correlación (COSPEC - COrrelation SPECtrometer) debido a su costo y a que la técnica de medición ha sido superada por las usadas actualmente.

Fuentes de emisión de gases a la atmósfera

Los gases en la atmósfera son emitidos por el suelo (e.g. metano, óxidos de nitrógeno NO y N_2O), por la vegetación (compuestos orgánicos volátiles), quema de biomasa (CO_2, CO_4). Hay contribuciones de gases por emisiones marinas, emisiones volcánicas (CO_2, SO_2, HCl, BrO, etc.) y antropogénicas.

Concentraremos la atención en las emisiones volcánicas porque son una de las mayores fuentes de SO_2 y este es el gas detectado en las imágenes de la cámara SO_2.

Aerosoles

Los aerosoles son partículas suspendidas y esparcidas en el aire, pueden ser partículas sólidas o gotas líquidas. Para permanecer suspendidas, deben tener estabilidad en la atmósfera, esto es determinado por sus velocidades, pues a mayor velocidad, mayor es la probabilidad de sedimentarse.

El límite superior para el tamaño de un aerosol es marcado, debido a que la velocidad terminal tiene un comportamiento cuadrático con el radio de la partícula. Así las partículas con radio mayor a 10 μm se consideran partículas de polvo gruesas y son removidas por sedimentación. El limite inferior en el tamaño de un aerosol para mantenerse estable depende de las condiciones de temperatura y humedad relativa, pero no es menor a 0.2 nm, que es aproximadamente el tamaño del radio de una molécula (N_2 o O_2) [8].

Las propiedades de los aerosoles dependen de su composición (elementos químicos solubles e insolubles en agua). En estas partículas ocurren varios procesos de transferencia de masa y calor, influenciando la termodinámica de la atmósfera. Su interacción con la radiación se presenta de varias maneras, pueden esparcir y absorber energía reflejando, difractando y refractando la radiación incidente.

1.2.2. Volcán

Un Volcán es una abertura en la corteza de un objeto planetario. En la Tierra tenemos miles de volcanes, estos pueden emitir lava, cenizas volcánicas, y emanar gases desde sus cámaras magmáticas. La principal razón por la que los volcanes aparecen en la tierra es debido a 17 placas tectónicas, que flotan sobre el manto. La mayoría se ubican en las zonas de convergencia o divergencia entre placas tectónicas. El magma puede subir a la corteza terrestre en forma de erupciones, con distintas intensidades, duración y frecuencia. Esto hace cambiar sus formas y comportamiento a través del tiempo.

El consenso mas general de los geólogos en los tipos de volcanes, es que existen: conos de escoria, volcanes compuestos, volcanes escudo y domos de lava.

Pluma volcánica

El flujo de gases emitidos por un volcán es conocido como pluma volcánica. Se trata de una mezcla de partículas sólidas y gases emitidos en una erupción producidas por la fragmentación del magma [14].

Los materiales sólidos pueden ser lava o fragmentos esparcidos (material piroclástico) estos son fáciles de estudiar porque pueden tomarse muestras alejadas de la abertura del volcán. En el caso de los gases se empezaron a estudiar recientemente (1950), la razón es que no se tenían instrumentos ni técnicas de muestreo confiables, además del peligro de acercarse y exponerse a temperaturas altas o gases tóxicos.

Estudiar los gases volcánicos tiene relevancia porque estos controlan la fuerza de las erupciones, son considerados como mensajeros del magma y los compuestos químicos influyen en la química atmosférica afectando el clima. En la Tabla 1.2 se resumen los principales componentes del gas magmático. La mezcla de gases que sale se diluye en la atmósfera en hasta concentraciones menores a 1%, esta es la pluma volcánica, la cual puede ser transportada por la atmósfera por miles de kilómetros.

La composición, origen y flujo de los gases en una pluma volcánica depende de varios factores. Los que influyen en su composición son

- Del contexto geodinámico
- Composición e historia del magma

Tabla 1.2: Principales gases característicos del gas magmático a su salida por el cráter [15].

Componente	Fórmula química	%/vol
Vapor de agua	H_2O	50-90
Bióxido de carbono	CO_2	1-40
Dióxido de azufre	SO_2	1-25
Ácido sulfhídrico	H_2S	1-10

- De la geometría del conducto
- La profundidad del magma que desgasifica
- Proporción de los gases liberados por el magma, los gases atraviesan varios km de rocas a veces saturadas con agua, producto de los sistemas hidrotermales. Parte de los gases se disuelven y ya no llegan a la superficie.

Y los que influyen el flujo de gases

- Grado de abertura del sistema
- La tasa de magma que desgasifica
- Abundancia de los elementos volátiles en el magma.

Los gases volcánicos como motor de las erupciones

Antes una erupción ocurren exsoluciones de gases disueltos en el magma (H, C, S, Cl, F, He, Ar, Br, B, Rn) en forma de burbujas, haciendo disminuir la densidad global del magma, es decir adquiere la flotabilidad necesaria para ascender a la superficie. Los gases proporcionan la fuerza para impulsar la roca fundida desde la chimenea volcánica.

Un indicio de una erupción es la dilatación de la cima, lo que significa que el magma se desplaza. Los componentes volátiles (agua principalmente) se acumulan en la parte superior de la cámara magmática.

La erupción comienza cuando el magma cargado de gases sale de la cámara magmática, por el conducto volcánico o chimenea. La reducción de presión

conforme el magma llega a la superficie, permite que los gases disueltos se liberen súbitamente.

Durante las erupciones hay fragmentación de lava y estos son lanzados a la atmósfera. La erupción depende de la viscosidad del magma, los magmas basálticos, muy fluidos, dejan que los gases en expansión migren hacia arriba y escapen por la chimenea con facilidad, impulsando a su paso lava incandescente. Por otro lado magmas viscosos expulsan de manera explosiva gases calientes cargados de cenizas que evolucionan a plumas con gran fuerza ascensional denominadas columnas eruptivas extendiéndose a varios kilómetros en la atmósfera. Un acontecimiento exclusivo suele ir seguido de una emisión tranquila de lavas desgasificadas. Luego el proceso de acumulación de gases vuelve a empezar.

La viscosidad del magma, la cantidad de gases disueltos y la facilidad con que escapan determina la naturaleza de la erupción. Las erupciones relativamente tranquilas corresponden a lavas líquidas y calientes de Hawaii y las erupciones explosivas provienen de lavas viscosas del tipo del monte de Santa Elena [14].

Los gases en los magmas constituyen de 1 a 6 % del peso total, y la mayor parte es vapor de agua. Decenas de miles de m^3 por día si se consideran condiciones típicas de presión y temperatura magmáticas (5 atm y 1200 K). Aunque el porcentaje es pequeño el flujo real puede superar varios miles de toneladas por día.

Varios volcanes tienen la característica de emitir gases antes y entre erupciones La dinámica y composición de estos gases contiene información sobre los procesos que estan ocurriendo al interior del volcán.

Efectos a la salud, vegetación, infraestructura y al clima

Aunque generalmente en una emisión volcánica el bióxido de carbono, bióxido de azufre y H_2S, HF, HCl y HBr no son tan abundantes como el vapor de agua, concentraciones altas si pueden representar un riesgo para personas, animales y agricultura.

Bióxido de carbono

El bióxido de carbono es una gas inodoro e incoloro, constituye aproximadamente 0.04 % del aire, en un año, en promedio los volcanes emiten entre 180 y 440 millones de toneladas. Cuando es emitido por un volcán típicamen-

te se diluye y las concentraciones bajan rápidamente al punto de no ser una amenaza, pero debido a que el dióxido de carbono a bajas temperaturas es más denso que el aire promedio, se puede acumular en altas concentraciones en condiciones atmosféricas estables. Respirar aire con mas de 3% de bióxido de carbono produce mareos y dolores de cabeza aumentando la frecuencia cardiaca y la dificultad para respirar. Mezclas con mas de 15% son letales.

Las emisiones volcánicas de CO_2 en promedio son menores a las emisiones producto de las actividades humanas [16]

Bióxido de azufre

Es un gas incoloro que irrita la piel, tejidos y membranas en los ojos, nariz y garganta.

Para cuestiones climáticas, el gas clave es el dióxido de azufre, cuando es

Figura 1.6: Modelo de una molécula de SO_2 [17]

lanzado a mas de 10 km en la estratósfera, se empieza a mezclar con agua y ocurren reacciones de oxidación con radicales libres, formando pequeñas partículas (aerosoles) de sulfatos.

Varias de las reacciones que destruyen el ozono ocurren en la superficie de aerosoles compuestos de sulfatos, también reflejan una fracción de la luz solar de regreso al espacio, la parte que no es transmitida se absorbe calentando la estratósfera y deja la superfice debajo con menor temperatura. Estos cambios afectan el clima. Un ambiente frio significa menos evaporación, y por tanto una disminución en las lluvias.

H_2S

Es un gas incoloro con olor a huevos podridos, el olfato humano es incluso mas sensible que los instrumentos para monitoreo de gases, hasta en proporciones de 0.000001% disueltas en aire.

En cantidades arriba de 0.01% se vuelve inodoro y muy tóxico. Causa irritación en el tracto respiratorio y edemas pulmonares si se tienen largos periodos de exposición.

Concentraciones mayores a 500 ppm pueden dejar inconsciente a una persona en 5 minutos y causar la muerte en una hora [16].

HF, HCl, HBr

Son ácidos fuertes que se disuelven en gotas de agua rápidamente, con potencial para producir lluvia ácida, afectando la vegetación, infraestructura humana y agricultura. Las cenizas pueden ayudar a esparcirlas y llegar a contaminar depósitos de agua.

1.2.3. Técnicas para el monitoreo de gases volcánicos

La recolección de muestras directas de lava fue una de las primeras técnicas aplicadas por geólogos. En áreas pequeñas donde hay actividad de lava incandescente, se pueden tomar muestras con pocas medidas de protección como guantes y botas.

Las muestras son analizadas posteriormente en un laboratorio para determinar su composición química. Los geólogos pueden estudiar los procesos que ocurrieron en la cámara magmática e incluso del manto donde el magma fue generado [18]. Sin embargo esta técnica es muy limitada, no se puede hacer de forma constante y los resultados se obtienen tarde. Por ello las mediciones remotas basadas en la obtención de espectros de absorción han tenido mayor éxito en el muestreo de gases volcánicos. Una limitación en estas mediciones es que los modelos usados de absorción y transferencia radiativa describen los fenómenos en condiciones ideales, por tanto para aplicarse a condiciones reales y conservar la confianza en los resultados se deben justificar suposiciones y hacer correcciones adaptando el modelo a las condiciones reales.

Espectro electromagnético

El espectro electromagnético se refiere al dominio de longitudes de onda de la radiación, es dividida en varias regiones (Ver Figura 1.7), la división de estas regiones es simple conveniencia y se ha ido definiendo con el estudio de la óptica y por sus aplicaciones.

El espectro visible es la región que el ojo humano percibe, típicamente de ~ 390 a ~ 750 nm. La región ultravioleta comprende las longitudes de onda de ~ 400 a ~15 nm.

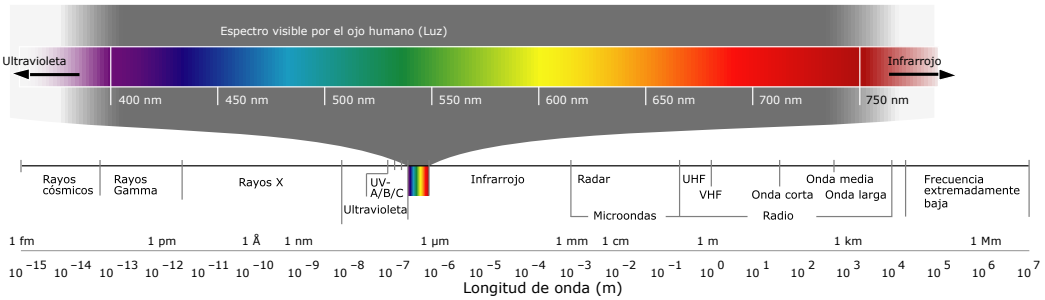


Figura 1.7: Espectro electromagnético [19].

COSPEC

Un espectrómetro de correlación mide la razón entre irradiancias. El sistema de elementos ópticos y mecánicos en el COSPEC es complejo, la selección de longitudes de onda necesarias es alcanzada con un arreglo de discos rotatorios y aberturas sincronizadas que hacen coincidir crestas y valles en pares de espectros de absorción de SO_2. La medición de densidades de columna se logra haciendo comparaciones con celdas de calibración que contienen cantidades de SO_2 conocidas.

Hay dos formas de tomar mediciones con el COSPEC, una es atravesar con la línea de visión perpendicular a la pluma, por debajo de la pluma y un desplazamiento horizontal. La otra es mantener fijo el equipo debajo de la pluma, y escanearla en distintos ángulos. El cálculo del flujo se obtiene haciendo consideraciones geométricas y multiplicando por la velocidad del viento.

Su principal desventaja es que es costoso y pesado, además de que funciona como una caja negra, porque no se tiene el acceso a la intensidad espectral en las distintas bandas en que se mide; es susceptible a errores por efectos de esparcimiento de la luz. Esto conduce a que su uso sea limitado.

DOAS

Desde principio de los 90's se extendió debido a la portabilidad de los espectrómetros UV. Dando lugar al mini-DOAS [20]. Este instrumento esta diseñado para que la luz incidente sea recogida por un telescopio y transferida al espectrómetro a través de fibra óptica. Dentro del espectrómetro la luz se colima, pasa por dos espejos y una rejilla de difracción donde la

luz es separada en sus longitudes de onda componentes y reflejada hacia un arreglo lineal de sensores CCD. La intensidad de la radiación es registrada en los sensores, los valores son digitalizados y pasan a almacenarse en una computadora.

Las principales ventajas del DOAS son el bajo costo, bajo consumo de energía, toma de datos rápida y directa. Ofrece la posibilidad de obtener información de otros gases como el BrO. Otra ventaja es que en la información que obtiene se pueden explorar varias bandas espectrales. La calidad de las mediciones da oportunidad de corregir efectos que ocurren en la transferencia radiativa [8].

Cámara SO_2

Es un instrumento que permite obtener imágenes de la absorción de luz ultravioleta por las moléculas en su campo de visión. Utiliza la luz solar esparcida por la atmósfera como fuente de luz ultravioleta (Ver Figura 1.8), sus componentes principales son un filtro y un sensor CCD. Una medición ideal busca un campo de visión libre de nubes, luz solar que atraviese perpendicularmente los gases volcánicos (Ver Figura 1.9)

Utilizando la ley de Beer Lambert Bouguer se convierten las imágenes 'crudas' de la cámara (valores de intensidad luminosa) a imágenes que contienen información de la distribución del dióxido de azufre, con el procesamiento de estos datos es posible obtener el flujo de dióxido de azufre en puntos cercanos al cráter del volcán.

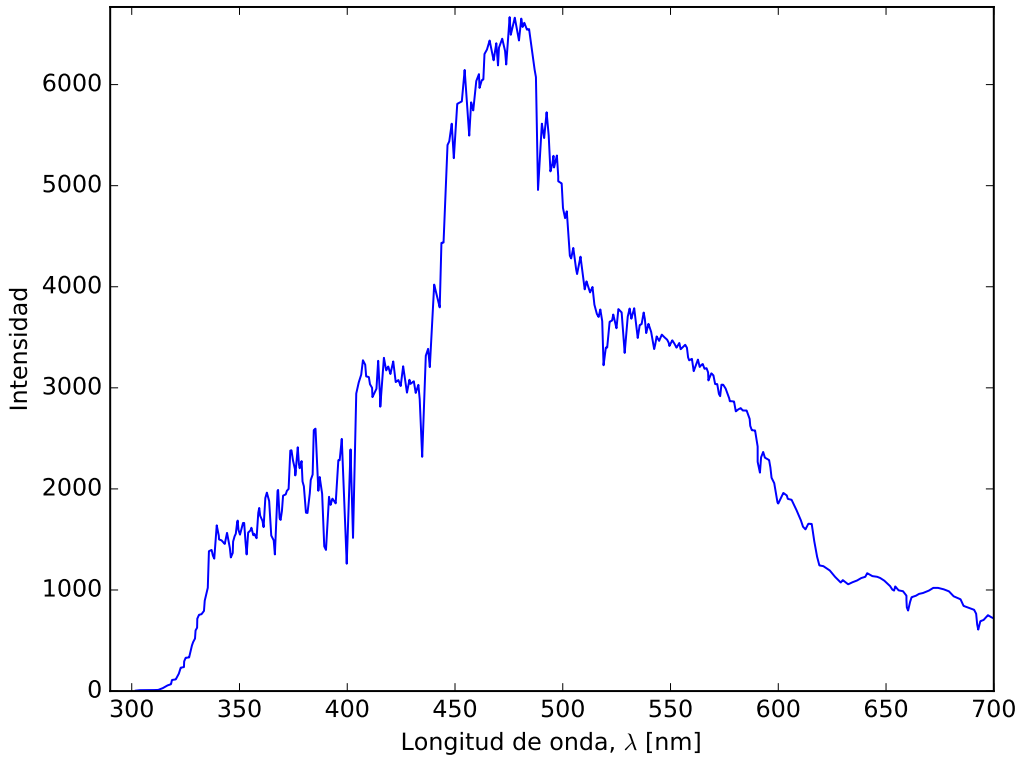


Figura 1.8: Espectro de radiación solar en la atmósfera. La fuente de radiación de la cámara con filtros de 300 y 330 nm, es la radiación solar difusa. El espectro medido depende del instrumento, ésta medición es consistente con el modelo para un cuerpo negro y la ley de Wien (cf. Fig. 1.4) que predice un pico de intensidad alrededor de 475 nm para un cuerpo negro a una temperatura de 6000 K.

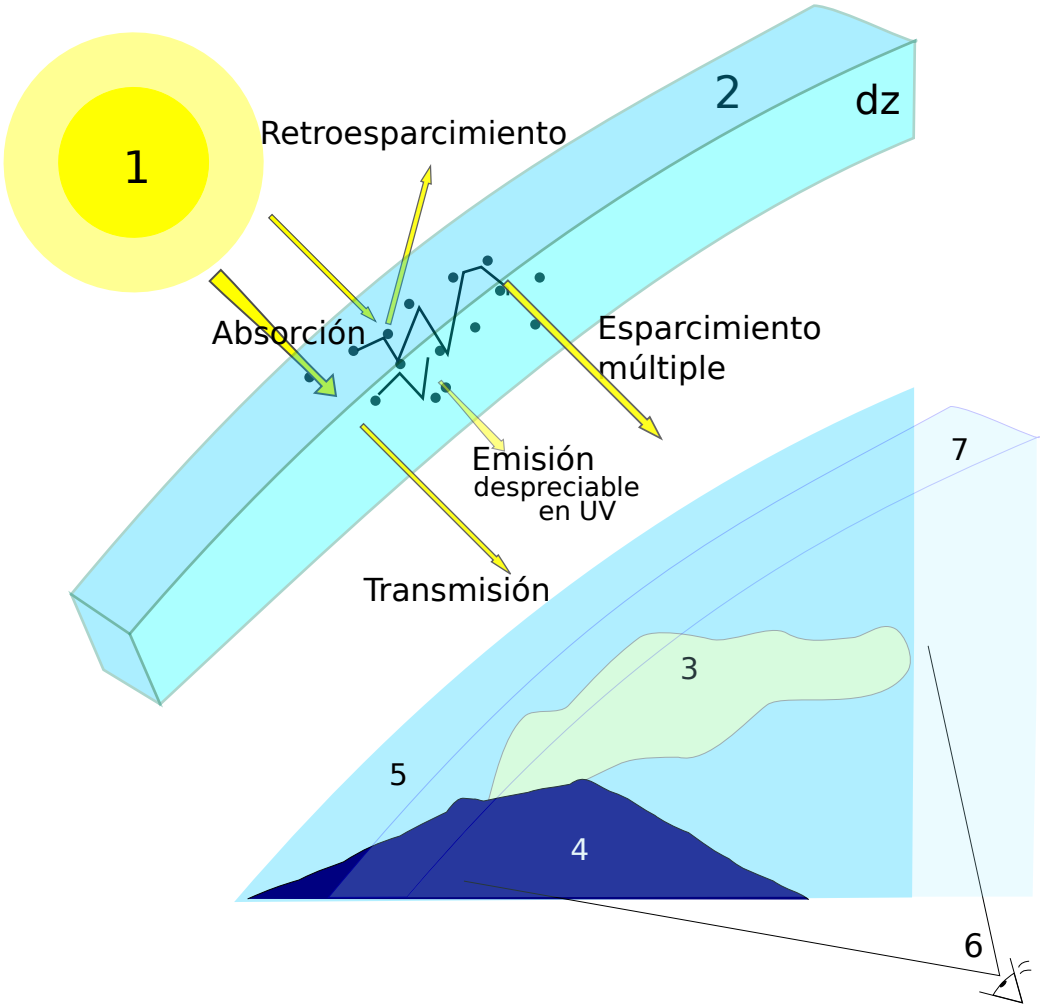


Figura 1.9: 1) fuente de radiación, 2) sección de una capa de la atmósfera (fuente de radiación difusa), 3) gases volcánicos, 4) edificio volcánico, 5) cielo (background), 6) campo visual del observador y 7) atmósfera intermedia pluma-cámara.

Capítulo 2

La cámara SO_2

2.1. ¿Qué es una cámara SO_2?

Es un dispositivo diseñado para medir la irradiancia, produciendo arreglos bidimensionales (imágenes 2D) de intensidades . Esto permite cuantificar la radiación ultravioleta absorbida por las moléculas de dióxido de azufre.

Para hacer mediciones remotas a la emisión de gases volcánicos, la cámara utiliza como fuente, la luz solar esparcida por la atmósfera. Aprovechando el hecho de que el dióxido de azufre suele ser abundante en los gases volcánicos en comparación a una atmósfera promedio.

Las imágenes procesadas y analizadas proporcionan información de la distribución bidimensional del bióxido de azufre.

2.1.1. Aplicaciones

Medir el flujo de SO_2 es una estrategia para conocer el comportamiento de un volcán, porque está relacionado con el flujo de magma.

Una de sus principales ventajas es que puede obtener imágenes cada segundo, esto permite estudiar la dinámica de la pluma en la atmósfera, monitorear el comportamiento de un volcán comparando con datos sísmicos y acústicos, de esta manera evaluar posibles riesgos.

Su alta resolución temporal y espacial permite distinguir si en la distribución de SO_2 aparece más de una fuente, D'Aleo et al. estudiaron las series de tiempo distinguiendo entre distintas fuentes de gas en el volcán Etna

[21]. La resolución espacial de la cámara utilizada es de aproximadamente 5 m para una distancia pluma-cámara de 10 km. Toda esta información puede usarse para corroborar y estudiar modelos termodinámicos y de esparcimiento de la pluma en la atmósfera. Además de usarse como complemento a mediciones satelitales.

2.1.2. Limitaciones y ventajas

La limitación más importante de la cámara es que no obtiene información espectral, la profundidad óptica es obtenida con las intensidades integradas sobre las longitudes de onda que pasan por el filtro, de esta forma los comportamientos de los espectros de absorción o de radiación difusa quedan reducidos a un solo valor de intensidad. Lo anterior vuelve importante usar filtros con rangos de transmitancia angostos. Las condiciones ideales en las mediciones de campo son ausencia de nubes y posición del sol fuera del campo visual.

Las técnicas DOAS y COSPEC son afectadas por cambios de la velocidad del viento en el tiempo, las cuales dan lugar a incertidumbres. En estos casos las velocidades se deben determinar adecuadamente cuidando de no atribuir los cambios de flujo de SO_2 a las condiciones meteorológicas y si al comportamiento del volcán. La sucesión de imágenes de la cámara contiene información de los desplazamientos en la distribución de SO_2, la velocidad es obtenida como un valor local en el tiempo para un par de imágenes consecutivas, esta es una ventaja respecto a las técnicas DOAS y COSPEC.

2.1.3. Antes de las cámaras SO_2

Antes que las cámaras SO_2 aparecieron el COSPEC y DOAS que se limitan a medir la densidad de columna del gas (trace gas column density) en una sola dirección, alrededor de 10^{-4} estereoradianes.

Las técnicas de imagenología aplicadas en gases volcánicos eran lentas, tardaban hasta 20 minutos. Es el caso del Imaging-DOAS que reproduce una imagen 2D escaneando un arreglo de direcciones, desafortunadamente la pluma volcánica puede tener movimientos significativos en periodos del orden de segundos, dependiendo de la velocidad del viento, y por tanto las plumas 2D obtenidas pueden verse distorsionadas [1].

La técnica con DOAS sigue vigente porque da información espectral y permite obtener concentraciones de otros gases. Las mediciones de la cáma-

ra SO_2 se complementan con mediciones DOAS para hacer calibraciones, y corroborar datos.

2.2. Principio de medición

La cámara SO_2 mide la radiancia UV transmitida a través de una pluma volcánica en dominios de longitud de onda donde la absorción por las moléculas de SO_2 es fuerte comparada con la de otras moléculas. Los componentes esenciales de una cámara SO_2, son un sensor CCD sensible en el ultravioleta y un filtro espectral pasa bandas.

La configuración con dos filtros resulta mas eficiente. Primero se describirá el proceso que conlleva usar un filtro. El filtro regula la sección del espectro de radiación que entra al sistema óptico, esto puede apreciarse en la curva de transmitancia $T(\lambda)$ que se muestra en la Fig. 2.1, en la misma figura se sobrepone la sección transversal de absorción del SO_2 $k_\alpha(\lambda)$, y la intensidad de la radiación solar difusa en la atmósfera I_S (la que vemos como cielo sin nubes y libre de pluma volcánica).

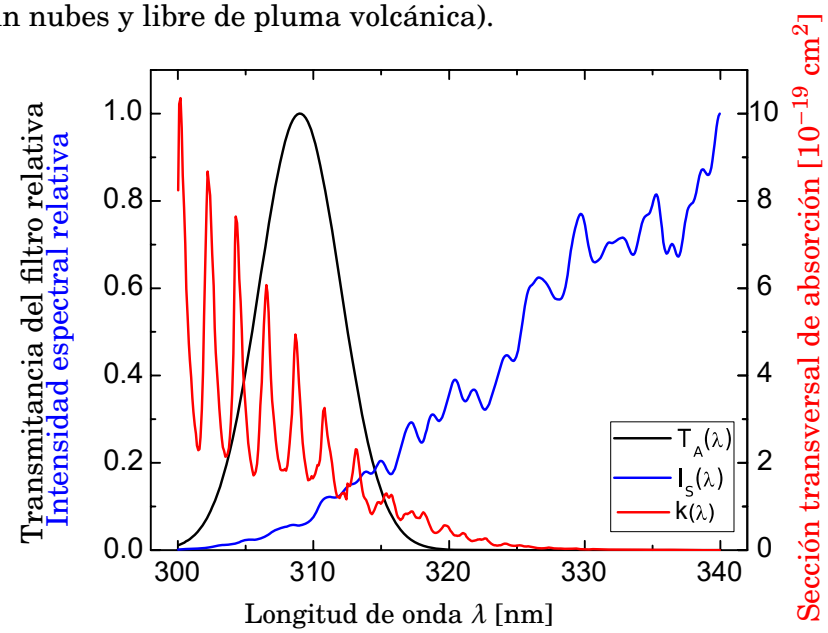


Figura 2.1: Curvas de transmitancia $T(\lambda)$, sección transversal de absorbción de SO_2 $k(\lambda)$ e intensidad solar espectral relativa $I_S(\lambda)$, la cual es una región pequeña de la gráfica en la Figura 1.8. Curvas extraídas de [2].

En una dirección libre de SO_2, a cada pixel del sensor CCD le llega radiancia $I_0(\lambda)$ (el subíndice cero se refiere a la radiación que llega al sensor, cuando se apunta a la fuente de radiación difusa que no atraviesa la pluma).

La radiancia $I_0(\lambda)$ depende de la transmitancia del filtro $T(\lambda)$, de la eficiencia cuántica $Q(\lambda)$ y de la radiación solar difusa $I_S(\lambda)$ [2]

$$I_0(\lambda) = I_S(\lambda) \cdot T(\lambda) \cdot Q(\lambda) \tag{2.1}$$

Si ahora en el camino óptico de la radiación incidente se encuentra gas SO_2, la intensidad espectral decrece a $I_p(\lambda)$, la atenuación se explica con la Ley de Beer-Lambert-Bouguer

$$I_p(\lambda) = I_0(\lambda) \cdot e^{-k_\alpha(\lambda) \cdot S(\lambda)} \tag{2.2}$$

Donde $k_\alpha(\lambda)$ es la sección transversal de absorción de SO_2, $S(\lambda)$ es conocida como densidad de columna y el índice p indica que la radiación se recibe después de que la fuente $I_S(\lambda)$ atraviesa la pluma (La figura 2.2 indica los puntos en los cuales un detector recibiría $I_0(\lambda)$ e $I_p(\lambda)$ si se ubicara en esas posiciones). La Ec. (1.7) considera la extinción como la suma de dos procesos: absorción y esparcimiento, por tanto es una expresión mas general para la Ec. (2.2).

La densidad de columna $S(\lambda)$ está dada por la integral de la concentración N de SO_2 a lo largo del camino óptico L .

$$S(\lambda) = \int_L N(x)dx \tag{2.3}$$

Idealmente L es la línea de visión que pasa a través de la pluma, pero en la realidad es distinta, y depende de factores como el esparcimiento por aerosoles, variaciones en la concentración de SO_2 y distancia de medición. Mas abajo se atenderán estas cuestiones y se propondrán las modificaciones a las ecuaciones que describen mejor las mediciones.

En condiciones ideales y con un instrumento que mida la profundidad óptica $\tau(\lambda)$ o la absorbancia $\tau(\lambda)/\ln 10$ en todo el espectro, se obtendría el valor de $k_\alpha(\lambda) \cdot S(\lambda)$ y por consiguiente, si se conocen los valores de $k_\alpha(\lambda)$, se puede estimar un valor de la densidad de columna S (en cada longitud de onda)

$$\tau(\lambda) = -\ln\left(\frac{I_p(\lambda)}{I_0(\lambda)}\right) = k_\alpha(\lambda) \cdot S(\lambda) \tag{2.4}$$

Sin embargo, la cámara no es capaz de dar la información espectral de intensidades como se describe en la Ec. (2.4), el sensor mide en su lugar, una suma sobre todo el espectro de radiación que le llega. Un esquema de un pixel en el sensor de la cámara y un filtro se bosqueja en la Figura 2.2, al sensor llega radiación con frecuencias seleccionadas por el filtro. Las primeras configuraciones para la cámara, como en [22], se construían con solo un filtro.

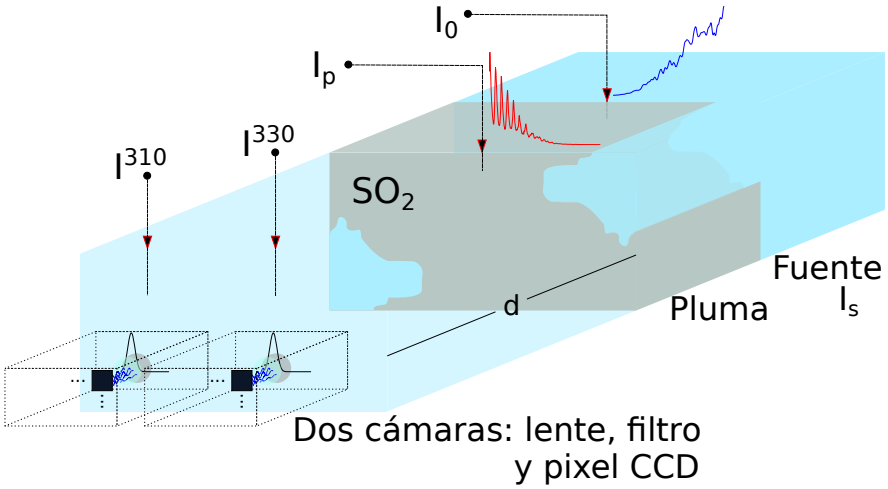


Figura 2.2: Proceso de medición: un sistema compuesto por dos cámaras adquiere un par de imágenes desde una distancia d de la pluma. En el campo de visión de cada cámara hay regiones en las que se mide la radiación difusa del cielo y regiones en las que la radiación difusa ha sido absorbida por las moléculas de SO_2 contenidas en la pluma volcánica. Las dos cámaras son representadas con cajas punteadas, cada cámara tiene un filtro. La intensidad que llega a cada pixel físico en el sensor CCD es la suma sobre un rango de longitudes de onda determinado por el filtro pasa bandas, a diferencia de un espectrómetro que permite descomponer un haz en sus longitudes de onda constituyentes.

Una cámara con un filtro centrado en 310 nm que toma imágenes en la misma dirección, primero con la radiación difusa como fuente atravesando la pluma y luego solo de la radiación difusa detrás de la pluma, medirá las intensidades integradas I_p^{310} e I_0^{310} (Nótese que a partir de ahora estas cantidades no dependen de una sola longitud de onda, sino de la suma de un

rango determinado por el filtro) respectivamente

$$I_p^{310} = \int_\lambda I_p(\lambda)\,\mathrm{d}\lambda \tag{2.5}$$

$$I_0^{310} = \int_\lambda I_0(\lambda)\,\mathrm{d}\lambda \tag{2.6}$$

Con estas cantidades definimos el peso promedio de la densidad óptica como

$$\hat{\tau}^{310} = -\ln\left(\frac{I_p^{310}}{I_0^{310}}\right) \tag{2.7}$$

Si el rango de transmitancia del filtro es lo suficientemente pequeño, la sección eficaz de absorción k_α y la densidad de columna S pueden ser considerados independientes. Entonces

$$\hat{\tau} = -\ln\left(\frac{I_0 \cdot \exp(-k_\alpha \cdot S)}{I_0}\right) = k_\alpha \cdot S \tag{2.8}$$

$$\Rightarrow \tau \approx k_\alpha \cdot S \tag{2.9}$$

La sección eficaz de absorción del dióxido de azufre k_α se obtiene de la literatura, de mediciones en el laboratorio, o incluso de calibraciones en el campo con DOAS y celdas que contienen concentraciones conocidas de SO_2 [3]. Una vez obtenida k_α, ésta es usada en los cálculos y procesamiento de imágenes de la cámara. Con esto se obtiene una buena aproximación a la densidad de columna S en cada pixel, este es uno de los objetivos para el software diseñado en este trabajo.

2.2.1. Corrección del efecto de esparcimiento en la pluma

Es muy importante que el SO_2 sea el compuesto con mayor contribución en absorción en la región de frecuencias transmitidas por el filtro de 310 nm. La pluma no está compuesta únicamente por moléculas de SO_2, y no sólo la absorción será el fenómeno relevante, también se presentarán esparcimientos. La mayor parte de los esparcimientos son causadas por aerosoles, éstos esparcimientos ocasionarán que las columnas de dióxido de azufre S estimadas no sean confiables.

Para contrarrestar el efecto de esparcimiento por gases y aerosoles contenidos en la pluma se hace una normalización de las intensidades I_0^{310}, I_p^{310}, para esto se utiliza un segundo filtro centrado en 330 nm, donde la absorción por el dióxido de azufre es despreciable (ver Figura 2.1).

El cociente de intensidades I_0^{310}/I_p^{310} es normalizado por el cociente de intensidades medidas con el filtro 330: I_0^{330}/I_p^{330}. Si la concentración y efecto de aerosoles en la pluma es baja $I_0^{330}/I_p^{330} \approx 1$, se tendrá

$$\left(\frac{I_0^{310}}{I_p^{310}} \Big/ \frac{I_0^{330}}{I_p^{330}}\right) \approx \frac{I_0^{310}}{I_p^{310}}$$

de otra forma, si I_0^{330}/I_p^{330} es muy distinto de 1, la normalización hará una corrección mas relevante.

El uso del filtro de 330 nm para la corrección se justifica si los valores de I_p^{310} y I_0^{310} se deben a la absorción por moléculas de dióxido de azufre y a la atenuación por los aerosoles, mientras que las intensidades I_p^{330} e I_0^{330} solo detectan la atenuación de los aerosoles [23]. La absorbancia por las moléculas de SO_2 a 330 nm es casi nula (Ver Figura 2.1) entonces el primer requisito se cumple. Para la segunda condición se considera que la atenuación debido a los aerosoles cambia poco en el rango de 310 a 330 nm. Para cuantificar la variación del efecto de aerosoles en este rango conviene revisar los coeficientes de esparcimiento Rayleigh σ_R y Mie σ_M, y aproximar cada coeficiente como $\sigma_R(\lambda) = \sigma_{R,0}\lambda^{-4}$ y $\sigma_M(\lambda) = \sigma_{M,0}\lambda^{-1}$, luego evaluando los cocientes en $\lambda_1 = 310$ y $\lambda_2 = 330$ [3]

$$\frac{\sigma_R(\lambda_1)}{\sigma_R(\lambda_2)} = \left(\frac{310}{330}\right)^{-4} \tag{2.10}$$

en el caso del esparcimiento Mie

$$\frac{\sigma_M(\lambda_1)}{\sigma_M(\lambda_2)} = \left(\frac{310}{330}\right)^{-1} \tag{2.11}$$

El cambio en los coeficientes de esparcimiento entre las longitudes de onda 310 y 330 significa que la profundidad óptica medida en estas dos longitudes de onda si cambiará por un factor pequeño. Puede decirse que la banda de extinción de los aerosoles es ancha o que tiene una dependencia débil con el cambio de longitudes de onda, en comparación a las bandas de absorción para SO_2 (ver sección eficaz de SO_2 en la Fig. 2.1) las cuales son estrechas y por tanto si tienen una alta susceptibilidad al cambio de longitudes de onda.

Las intensidades con la pluma I_p^{330} y sin la pluma I_0^{330} deben tomarse idealmente con el mismo campo de visión usado para el filtro de 310 nm. Los puntos de medición I_p, I_0 están señalados en la Figura 2.2.

Con la normalización de intensidades se define la absorbancia aparente AA, su significado físico es equivalente al de la profundidad óptica, excepto que la AA ha sido corregida de la contribución del esparcimiento por aerosoles contenidos en los gases del volcán.

$$AA = \ln\left(\frac{I_0^{310}}{I_p^{310}} \Big/ \frac{I_0^{330}}{I_p^{330}}\right) \tag{2.12}$$

La absorbancia aparente puede reescribirse en términos de la absorbancia para el filtro 330 $\hat{\tau}^{330}$ y de la absorbancia obtenida con el filtro de 310 nm $\hat{\tau}^{310}$ definida anteriormente en la (2.7)

$$\begin{aligned} AA &= \hat{\tau}^{310} - \hat{\tau}^{330} \\ &= \ln\left(\frac{I_0^{310}/I_p^{310}}{I_0^{330}/I_p^{330}}\right) \\ &= \ln\left(\frac{I_0^{310} I_p^{330}}{I_0^{330} I_p^{310}}\right) \end{aligned} \tag{2.13}$$

La expresión de la absorbancia aparente se reordena para ajustarse a la metodología sugerida en [4], tal que los cálculos y el código en el software se identifiquen de forma paralela.

$$AA = \ln\left(\frac{I_p^{330}}{I_p^{310}}\right) + \ln\left(\frac{I_0^{310}}{I_0^{330}}\right) = \ln\left(\frac{I_p^{330}}{I_p^{310}}\right) - \ln\left(\frac{I_0^{330}}{I_0^{310}}\right) \tag{2.14}$$

$$AA = \ln\left(\frac{I_p^{330}}{I_p^{310}}\right) - \ln B \tag{2.15}$$

2.2.2. Corrección del esparcimiento en la atmósfera

Medir con dos filtros corrige a primer orden el efecto de esparcimiento de la luz por los aerosoles dentro de la pluma. Hasta este momento una medición con la cámara y los dos filtros será satisfactoria si consideramos que en la distancia d recorrida por la radiación desde la frontera de la pluma

hasta la cámara (dentro de su campo de visión) no ocurren mas procesos de absorción ni de esparcimiento, o al menos no que afecten las longitudes de onda detectadas en la cámara. Este caso hipotético es equivalente a decir que las mediciones con la cámara se hacen a distancias muy cortas de la pluma $d \approx 0$, porque de esta manera se asegura que no aparecerán mas fuentes de esparcimiento o absorción que interfieran. Considerar distancias cortas no es deseable por razones de seguridad, sería ineficiente reducir el área en el campo de visión y eliminaría el potencial de la cámara como instrumento de mediciones remotas.

Las mediciones se realizan a una distancia d desde la pluma, ver Figura 2.2. El espacio entre la pluma y la cámara no es translúcido y el esparcimiento de fotones en el campo de visión de la cámara induce una subestimación sistemática en las mediciones que se incrementa con la distancia d. Es un efecto similar al que se corrigió con la normalización de intensidades, pero esta vez ocurre en el campo de visión de la cámara y no en la pluma.

La subestimación varía con la distancia, la densidad de columna de SO_2, la presión atmosférica y turbicidad. El efecto es mas notorio para longitudes de onda cortas y a altas densidades de columna S porque hay mayor probabilidad de esparcimiento y la longitud de camino óptico aumenta.

Las técnicas utilizadas para cuantificar la dilución de luz producida en la atmósfera consisten en hacer mediciones a diferentes distancias y encontrar la relación entre la atenuación y la distancia. En [22] se hace una primera corrección midiendo 4 distancias, la desventaja es que esta técnica supone que la tasa de emisión y el esparcimiento en la atmósfera no cambian en el lapso en el que se toman las series de mediciones en distintos lugares (orden de horas). Otras técnicas mas complejas y costosas incluyen mediciones a diferentes distancias usando un espectrómetro UV sobre un avión [24].

En [4] se hace una propuesta para hacer una corrección. Consiste en obtener los coeficientes de esparcimiento directamente en las dos imágenes obtenidas con los dos filtros. Durante el procesamiento de imágenes se adquiere un perfil de intensidades trazando una línea sobre pixeles que corresponden a una sección del edificio volcánico. Por esta razón para obtener las imágenes en el campo, se busca que una parte de la ladera del volcán sea visible, y que esa sección presente la menor cantidad posible de irregularidades en el terreno. Se elige este método porque la atenuación de intensidades en el perfil trazado depende de la distancia, y las distancias sobre la línea de pixeles pueden ser aproximadas si se conocen las distancias al punto inicial y final. Para recuperar los coeficientes σ el paso siguiente es hacer un ajuste

de los valores de intensidad y distancias al modelo siguiente

$$I(\lambda) = I_0(\lambda)\left(e^{-\sigma(\lambda)d}\right) + I_A(\lambda)\left(1 - e^{-\sigma(\lambda)d}\right) \tag{2.16}$$

La Ec. (2.16) describe la atenuación en función de la distancia, y es aplicable a una atmósfera que esparce fotones. La radiancia $I(\lambda)$ se mide a una distancia d de la fuente u objeto que irradia con intensidad $I_0(\lambda)$. El primer término en la Ecuación (2.16) supone que una fracción de $I_0(\lambda)$ se esparce en direcciones fuera del campo de visión (CV) de la cámara, el segundo término expresa la intensidad de luz esparcida en el ambiente I_A que entra al CV del instrumento, modificando la medición de $I(\lambda)$. σ es el coeficiente de esparcimiento en la atmósfera, que cambia en función de la longitud de onda λ, de la presión atmosférica y de la concentración de aerosoles. El coeficiente σ es el resultado de una combinación de esparcimiento Mie y Rayleigh, el primero con una débil dependencia con la longitud de onda y el segundo con una dependencia en λ^{-4}. Considerando que la fuente para I_A está muy lejana ($d \to \infty$), entonces I_A es aproximada con la radiancia del cielo en la dirección de medición. I_0 es la intensidad inicial, la que se mediría frente a la pluma en ($d = 0$).

2.2.3. Otras consideraciones en las mediciones

Kern et al. [2] hizo otras observaciones, de efectos que conducen a mediciones con errores sistemáticos

- El camino óptico que recorre la luz solar varía con el ángulo solar cenital (ASC), si el ASC crece, también crece la longitud de camino óptico porque debe atravesar una sección mas ancha de la capa de ozono, causando que la intensidad en el espectro a longitudes cortas decrezca. Ver Figura 2.3.

- La variación de la función de transmitancia en el filtro depende del ángulo de iluminación y de la longitud de onda. Mayores ángulos de iluminación respecto al centro del campo de visión y longitudes de onda cortas disminuyen la transmitancia, dando lugar a mayor sensibilidad en el centro de la imagen, y menor sensibilidad en los bordes. Ver Figura 2.4.

Para este trabajo la variación de la intensidad de radiación solar I_S con el ángulo solar cenital se tomó como despreciable, después de que algunas

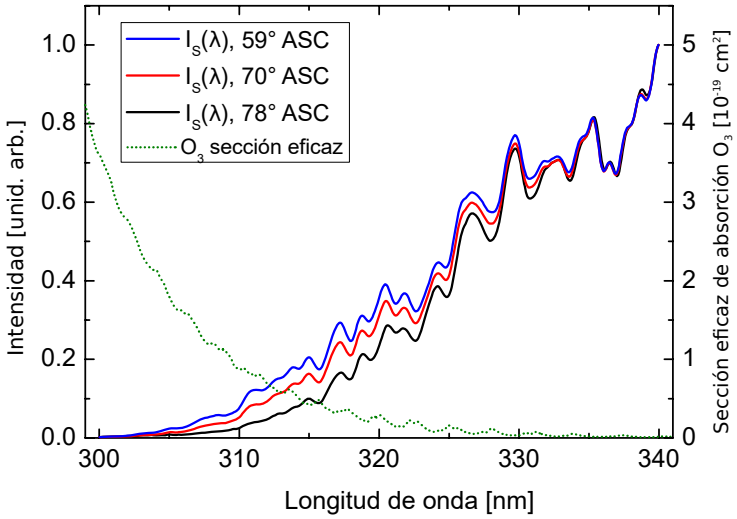


Figura 2.3: La intensidad de radiación solar I_S depende del ángulo solar cenital. Gráfica recuperada y adaptada de [2].

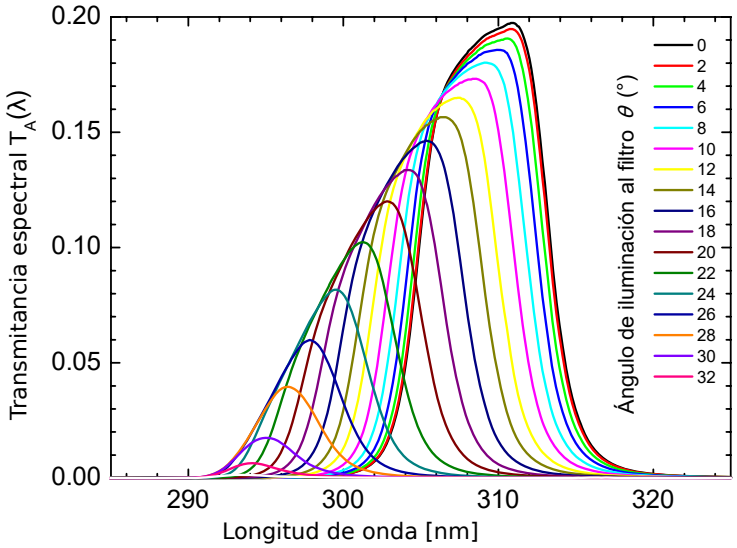


Figura 2.4: Dependencia de la transmitancia en el filtro con el ángulo de iluminación, gráfica adaptada de [2].

mediciones en un rango de ASCs cercanos (usadas en mediciones típicas) arrojaran intensidades similares. La gráfica en 2.5 muestra la variación del

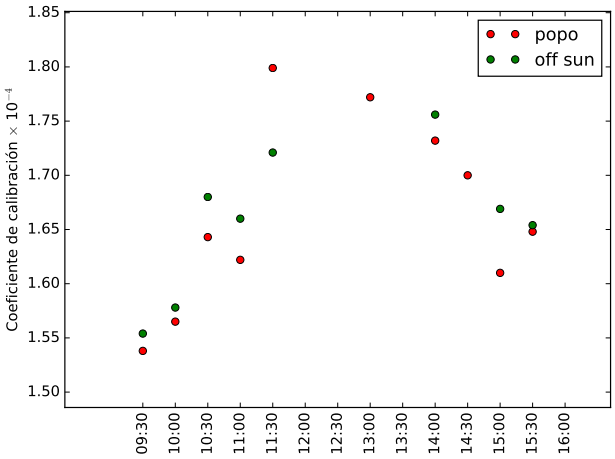


Figura 2.5: La variación del ángulo solar cenital ASC durante el día implica una variación al coeficiente de calibración, este cambia en menos de 10 % en una hora.

coeficiente de calibración a distintas horas del día. La variación de sensibilidad en la imagen debida a la configuración del filtro y arreglo óptico, fue solucionada haciendo una normalización, esto se revisa en la Subsección 3.2.4.

El conjunto de expresiones en ((2.13) – (2.15)) describen la absorbancia aparente para un pixel. La cámara cuenta con 512×512 pixeles, lo que produce una matriz de intensidades. Las expresiones siguen siendo válidas, solo se debe tomar en cuenta que el cálculo se hace para cada uno de los pixeles, por tanto las operaciones con las matrices se hacen entrada a entrada. Si se utiliza la notación $\mathbf{M} = (M_{ij})$ para un arreglo bidimensional, la absorbancia aparente en términos de las matrices correspondientes a cada imagen se convierte en

$$\mathbf{AA} = \ln\left(\frac{\mathbf{I}_p^{330}}{\mathbf{I}_p^{310}}\right) - \ln\mathbf{B} \tag{2.17}$$

Se ha elegido esta expresión porque con el software se extraen regiones de cielo originadas con las mediciones de los dos filtros, se calcula el cociente de las matrices entrada a entrada y se hace una interpolación 2D para obtener $\mathbf{B} = \mathbf{I}_0^{330}/\mathbf{I}_0^{310}$.

Con la cantidad de datos a procesar se hace evidente la necesidad de hacer los cálculos con una computadora.

2.3. Diseño del instrumento

Los componentes principales de esta cámara son un filtro de interferencia pasa bandas y un detector CCD bidimensional sensible a la luz UV.

El diseño del instrumento influye en el desempeño de la cámara, Mori y Burton [1], los pioneros de la cámara detectaron la necesidad de usar dos filtros y así minimizar los efectos de los aerosoles.

Para este trabajo se procesaron imágenes de una cámara con filtros de 310 y 330 nm, esta cámara es utilizada en [4].

La cámara SO_2 usada está compuesta de 2 cámaras modelo Apogee Alta U260, con 512×512 pixeles de $26\mu m$. Cada cámara es equipada con lentes silica PentaxB2528-UV. La longitud focal es de 25 mm. Rendijas con filtros pasa bandas centradas en 310 y 330 nm (Un filtro a cada cámara). Los filtros son Asahi XBPA310 y ZBPA330, con FWHM de 10 nm y pico de transmitancia de 0.7.

El campo de visión de la cámara es de 23^o y junto a las cámaras también está implementado un DOAS Avantes 2048. El filtro de 310 nm es sensible a la absorción de SO_2 y a la atenuación debida a los aerosoles, mientras que el filtro de 330 nm solo es sensible a la atenuación por aerosoles. Las imágenes se toman al mismo tiempo, para calcular la absorbancia se usan las dos imágenes y se elimina gran parte del efecto de atenuación por los aerosoles. Aun así persiste la subestimación por la no idealidad del camino óptico, originada por el esparcimiento Mie y Rayleigh.

Nuevos métodos para la detección de gases volcánicos siguen desarrollándose, por ejemplo se está diseñando una cámara que mapea las distribuciones de SO_2 con un Fabry-Perot [25] mejorando las debilidades de la cámara SO_2. O bien existen otros instrumentos que pueden medir otros gases.

Capítulo 3

Análisis y procesamiento de imágenes

En las siguientes secciones se describe el procesamiento de las imágenes, para organizar este capítulo, se divide en cinco puntos

- En la primera parte Sección 3.1 se hace un resumen del procesamiento, tomando como guía un diagrama de flujo. También se presenta la interfaz gráfica y parámetros que verá un usuario del programa.
- La descripción de las imágenes crudas de la cámara y su preparación antes del procesamiento se explica en la Sección 3.2.
- Para identificar las regiones de la pluma, el cielo y el volcán se utiliza una secuencia de absorbancias. La metodología para lograr este paso se revisa en la Sección 3.3.
- La implementación de los métodos de corrección, y distribuciones de las concentraciones de SO_2 se detalla en la Sección 3.4.
- Por último en la Sección 3.5 se revisa el método para obtener el campo de velocidades y el cálculo del flujo.

.

3.1. Diagrama de flujo

El diagrama de flujo en la Figura 3.1 es una idea general del funcionamiento del programa. Una de las primeras cosas que salta a la vista es que el procesamiento de las imágenes se encuentra dentro de un ciclo.

Antes de iniciar el procesamiento, el programa pide al usuario especificar las rutas de las carpetas que contienen los archivos que se van a procesar. Después de que el programa ha reconocido que hay suficientes archivos, se pide al usuario algunos parámetros que dependen de cada volcán y las condiciones de la medición. Para el caso de una estación de medición fija, los parámetros no deberían cambiar, por eso el programa tiene incluido un archivo de configuración donde pueden introducirse los parámetros que se usarán por defecto.

Con la lista de archivos y parámetros cargados, comienza el procesamiento. El ciclo proporciona un índice p que hace recorrer secuencias con todas las imágenes disponibles en un directorio. Cada secuencia lleva un preprocesamiento que incluye eliminación de líneas de pixeles sin información, rotaciones, desplazamientos para emparejar los pares de imágenes tomados con los filtros de 310 y 330 nm, y otras calibraciones. Una secuencia de imágenes con este preprocesamiento está lista para generar una segunda secuencia, esta vez de absorbancias, las cuales son usadas para localizar las regiones de cielo, pluma y edificio volcánico. Sobre cada una de estas regiones hay información que es aprovechada para recuperar los coeficientes de esparcimiento, corregir la dilución de luz y obtener las densidades de columna de SO_2.

Una vez que se tienen disponibles las distribuciones de las densidades de columna, es posible usar un par de imágenes (selecionadas de una secuencia) para calcular el campo de velocidades. Aparecerán vectores erráticos asociados a pixeles, especialmente en los bordes de la imágen, si estos vectores se usan interferirán negativamente al cálculo del flujo, entonces es necesario usar una máscara que considere solo los vectores asociados a los valores de la pluma volcánica.

Con la selección de la región en el campo vectorial de velocidades y la distribución de concentraciones se estima el flujo de SO_2. Finalmente se genera una animación GIF, que sirve para visualizar los campos de velocidades y concentraciones procesados por el programa. Puede usarse como una forma rápida de evaluar superficialmente el desempeño del programa.

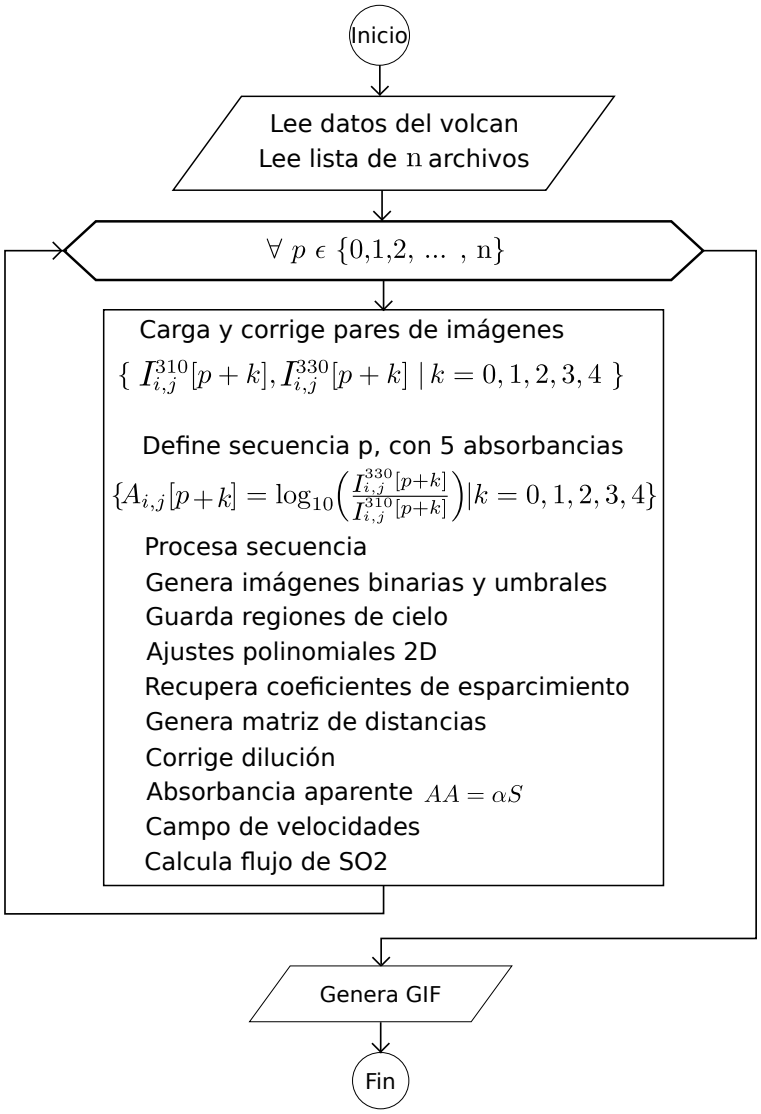


Figura 3.1: Diagrama de flujo, funcionamiento global del programa, el rectángulo contiene los cálculos y procedimientos que se realizan sobre cada secuencia de cinco imágenes.

3.1.1. Interfaz de inicio

El usuario proporciona las rutas, datos del volcán, parámetros para calcular el flujo y parámetros para el comportamiento general del programa por

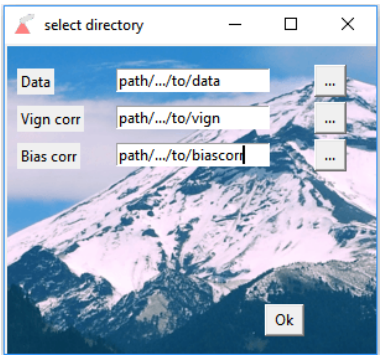


Figura 3.2: Interfaz gráfica creada con el módulo Tkinter.

medio de una interfaz gráfica (ver Fig. 3.2). Se utiliza la biblioteca TKinter de PYTHON para implementarla.

La interfaz modifica el archivo *config.ini*, que también puede ser modificado directamente con un editor de textos. Con las rutas especificadas, el programa busca los archivos, enumera los archivos disponibles y guarda en una lista los nombres de todos ellos. El archivo de configuración *config.ini* debe ubicarse en el mismo directorio que el script principal del programa. Un ejemplo del contenido de *config.ini* es el siguiente

```
[varPaths]
data = C:/Path/to/.../data
vign = C:/Path/to/.../VCorrection
bias = C:/Path/to/.../BiasCorrection

[datosVolcan]
distance cam-plume = 2500
ang fov-plume = 30
profile d0 = 1500
profile d1 = 2000

[parametrosOpticalFlow]
pyr_scale = 0.5
levels = 4
winsize = 10
iterations = 10
poly_n = 7
```

```
poly_sigma = 1.5
flux_conversion_factor = 2.66e-06
pixel_size = 1.02
```

Los directorios data, VCorrection y BiasCorrection contienen el conjunto de archivos raw y txt (Fig. 3.3), un par de imágenes de calibración para corregir el efecto de Vignetting, y otro par usado para contrarrestar el desplazamiento (en inglés se le llama *Bias*) de las intensidades debida a las fluctuaciones que ocurren en la electrónica del sensor CCD. Todos los parámetros son revisados en las siguientes secciones.

3.2. Preparación de las imágenes

3.2.1. Las imágenes obtenidas de la cámara

La cámara es configurada para adquirir un par de imágenes con tiempos de exposición e intervalos de tiempo fijo, ajustados de acuerdo a las condiciones climáticas y factores como la velocidad del viento en el día de la medición. Si la velocidad del viento es alta se requiere tomar mediciones a intervalos de tiempo mas cortos.

La resolución temporal típica de la cámara para una velocidad de viento de alrededor de 5 m/s, es de 0.1 Hz. Es decir cada 10 segundos se adquiere el siguiente conjunto de archivos (Ver Figura 3.3).

Figura 3.3: Archivos obtenidos en una sola medición de la cámara.

- Una imagen de tipo *raw* a 310 nm, nombrada con el formato de tiempo epoch.[1]

[1]El tiempo Epoch es la hora actual medida en número de segundos desde el Epoch Unix. Epoch 0 es enero 1 1970 00:00:00

- Una imagen de tipo *raw* a 330 nm, igual que el archivo para 310, ocupa un espacio de 512Kb y está formado de 512×512 datos divididos en 8 columnas, es decir 3768 líneas. Cada dato esta representado por números enteros sin signo en el sistema hexadecimal (base 16), esta es la razón por la que en PYTHON se lee con el tipo de dato `uint16`.
- Un archivo de texto con especificaciones de tiempos de exposición y temperaturas en las dos cámaras.

3.2.2. Estructura de una secuencia

En PYTHON pueden declararse objetos llamados *data type* que sirven para definir estructuras de arreglos, especificar sus nombres y tipos de datos que la componen. En el programa se define una estructura 'pairtype' que contiene el tiempo, matrices correspondientes a las imágenes de 310 y 330, y los tiempos de exposición.

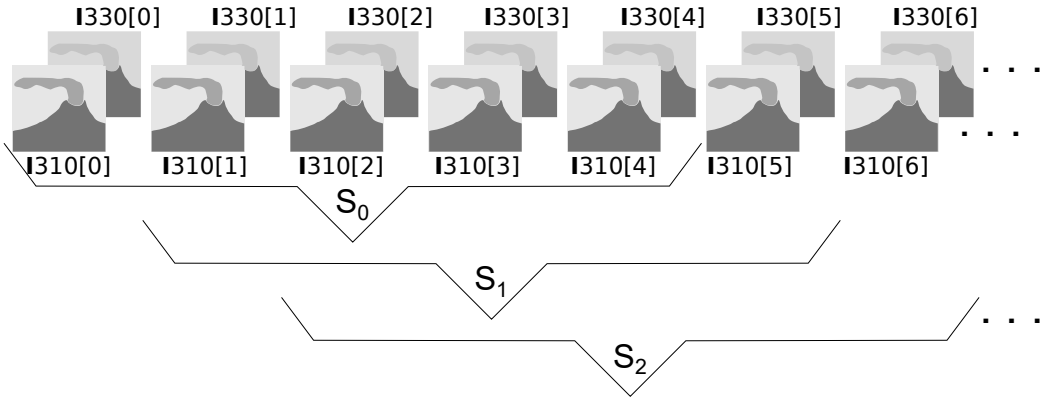


Figura 3.4: Secuencias S_i, el índice i se recorre con un ciclo for, hasta cubrir todas las secuencias disponibles.

En cada iteración del ciclo *for* se generan nuevas secuencias S_i de 5 pares de imágenes, cada par (**I**310 e **I**330) va acompañada del tiempo epoch, y tiempos de exposición formando una estructura 'pairtype'.

El índice i del ciclo *for* recorre todas las imágenes contenidas en el directorio *data*. Por ejemplo, si el directorio contiene 7 pares de imágenes, pueden generarse tres secuencias S_0, S_1 y S_2 (ver Figura 3.4), de manera que cada

secuencia tenga cinco pares y se recorran todas las imágenes contenidas en el directorio *data*.

A partir de cada secuencia generada se obtiene una secuencia de absorción, esta es con la que se hacen los cálculos para identificar umbrales de detección del cielo, edificio volcánico y pluma, los cálculos se detallan en la siguientes secciones.

3.2.3. Preparando las imágenes para llenar una secuencia

Cinco estructuras de datos tipo 'pairtype' forman una secuencia, y cada estructura se llena con la información de la hora en que fue tomada, tiempos de exposición y el par de imágenes a 310 y 330 nm, el problema es que al desplegar un par de imágenes, tal como sale de la cámara (archivos *raw*, ver Figura 3.5) se observan características que provocarían que los cálculos no fueran fiables. Estos efectos son causados por la configuración y electrónica del hardware en la cámara.

- Las imágenes se encuentran rotadas en distintos sentidos, debido a la configuración inicial del hardware,
- aparece el efecto de borde vignetting causado por la disminución en la transmisión de los filtros y lentes para ángulos de incidencia elevados,
- hay líneas verticales obscuras de pixeles a los lados (probablemente originado por un defecto en los sensores),
- si superponemos un par de imágenes resulta que no están emparejados, tienen un desplazamiento (*offset*) que debe ser eliminado, con la extracción de regiones con coordenadas apropiadas, el origen del offset se debe a un ligero desalineamiento en las cámaras.
- Otro efecto que no es apreciable en las imágenes, pero es importante, es el desplazamiento 'bias' de los valores de intensidad.

La Figura 3.5 muestra una versión que simplifica algunos rasgos característicos de las imágenes 310 y 330 nm. Sin aplicar algún tipo de procesamiento, esto es lo que se tendría al visualizarlas con un software que permita leer archivos '*.raw'.

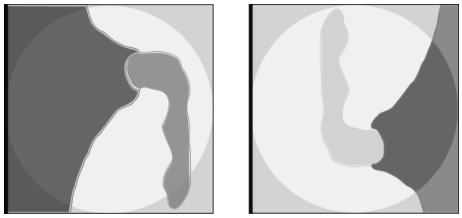


Figura 3.5: Simplificación de las imágenes $\mathbf{I}^{310}$ e $\mathbf{I}^{330}$ arrojadas por la cámara SO_2 (Aún no están preparadas para formar una secuencia).

Las imágenes reales, internamente son matrices de 512×512 entradas, correspondientes a pixeles que tienen valores enteros de intensidad en un intervalo de 0 a $2^{16} - 1$. En PYTHON se utilizaron las funciones[2] *fromfile*, *contourf* e *imshow* para su visualización.

En escala de grises, los valores de intensidad altos son mas claros, y el valor de intensidad cero es negro. En la comparación de imágenes a 310 y 330 nm (Fig. 3.5) se puede identificar el cielo, que está detrás del edificio volcánico, los gases volcánicos (pluma) y edificio volcánico. Nótese que hay una mayor intensidad en las regiones que pertenecen al cielo, (regiones libres de pluma), esto es porque en esas regiones la concentración de SO_2 es nula o despreciable. La pluma en la imagen 310 es prominente respecto al cielo, mientras que en la imagen 330 las intensidades de la pluma y cielo son más cercanas. Esto último se explica porque en longitudes de onda de 330 nm, las absorción de luz por la pluma es baja, entonces la pluma actúa mas como si fuera un medio transparente.

Un par de imágenes debe ser preparado antes de llenar la estructura 'pairtype'. En la Figura 3.6 se representa una imagen tomada con el filtro de 310 nm cargada al programa y desplegada sin ninguna corrección, y a su derecha la imágen corregida, que es la que terminará formando parte de una estructura de datos 'pairtype' y por lo tanto de una secuencia.

Que ambas imágenes $\mathbf{I}^{310}$ e $\mathbf{I}^{330}$ sean transformadas, como en la Figura 3.6, implica varias operaciones matriciales. Los pasos en la preparación o preprocesamiento de imágenes se hacen explícitos a continuación con las Figuras 3.7, 3.8.

En el paso cero (Figura 3.7), el par de imágenes se carga al programa, al desplegarlas se hicieron visibles líneas obscuras. Estas líneas verticales de pixeles malas se presentaban para cada par de imágenes en las mismas

[2]Disponibles en PYTHON después de importar los módulos numpy y matplotlib.

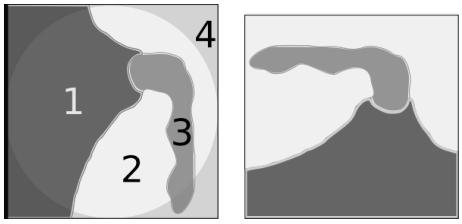


Figura 3.6: Una imágen cruda $\mathbf{I}^{310}$ simplificada, En la izquierda se identifican las regiones y efectos de una imagen sin correciones: 1) Edificio volcánico, 2) región de cielo, 3) gases volcánicos o pluma, 4) efecto de borde vignetting. En la derecha: la imagen ha sido rotada, líneas de pixeles innecesarias fueron eliminadas, la imágen se recortó y se aplicaron correcciones de vignetting y desplazamiento 'bias'; entonces la imágen de la derecha ya está preparada para estar dentro de una secuencia.

posiciones. Probablemente sea un defecto en el detector de la cámara, estos pixeles no aportarían información, y conducirían a un error sistemático en la detección del offset. Entonces en el paso 1 se eliminaron 6 líneas de pixeles correspondientes a las zonas obscuras en cero.

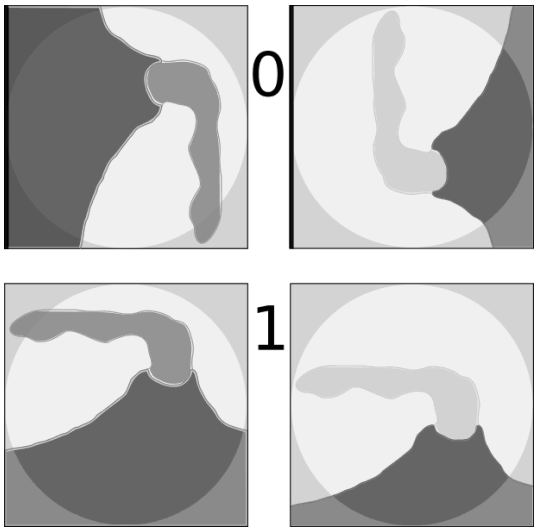


Figura 3.7: Operaciones: Transpuesta y rotación de matrices. Desaparecen líneas verticales de pixeles obscuras.

En el paso 2 (Figura 3.8) se ha corregido el efecto de borde vignetting y

bias (Subsección 3.2.4), luego en el paso 3 se recortan las regiones de forma que si se sobreponen una sobre otra (Subsección 3.2.5) las siluetas visibles del volcán coinciden.

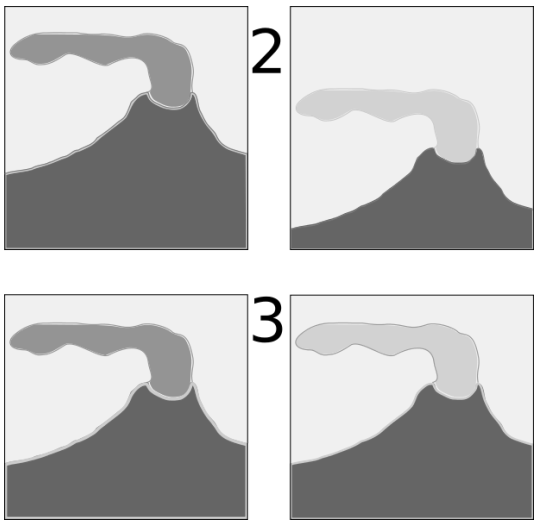


Figura 3.8: Vignetting y bias corregidos. Emparejando las imágenes con matchmatrix.

3.2.4. Corrección de vignetting y bias

El método usado para eliminar el efecto de borde vignetting consistió en obtener imágenes del cielo con la cámara SO_2. Se hizo rotar la cámara sobre un eje, tomando imágenes cada 5 grados. Luego se obtuvieron matrices promedio normalizadas para las imágenes a 310 nm y 330 nm. Puesto que se usa la misma cámara con la que se obtienen imágenes de la pluma, las imágenes promedio deben ser sometidas al mismo procedimiento de enderezar, quitar líneas de pixeles defectuosos y obtener segmentos de imagen apropiados, de manera que las siluetas del volcán y pluma en la imagen 310 nm coincidan en posición con las respectivas siluetas en la imagen de 330 nm. El código escrito en PYTHON para obtener las imágenes promedio se encuentra en el Apéndice C. Análogamente se producen imágenes de calibración bias, con un promedio de imágenes tomadas con la cámara impidiendo la entrada de luz.

En el programa principal, se utilizan las dos imágenes promedio para corregir las imágenes originales, la expresion que se utiliza para obtener las imágenes de calibración de vignetting es la siguiente

$$\mathbf{M} = \frac{\mathbf{M}_{\text{vigntt}} \cdot \max(\mathbf{M}_{\text{avg}})}{\mathbf{M}_{\text{avg}}} = \frac{\mathbf{M}_{\text{vigntt}}}{\hat{\mathbf{M}}_{\text{avg}}} \tag{3.1}$$

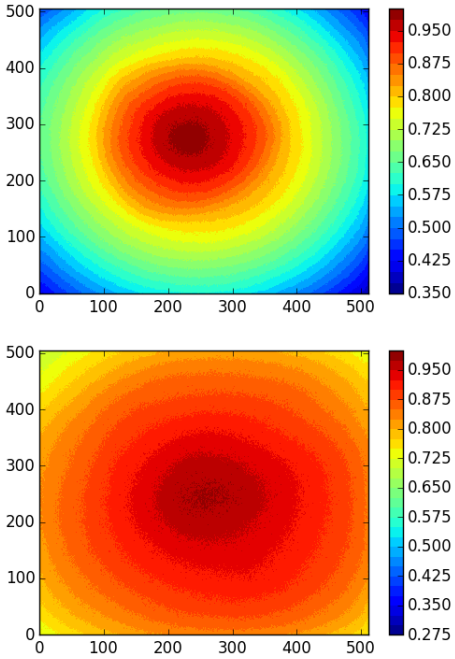


Figura 3.9: Imágenes con efecto vignetting normalizadas, en los ejes se especifican las filas y columnas de pixeles.

3.2.5. Emparejando los pixeles en un par de imágenes

El objetivo de esta función es encontrar el offset o desplazamiento en un par de imágenes. El campo visual de las dos cámaras no es el mismo, entonces se requiere emparejarlas, es decir buscar un desplazamiento horizontal y vertical que haga que una de las imágenes se pueda sobreponer en la otra, y

que los pixeles coincidan en el mismo campo visual. La función matchMatrix consiste en un algoritmo que identifica rasgos prominentes de las imágenes, y busca la mejor coincidencia haciendo diferencias con una región fija de la imágen.

3.3. Identificando el cielo y la pluma volcánica

Identificar una porción de la imagen que pertenece al cielo es importante para obtener el término $\ln \mathbf{B}$ en la Ecuación (2.17). Con la porción de cielo recuperada se busca simular todo el cielo, incluyendo lo que está detrás de la pluma y del edificio volcánico. Otro beneficio de obtener la región del cielo es que con ella se puede obtener un promedio de los valores de intensidad y usarlos en la Ecuación (2.16). Este apartado describe como se construyen nuevas imágenes que resaltan las regiones de cielo y pluma que posteriormente se analizan con histogramas. Se ha detectado un patrón que sigue las imágenes y con esta información se calculan umbrales de forma automática, así en cada secuencia se obtienen regiones de cielo actualizados. Anteriormente se tenía un programa (utilizado en [4]) en el que la selección del cielo era manual, y dependía del criterio del usuario seleccionar solo una región rectangular de cielo (otra desventaja superada).

Para identificar la pluma se generará una nueva secuencia de absorbancias $\mathbf{A}$, a partir de las secuencias de pares de imágenes 'pairtype', preparados como se describió en la sección anterior. En cada iteración del ciclo *for* (en el diagrama de flujo en Fig. 3.1) se obtiene una nueva secuencia $\mathbf{A}$, con la que se detectarán la pluma, el cielo y el edificio volcánico.

3.3.1. Cálculo de la absorbancia

La (3.2) expresa una secuencia de 5 imágenes de absorción generada por el programa a partir imágenes contenidas en alguna secuencia S_i (como las de la Figura 3.4). La Figura 3.10 es un ejemplo de una secuencia de absorbancias desplegada por el programa.

$$\mathbf{A}[k] = \log_{10} \frac{\mathbf{I}^{330}[k]}{\mathbf{I}^{310}[k]} \Big|_{k=0,1,2,3,4} \tag{3.2}$$

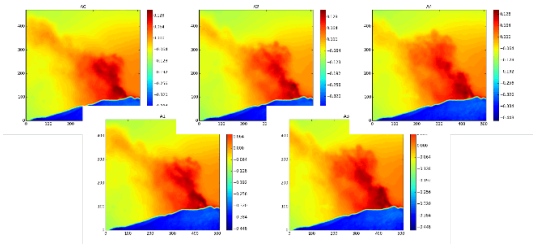


Figura 3.10: Una secuencia de absorción $\{\mathbf{A}[0], \mathbf{A}[1], \mathbf{A}[2], \mathbf{A}[3], \mathbf{A}[4]\}$

3.3.2. Timedif y Suma de absorbancias

De cada secuencia de absorción obtenemos tres imágenes nuevas, útiles en la automatización de la identificación de la pluma. Estas imágenes o matrices se guardan en las variables timeDiffAbso, sumAbso y sumI330, y son obtenidas respectivamente con las Ecs. (3.3), (3.4) y (3.5).

$$\text{timeDiffAbso} = \sum_{i=0}^{3} |\mathbf{A}[i] - \mathbf{A}[i+1]| \tag{3.3}$$

$$\text{sumAbso} = \sum_{i=0}^{4} \mathbf{A}[i] \tag{3.4}$$

$$\text{sumI330} = \sum_{i=0}^{4} \mathbf{I}^{330}[i] \tag{3.5}$$

El límite superior e inferior de las sumas son los que corresponden a los índices en las listas y arreglos de PYTHON (el primer elemento en una lista tiene índice cero).

Las sumas de arriba se obtienen con PYTHON utilizando el código siguiente:

```
sumabsorbancia = np.sum(absorbancia, axis=0)
timeDiffAbso=0
imagen330=0
for i in range(4):
    timeDiffAbso+=np.absolute(absorbancia[i]-absorbancia[i+1])
timeDiffAbso = cv2.blur(timeDiffAbso,(7,7))
for i in range(5):
    imagen330+=img330[i]
```

3.3.3. Histogramas

Como se revisó en el Capítulo 2 la cámara mide una sección del espectro UV, y las imágenes producidas son las intensidades provenientes de cada región del campo visual de la cámara.

Hay tres regiones que se pueden identificar en las imágenes, porque estas interactúan con la luz UV de formas diferentes. La radiación difusa en el cielo es considerada la fuente UV, y en las imágenes tendrán los valores de intensidad más altos. El edificio volcánico refleja muy poca cantidad de radiación (en la región UV la reflectividad típica del basalto es de 3% y la vegetación 5%) entonces en la imagen se tendrán valores más bajos de intensidad. La pluma volcánica con moléculas de SO_2 absorbe una cantidad de radiación UV, entonces en las imágenes se observará una 'sombra' que se desplaza en una sucesión de imágenes.

Conocer las coordenadas de los pixeles que pertenecen a cada una de las tres regiones fue uno de los principales objetivos de este trabajo, y su importancia radica en que manejar estas regiones por separado hace los cálculos para obtener las correcciones, distribuciones y flujo de SO_2 mas rápidos y robustos. La herramienta básica para lograr este objetivo, fue el uso de histogramas. Los histogramas permiten apreciar como se distribuyen los valores de intensidad, los valores de intensidad con frecuencia alta son los máximos de la distribución. Éstos máximos permiten diferenciar las regiones en la imágen.

Se obtuvieron los histogramas de las nuevas variables timeDiffAbso, sumAbso y sumI330 (definidas en la Subsección 3.3.2), y esto fue la clave, pues estas imágenes resaltan las características en las regiones de una imagen. timeDiffAbso es una imagen de una suma de diferencias entre absorbancias de una secuencia, los valores más altos corresponden a los lugares de la imagen donde hay mayor movimiento, mientras que valores de absorbancias altas debidas a las variaciones de intensidad del fondo permanecen idénticas en la secuencia. Las mediciones desde la cámara se toman desde una posición fija, entonces el movimiento que se observa es solo de la pluma. sumAbso es una imagen de la suma de las cinco absorbancias, esto remarca las zonas donde hay mayor absorbancia, es decir de la pluma.

La variable sumI330 es la suma de imágenes 330 de una secuencia, idealmente la absorción para esta imagen no debe ser observable, por lo tanto en un histograma de esta imagen se deben diferenciar dos picos anchos, los valores de intensidad que corresponden a estos picos se usan para diferenciar

el cielo y el volcán. El pico correspondiente a intensidades altas será de los pixeles del cielo, y el pico de intensidades bajas es por los pixeles del edificio volcánico.

Los histogramas de las imágenes timeDiffAbso y sumAbso se analizaron con un programa que se escribió para facilitar la visualización (Figura 3.11). El programa permite seleccionar dos umbrales en las intensidades (una cota para intensidad mínima y otra para la intensidad máxima) con esto se puede observar el histograma y al mismo tiempo la imagen que se está analizando muestra los pixeles con los valores de intensidad que se encuentran entre los dos umbrales. El programa sirvió para visualizar (Figura 3.11) y comparar los rasgos en las curvas generadas de los histogramas. Sirvió como una guía para el conjunto de reglas que se debía programar y lograr identificar cada patrón a partir de sus máximos y mínimos locales.

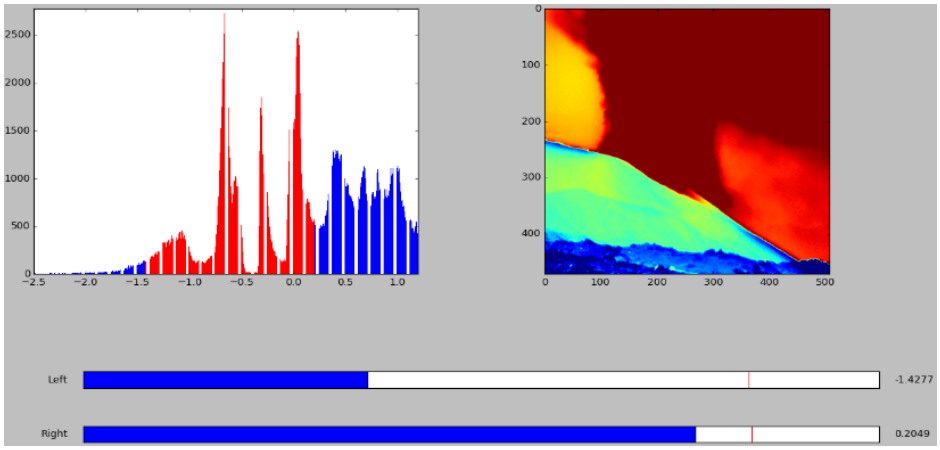


Figura 3.11: Programa escrito en PYTHON para visualizar y explorar distintos umbrales, las barras inferiores permiten acotar rangos de intensidades. Cada región en la imagen es dominado por un rango de intensidades en el histograma.

3.3.4. Umbrales

Las curvas definidas por los histogramas fueron suavisadas con el algoritmo de Savitzky-Golay filter y se obtuvieron las derivadas para extraer una lista de máximos locales, los algoritmos fueron adaptaciones de códigos encontrados en varias fuentes [26, 27, 28].

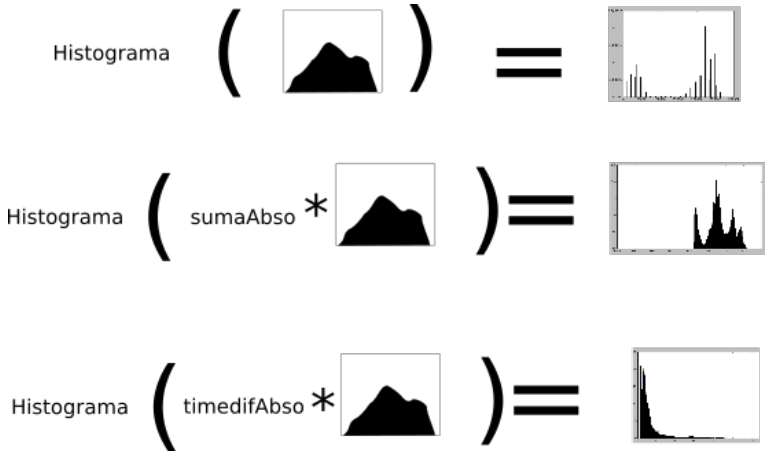


Figura 3.12: Los umbrales son obtenidos del análisis en los histogramas.

Máscara de volcán y cielo

De la suma de cinco imágenes preparadas **I**330, se obtuvo un histograma; la curva generada se suavizó con el fin de obtener los máximos locales mas prominentes. El valor del umbral `u_sumI330` es un punto medio localizando entre los dos máximos de intensidad correspondientes al cielo y al volcán.

Se genera una imagen binaria o máscara de volcán y cielo, una matriz del mismo tamaño que una imagen 310, solo que con entradas 1 y 0. Si `sumI330[i,j]` $>$ `u_sumI330[i,j]` el valor de `mask_sumI330[i,j]` es 1. Si `sumI330[i,j]` $\leq$ `u_sumI330[i,j]` el valor de `mask_sumI330[i,j]` es 0. Los valores de la imagen que pertenecen al volcán son ceros y los que pertenecen al cielo son unos.

sumAbso

Para la obtener el umbral de la imagen `sumAbso` se requiere un histograma del producto `mask_sumI330` $\times$ `sumAbso`. (Ver Figura 3.12).

Se crea una curva suavizando los valores del histograma y se obtienen los máximos locales, el primer máximo corresponderá a la intensidad umbral buscada.

timeDiffAbso

Para la obtener el umbral de la imagen `timeDiffAbso` se requiere un histograma del producto `mask_sumI330` × `timeDiffAbso`.

```
class Histograma:
    """Defino la clase Histograma, para obtener
    un histograma y las propiedades
    que me interesan: máximos locales LML, MaxList.
    """

    def __init__(self,data,nbins,wl = 5 , po = 2):
        self.hist, self.bins = np.histogram(data[ data !=0 ]\
        .ravel(), nbins, density=True)
        self.binz = shiftx ( self.bins )    \
        #binz es un desplazamiento usado
        #para que pueda graficarse
        self.max = max(self.hist)
        self.LML = max_of (\
        savgol_filter(self.hist,wl,po, mode = 'nearest' ))

    def MaxList(self,frac=0.1):
        return self.LML[ self.hist[ self.LML ]\
         > frac*self.max ]
```

3.3.5. Cielo

La región del cielo se extrae con el producto `mask_sumI330` × `sky_sumAbso` × `sky_timeDiffAbso` (Figura 3.13)

La región de cielo recuperada es sometida a un ajuste polinomial bidimensional para obtener la matriz **B** en Ecuación (3.8)

3.4. Implementando métodos de corrección

Hasta este punto se tienen las secuencias de absorbancias y las coordenadas de las regiones de cielo, volcán y pluma.

Las absorbancias calculadas requieren ser corregidas por el efecto de dilución.

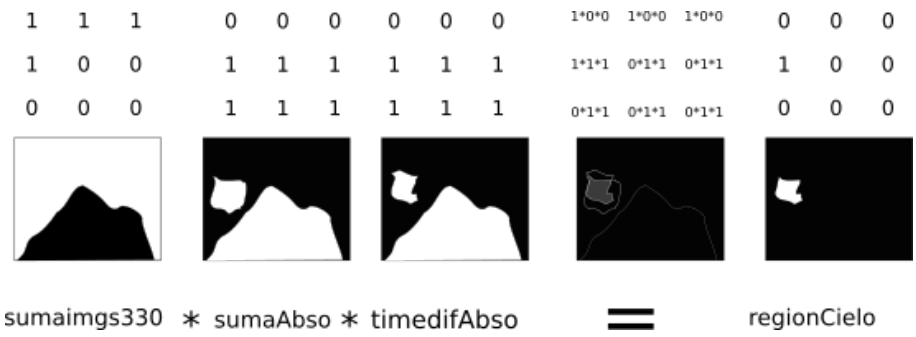


Figura 3.13: Ilustrando el producto `mask_sumI330` × `sky_sumAbso` × `sky_timeDiffAbso`.

3.4.1. Coordenadas de un perfil en la imagen

Al inicio del programa se obtienen dos parámetros que son las coordenadas de dos puntos en una línea vertical sobre la imagen del edificio volcánico, estos valores sirven para obtener un perfil de intensidades sobre la linea que une el par de coordenadas.

3.4.2. Coeficientes de esparcimiento

Para hacer la corrección se recuperan los coeficientes de esparcimiento σ. El perfil de intensidades y distancias reales correspondiente al perfil son sometidos a un ajuste con la Ecuación (2.16) que aparece en la Subsección 2.2.2.

La Ecuación (2.16) es transformada a una función definida en PYTHON como `ajuste_perfil310( d , I0 , sigma ):`

```
1 def ajuste_perfil310( d , I0 , sigma ):
2     return I0 * np.exp( sigma * d ) + \
3     SKY[0] * ( 1 - np.exp( sigma * d ) )
4
5 coeficientes, __ = curve_fit( \
6 ajuste_perfil310, distancias_profile, profileI310, p0 )
```

donde `coeficientes` contiene los coeficientes del ajuste `I0, sigma` = I_0, $-\sigma$. De forma análoga se escribe otra función `ajuste_perfil330( d , I0 , sigma )` para obtener los coeficientes usados en la corrección de la imagen $\mathbf{I}^{330}$.

3.4.3. Matriz de distancias

Se han capturado desde el inicio del programa los datos de la distancia d_0 de la cámara a la pluma volcánica, y el ángulo θ que se forma con un vector que tiene la dirección de propagación de la pluma y la línea central[3] en el campo de visión de la cámara en un plano horizontal.

Para aplicar la Ecuación (3.7) se calcula una matriz de distancias, donde cada entrada corresponda a la distancia pixel-en-$\mathbf{I}^{310}$ – cámara.

$$\mathbf{d} = d_{i,j} = d_0 \left(\cos((i - \frac{n}{2}) \cdot \frac{\xi}{n}) + \frac{\sin((i - \frac{n}{2}) \cdot \frac{\xi}{n})}{\tan(\theta - (i - \frac{n}{2}) \cdot \frac{\xi}{n})} \right) \tag{3.6}$$

ξ es el ángulo de visión de la cámara y n es el número de pixeles en el ancho de la imagen.

Comparaciones con imágenes satelitales han confirmado que la estimación visual del ángulo θ de la pluma, respecto a la linea central en el CV de la cámara, conduce a lo mas 15% de error en la estimación de la distancia d_0 entre la pluma y la cámara [4].

3.4.4. Corrección de intensidades en las imágenes 310 y 330

$$I_{\text{corr}_{i,j}} = \frac{I_{\text{uncorr}_{i,j}} - I_{A_{i,j}}(1 - e^{-\sigma d_{i,j}})}{e^{-\sigma d_{i,j}}} \tag{3.7}$$

```
def corr_plume_intensity(Iuncorr,I_A,sigma,d):
    Icorr = np.zeros(np.shape(Iuncorr))
    for i in np.arange( np.shape(d)[0] ):
        for j in np.arange( np.shape(d)[1] ):
            Icorr[i,j] = ( Iuncorr[i,j] - I_A[i,j]\
            * ( 1 - math.exp( sigma*d[i,j] ) ) )\
            / ( math.exp( sigma*d[i,j] ) )
    return Icorr
```

[3] La línea central es un vector de magnitud d_0 que inicia en la cámara y termina en un punto medio de la pluma

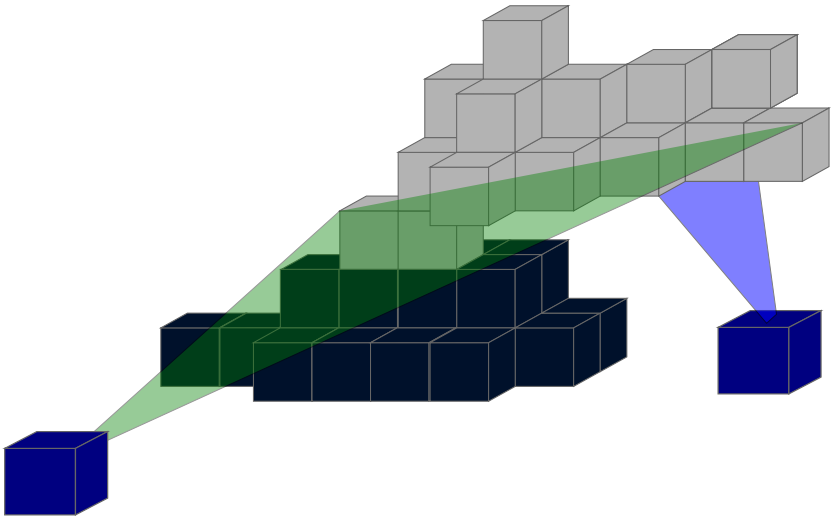


Figura 3.14: Las distancias de la cámara a los pixeles de la pluma dependen del ángulo del campo de visión. La cámara puede ver a la pluma desplazarse perpendicularmente al campo de visión o la pluma puede pasar encima de la cámara.

3.4.5. Absorbancia aparente

Retomando la Ecuación (2.17), se programó el cálculo de la absorbancia aparente, con la primera imagen de una secuencia con cinco imágenes $\mathbf{I}^{310}$ y la primera imágen de una secuencia de cinco imágenes $\mathbf{I}^{330}$, es decir se hizo el cálculo con `I310[0]` e `I330[0]`. El segundo término ya se ha descrito en la Subsección 3.3.5.

$$\mathbf{AA} = \log_{10}\left(\frac{\mathbf{I}_p^{330}}{\mathbf{I}_p^{310}}\right) - \log_{10}\mathbf{B} \tag{3.8}$$

Finalmente para obtener la distribucion de concentraciones se multiplica la absorbancia aparente por el coeficiente de calibración.

3.5. Campo de velocidades y flujo de SO_2

El programa desarrollado por Robin et al. [4] utilizaba perfiles paralelos a la pluma entre imágenes sucesivas, esto es suceptible a errores cuando los vientos resultan ser heterogeneos, por esta razón se buscó una opción que

permitiera obtener el campo de velocidades en toda la región que corresponde al cielo y pluma. El cálculo del campo de velocidades fue posible gracias a un algoritmo de flujo óptico desarrollado por Farneback, este tipo de algoritmos son estudiados en una rama de la computación llamada computación visual. La entrada para la función de Farneback implementada en OpenCV es la distribución de densidades de columna SO_2 de un par de imágenes consecutivas. La salida es una campo vectorial de desplazamientos en pixeles, que multiplicados por el tamaño de un pixel en metros y dividido por el tiempo en segundos transcurrido entre el par de imágenes, se obtiene el campo de velocidades $\mathbf{v} = (v_{i,j})$ asociado a la distribución de SO_2.

3.5.1. Cálculo del flujo de SO_2

El campo de velocidades y la distribución de concentraciones son la base para el cálculo del flujo. La estrategia seguida fue una adaptación al modo en que lo hicieron Peters et al. [29], y se adaptó parte de su código para hacer el cálculo en el flujo de SO_2. Este algoritmo consistió en trazar las coordenadas de una curva[4] de integración Γ en una imagen, para rodear el cráter del volcán. Los puntos de la curva dados por el usuario son interpolados para tener una mayor resolución (~ un punto por pixel). Luego se obtiene la suma sobre todas las componentes perpendiculares de los vectores de velocidad que atraviesan la curva, y se multiplican por las concentraciones correspondientes a esos vectores. En resumen el algoritmo sigue la ley de Gauss de la divergencia en dos dimensiones, en la Ecuación (3.9) el algoritmo es descrito por el término de la izquierda,

$$\int_{\Gamma} \vec{\mu} \cdot \vec{n} \, \mathrm{d}\Gamma = \int \int_{W} \nabla \vec{\mu} \, \mathrm{d}V \tag{3.9}$$

W es una región acotada por la curva frontera[5] Γ, y $\vec{\mu}$ es un campo vectorial generado por las densidades de columna en cada pixel de la imagen y sus velocidades asociadas, *i.e.* $\vec{\mu}_{i,j} = S_{i,j}\vec{v}_{i,j}$.

El siguiente extracto del código es la esencia del cálculo del flujo de SO_2, se obtienen los valores de los pixeles en una imágen interpolada de la distribución de SO_2 y las componentes perpendiculares de las velocidades en esos

[4] Si los datos de distribución de concentraciones fueran tridimensionales la curva se vería como una sabana o superficie que cubre la abertura del volcán.

[5] $\mathrm{d}\Gamma$ y $\mathrm{d}V$ tienen unidades de longitud y área respectivamente, pues las imágenes son bidimensionales.

puntos. La curva Γ se divide en un número de segmentos aproximadamente igual al número de pixeles que atraviesa y se obtienen las longitudes de esos segmentos (se guardan en la lista `length_of_segments`). El flujo por un pixel es el producto del valor de la densidad de columna SO_2 por un factor de conversión ($1/(1.5\times10^{-4}$ [4]) para convertir [ppm m] a [Kg/m^2]), la velocidad [m/s] perpendicular a un segmento de la curva dΓ y la longitud [m] correspondiente al pixel. Haciendo la suma sobre cada pixel que atraviesa la curva se obtiene el flujo total de masa SO_2 [kg/s], término de la izquierda en la Ecuación (3.9).

```
#x_coord, y_coord son coordenadas que recorren la curva Gamma
for i in range(number_of_segments):
    x_coord = mid_points[i,0]
    y_coord = mid_points[i,1]
    pix_vals[i] = image_interp(x_coord, y_coord)
    err_vals[i] = errors_interp(x_coord, y_coord)
    velocities[i,0] = x_flow_interp(x_coord, y_coord)
    velocities[i,1] = y_flow_interp(x_coord, y_coord)

#Calcular velocidades perpendiculares a la curva
perp_vel = numpy.diag(numpy.dot(velocities, int_norms.T))

#cacular flujo
fluxes = pix_vals * conversion_factor * perp_vel * length_of_segments
total_flux = numpy.sum(fluxes)
```

Capítulo 4

Resultados

4.1. Software desarrollado

El programa escrito en PYTHON logra el objetivo inicial, detecta las distintas regiones en las imágenes y es posible procesar pares de imágenes con solo introducir los parámetros esenciales al inicio del programa. En este capítulo se revisan los requerimientos básicos para ejecutar el software desarrollado y se muestra parte de los resultados más relevantes en los cálculos: imágenes intermedias en los cálculos, umbrales, obtención de los coeficientes de esparcimiento, corrección de la dilución, campos de velocidades y el resultado final: los datos de la variación de flujo en el tiempo, que demuestran el potencial de un programa de este tipo usado junto con la cámara, para monitorear el comportamiento de un volcán.

4.1.1. Requerimientos y especificaciones

El programa fue escrito en lenguage PYTHON para la versión 3.4, en una arquitectura de 64 bits, y con instalaciones de los módulos de matplotlib (1.5.1), numpy (1.11.1), cv2 (1.0) y scipy (0.16.0).

Versiones de PYTHON 3.4.X con versiones anteriores de Matplotlib, NumPy y SciPy, o instalados en una arquitectura de 32 bits deben ser compatibles con el software de este trabajo. En el caso de versiones anteriores de opencv (cv2) la función calcOpticalFlowFarneback debe consultarse en su documentación. Si se desea usar PYTHON 2.7 la sintaxis para print y funciones de tkinter deben revisarse.

El interprete de PYTHON y su biblioteca estándar se puede descargar de forma gratuita en www.python.org, la documentación y otros links para descargar los módulos también están disponibles. La instalación de los módulos cambia de acuerdo al sistema operativo, una guia para esto: http://-www.scipy.org/install.html, para windows paquetes binarios no oficiales están en http://www.lfd.uci.edu/~gohlke/pythonlibs/.

Una vez que se cuenta con la instalación de PYTHON adecuada, el programa puede ejecutarse desde una consola o la terminal (Linux o Windows):

```
python main.py
```

también es necesario conocer las rutas donde se encuentran almacenados los archivos de calibración de vignetting y bias, asi como del directorio data en el que deben estar los pares de archivos *-1.raw y *-2.raw y sus correspondientes archivos de texto *.txt. Deben existir al menos cinco pares de imágenes. Un archivo de configuración: config.ini debe estar dentro del mismo directorio que main.py.

4.1.2. Guía de uso y documentación básica

El usuario puede decidir modificar los parámetros esenciales dentro del archivo de configuración config.ini, y asegurarse de poner los parámetros apropiados para el volcán que se está monitoreando. Algunos parámetros podrán ser modificados directamente al iniciar el programa, si estos son modificados por el usuario esos nuevos datos se respaldarán automáticamente en el archivo de configuración.

Al iniciar el programa, este pide las rutas del par de archivos de calibración de vignetting, del par de archivos bias y del directorio principal `data`, despues de aceptar, si el programa detecta rutas válidas, cuenta el número de pares de imágenes disponibles y pide al usuario escribir un rango de pares de imágenes a procesar.

El usuario decide si al final de la ejecución necesita visualizar un archivo *.gif, donde aparezca el movimiento general de la pluma o revisar cada par de imágenes procesadas.

El usuario tiene que elegir un par de puntos necesarios para la corrección de dilución de luz. Sobre el edificio volcánico se trazan los puntos con distancias conocidas y especificadas en los datos en el archivo de configuración como $d0$ y $d1$.

Por último se debe elegir una curva frontera con 6 puntos encima del cráter del volcán, esta curva será la línea de integración sobre la que se calcula el flujo de SO_2. Al cerrar la ventana comienza el procesamiento de todos los pares de imágenes colicitados, y al final en el directorio `data` se crea un nuevo directorio `results` con las imágenes y archivo gif (si fue marcado al inicio del programa). Además un archivo `flujos.txt` con dos columnas de datos, la primera para los flujos calculados y la segunda para sus errores asociados en porcentaje.

Los datos en el archivo `flujos.txt` son graficados para ver el cambio de flujo en el tiempo. Esto se puede utilizar para estudiar el comportamiento del volcán.

4.2. Mediciones al volcán Popocatépetl

El volcán Popocatépetl está ubicado en México con una elevación de 5452 m sobre el nivel del mar, es uno de los más activos en el mundo. Los datos de flujo promedio reportados en otras mediciones de [4] son de ~40 kg/s.

Se procesaron 100 pares de imágenes de mediciones que se llevaron a cabo el día 31/01/2013 10 AM Hora local.

Lo que sigue son los resultados que se obtuvieron durante la ejecución del programa con el primer par de imágenes y la primera secuencia. En todas las imágenes procesadas se especifican las líneas y columnas de pixeles en los ejes vertical y horizontal respectivamente. En las barras de color los valores son unidades arbitrarias dadas por la intensidad percibida en los sensores CCD (detectando valores de 0 a $2^{16} - 1$).

El par de imágenes de la Figura 4.1, expone las imágenes crudas 'raw' cargadas al programa, este par requiere una preparación antes de formar parte de una secuencia.

La preparación consiste en rotar, trasladar, recortar y usar las imágenes de calibración bias y vignetting como se revisó en la Sección 3.2 del Capítulo 3. El preprocesamiento se refleja en la Figura 4.2.

Después de preparar cinco pares de imágenes el programa llena una secuencia, y con ella se calcula una secuencia de absorbancias.

Se construyen tres imágenes nuevas, que se guardan en las variables `sumI330`, `sumAbsor` y `timediffAbso`, las tres son generadas a partir de una secuencia de absorbancias, y cada una es analizada con un histograma para detectar umbrales, los detalles del cálculo están en la Sección 3.3.

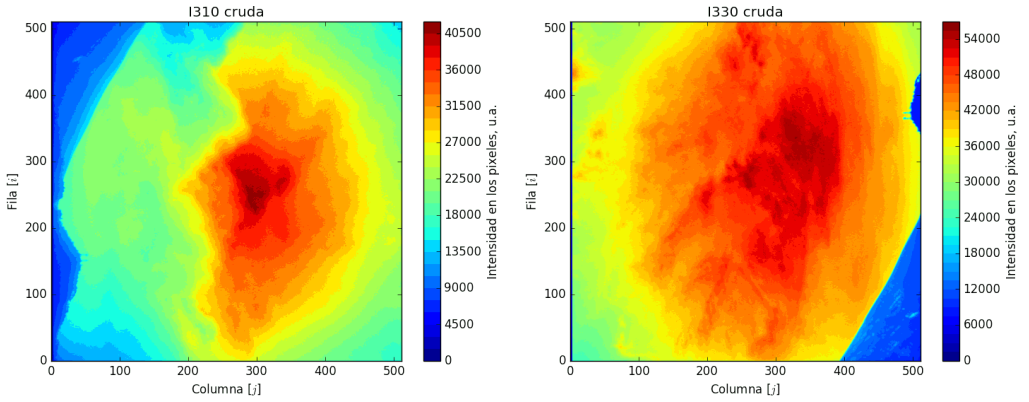


Figura 4.1: Visualización de las imágenes de entrada al programa, los archivos se han leído en el programa y se han representado con la función `contourf`, en este punto las imágenes necesitan prepararse antes de empezar el procesamiento.

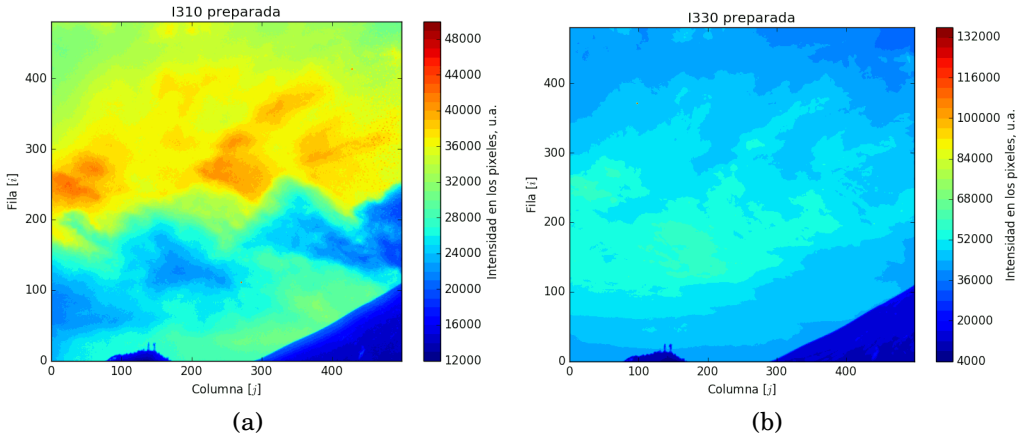


Figura 4.2: (a) Imagen I310 e (b) I330 preparadas para hacer correcciones y procesamientos.

Los umbrales permiten obtener imágenes binarias, por ejemplo el histograma de `sumI330` (Fig. 4.4a) sirve para detectar el cielo del volcán, asignando valores 1 a pixeles del cielo y 0 a los del edificio volcánico (4.4b).

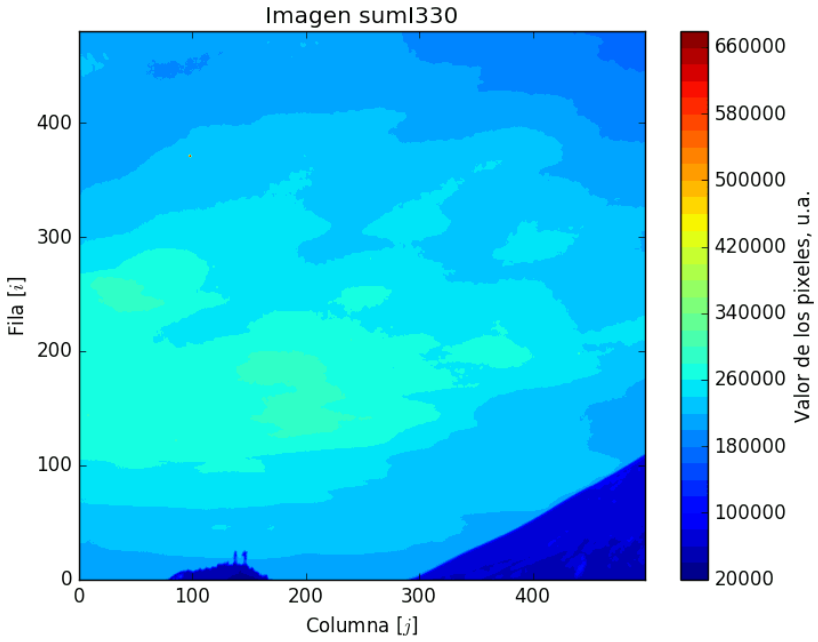
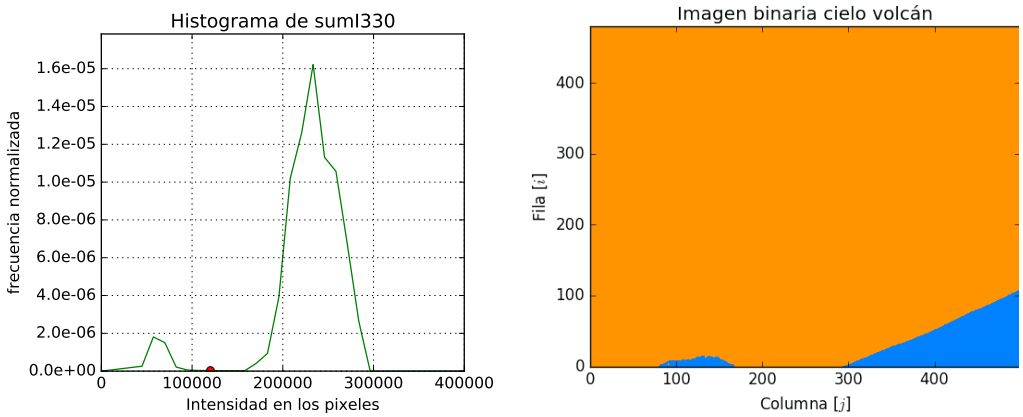


Figura 4.3: Imagen sumI330.



(a) (b)

Figura 4.4: (a) Histograma de la imagen sumI330 obtenida de una secuencia, y (b) la imagen binaria obtenida con el umbral de 110182 marcado en el histograma, los valores de intensidad en los pixeles se convierten en: Cielo = 1 y edificio volcánico del Popocatépetl = 0.

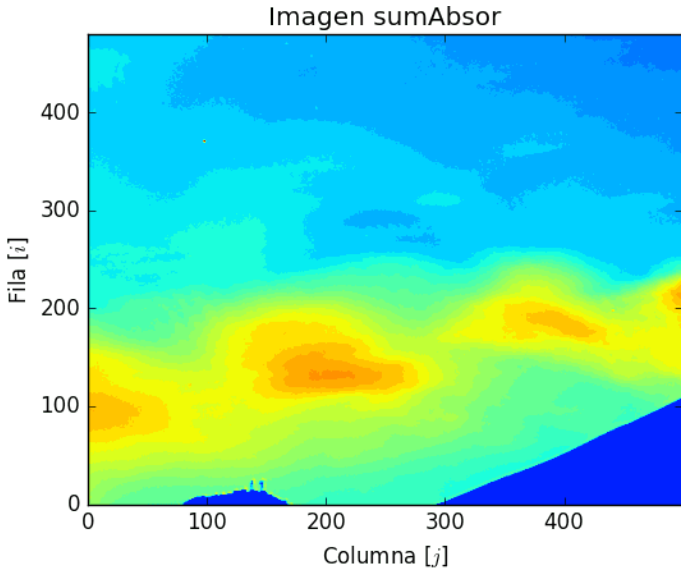


Figura 4.5: Imagen almacenada en una variable sumAbsor.

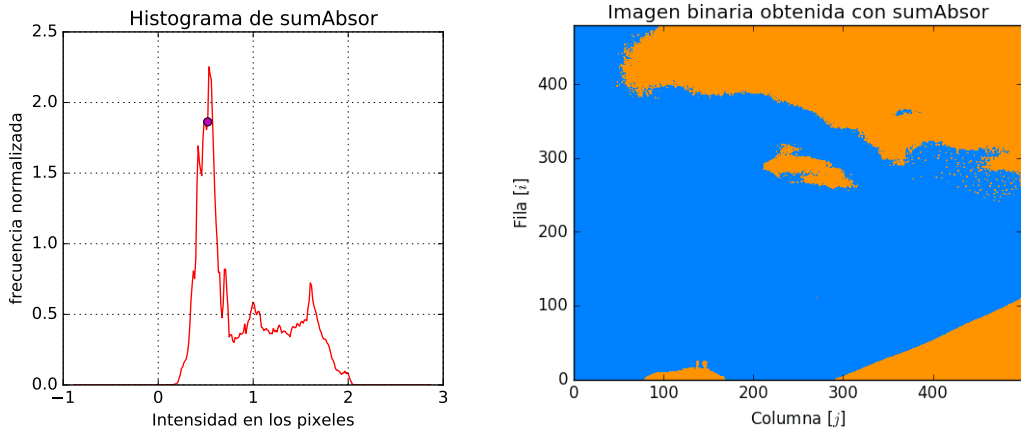


Figura 4.6: Se obtiene un umbral de la imagen sumAbsor.

También de las variables `sumAbsor` y `timediffAbso` se obtienen regiones de cielo. Multiplicando las tres imágenes binarias entrada a entrada, se recupera una región en la que se tiene certeza que pertenece a pixeles del cielo. Ésta región es multiplicada por una imagen de absorbancias. La absorbancia por la región de cielo recuperada es extrapolada. Se hace con el fin

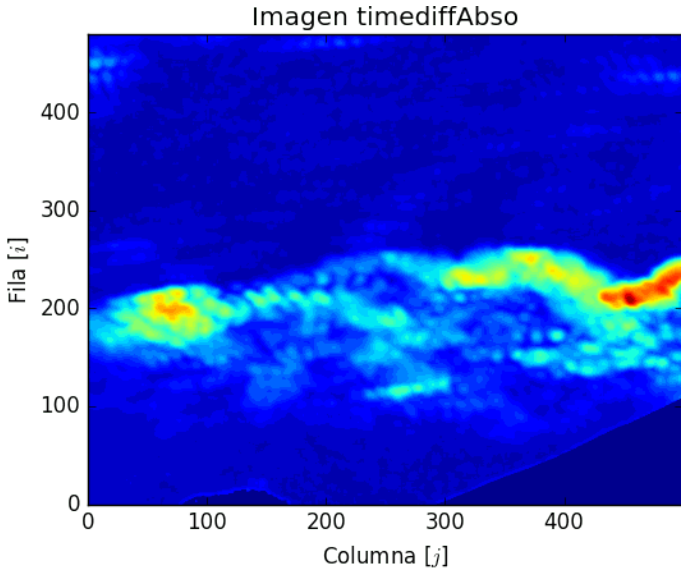


Figura 4.7: Imagen almacenada en una variable timediffAbso.

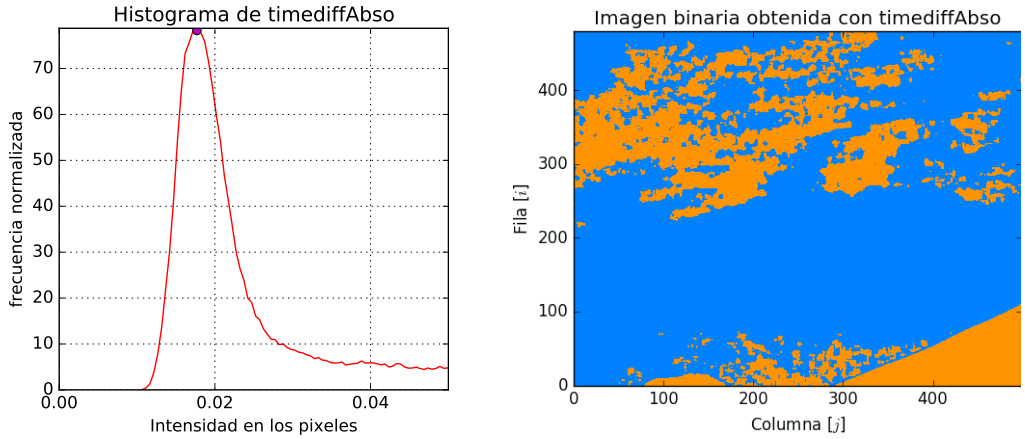
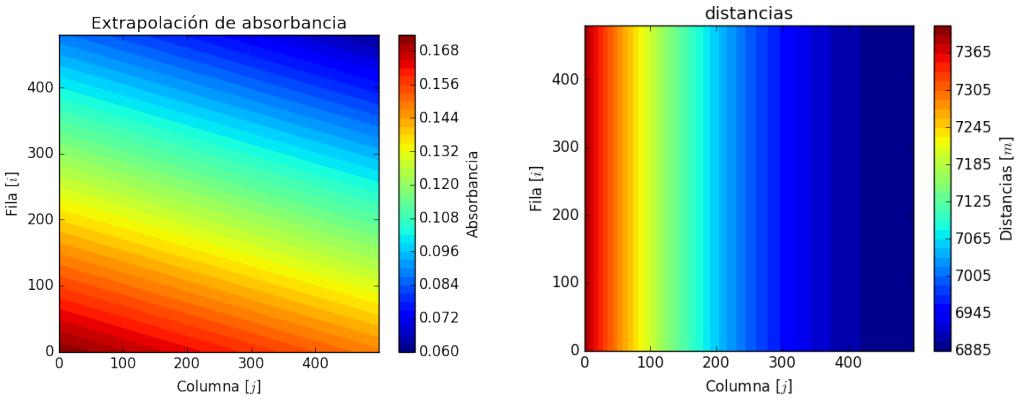


Figura 4.8: Se obtiene un umbral de timediffAbso.

de obtener una aproximación a la absorbancia del cielo detrás de la pluma volcánica.

En la Figura 4.9a se visualiza la imagen extrapolada, proporcional al término $\log_{10}\mathbf{B}$ en la Ecuación (2.17).

Para obtener los coeficientes de esparcimiento (Sección 3.4) se trazan dos



(a) (b)

Figura 4.9: a) Extrapolación de la absorbancia en la región de cielo detectada. b) Distancias calculadas para la imágenes del volcán Popocatépetl, obtenidas aplicando la Ecuación (3.6).

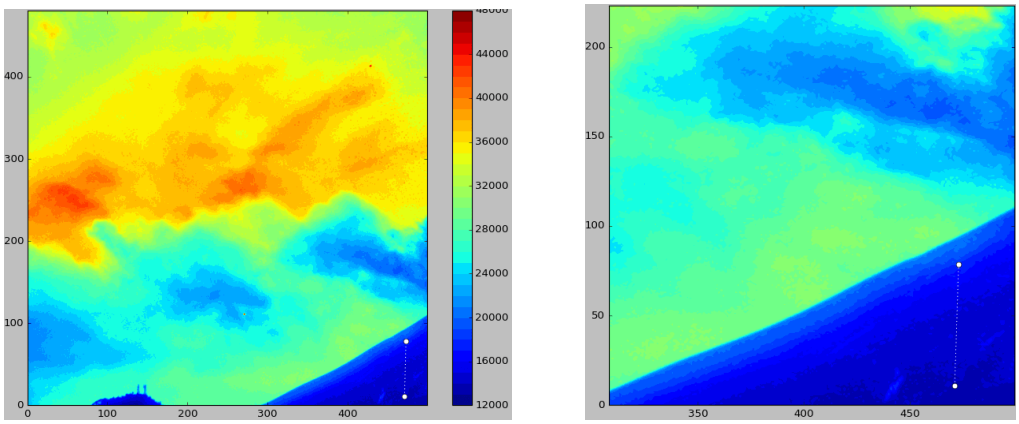
puntos con distancias conocidas (Ver Figura 4.10). Éstas aproximaciones son válidas para regiones de la imágen en las que el edificio volcánico presente una pendiente homogenea.

Se usaron los siguientes datos en el archivo de configuración:

```
[datosVolcan]#Popocatepetl
distance cam-plume = 7000
ang fov-plume = 100
profile d0 = 5000
profile d1 = 7000
```

Los coeficientes de esparcimiento obtenidos del ajuste de intensidades a la Ecuación (2.16) fueron $\sigma_{310} = 6.6 \times 10^{-5} \mathrm{m}^{-1}$ y $\sigma_{330} = 3.4 \times 10^{-5} \mathrm{m}^{-1}$, valores del orden de magnitud reportados en [8]. Además $\sigma_{310} > \sigma_{330}$ es un resultado que se esperaba, porque esto es coherente con las Ecuaciones (2.10) y (2.11).

Los coeficientes obtenidos se usaron para hacer la corrección de dilución con la Ecuación (3.7). El efecto de la corrección es que los pixeles menores en intensidad que los pixeles de cielo disminuyen aun más su intensidad, debido a que con la dilución (mezcla de esparcimiento Rayleigh y Mie) aumenta la cantidad de fotones en el campo de visión de la cámara y esto hace parecer que el edificio volcánico refleja mas fotones de lo que en realidad es. Si las mediciones se realizan mas cerca de la pluma, la correccion por dilu-



(a) (b)

Figura 4.10: Puntos d0 y d1, de un perfil sobre el edificio volcánico. (b) es una ampliación de (a) en la región de interés.

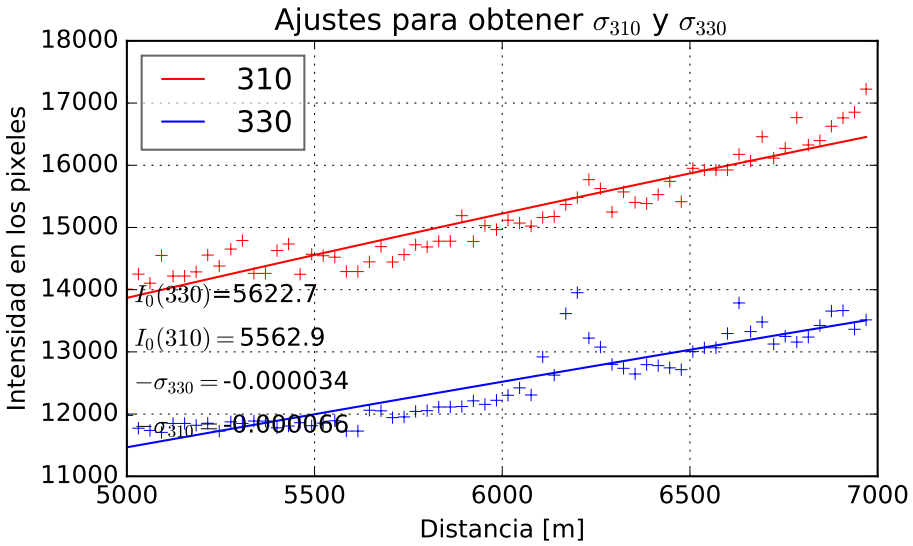


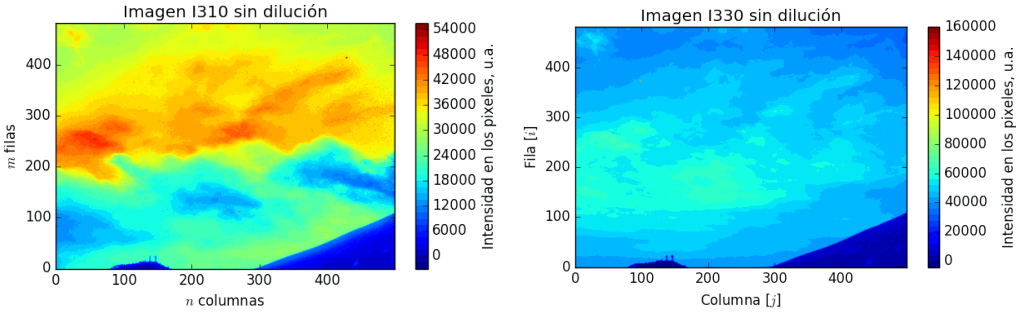
Figura 4.11: Ajustes para obtener coeficientes de esparcimiento. d0 = 5000 y d1 = 7000.

ción es mas débil, pues la corrección aumenta con la distancia de acuerdo con Ecuación (3.7).

También fue necesario tener un arreglo con las distancias que correspon-

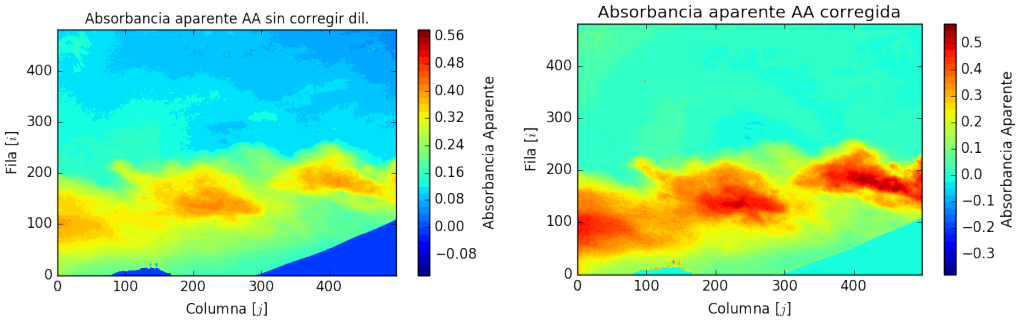
den a cada pixel en la imagen (Ver Figura 4.9b), a partir de esta matriz se generó otra, que contiene información del tamaño de cada pixel en metros.

Después de aplicar la corrección por dilución Figura 4.12, se hace el cálculo de la absorbancia aparente AA con la Ecuación (2.17) (véase la Figura 4.13).



(a) (b)

Figura 4.12: a) Imagen I310 corregida e b) Imagen I330 corregidas, *cf.* Figura 4.2.



(a) (b)

Figura 4.13: a) Absorbancia Aparente **AA** sin corrección y b) con corrección de dilución.

Multiplicando la absorbancia aparente por el coeficiente de calibración de 6250 ppm m se obtiene el mapeo de densidades de columna SO_2 que se presenta en la Figura 4.14.

Se utiliza un par de imagenes consecutivas `I310[0]`, `I310[1]` para obtener el campo de velocidades de la Figura 4.15

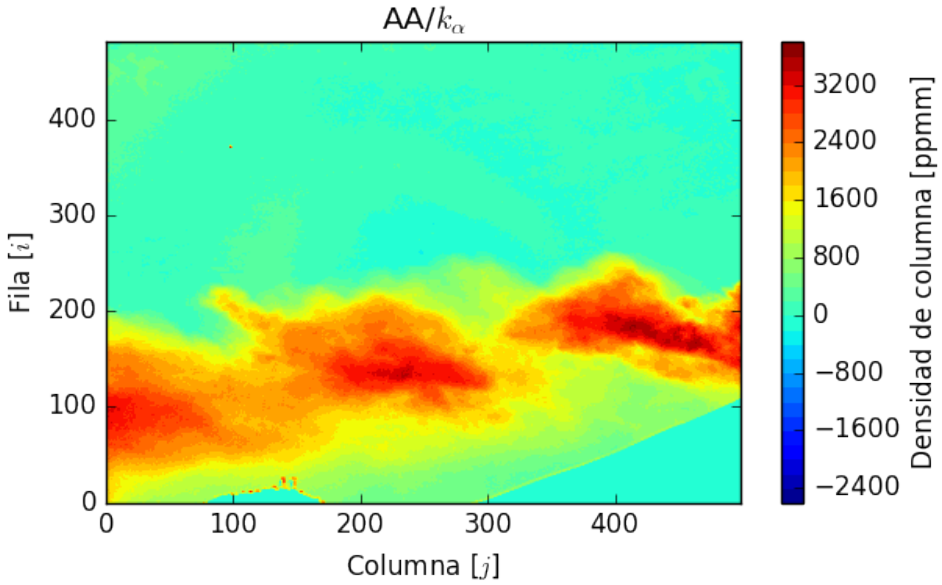


Figura 4.14: Distribución de densidades de columna SO_2 en ppm m, con corrección de dilución.

Los ingredientes para calcular el flujo que pasa por una curva frontera alrededor del crater es dictado por la ley de divergencia de Gauss (Ecuación (3.9)). Los ingredientes son el campo de velocidades, la distribución de concentraciones SO_2 obtenidos anteriormente y la curva frontera de integración Γ dibujada durante la ejecución del programa. Se observó que el cálculo del flujo óptico por el algoritmo Farneback tiende a ser inestable en los bordes de la imágen, por tanto el programa podría sobrestimar el flujo en el caso de que se tracen curvas sobre el borde y las cantidades de SO_2 sean altas en esos puntos. El flujo para el volcán Popocatépetl se visualiza en la Figura 4.17, las barras de error están asociadas a la incertidumbre en el cálculo del flujo óptico. El cálculo del flujo de SO_2 por el programa es comparable con datos reportados por Campion et. al [4] del orden de 40 kg/s.

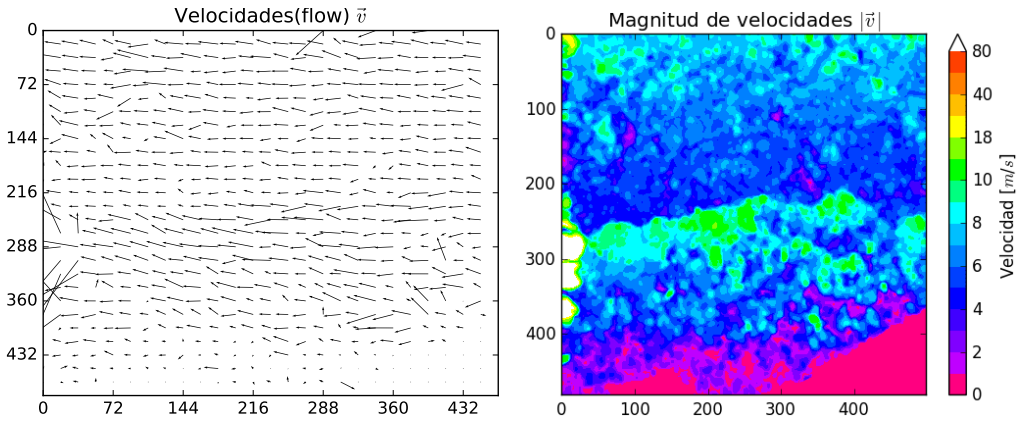


Figura 4.15: a) Campo de velocidades y b) sus magnitudes. El error en el cálculo de flujo óptico en los bordes, tiende a ser mayor. *Los vectores en (a) muestran las direcciones de las velocidades, tienen escalas y densidades ajustadas para su visualización (un vector cada 18 pixeles en filas y columnas).

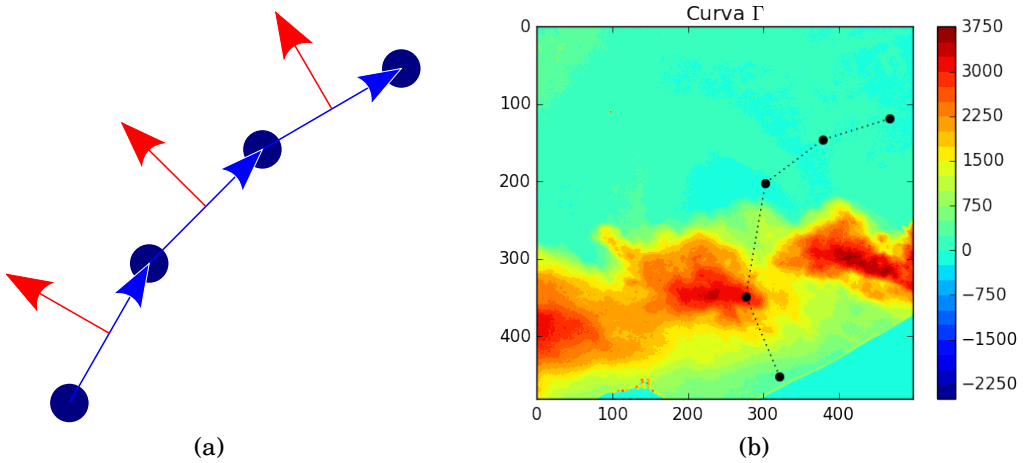


Figura 4.16: Curva de integración Γ, se hace una interpolación de estos puntos para obtener un punto en cada pixel.

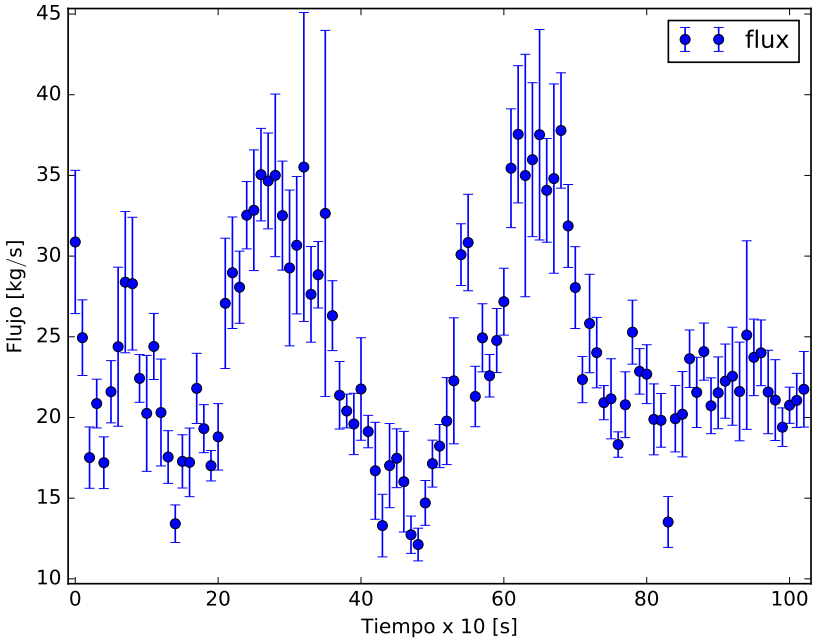


Figura 4.17: Flujo de SO_2 en el volcán Popocatépetl.

4.3. Pacaya

El volcán Pacaya se encuentra en Guatemala (14.381^o N, 90.601^o O) y tiene una elevación de 2552 m. Existen pocos datos de referencia para este volcán, las imágenes procesadas son de las primeras mediciones realizadas desde la reactivación del volcán despues de una erupción fuerte en el 2010. Datos de mediciones con COSPEC reportadas en [22] del año 2005 indican de 300 a 400 toneladas por día.

Los resultados para esta sección están mas condensados que la sección anterior, para mostrar lo resultados relevantes. En la Figura 4.18 están las imágenes creadas para resaltar las regiones del cielo y en sus histogramas (Figura 4.19) se muestran los umbrales. El programa detecta la región del cielo y el volcán con la imagen 4.18a, pues su histograma 4.19a muestra como hay dos acumulaciones prominentes, el cúmulo a mayor intensidad corresponde al cielo y el cúmulo de la izquierda son las intensidades del volcán. El umbral es un punto medio en el eje de las intensidades entre los puntos que corresponden a los máximos de la distribución. De esta manera se crea una imagen binaria 4.18d que es usada como máscara para operar sobre las otras imágenes y obtener la intersección que corresponde a una región de cielo.

El primer par de una secuencia se muestra en la Figura 4.20,

después de hacer el ajuste de los perfiles de intensidades sobre el edificio volcánico y usar los datos siguientes:

```
[datosVolcan]#Pacaya
distance cam-plume = 1000
ang fov-plume = -30
profile d0 = 400
profile d1 = 800
```

se obtuvieron los coeficientes $\sigma_{310} = 1.9 \times 10^{-4}\text{m}^{-1}$ y $\sigma_{330} = 1.6 \times 10^{-4}\text{m}^{-1}$. Las imágenes son corregidas con los coeficientes obtenidos (ver Figura 4.22), y con ellas se obtienen las concentraciones SO_2 en ppm m (Figura 4.23).

Dos imágenes consecutivas $\mathbf{I}^{310}$ en una secuencia son usadas para calcular el campo de velocidades (Figura 4.24), luego se calcula el flujo de SO_2 que pasa por una curva frontera rodeando la salida de los gases en la imágen de concentraciones de SO_2. El resultado final es la gráfica en la Figura 4.25.

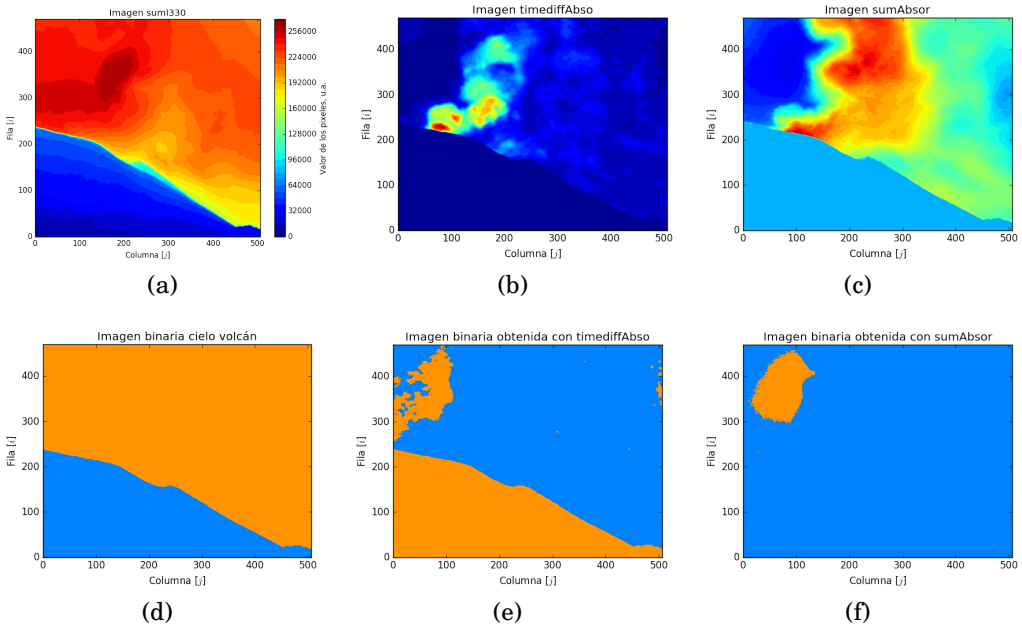


Figura 4.18: Imágenes SumI330, timediffAbso, sumAbso

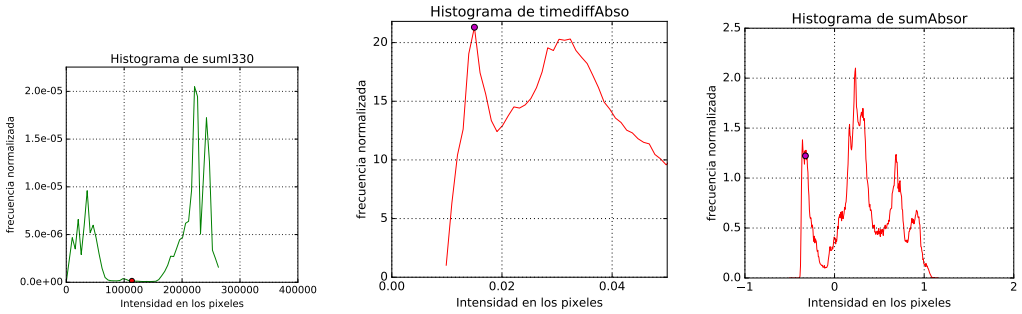
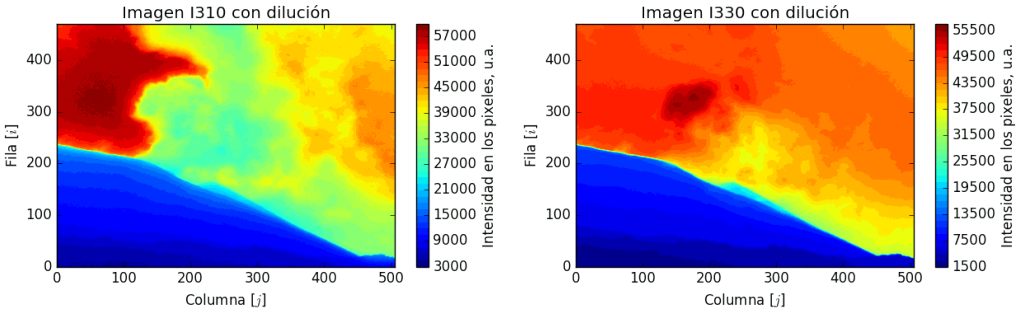


Figura 4.19: Histogramas con los que se calculan los umbrales, estas corresponden a una secuencia de cinco absorbancias, por lo tanto en cada secuencia se obtienen umbrales actualizados.



(a) (b)

Figura 4.20: Un par de imágenes I310 e I330 sin corrección de dilución.

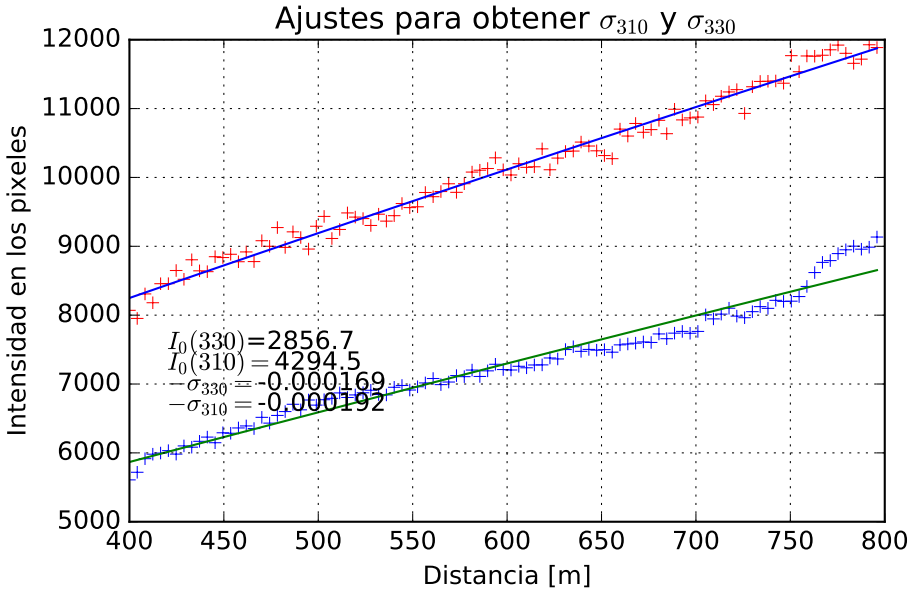
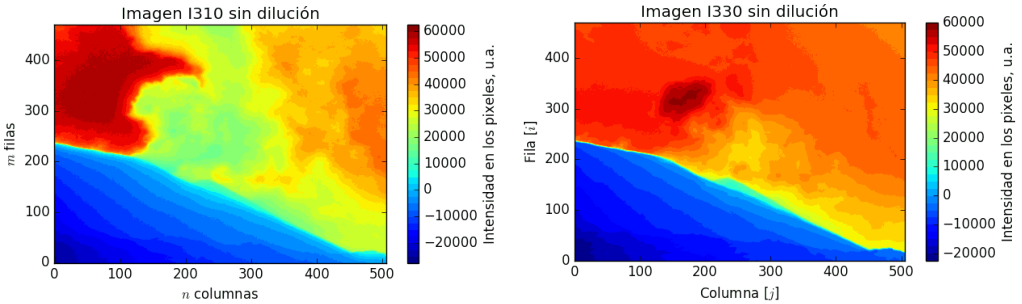


Figura 4.21: Ajustes para obtener coeficientes de esparcimiento.



(a) (b)

Figura 4.22: a) Imagen I310 corregida e b) Imagen I330 corregidas, *cf.* Figura 4.20.

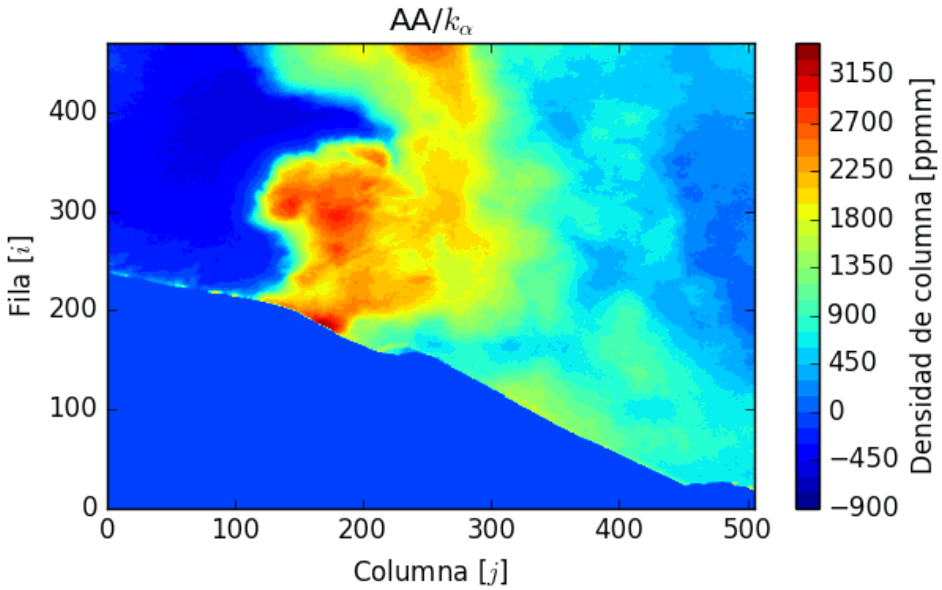


Figura 4.23: Distribución de densidades de columna SO_2 con corrección de dilución.

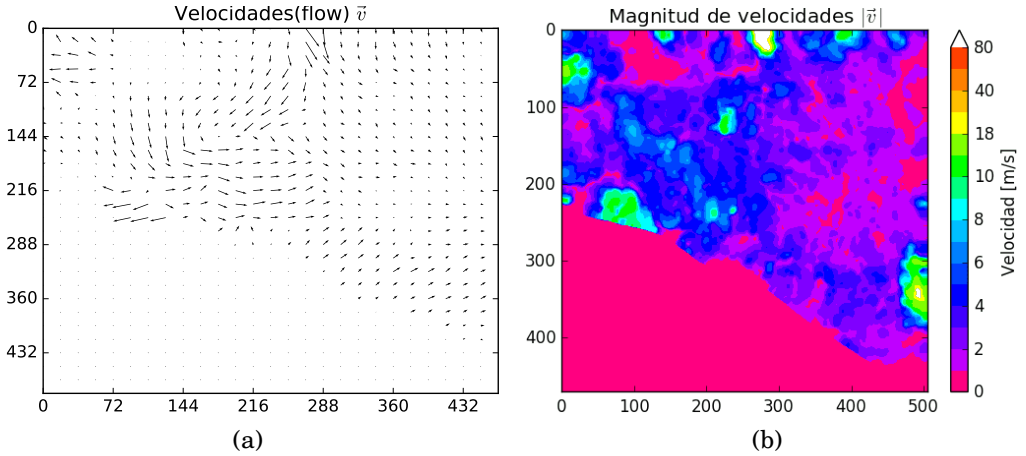


Figura 4.24: Campo de velocidades y sus magnitudes [m/s]. El error en el cálculo de flujo óptico en los bordes, tiende a ser mayor.

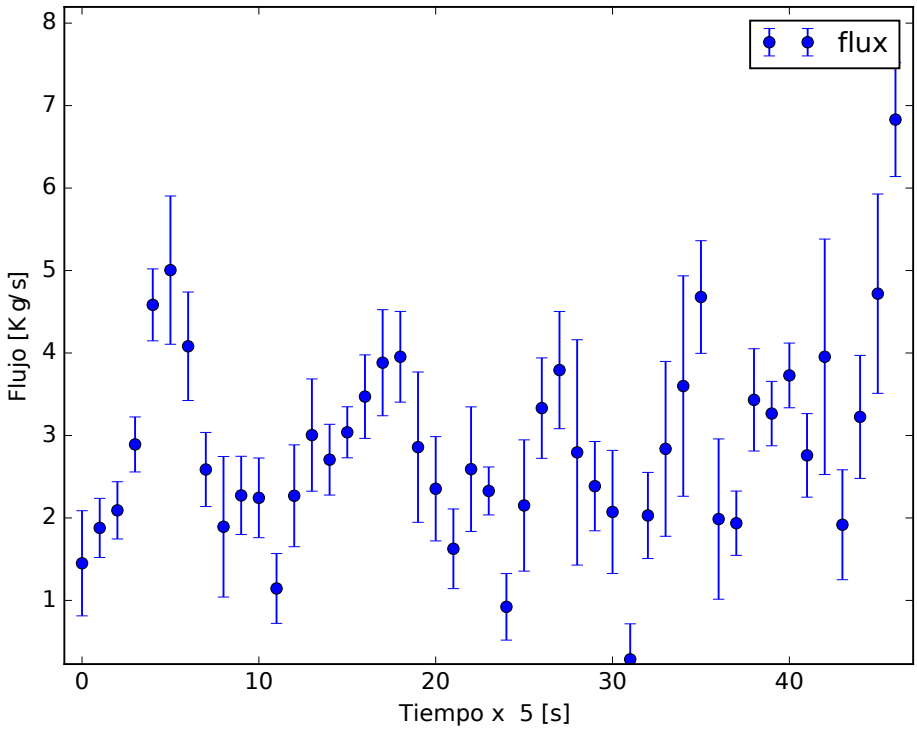


Figura 4.25: Sat, 16 Jan 2016 14:44:46 GMT. Flujo del volcán Pacaya.

4.4. Ubinas

El volcán ubinas está ubicado al sur de Perú, en la cordillera de los Andes, a 5672 m.s.n.m. Las mediciones realizadas el 8 de julio de 2015 19:25:10 GMT, se procesaron con el programa, el cual arrojó los resultados que se presentan en esta sección

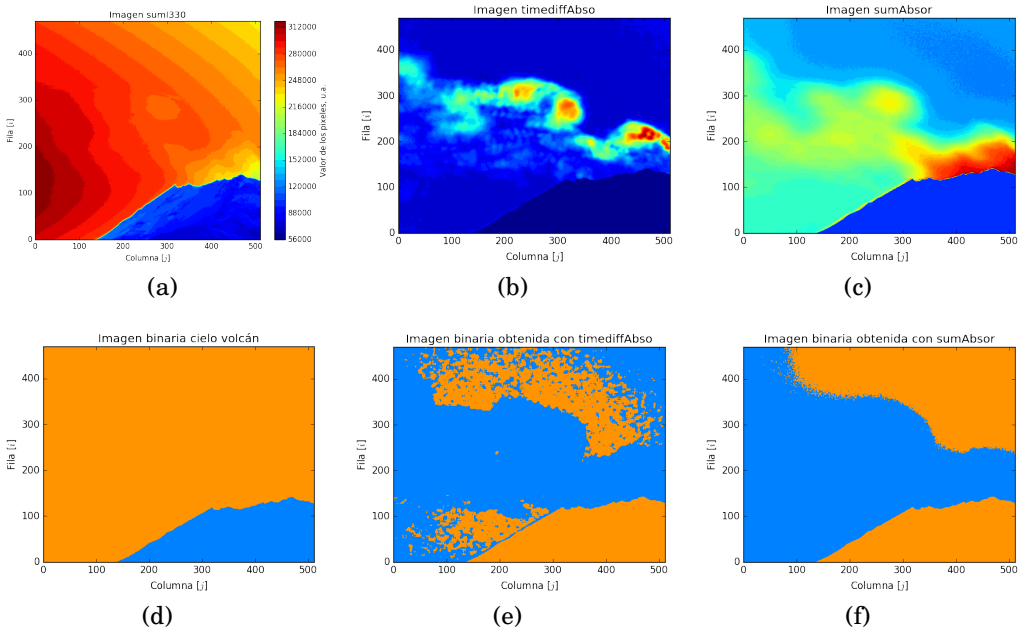


Figura 4.26: SumI330, timediffAbso, sumAbso

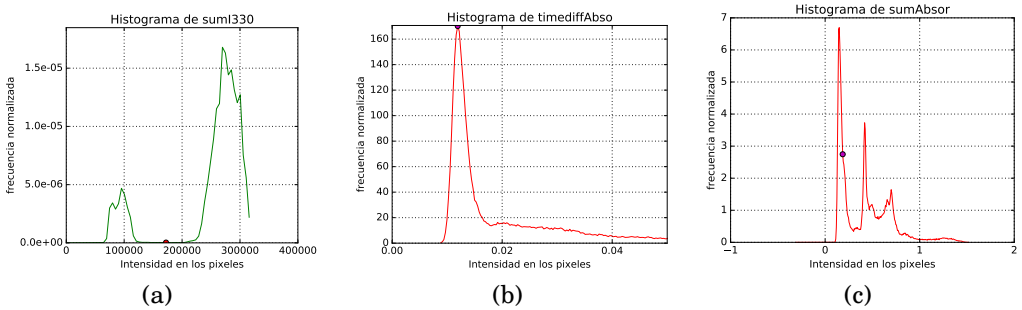
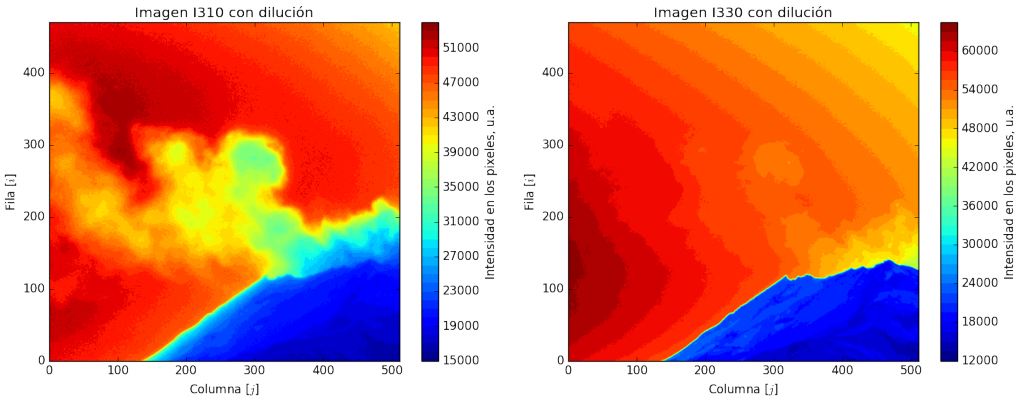


Figura 4.27: histSumI330, histtimediffAbso, histsumAbso



(a) (b)

Figura 4.28: Un par de imágenes I310 e I330 sin corrección de dilución.

Las imágenes con y sin correción de dilución (esparcimiento Mie y Rayleigh) se pueden comparar en los pares de imágenes de la Figura 4.29 y la Figura 4.28.

Después de hacer el ajuste de los perfiles de intensidades sobre el edificio volcánico y usar los datos siguientes:

```
[datosVolcan] #Ubinas
distance cam-plume = 4000
ang fov-plume = 90
profile d0 = 1500
profile d1 = 3500
```

se usaron los coeficientes $\sigma_{310} = 4.6 \times 10^{-5}\text{m}^{-1}$ y $\sigma_{330} = 3.7 \times 10^{-5}\text{m}^{-1}$.

El flujo de SO_2 del volcán (Ubinas Figura 4.32) se calcula usando la informacón de la Figura 4.30 y la Figura 4.31.

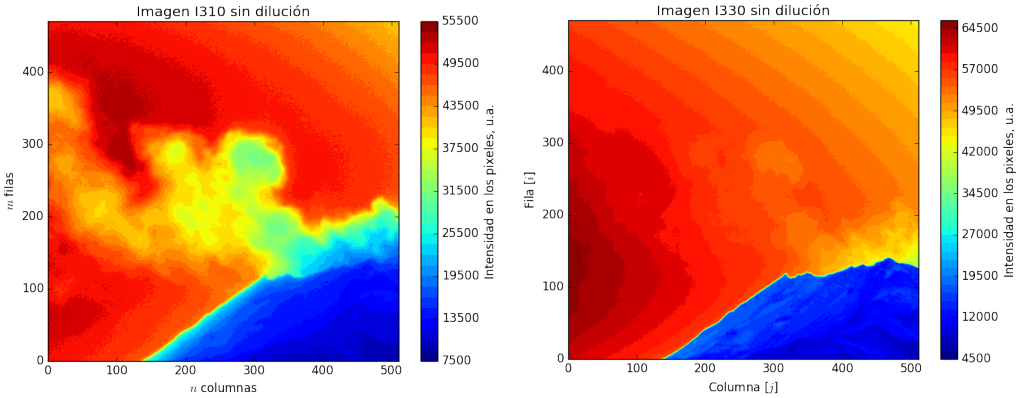


Figura 4.29: a) Imagen I310 corregida e b) Imagen I330 corregidas, *cf.* Figura 4.28.

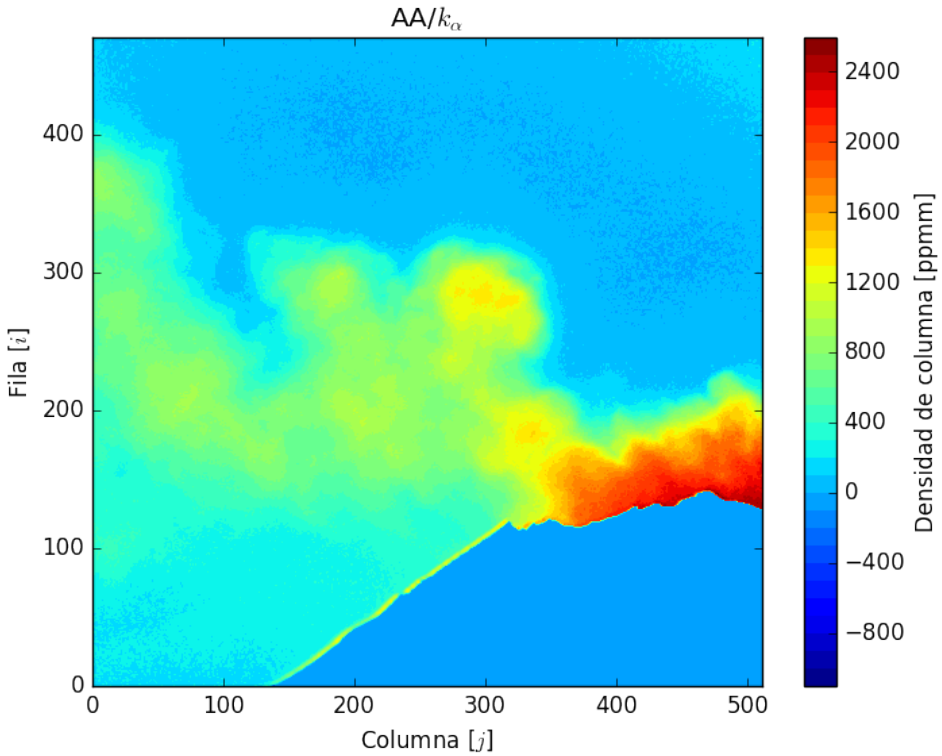


Figura 4.30: Distribución de densidades de columna SO_2 en ppm m, con corrección de dilución.

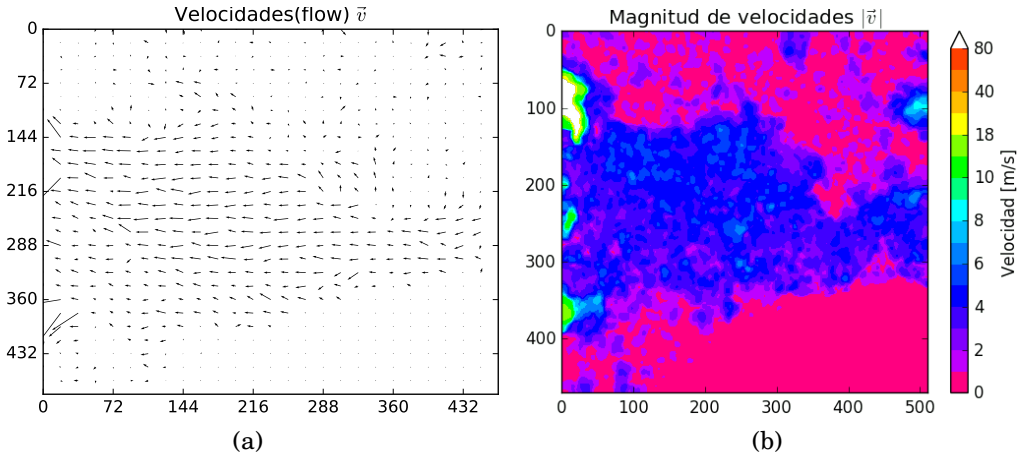


Figura 4.31: Campo de velocidades y sus magnitudes [m/s]. El error en el cálculo de flujo óptico en los bordes, tiende a ser mayor.

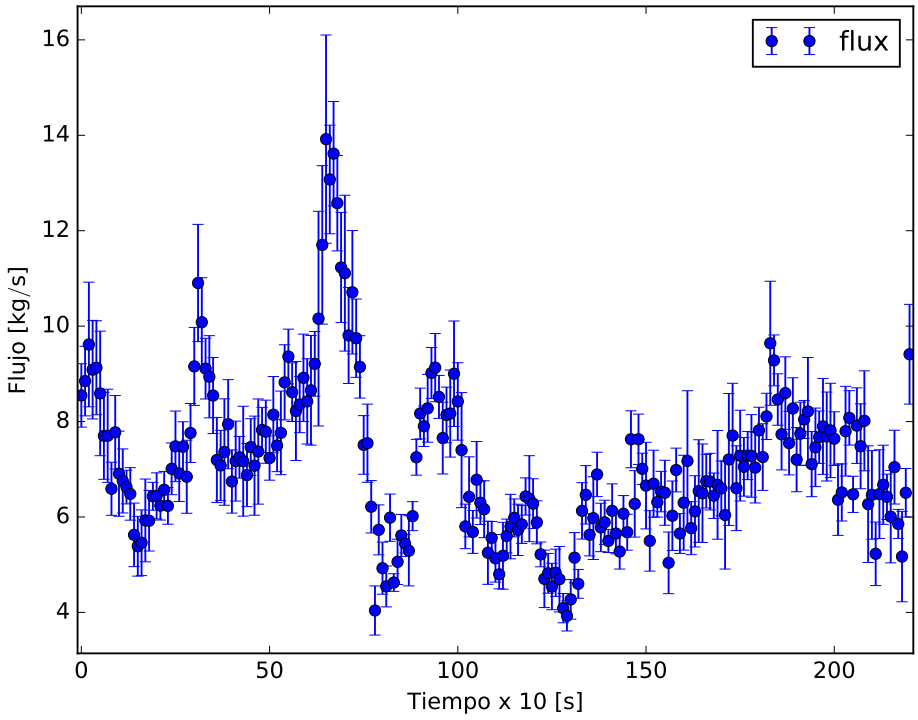


Figura 4.32: Flujo del volcán Ubinas.

Capítulo 5

Conclusiones y recomendaciones

En las erupciones volcánicas se emiten gases y componentes sólidos, la medición de los gases es importante porque estos controlan la intensidad en la erupción. Un gas común en las plumas volcánicas es el dióxido de azufre SO_2, se encuentra típicamente en concentraciones de 50 ppm, en contraste con la atmósfera promedio donde su concentración es del orden de 50 ppb.

La cámara SO_2 es un instrumento que permite hacer mediciones de las densidades de columna SO_2 en plumas volcánicas. Su resolución temporal de hasta 1 Hz y las distribuciones bidimensionales de intensidades que adquiere ofrecen la posibilidad de calcular distribuciones de velocidades, necesarias para calcular el flujo.

En esta tesis se trabajó con imágenes crudas obtenidas por una cámara SO_2, compuesta por dos cámaras con sensores CCD sensibles a la radiación ultravioleta y equipada con dos filtros de banda estrecha centrados en 310 y 330 nm. En el primer filtro la absorción por moléculas de SO_2 es fuerte mientras que con el filtro de 330 nm las moléculas de SO_2 son idealmente invisibles.

Se escribió un programa en Python y se logró el objetivo de rastrear automáticamente las regiones de cielo, pluma y edificio volcánico en cada imagen. Para conseguirlo se hizo el análisis a los histogramas de imágenes creadas a partir del cálculo de absorbancias. En el procesamiento de imágenes para obtener las densidades de columna de SO_2 se usaron algoritmos de filtrado de imágenes con operadores matriciales, derivadas, funciones de estadística, filtrado de señales para suavizar curvas con el algoritmo Savitzky-Golay, ex-

trapolación de imágenes bidimensionales y morfología de imágenes binarias.

Otro aporte de este trabajo fue adaptar un algoritmo para calcular el flujo, basado en el teorema de Gauss, para este procesamiento primero se obtuvo el campo de velocidades usando el cálculo de flujo óptico con el algoritmo Gunnar Farneback, esto se aplicó a un par de imágenes consecutivas. También se hicieron interpolación de datos y operaciones vectoriales.

En los procesamientos: para obtener distribuciones de las concentraciones de SO_2, distribución de velocidades y flujo, se utilizaron principalmente los módulos SciPy, NumPy y Matplotlib, este último permitió hacer representaciones gráficas para visualizar e interpretar los datos e imágenes generadas en cada etapa del procesamiento.

Para lograr mediciones eficientes y confiables, es importante una combinación de software y hardware. El software debe tomar en cuenta los principios de medición y fuentes de error. La revisión de la teoría fue clave para estos propósitos. Fuentes de error sistemáticos como el efecto de esparcimiento por aerosoles en el campo de visión y dentro de la pluma fueron reducidos, aplicando aproximaciones y calculando coeficientes de esparcimiento.

Entre las ventajas están la automatización y rapidez de procesamiento. Además existe un beneficio por haber escrito el programa en Python que cuenta con una licencia de código abierto. El código es versátil y puede adaptarse a otras cámaras con solo algunos ajustes en la preparación a las imágenes de entrada.

En este trabajo se confirmó que la tasa de emisión de SO_2 por el volcán Popocatépetl de hasta ≈ 40 kg/s, da lugar a tasas del orden de miles de toneladas liberadas diariamente. Es un valor alto respecto a los valores promedio calculados en otros volcanes, ~ 3 kg/s en Pacaya y ~ 8 kg/s en Ubinas.

El monitoreo volcánico remoto con cámaras SO_2 en conjunto con un software de procesamiento de datos que genera series de tiempo da la posibilidad de estudiar patrones de comportamiento en la desgasificación magmática. Estos datos pueden ayudar a predecir comportamientos futuros.

Recomendaciones

El trabajo realizado en ésta tesis se puede extender desde varios puntos de vista, se puede mejorar el hardware, optimizar el software y mejorar las aproximaciones a los modelos de transferencia radiativa para obtener densidades de columna del gas SO_2 más confiables. Eso implica tener conocimientos de óptica, electrónica, ciencias de la computación, química y física.

Algunas recomendaciones para trabajos futuros son los siguientes:

- Agregar la capacidad al software para dibujar mas perfiles de intensidades en el edificio volcánico y con ello recuperar coeficientes de esparcimiento más confiables.
- Modificar el software para dibujar mas líneas de integración y comparar cómo cambia el flujo en distintas regiones de la pluma.
- En este trabajo se adaptó el cálculo del flujo con la ley de Gauss usando las velocidades perpendiculares a la curva que rodea el crater, otra opción que podría implementarse es ajustar los campos vectoriales cerca del crater a funciones que permitan calcular la divergencia, y de esta manera se use el otro término en el teorema de la divergencia de Gauss.
- La detección de las regiones de cielo es importante para tener la información de la radiación difusa detrás de la pluma. Un trabajo interesante sería usar algoritmos de 'learning machine' del paquete Scikit-learn de Python.
- Es deseable unificar el programa que adquiere las imágenes de la cámara y el software que calcula el flujo de SO_2 escrito en este trabajo, y con ésto planear una estación fija de monitoreo permanente.
- El cálculo del flujo óptico funciona mejor si la deformación de la pluma entre imágenes sucesivas es moderado. Por lo tanto de debe adaptar la frecuencia de adquisición para tener desplazamientos de ~ 10 pixeles. O bien modificar los parámetros de la función que calcula el flujo óptico dentro del archivo de configuración.
- Parte del resultado de este trabajo son las series de tiempo correspondientes al flujo de SO_2, comparar datos de sismicidad y analizar las series con métodos estadísticos y transformadas de fourier es el siguiente paso para conocer las tendencias en el comportamiento de un volcán.

Bibliografía

[1] Toshiya Mori and Mike Burton. The SO2 camera: A simple, fast and cheap method for ground-based imaging of SO2 in volcanic plumes. *Geophysical Research Letters*, 33(24), 2006. L24804 >http://onlinelibrary.wiley.com/doi/10.1029/2006GL027916/full.

[2] C. Kern, F. Kick, P. Lübcke, L. Vogel, M. Wöhrbach, and U. Platt. Theoretical description of functionality, applications, and limitations of so_2 cameras for the remote sensing of volcanic plumes. *Atmospheric Measurement Techniques*, 3(3):733–749, 2010. >http://www.atmos-meas-tech.net/3/733/2010/.

[3] P. Lübcke, N. Bobrowski, S. Illing, C. Kern, J. M. Alvarez Nieves, L. Vogel, J. Zielcke, H. Delgado Granados, and U. Platt. On the absolute calibration of so_2 cameras. *Atmospheric Measurement Techniques*, 6(3):677–696, 2013. >http://www.atmos-meas-tech.net/6/677/2013/amt-6-677-2013.pdf.

[4] Robin Campion, Hugo Delgado-Granados, and Toshiya Mori. Image-based correction of the light dilution effect for SO2 camera measurements. *Journal of Volcanology and Geothermal Research*, 300:48 – 57, 2015.

[5] Murry L. Salby. *Physics of the Atmosphere and Climate*. Cambridge University Press, 2 edition, 2012.

[6] E. Thomas Gary and Stamnes Knut. *Radiative Transfer in the Atmosphere and Ocean*. Cambridge University Press, 1999.

[7] Annamaneni Peraiah. *An Introduction to Radiative Transfer: Methods and applications in astrophysics*. Cambridge University Press, 2002.

[8] Ulrich Platt and Jochen Stutz. *Diferential Optical Absorbtion Spectroscopy Principles and Applications*. Springer, 2008.

[9] Pedro Lilienfeld. A blue sky history. *Opt. Photon. News*, 15(6):32–39, Jun 2004.

[10] Irina N. Sokolik. Remote sensing of the atmosphere and oceans. pdf. http://irina.eas.gatech.edu/EAS_Fall2008/Lecture6.pdf.

[11] Beer–Lambert–Bouguer law. IUPAC Compendium of Chemical Terminology. [web], feb 2014. Recuperado de http://goldbook.iupac.org/B00626.html Consultado en septiembre de 2016.

[12] John E. Frederick. *Principles Of Atmospheric Science*. Jones & Bartlett Learning, 2007.

[13] Vicente Araña Saavedra and Ramon Ortiz Ramis. *Volcanologia*. Editorial Rueda, 1984.

[14] Edward J. Tarbuck and Frederick K. Lutgens. *Earth: an introduction to physical geology*. Pearson Education, 8th edition, 2005.

[15] Christiane Textor, Hans-F. Graf, Claudia Timmreck, and Alan Robock. *Emissions from volcanoes*, pages 269–303. Springer Netherlands, Dordrecht, 2004.

[16] Volcanic gases can be harmful to health, vegetation and infrastructure. [web], feb 2016. Recuperado de https://volcanoes.usgs.gov/vhp/gas.html Consultado en marzo de 2016.

[17] Ben Mills. Sulfur-dioxide-3d-vdw.png, 2007. Recuperado de https://commons.wikimedia.org/w/index.php?curid=969206 Descargado el 17 de marzo de 2016.

[18] Working on volcanoes. web. Recuperado de http://volcano.oregonstate.edu/book/export/html/168 Consultado en marzo de 2016.

[19] Por Horst Frank (GFDL o CC-BY-SA-3.0). Electromagnetic spectrum, 2005. Recuperado de https://commons.wikimedia.org/wiki/File:Electromagnetic_spectrum.svg, GFDL (http://www.gnu.org/copyleft/fdl.html) Descargado el 19 de marzo de 2016.

[20] Bo Galle, Mattias Johansson, Claudia Rivera, Yan Zhang, Manne Kihlman, Christoph Kern, Thomas Lehmann, Ulrich Platt, Santiago Arellano, and Silvana Hidalgo. Network for observation of volcanic and atmospheric change (novac)—a global network for volcanic gas monitoring: Network layout and instrument description. *Journal of Geophysical Research: Atmospheres*, 115(D5):n/a–n/a, 2010. D05304.

[21] R. D'Aleo, M. Bitetto, D. Delle Donne, G. Tamburello, A. Battaglia, M. Coltelli, D. Patanè, M. Prestifilippo, M. Sciotto, and A. Aiuppa. Spatially resolved so2 flux emissions from mt etna. *Geophysical Research Letters*, 43(14):7511–7519, 2016. 2016GL069938.

[22] G.J.S. Bluth, J.M. Shannon, I.M. Watson, A.J. Prata, and V.J. Realmuto. Development of an ultra-violet digital camera for volcanic SO2 imaging. *Journal of Volcanology and Geothermal Research*, 161(1–2):47 – 56, 2007.

[23] Toshiya Mori and Mike Burton. Quantification of the gas mass emitted during single explosions on stromboli with the SO2 imaging camera. *Journal of Volcanology and Geothermal Research*, 188(4):395 – 400, 2009.

[24] L. Vogel, B. Galle, C. Kern, H. Delgado Granados, V. Conde, P. Norman, S. Arellano, O. Landgren, P. Lübcke, J. M. Alvarez Nieves, L. Cárdenas Gonzáles, and U. Platt. Early in-flight detection of so_2 via differential optical absorption spectroscopy: a feasible aviation safety measure to prevent potential encounters with volcanic plumes. *Atmospheric Measurement Techniques*, 4(9):1785–1804, 2011.

[25] J. Kuhn, N. Bobrowski, P. Lübcke, L. Vogel, and U. Platt. A fabry-perot interferometer based camera for two-dimensional mapping of SO2 distributions. *Atmospheric Measurement Techniques Discussions*, 7(5):5117 – 5145, 2014.

[26] Non-linear least-squares minimization and curve-fitting for python, n.d. Recuperado de http://cars9.uchicago.edu/software/python/lmfit/builtin_models.html.

[27] how to smooth a curve in the right way, 2013. Recuperado de http://stackoverflow.com/q/20618804 Consultado el 9 de febrero de 2016.

[28] Simple loops to define finite difference derivatives. Recuperado de http://gilgamesh.cheme.cmu.edu/doc/software/jacapo/9-numerics/9.1-numpy/9.2-integration.html Consultado el 12 de febrero de 2016.

[29] Nial Peters, Alex Hoffmann, Talfan Barnie, Michael Herzog, and Clive Oppenheimer. Use of motion estimation algorithms for improved flux measurements using SO2 cameras. *Journal of Volcanology and Geothermal Research*, 300:58 – 69, 2015. >http://www.sciencedirect.com/science/article/pii/S0377027314002807.

[30] Allen Downey. *Think Python, How To Think Like a Computer Scientist*. Green Tea Press, Needham, Massachusetts, 2012. version 2.0.17 Recuperado en www.thinkpython.com.

[31] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[32] Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011. Disponible en http://aip.scitation.org/doi/pdf/10.1109/MCSE.2011.37.

[33] Scipy-0.18.1. web, sept 2015. https://docs.scipy.org/doc/scipy-0.16.0/reference/.

[34] Nicolas Rougier, Mike Müller, and Gaël Varoquaux. Matplotlib: plotting – scipy lecture notes. web, oct 2016. http://www.scipy-lectures.org/intro/matplotlib/matplotlib.html.

[35] R. Szeliski. Computer Vision: Algorithms and Applications, 2010. Recuperado de http://szeliski.org/Book.

Apéndice

Apéndice A

Lenguaje de programación Python

Python es un lenguaje de alto nivel, que facilita la programación, su sintaxis es limpia y tiene portabilidad, lo que significa que los programas escritos con este lenguaje pueden ser ejecutados en distintas plataformas con nula o casi ninguna modificación [30].

Tiene soporte e integración con otros lenguajes y es multiparadigma de programación. Tiene una licencia de código abierto PSFL (PYTHON Software Foundation License).

Los programas de PYTHON son ejecutados por un intérprete, el cual puede ser en modo interactivo o script, este último es el modo en que se ejecuta el programa escrito para el presente trabajo.

Este apéndice presenta algunas de las bibliotecas, módulos, tipos de datos y funciones incorporados en PYTHON que fueron utilizados durante el desarrollo del programa, haciendo posible el procesamiento de imágenes.

A.1. Paradigmas de programación

Existen dos paradigmas muy extendidos: la programación estructurada y la orientada a objetos. Cada uno de ellos es un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de un programa.

La programación orientada a objetos busca simular el mundo real a través de objetos con características y funciones (atributos y métodos). Un objeto es una combinación de variables locales y procedimientos que forman una

entidad de programación.

La programación estructurada esta compuesta de funciones, y módulos que tienen un inicio y un final. El código es ejecutado en orden hasta llegar al final. El flujo de instrucciones ejecutadas está gobernado por estructuras de control. Un programa diseñado con este paradigma es representado gráficamente con un diagrama de flujo.

Hay tres estructuras fundamentales. La estructura lineal es una secuencia de órdenes. La estructura selectiva consiste en ejecutar una orden u otra que depende de una condición. La estructura iterativa es utilizada para hacer repeticiones de una instrucción; también es común usar una condicional para evaluar cuantas veces debe iterar una instrucción o bloque de código.

A.2. math, NumPy y SciPy

El módulo math está incluido en la librería estandar de PYTHON, y proporciona funciones matemáticas básicas pero eficientes como la exponencial, logaritmos, funciones trigonométricas e hiperbólicas, y valores constantes como el número e y π.

Los módulos NumPy y SciPy [31, 32] están entre los mas difundidos para hacer computo científico en PYTHON. NumPy está especializado en cálculos numéricos entre arreglos de uno y mas dimensiones. Álgebra lineal, transformadas de fourier y funciones para obtener números aleatorios. SciPy [33] contiene funciones especiales, módulos para integración, optimización, interpolación transformadas de fourier, procesamiento de señales, estadística, etc.

A.2.1. Listas

Python tiene incorporado un tipo de dato llamado 'list' para almacenar secuencias y numpy tiene una versión 'numpy.array' con propiedades parecidas a las de 'list', pero con mas funciones, y más eficiente para el caso de vectores o arreglos de vectores con gran cantidad de datos.

Los elementos en una lista se separan con comas, y se encierran entre corchetes. El primer elemento tiene índice cero.

Ejemplos de métodos de un objeto tipo lista son:

- `list.append(elem)` – Agrega un elemento al final de la lista. No regresa una nueva lista, solo modifica la lista original.

- `list.insert(index,elem)` – Inserta un elemento en un índice dado, desplazando los elementos hacia la derecha.

Para obtener partes de una lista:

```
list = ['a','b','c','d']
print(list[1:-1]) # Imprime ['b','c']
print(list[-1]) # Imprime el último
                #elemento de la lista.
```

Una de las ventajas del lenguaje PYTHON es su legibilidad; en la creación de listas hay un método llamado 'comprensión de listas' que se utiliza para generar una lista de forma eficiente. Por ejemplo el código, ejecutado en un interprete interactivo de Python:

```
>>> for x in range(10):
...         cubos.append(x**3)
```

se reemplaza por la 'comprensión de lista' siguiente:
`cubos = [x ** 3 for x in range(10)]`

Una lista de comprensión consiste en una expresión seguida de la declaración `for` y luego cero o más declaraciones `for` o `if` encerrados por corchetes. El resultado es una nueva lista que sale de evaluar la expresión en el contexto de los `for` o `if` contenidos.

A.2.2. Operaciones con arreglos

El módulo numPy maneja arreglos de datos n-dimensionales. Arreglos que se pueden crear con numPy son:

- `identity(n,dtype)`. Devuelve la matriz identidad de $n \times n$, una matriz cuadrada nula excepto en su diagonal. `dtype` es el tipo de dato, por defecto se toma como flotante.

- `ones(shape,dtype)`. Crea un arreglo de unos, compuesto de `shape` elementos.

- `zeros(shape,dtype)`. Crea un arreglo de ceros, compuesto de `shape` elementos.

- `arange([start,]stop[,step,],dtype=None)`. Crea un arreglo con valores distanciados `step` entre el valor inicial `start` y el valor final `stop`. `step = 1` por defecto.
- `linspace(start,stop,num,endpoint=True)`. Crea un arreglo con valor inicial `start`, valor final `stop` y `num` elementos.

Sea `A = numpy.array([1,2],[3,4])`, las operaciones básicas como rotaciones y transpuestas, se logran con `rot90(A,1)` y `A.T`. Las operaciones básicas como suma y multiplicación se hacen elemento a elemento. Para hacer operaciones de matrices se usa `A=numpy.matrix([1,2],[3,4])`.

A.3. matplotlib.pyplot, opencv

Matplotlib es el paquete por defecto en PYTHON para gráficas 2D, sus módulos están diseñados para la visualización e interpretación de datos, las figuras resultantes en distintos formatos son de alta calidad. 'Pyplot' es la interfaz para la biblioteca de objetos de matplotlib, las posibilidades que ofrece son inmensas, varias de sus funciones pueden verse en [34]. Aquí solo se describen algunas funciones vitales para el desarrollo de esta tesis, usando un ejemplo para visualizar una gráfica del espectro de emisión que se muestra en la Figura 1.4.

OpenCV ofrece funciones similares a matplotlib, además de algoritmos especializados para el procesamiento de imágenes bidimensionales, por ejemplo la morfología de imágenes binarias, gaussianas y laplacianos, cálculo de flujo óptico y otros.

A.4. tkinter, configparser

Están incluidas en la biblioteca estándar de PYTHON, tkinter es útil en la creación de una interfaz gráfica de usuario. Esta característica es importante para un usuario porque oculta el extenso código del programa y permite modificar los parámetros importantes.

`configparser` es otro módulo de PYTHON que facilita el manejo de archivos de configuración. Combinando tkinter y configparser se simplifica la interacción del usuario con el programa.

A.5. Codificando en Python

Escribir código en PYTHON puede requerir algunos trucos, que hagan mas fácil y presentable su lectura. Aquí algunos consejos adquiridos por la experiencia en este trabajo.

- Los comentarios en el código pueden ser de una línea comentados con #, o de varias líneas entre comillas ‘‘‘ ’’’.
- Para cortar una línea sin repercursión en la sintaxis o identación reconocida por el interprete de PYTHON, se usa un ‘backslash’ \. Para unir dos lineas, usar punto y coma ‘;’
  Por ejemplo `plt.title('I310');plt.contourf(I310,40)`
- El símbolo ‘underscore’ _ es usado convencionalmente para desechar valores que son arrojados por funciones y no serán necesarios más adelante. por ejemplo un histograma:
  `hist, bins, __ = numpy.histogram(argumentos)`
- Si se hace matriz **U** > matriz **V** se produce una matriz con entradas lógicas (True, False) → (1,0).
- Las funciones con el módulo `math` son mas rápidas que las de `NumPy` cuando se trata de entradas escalares.
- Se debe tener cuidado cuando se usan referencias a la memoria que corresponde a una variable.

  ```
  d=[2,3,4]
  a=d # a y d apuntan a un mismo dato.
  a[0]=1
  ```

  entonces `d = [1,3,4]`, si el resultado deseado es que `a` y `d` sean independientes, hay que usar `a = d.copy()`.

  Cuando se hace una operación a la variable `d` y se guarda en `a`; por ejemplo haciendo `b = d[1:2]` entonces ya no es necesario usar `copy()`, porque se vuelven independientes, es decir hacen referencia a distintas regiones de memoria, entonces `d = [2,3,4]` y `b = [3]`.
- Es más eficiente usar `[i,j]` que `[i][j]` para acceder a un elemento de un arreglo bidimensional en NumPy.

- Convertir una estructura tipo lista `[1,2,3]` a un arreglo de NumPy y viceversa:

```
arregloNumPy = numpy.array( [1,2,3] )
listaNativaPython = arregloNumPy.tolist()
```

- El arreglo correspondiente a una imagen puede ser visualizado de dos maneras:
  para un arreglo `A = numpy.array([[a00, a01],[a10, a11]])`, con `plt.imshow(A)` se despliega como `[[a00, a01],[a10, a11]]` mientras que con `plt.contourf` se despliega como
  `[[a10, a11],[a00, a01]]` es decir las filas se invierten. Si se desea que `contourf` tenga el mismo comportamiento que `imshow` se usa la instrucción `plt.gca().invert_yaxis()`.

A.6. Procesamiento de imágenes

Los métodos para procesar imágenes son complejos. Se basan en modelos probabilísticos y consideraciones físicas [35]. Revisando estos temas podemos darnos una idea de cuanto se subestima la capacidad visual humana. Problemas para niños de dos años como contar el número de animales de cierta especie en una imagen, siguen siendo un reto para la computación visual.

Se usaron algoritmos implementados en el módulo OpenCV. Blur para suavisar las imágenes, y el operador laplaciano para remarcar bordes. Esto fue útil para emparejar un par de imágenes tal que las coordenadas de las distintas regiones de pixeles coincidan.

Se aplicaron herramientas de morfología de imágenes binarias, las cuales consisten en regiones de ceros y unos, a los que se les puede aplicar operaciones como *closing*, tal que las regiones de unos se ensanchan para abarcar regiones pequeñas de ceros. Otra herramienta esencial fue calcular histogramas de las intensidades en cada imagen. Con esta técnica se pueden identificar patrones correspondientes a las regiones sobresalientes de una imagen, es decir de regiones que pertenecen a rangos de intensidades vecinos, como el cielo y el volcán. También se usó el algoritmo de Farneback extraído de OpenCV, el cual fue programado para calcular el flujo óptico entre dos imágenes.

Apéndice B

Programa principal

```
#!/usr/bin/env python
"""
    LookingForSky.py
    @author Salvador Pedraza Espitia
    @version 1.0
"""
import matplotlib.pyplot as plt
import matplotlib.ticker as tck
import numpy as np
import cv2
import glob
import os
import fileinput
import time, gc
from scipy.optimize import curve_fit
from scipy.signal import savgol_filter
import scipy.ndimage
from tkinter import *
import ast
import re
import warnings
import math

from scipy.interpolate import interp1d, interp2d
import numpy
import collections

try:
    from configparser import ConfigParser
except ImportError:
    from ConfigParser import ConfigParser  # ver. < 3.0

class FluxEngineBase(object):
    def __init__(self, config):
        """
        Base class for flux engine classes. Creates integration lines based on
        the values in config. Subclasses must override the compute_flux method.
        """
        self._int_lines = []
        xpts = config[0]
        ypts = config[1]
        print (xpts)
        print (ypts)
        self._int_lines.append(IntegrationLine( 'line', xpts, ypts, 1 ))

        self._conversion_factor = 0.00000266# convierte ppm m  a  Kg/m**2
        self._low_pix_threshold = 0

    def get_integration_lines(self):
        return self._int_lines



    def compute_flux(self, current_image, flow, delta_t):
        raise TypeError("FluxEngineBase must be subclassed, and the "
                        "compute_flux() method should be defined in the "
                        "subclass.")

```

```
class FluxEngine1D(FluxEngineBase):

    def __init__(self, config):
        """
        One dimensional flux calculation engine. This works by multiplying each
        pixel on the integration line by its corresponding velocity and then
        integrating along the line. This is the "traditional" flux calculation
        method.
        """
        super(FluxEngine1D, self).__init__(config)


    def compute_flux(self, current_image, next_image,flow, lpix,regionPlumaCielo, AA,delta_t,angle,distancia,num):
        """
        Returns the fluxes and associated errors across each defined integration
        line.
        """
        pix_errors = compute_error_map(current_image, next_image, flow)


        #convert the displacements to velocities in m/s
        flow[...,0] *= ( regionPlumaCielo*lpix / delta_t)
        flow[...,1] *= ( regionPlumaCielo*lpix / delta_t)

        U = flow[...,0]
        V = flow[...,1]

        # fig = plt.figure(1)
        # fig.set_size_inches(8,6)
        # ax = fig.add_subplot(121)
        # ax.set_title(r'Velocidades(flow) $\vec{v}$')
        # #ax.contourf(numpy.rot90(regionPlumaCielo*AA,-1).T,30)
        # ax.quiver(U[0:27*18:18,0:26*18:18],V[0:27*18:18,0:26*18:18],scale=200,width = 0.0016)
        # # set ticks and tick labels
        # ax.set_xlim((0, 26))
        # ax.set_xticks(numpy.arange(0,26,4))
        # #plt.subplots_adjust(bottom=0.15)#en caso de que xticks sean verticales y no se corten.
        # ax.set_xticklabels(numpy.arange(0,26*18,18*4))
        # ax.set_ylim((0, 27))
        # ax.set_yticks(numpy.arange(0,27,4))
        # ax.set_yticklabels(numpy.arange(0,27*18,18*4))
        # ax.axis('scaled')
        # #plt.xticks(numpy.arange(0,26*18,18)); plt.yticks(())
        # ax.invert_yaxis()
        # ax2 = fig.add_subplot(122)
        # ax2.set_title(r'AA/$k_\alpha$')
        # levels = numpy.arange(0,numpy.amax(AA),50)
        # rr = ax2.contourf(numpy.rot90(regionPlumaCielo*AA,-1).T, levels = levels, cmap=plt.get_cmap('plasma'), extend="both")#ax2.imsho
        # ax2.axis('scaled')
        # cbar_rr = fig.colorbar(rr)
        # cbar_rr.ax.set_ylabel('Densidad de columna [ppmm]')
        # plt.savefig(num[1]+'Ubin'+"quivvelocidades"+str(num[0])+".png", dpi = 200, bbox_inches='tight', pad_inches=0);plt.clf()

#         #----Graficar magnitudes de velocidades usando contourf----#
#         fig = plt.figure(1)
#         fig.set_size_inches(6,4.6)
#         ax = fig.add_subplot(111)
#         ax.set_title(r'Magnitudes de velocidades $|\vec{v}\,|$')
```

```
117 #         ax.invert_yaxis()
118 #         Z1 = numpy.sqrt(U*U+V*V)
119 #         rrb = ax.pcolormesh(Z1, norm=colors.PowerNorm(gamma=1./5.),cmap=plt.cm.plasma)
120 #         cbb = fig.colorbar(rrb,ticks=[0,0.2,0.5,1,2,5,11,32,64])
121 #         plt.savefig(pathTemp+prefix+"velocidades3.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
122 
123         #interpolate the current image and error values to find the values at
124         #the points on the integration line
125 
126         image_interp = interp2d(numpy.arange(current_image.shape[1]),
127                                 numpy.arange(current_image.shape[0]),
128                                 AA, copy=False, fill_value=0.0)
129 
130         errors_interp = interp2d(numpy.arange(current_image.shape[1]),
131                                 numpy.arange(current_image.shape[0]),
132                                 pix_errors, copy=False, fill_value=0.0)
133 
134         #interpolate the flow field to find the velocities on the integration
135         #line
136         x_flow_interp = interp2d(numpy.arange(flow.shape[1]),
137                                  numpy.arange(flow.shape[0]),
138                                  flow[...,0], copy=False, fill_value=0.0)
139 
140         y_flow_interp = interp2d(numpy.arange(flow.shape[1]),
141                                  numpy.arange(flow.shape[0]),
142                                  flow[...,1], copy=False, fill_value=0.0)
143 
144         #now compute the flux for each integration line in the list
145         for integration_line in self._int_lines:
146 
147             #work out how many segments to split the integration line into, to get
148             #approximately single pixel resolution
149             line_length = integration_line.get_length()
150             number_of_segments = int(round(line_length, 0))
151 
152             int_pts, int_vecs, int_norms = integration_line.get_poly_approx(n=number_of_segments)
153 
154 
155             #------------ get length of segments -------------
156             cols = current_image.shape[1]
157             rows = current_image.shape[0]
158 
159             distancias_x = numpy.zeros(cols)
160             distancias_y = numpy.zeros(rows)
161             for i in range(cols):
162                 distancias_x[i] = distancia * ( math.cos( math.radians( (float(i)-(cols/2.))*23./512 ) ) +\
163                                       ( math.sin( math.radians( (float(i)-(cols/2.)) * 23./512 )) / math.tan( math.radians( angle - (fl
164             for i in range(rows):
165                 distancias_y[i] = distancia *  math.cos( math.radians( (float(i)-(rows/2.))*23./512 ) ) * math.radians(23.0)/512
166 
167             lxpix = numpy.zeros(cols)
168             lypix = numpy.zeros(rows)
169             lxpix = numpy.cumsum(distancias_x)
170             lypix = numpy.cumsum(distancias_y)
171 
172             interplx = interp1d(numpy.arange(cols),lxpix)
173             interply = interp1d(numpy.arange(rows),lypix)
174 
```

```
175             realx = interplx(int_pts[:,0])
176             realy = interplx(int_pts[:,1])
177             length_of_segments = numpy.zeros_like(realx,numpy.float)
178             length_of_segments[1:] =  numpy.sqrt((realx[1:]-realx[:-1])**2 + (realy[1:]-realy[:-1])**2)
179 
180             #calculate the mid-points of each line segment
181             mid_points = int_pts + (0.5*int_vecs)
182 
183             pix_vals = numpy.zeros((number_of_segments,), dtype='float')
184             err_vals = numpy.zeros((number_of_segments,), dtype='float')
185             velocities = numpy.zeros((number_of_segments,2), dtype='float')
186 
187             #note that we have to do this in a for loop rather than calling the
188             #interpolator with arrays of values - because the mid-points may not be
189             #sorted (or even sortable - e.g. the x coordinates may increase as the
190             #y coordinates decrease)
191             x_coord2=numpy.zeros(number_of_segments)
192             y_coord2=numpy.zeros(number_of_segments)
193 
194             for i in range(number_of_segments):
195                 x_coord = mid_points[i,0]
196                 y_coord = mid_points[i,1]
197                 pix_vals[i] = image_interp(x_coord, y_coord)
198                 err_vals[i] = errors_interp(x_coord, y_coord)
199                 velocities[i,0] = x_flow_interp(x_coord, y_coord)
200                 velocities[i,1] = y_flow_interp(x_coord, y_coord)
201                 ##print(pix_vals[i],velocities[i,0],velocities[i,1])
202                 x_coord2[i]=mid_points[i,0]
203                 y_coord2[i]=mid_points[i,1]
204 
205 
206             #find component of velocities perpendicular to integration line
207             perp_vel = numpy.diag(numpy.dot(velocities, int_norms.T))
208 
209             #caculate flux
210             line_length_metres = line_length
211             line_length2 = numpy.sum(lpix[numpy.array(y_coord2).astype(numpy.int), numpy.array(x_coord2).astype(numpy.int)])
212             fluxes = pix_vals * self._conversion_factor * perp_vel * length_of_segments
213             ###print('velocidades',perp_vel)
214             #print('fluxes',fluxes)
215 
216             #sanity check
217             assert fluxes.shape == (number_of_segments,),"incorrect shape for fluxes array %s,\
218 expecting %s"%(str(fluxes.shape), str((number_of_segments,)))
219             #only sum the values with pixel values above the low pixel threshold
220             if self._low_pix_threshold > -1:
221                 ###print('pix_vals > -1')
222                 mask = numpy.where(pix_vals > self._low_pix_threshold)
223                 total_flux = numpy.sum(fluxes[mask])
224 
225                 #total error is quadrature sum of pixel errors
226                 error = math.sqrt(numpy.sum(((err_vals[mask]/100.0) * fluxes[mask])**2))
227                 error = (error/total_flux) * 100.0 #convert to percentage
228             else:
229                 total_flux = numpy.sum(fluxes)
230                 error = math.sqrt(numpy.sum(((err_vals/100.0) * fluxes)**2))
231                 error = (error/total_flux) * 100.0 #convert to percentage
232 
```

```
233         return total_flux, error
234 
235 class IntegrationLine:
236     def __init__(self, name,xpts, ypts, direction):
237         """
238         Class to represent an integration line in an image. Note that the xpts and ypts
239         are *not* in pixel coordinates, they are in line coordinates - which are pixel boundary
240         coordinates where (0,0) is the top left of the image.
241 
242         For example, for a 512x512 pixel image, if you want a horizontal integration line
243         that spans the entire width of the image you would specify its x points as
244         [0, 513].
245 
246         Integration lines can extend beyond the boundaries of the image without
247         problems.
248         """
249         self.name = name
250 
251         if len(xpts) != len(ypts):
252             raise ValueError("xpts and ypts must contain the same number of elements")
253 
254         if len(xpts) < 2:
255             raise ValueError("At least 2 points are required for interpolation")
256 
257         self.__xpts = numpy.array(xpts, dtype='float')
258         self.__ypts = numpy.array(ypts, dtype='float')
259 
260         self.direction = direction
261         self.__dist = numpy.zeros_like(self.__xpts)
262         self.__dist[1:] = numpy.cumsum(numpy.sqrt((self.__xpts[1:]-self.__xpts[:-1])**2 + (self.__ypts[1:]-self.__ypts[:-1])**2))
263 
264 
265         self._interp_x = interp1d(self.__dist, self.__xpts, kind='linear')
266         self._interp_y = interp1d(self.__dist, self.__ypts, kind='linear')
267 
268 
269 
270 
271 
272     def get_length(self):
273         """
274         Returns the cumulative length of all the line segments in pixels (not
275         in metres!)
276         """
277         return self.__dist[-1]
278 
279 
280     def get_n_points(self, n=-1):
281         """
282         Returns an n by 2 array of of x y coordinates of points along the line.
283         If n==-1 then it returns the actual points of the line rather than
284         interpolating. Note that the points returned are in pixel coordinates,
285         *not* line coordinates.
286         """
287         if n == -1:
288             result = numpy.zeros((len(self.__xpts), 2), dtype='float')
289             result[:,0] = self.__xpts - 0.5
290             result[:,1] = self.__ypts - 0.5
```

```
291
292         else:
293
294             result = numpy.zeros((n,2), dtype='float')
295
296             pts = numpy.linspace(0, self.__dist[-1],n)
297
298             #print(self._interp_x(pts))
299             result[:,0] = self._interp_x(pts) - 0.5
300             result[:,1] = self._interp_y(pts) - 0.5
301             #print(result[:,0])
302
303         return result
304
305
306
307     def get_poly_approx(self, n=-1):
308         """
309         Returns a tuple of arrays (startpoints, vectors, normals). Note that the
310         points returned are in pixel coordinates, *not* line coordinates.
311         """
312         if n < 1 and n != -1:
313             raise ValueError("n must be greater than 1 (or -1 for defaults)")
314
315         if n == -1:
316             pts = self.get_n_points(n)
317
318         else:
319             pts = self.get_n_points(n+1)
320
321         vectors = pts[1:] - pts[:-1]
322
323         normals = numpy.cross(vectors, numpy.array([0.0,0.0,self.direction]))
324         normals /= numpy.sqrt((normals ** 2).sum(-1))[..., numpy.newaxis]
325         normals = numpy.array(normals[...,:2]) #only return the xy components of the normals
326
327         return pts[:-1], vectors, normals
328
329
330 class MotionEngine(object):
331     def __init__(self, config):
332         self.pyr_scale = config['pyr_scale']
333         self.levels = config['levels']
334         self.winsize = config['winsize']
335         self.iterations = config['iterations']
336         self.poly_n = config['poly_n']
337         self.poly_sigma = config['poly_sigma']
338
339     def compute_flow(self, current_image, next_image):
340         """
341         Uses the Farneback algorithm to compute the flow field that maps pixels
342         in current_image, to pixels in next_image. Both image arguments should be
343         numpy arrays.
344         """
345         return cv2.calcOpticalFlowFarneback(current_image, next_image,None,#recordar que en opencv python 2.7 No va None.
346                                             self.pyr_scale,
347                                             self.levels,
348                                             self.winsize,
```

```
                                        self.iterations,
                                        self.poly_n,
                                        self.poly_sigma,
                                        flags=cv2.OPTFLOW_FARNEBACK_GAUSSIAN)


def compute_error_map(current_image, next_image, flow):
    """
    This function attempts to estimate the error associated with an image pair
    and their corresponding motion field. It works by considering conservation of
    mass. For each pixel in current_image, it uses the motion vector to compute
    where the mass of SO2 in the pixel should end up - eventually creating a
    "modelled next image". This is then compared to the real next image to
    estimate the error. The motion field is then used to map the errors back
    onto their source pixels in current_image.
    """
    flat_len = numpy.prod(current_image.shape)

    flat_error_map = numpy.zeros(flat_len, dtype='float')

    src_coords = numpy.dstack(numpy.meshgrid(numpy.arange(current_image.shape[0]),numpy.arange(current_image.shape[1]),\
                                         indexing='ij'))

    primary_dest_coords = src_coords + flow[...,::-1] #reverse x and y flows since we are dealing with row and columns now

    #calculate the fractional contribution to each destination pixel
    fraction = ((primary_dest_coords + 1).astype('int') - primary_dest_coords)

    #convert the destination coordinates into coordinates for a flattened image
    flat_dest_coords = (primary_dest_coords[...,1].astype('int').ravel() +
                        primary_dest_coords[...,0].astype('int').ravel() * current_image.shape[1])

    #add up the various contributions. Note that this is slightly complicated by the fact
    #that multiple pixels in the current image can contribute to the pixels in the next
    #image. This means that our dest_coords array can contain duplicate entries, and so
    #we have to use the bincount function to add it to our error map.

    #11 contribution
    dest_coords = flat_dest_coords
    valid_mask = numpy.where(numpy.logical_and(dest_coords > 0, dest_coords < flat_len))
    contribution = (current_image * fraction[...,0] * fraction[...,1]).ravel()
    flat_error_map += numpy.bincount(dest_coords[valid_mask],weights=contribution[valid_mask], minlength=flat_len)

    #12 contribution
    dest_coords = flat_dest_coords + 1 #plus one column
    valid_mask = numpy.where(numpy.logical_and(dest_coords > 0, dest_coords < flat_len))
    contribution = (current_image * fraction[...,0] * (1.0 - fraction[...,1])).ravel()
    flat_error_map += numpy.bincount(dest_coords[valid_mask], weights=contribution[valid_mask], minlength=flat_len)

    #21 contribution
    dest_coords = flat_dest_coords + current_image.shape[1] #plus one row
    valid_mask = numpy.where(numpy.logical_and(dest_coords > 0, dest_coords < flat_len))
    contribution = (current_image * (1.0 - fraction[...,0]) * fraction[...,1]).ravel()
    flat_error_map += numpy.bincount(dest_coords[valid_mask], weights=contribution[valid_mask], minlength=flat_len)

    #22 contribution
    dest_coords = flat_dest_coords + current_image.shape[1] + 1#plus one row and one col
    valid_mask = numpy.where(numpy.logical_and(dest_coords > 0, dest_coords < flat_len))
```

```
407     contribution = (current_image * (1.0 - fraction[...,0]) * (1.0 - fraction[...,1])).ravel()
408     flat_error_map += numpy.bincount(dest_coords[valid_mask], weights=contribution[valid_mask], minlength=flat_len)
409 
410     error_map = flat_error_map.reshape(current_image.shape)
411 
412     error_map -= next_image
413 
414     percentage_errors = numpy.zeros_like(error_map)
415     valid_mask = numpy.where(next_image != 0)
416     percentage_errors[valid_mask] = numpy.abs(100 * error_map[valid_mask] / next_image[valid_mask])
417 
418     #error map now contains the computed error in next_image following the motion
419     #however, what we want is a map of where this error originated from (i.e.
420     #the error in current_image). So, we now invert the error map, splitting the
421     #error in each pixel in next_image between the pixels in current_image that
422     #contributed to it.
423     flat_err_map = percentage_errors.ravel()
424 
425     flat_inverted_error_map = numpy.zeros_like(flat_err_map)
426 
427     #11 contribution
428     dest_coords = flat_dest_coords
429     valid_mask = numpy.where(numpy.logical_and(dest_coords > 0, dest_coords < flat_len))
430     flat_inverted_error_map[valid_mask] += (fraction[...,0] * fraction[...,1]).ravel()[valid_mask] * flat_err_map[dest_coords[valid_mask]
431 
432     #12 contribution
433     dest_coords = flat_dest_coords + 1 #plus one column
434     valid_mask = numpy.where(numpy.logical_and(dest_coords > 0, dest_coords < flat_len))
435     flat_inverted_error_map[valid_mask] += (fraction[...,0] * (1.0 - fraction[...,1])).ravel()[valid_mask] * flat_err_map[dest_coords[val
436 
437     #21 contribution
438     dest_coords = flat_dest_coords + current_image.shape[1] #plus one row
439     valid_mask = numpy.where(numpy.logical_and(dest_coords > 0, dest_coords < flat_len))
440     flat_inverted_error_map[valid_mask] += ((1.0 - fraction[...,0]) * fraction[...,1]).ravel()[valid_mask] * flat_err_map[dest_coords[val
441 
442     #22 contribution
443     dest_coords = flat_dest_coords + current_image.shape[1] + 1#plus one row and one col
444     valid_mask = numpy.where(numpy.logical_and(dest_coords > 0, dest_coords < flat_len))
445     flat_inverted_error_map[valid_mask] += ((1.0 - fraction[...,0]) * (1.0 - fraction[...,1])).ravel()[valid_mask] * flat_err_map[dest_co
446 
447 
448     #returns % errors.
449     return flat_inverted_error_map.reshape(current_image.shape)
450 
451 
452 def create_flux_engine(config):
453     '''
454 Aqui puedo definir otro algoritmo para el calculo del flujo.
455     '''
456     return FluxEngine1D(config)
457 
458 def procesaunpar(img0, img1 , motion_engine,flux_engine,lpix,regionPlumaCielo,AA,delta_t,angle,distancia, num):
459     flow = motion_engine.compute_flow( img0, img1 )
460     #aqui codigo para Desplegar campo vectorial!
461     integration_mask = numpy.zeros_like( img0 )
462     current_masked_im = img0
463     next_masked_im = img1
464     #only include pixels in the non-masked regions in the flux calculation
```

```
    current_masked_im *= numpy.logical_not(integration_mask)#*=regionPlumaCielo
    next_masked_im *= numpy.logical_not(integration_mask)#*=regionPlumaCielo
    flujo_so2_calculado = flux_engine.compute_flux(current_masked_im, next_masked_im, flow,lpix,regionPlumaCielo,AA,delta_t,angle,distanc
    return flujo_so2_calculado

def corr_plume_intensity(Iuncorr,I_A,sigma,d):
        Icorr = np.zeros(np.shape(Iuncorr))
        for i in np.arange( np.shape(d)[0] ):
                for j in np.arange( np.shape(d)[1] ):
                        Icorr[i,j] = ( Iuncorr[i,j] - I_A * ( 1 - math.exp( sigma*d[i,j] ) ) ) / \
                                ( math.exp( sigma*d[i,j] ) )
        #Icorr = ( Iuncorr - I_A * ( 1 - np.exp( sigma*d ) ) ) / ( np.exp( sigma*d ) )
        return Icorr

def matchmatrix(raw1,raw2,mini,maxi,minj,maxj,dname):
        M1=np.fromfile(dname+raw1,dtype=np.uint16)
        M1=M1.reshape(512,512)
        M2=np.fromfile(dname+raw2,dtype=np.uint16)
        M2=M2.reshape(512,512)
        M1=M1.T
        M2=M2.T
        M1=M1[5:511]
        M2=M2[5:511]
        M2=np.rot90(M2,2)
        b=np.shape(M1)
        M1=cv2.Laplacian(cv2.blur(M1,(5,5)),cv2.CV_16U)
        M2=cv2.Laplacian(cv2.blur(M2,(3,3)),cv2.CV_16U)
        subset2=M2[abs(mini):(b[0]-1-abs(maxi)),abs(-minj):(b[1]-1-abs(maxj))]
        Smldelta=9999999
        matrixcor=np.zeros([maxi-mini+1,maxj-minj+1],dtype=np.float32)
        for i in range(mini,maxi+1):# puse maxi+1, para que i llegue hasta i==maxi
                for j in range(minj,maxj+1):
                        subset1=M1[-mini+i:(b[0]-1-maxi+i),-minj+j:(b[1]-1-maxj+j)]
                        delta=np.sum(abs(subset2-subset1))/(b[0]*b[1])

                        matrixcor[i-mini,j-minj]=delta
                        if delta < Smldelta:
                                Smldelta=delta
                                igood=i
                                jgood=j
        match=[-igood,-jgood]
        return match

def shiftx(x):
    return 0.5*(x[1:]+x[:-1])

def enderezayrecorta(M1,M2,BiasAve310,BiasAve330,VigAve310,VigAve330,offset):#incluye correccion de vignetting.
    M2 = np.rot90(M2,-1)
    M2 = M2[5:511]
    M1=np.rot90(M1)
    M1=M1[1:507]

    M1 = np.rot90(M1.T)

    M1=M1-BiasAve310
    VigAve310 = np.rot90(VigAve310.T)
    M1=np.divide(M1,VigAve310)

```

```
523     M2 = np.rot90(M2.T)
524     M2 = M2-BiasAve330
525     VigAve330 = np.rot90(VigAve330.T)
526 
527     M2=np.divide(M2,VigAve330)#imgs oct 2, 2015.
528     M1=M1[0-(offset[0] < 0)*offset[0]:506-(offset[0] > 0)*offset[0],0-(offset[1] < 0)*offset[1]:512-(offset[1] > 0)*offset[1]]
529     M2=M2[0+(offset[0] > 0)*offset[0]:506+(offset[0] < 0)*offset[0],0+(offset[1] > 0)*offset[1]:512+(offset[1] < 0)*offset[1]]
530 
531 
532     return M1,M2
533 class Histograma:
534     """Defino la clase Histograma, para obtener un histograma y las propiedades
535     que me interesan: maximos locales LML, etc.
536     """
537 
538     def __init__(self,data,nbins,wl = 5 , po = 2):
539         self.hist, self.bins = np.histogram(data[ data !=0 ].ravel(), nbins, density=True)
540         self.binz = shiftx ( self.bins )    #binz es un desplazamiento en los
541                                             #compartimentos para que pueda graficarse
542         self.max = max(self.hist)
543         self.LML = max_of (savgol_filter(self.hist,wl,po, mode = 'nearest' ))
544 
545     def MaxList(self,frac=0.1):
546         return self.LML[ self.hist[ self.LML ] > frac*self.max ]
547 
548 def split_slash(cad):
549     return re.split('\W+',cad)
550 
551 def max_of(data):
552     return (np.diff(np.sign(np.diff( data ))) < 0).nonzero()[0] + 1
553 
554 def indxMax_of(valores,lista_indices):
555     valor_max = max(valores[lista_indices])
556     return lista_indices[ np.where (valores[lista_indices] == valor_max)[0][0] ]#regresa un indice
557 
558 def ajuste2d(region, absorb, rows, cols):
559     X, Y = np.where( region != 0 )
560     z = absorb[X,Y].flatten()
561     coef, r, rank, s = np.linalg.lstsq( np.array([X*0+1, X, Y]).T , np.array(z) )
562     m = np.linspace(0,rows,rows,0)
563     n = np.linspace(0,cols,cols,0)
564     M,N = np.meshgrid(m,n,indexing='ij')
565     return coef[0] + coef[1]*M + coef[2]*N
566 
567 ##Funcion para llenar una estructura
568 def estructura(cincoNombres,rows,cols,path,BiasAve310,BiasAve330,VigAve310,VigAve330,offset):#normalmente recibe 5 nombres de archivo.
569     ##composicion de una variable: guarda un par de imagenes con sus respectivos datos
570     #de exposicion. y el ID es time.
571     pairtype=np.dtype([('time',int),('m310',np.float64,(rows,cols)),\
572                     ('m330',np.float64,(rows,cols)),('expos310',float),('expos330',float)])
573     cinco=len(cincoNombres)
574     secuencia=np.zeros((cinco,),dtype=pairtype)  #definimos una secuencia de tipo pairtype
575     ##Lenar secuencia con 5 pares:
576     for i in range(cinco):
577         #print ('loading...',cincoNombres[i])
578         tiempo=int(cincoNombres[i].split('.')[0].strip())
579         #start=time.clock()
580         ##Cargando un par > de imagenes raw
```

```
581         M1=np.fromfile(path+cincoNombres[i]+'-1.raw',dtype=np.uint16).reshape(512,512)
582         M2=np.fromfile(path+cincoNombres[i]+'-2.raw',dtype=np.uint16).reshape(512,512)
583 
584         M310,M330=enderezayrecorta(M1,M2,BiasAve310,BiasAve330,VigAve310,VigAve330,offset)
585         #llenamos tiempo de exposicion con info de archivos txt
586         for ind,line in enumerate(fileinput.input(path+str(cincoNombres[i])+"-I.txt")):
587             if ind == 1:
588                 expos310=float(line.split('=')[1].strip())
589             if ind == 5:
590                 expos330=float(line.split('=')[1].strip())
591         ##LLENANDO PAR [i]
592         secuencia[i] = np.array([(tiempo,M310,M330,expos310,expos330)], dtype=pairtype)
593     return secuencia
594 def profilecoords(img310):
595         coords = []
596         fig=plt.figure(1)
597         plt.title('Marca el primer punto del perfil')
598         plt.contourf(img310,40)
599         plt.colorbar()
600         contour_axis = plt.gca()
601 
602         def onclick(event,n):
603                 coords.append((event.xdata, event.ydata))
604                 contour_axis.clear()
605                 contour_axis.contourf(img310,40)
606                 xx,yy = zip(*coords)
607                 plt.title('Marca el segundo punto, y cierra esta ventana.')
608                 plt.plot(xx,yy, 'w',linestyle=':', marker='o')
609                 plt.draw()
610                 if len(coords) == n:
611                         fig.canvas.mpl_disconnect(interactividad)
612                         #plt.close(1)
613         interactividad = fig.canvas.mpl_connect('button_press_event', lambda event: onclick(event,2))
614         plt.show()
615         winProfile=Tk()
616         winProfile.geometry("250x100+250+200")
617         winProfile.title(" Check P0, P1 ")
618         config = ConfigParser()
619         config.read('config.ini')
620         P0 = config.getfloat('datosVolcan','profile d0')
621         P1 = config.getfloat('datosVolcan','profile d1')
622         Label(winProfile, text = 'P0').place(x=5 , y=5)
623         Label(winProfile, text = 'P1').place(x=5 , y=35)
624         varP0 = DoubleVar(winProfile)
625         varP1 = DoubleVar(winProfile)
626         varP0.set( P0 )
627         varP1.set( P1 )
628         Entry( winProfile, textvariable = varP0 ).place( x=65 , y=5 )
629         Entry( winProfile, textvariable = varP1 ).place( x=65 , y=35 )
630         botOnK = Button ( winProfile, text=" Ok ", command = winProfile.destroy ).place(x=100,y=70)
631         winProfile.mainloop()
632 
633         #Leyendo datos del Volcan desde el archivo de configuracion
634         config.set('datosVolcan','profile d0', str(varP0.get()))
635         config.set('datosVolcan','profile d1', str(varP1.get()))
636         with open('config.ini', 'w') as configfile:
637                 config.write(configfile)
638 
```

```
639         x0,y0 = coords[0]
640         x1,y1 = coords[1]
641         length_profile = int(np.hypot(x1-x0, y1-y0))
642         x, y = np.linspace(x0, x1, length_profile), np.linspace(y0, y1, length_profile)
643         x=x.astype(np.int)
644         y=y.astype(np.int)
645 
646         return x,y,length_profile,varP0.get(),varP1.get()
647 
648 def coords_points(img310,mpoints):
649         img310 = numpy.rot90(img310,-1).T
650         coords = []
651         fig=plt.figure(1)
652         plt.title('Dibuja 6 puntos alrededor del crater')
653         plt.contourf(img310,40)#imshow(img310)
654         plt.gca().invert_yaxis()
655         plt.colorbar()
656         contour_axis = plt.gca()
657 
658         def onclick(event,n):
659                 coords.append((event.xdata, event.ydata))
660                 contour_axis.clear()
661                 contour_axis.contourf(img310,40)
662 
663                 xx,yy = zip(*coords)
664                 if (6-len(coords))==1:
665                     plt.title('Falta '+str(6-len(coords))+' punto')
666                 else:
667                     plt.title('Faltan '+str(6-len(coords))+' puntos')
668                 plt.plot(xx,yy,'g',linestyle=':', marker='o')
669                 plt.draw()
670                 if len(coords) == n:
671                         plt.title('6 puntos leidos. Cierra esta ventana.')
672                         fig.canvas.mpl_disconnect(interactividad)
673         interactividad = fig.canvas.mpl_connect('button_press_event', lambda event: onclick(event,mpoints))
674 
675         plt.show()
676         x,y = zip(*coords)
677         x = numpy.array(x)
678         y = numpy.array(y)
679         x=x.astype(numpy.int)#around(x,0)
680         y=y.astype(numpy.int)
681         return x,y
682 
683 class CreateToolTip(object):
684     """
685     create a tooltip for a given widget
686     """
687     def __init__(self, widget, text='widget info'):
688         self.waittime = 500     #miliseconds
689         self.wraplength = 180   #pixels
690         self.widget = widget
691         self.text = text
692         self.widget.bind("<Enter>", self.enter)
693         self.widget.bind("<Leave>", self.leave)
694         self.widget.bind("<ButtonPress>", self.leave)
695         self.id = None
696         self.tw = None
```

```
697
698     def enter(self, event=None):
699         self.schedule()
700
701     def leave(self, event=None):
702         self.unschedule()
703         self.hidetip()
704
705     def schedule(self):
706         self.unschedule()
707         self.id = self.widget.after(self.waittime, self.showtip)
708
709     def unschedule(self):
710         id = self.id
711         self.id = None
712         if id:
713             self.widget.after_cancel(id)
714
715     def showtip(self, event=None):
716         x = y = 0
717         x, y, cx, cy = self.widget.bbox("insert")
718         x += self.widget.winfo_rootx() + 25
719         y += self.widget.winfo_rooty() + 20
720         # creates a toplevel window
721         self.tw = Toplevel(self.widget)
722         # Leaves only the label and removes the app window
723         self.tw.wm_overrideredirect(True)
724         self.tw.wm_geometry("+%d+%d" % (x, y))
725         label = Label(self.tw, text=self.text, justify='left',
726                       background="#ffffff", relief='solid', borderwidth=1,
727                       wraplength = self.wraplength)
728         label.pack(ipadx=1)
729
730     def hidetip(self):
731         tw = self.tw
732         self.tw= None
733         if tw:
734             tw.destroy()
735
736 def main():
737     corre_unaVez = 0
738     pathTemp='C:/Users/salvador/Documents/UNAM/'
739     prefix = 'Popo-'
740     datosVolcan = { 'distance cam-plume':7000. ,\
741                     'ang FOV-plume':90. ,\
742                     'profile d0':6000., \
743                     'profile d1':6300. }
744     preloadAvailable = 0 #si =1 entonces usa las que ya estan precargadas en path
745
746     def askdirectory(pathini,a=1):
747             dirname = filedialog.askdirectory( initialdir = pathini )
748             if dirname:
749                     varPaths[a].set(dirname)
750
751     def UserFileInput(status,name, xx=10, yy=20):
752             optionLabel = Label(ventana1, text = name).place(x=xx , y=yy)
753             var = StringVar(ventana1)
754             var.set( status )
```

```
755             Entry( ventana1, textvariable = var ).place( x=xx+80 , y=yy )
756             return var
757 
758     ventana1=Tk()
759     ventana1.geometry("300x250+250+200")
760     ventana1.title(" select directory ")
761     imagen=PhotoImage(file="canvas.PNG")
762     fondo=Label(ventana1,image=imagen).place(x=0,y=0)
763     ventana1.iconbitmap(default = 'volcan.ico')
764     config = ConfigParser()
765     config.read('config.ini')
766 
767     datosVolcan['distance cam-plume'] = config.getfloat('datosVolcan','distance cam-plume')
768     datosVolcan['ang FOV-plume'] = config.getfloat('datosVolcan','ang FOV-plume')
769     datosVolcan['profile d0'] = config.getfloat('datosVolcan','profile d0')
770     datosVolcan['profile d1'] = config.getfloat('datosVolcan','profile d1')
771 
772     varPaths = ['','','']
773     varPaths[0] = config.get('varPaths', 'Data')
774     varPaths[1] = config.get('varPaths', 'Vign')
775     varPaths[2] = config.get('varPaths', 'Bias')
776 
777     varPaths[0] = UserFileInput(varPaths[0], "Data",10,20)
778     dirBut0 = Button(ventana1, text=' ... ', command = lambda:askdirectory(varPaths[0].get(),0)).place(x=250,y=17)
779 
780     varPaths[1] = UserFileInput(varPaths[1], "Vign corr",10,50)
781     dirBut1 = Button(ventana1, text=' ... ', command = lambda:askdirectory(varPaths[1].get(),1)).place(x=250,y=47)
782 
783     varPaths[2] = UserFileInput(varPaths[2], "Bias corr",10,80)
784     dirBut2 = Button(ventana1, text=' ... ', command = lambda:askdirectory(varPaths[2].get(),2)).place(x=250,y=77)
785 
786     #input datos
787     datosVolcan['distance cam-plume'] = UserFileInput(datosVolcan['distance cam-plume'], "distance  ",10,110)
788     datosVolcan['ang FOV-plume'] = UserFileInput(datosVolcan['ang FOV-plume'], "ang FOV  ",10,135)
789     datosVolcan['profile d0'] = UserFileInput(datosVolcan['profile d0'] , "profile P0  ",10,160)
790     datosVolcan['profile d1'] = UserFileInput(datosVolcan['profile d1'], "profile P1  ",10,185)
791 
792     botOnK = Button ( ventana1, text=" Ok ", command = ventana1.destroy ).place(x=250,y=210)
793     btn1 = Label(ventana1, text="?")
794     btn1.pack(side = BOTTOM)
795     botOnK_ttp = CreateToolTip(btn1, \
796     'Se puede omitir P0 y P1, '
797     'y definirlos '
798     'mas adelante .')
799     ventana1.mainloop()
800 
801     for i in [0,1,2]:
802         varPaths[i] =  varPaths[i].get()
803     datosVolcan['distance cam-plume'] = float(datosVolcan['distance cam-plume'].get())
804     datosVolcan['ang FOV-plume'] = float(datosVolcan['ang FOV-plume'].get())
805     datosVolcan['profile d0'] = float(datosVolcan['profile d0'].get())
806     datosVolcan['profile d1'] = float(datosVolcan['profile d1'].get())
807 
808     config.set('varPaths', 'Data', varPaths[0])
809     config.set('varPaths', 'Vign', varPaths[1])
810     config.set('varPaths', 'Bias', varPaths[2])
811     config.set('datosVolcan','distance cam-plume',str(datosVolcan['distance cam-plume']))
812     config.set('datosVolcan','ang FOV-plume',str(datosVolcan['ang FOV-plume']))
```

```
813     config.set('datosVolcan','profile d0',str(datosVolcan['profile d0']))
814     config.set('datosVolcan','profile d1',str(datosVolcan['profile d1']))
815     with open('config.ini', 'w') as configfile:
816             config.write(configfile)
817 
818     path=varPaths[0]+"/"#PARA LINUX cambiar todos los \\ por /
819     pathVC=varPaths[1]+"/"
820     pathBias=varPaths[2]+"/"
821 
822     #parametros para el algoritmo Farneback
823     pyr_scale = config.getfloat('parametrosOpticalFlow','pyr_scale')
824     levels = config.getint('parametrosOpticalFlow','levels')
825     winsize = config.getint('parametrosOpticalFlow','winsize')
826     iterations = config.getint('parametrosOpticalFlow','iterations')
827     poly_n = config.getint('parametrosOpticalFlow','poly_n')
828     poly_sigma = config.getfloat('parametrosOpticalFlow','poly_sigma')
829 
830     configof = { 'pyr_scale':pyr_scale ,\
831                             'levels':levels ,\
832                             'winsize':winsize, \
833                             'iterations':iterations, \
834                             'poly_n':poly_n, \
835                             'poly_sigma':poly_sigma }
836 
837     '''
838     Este programa se escribio para python 3.4
839     Debe existir un directorio data* con los archivos raw y txt
840     Y un directorio VCorrection con las imagenes de correccion de Vignetting
841     '''
842     ##lista de archivos de data en  una lista filenames
843     filelist=glob.glob(path+'*.raw')
844     if len(filelist) < 5:
845             print("NO HAY SUFICIENTES ARCHIVOS PARA PROCESAR \n INTENTA OTRO PATH")
846     filenames=[]
847     filelist.sort()
848     for sfile in filelist:
849         filenames.append(sfile.split('\\')[-1][:19])#ULTIMA_SECCION.19_CARACTERES
850 
851     ##Se obtiene lista de nombres(se obtienen solo los pares para que no se repitan)
852     ##Los nombres son de time
853     nombrespares=[]#lista de la forma stime.xx
854     for i in range(0,len(filenames),2):
855         nombrespares.append(filenames[i].split('-')[0].strip())
856     numplots=len(nombrespares)-6
857     if numplots < 0:
858       numplots = 0
859 
860 
861     ventana2=Tk()
862     valor1=IntVar()
863     valor2=IntVar()
864     pltshow1=IntVar()
865     makegif1 = IntVar()
866 
867     ventana2.geometry("250x250+250+180")
868     ventana2.title(" Parametros de prueba ")
869     imagen=PhotoImage(file="canvas.PNG")
870     fondo=Label(ventana2,image=imagen).place(x=0,y=0)
```

```
871     etiqueta = Label( ventana2,text=str(numplots)+' Pairs found in '+' .../.../'+path.split('/')[-2], bg= "#ffffff" ).place(x=20,y=10)
872
873     etiqueta2 = Label( ventana2,text="Start").place(x=20,y=65)
874     combo1= Spinbox(ventana2, from_=0,to=numplots,textvariable=valor1).place(x=65,y=65)
875     etiqueta3 = Label( ventana2,text="End").place(x=20,y=95)
876     combo2 = Spinbox(ventana2, from_=0,to=numplots,textvariable=valor2).place(x=65,y=95)
877     etiqueta4 = Label( ventana2,text="Show plots?").place(x=20,y=130)
878     check1 = Checkbutton(ventana2,state=ACTIVE,variable=pltshow1).place(x=100,y=130)
879     etiqueta4 = Label( ventana2,text="make gif?").place(x=20,y=155)
880     check2 = Checkbutton(ventana2,state=ACTIVE,variable=makegif1).place(x=100,y=155)
881     boton2 = Button (ventana2, text=" Ok ", command=ventana2.destroy).place(x=145,y=190)
882     ventana2.mainloop()
883
884     filenum=[valor1.get(),valor2.get()]#preloaded images [initial,final]
885     if filenum[1] < filenum[0]:
886         print('end < start ! !\n using start value')
887         filenum[1] = filenum[0]
888     print("»> "+str(filenum)+path+" de:"+str(numplots)+" disponibles.")
889
890     pltshow=pltshow1.get()#show plots True or False
891     makegif = makegif1.get()
892
893     print("Loading images for vignetting correction...")
894     ##Cargar archivos de correccion de Vignetting (img constante)
895     VigAve310=np.fromfile(pathVC+'VignettingAverage310',dtype=np.float64).reshape(506,512)
896     VigAve330=np.fromfile(pathVC+'VignettingAverage330',dtype=np.float64).reshape(506,512)
897     print("Loading images for dark current...")
898     ##Cargar archivos de correccion de Bias (img constante)
899     BiasAve310=np.fromfile(pathBias+'BiasAverage310',dtype=np.float64).reshape(506,512)
900     BiasAve330=np.fromfile(pathBias+'BiasAverage330',dtype=np.float64).reshape(506,512)
901
902
903     start=time.clock()
904     print("running matchmatrix...")
905     offset=matchmatrix(filenames[4],filenames[5],-12,50,-12,20,path)##filenames[2*i],[2*i+1]
906     #offset=[-35,-1]#podria usarse de esta manera un offset manipulado.
907     rows=506-abs(offset[0])
908     cols=512-abs(offset[1])
909     end=time.clock()
910     print(offset,"\t time for matchmatrix:",end-start)
911
912     localdir = split_slash(path)[-3]
913     if not os.path.exists(path[:-6] + r'\Results'):
914         os.makedirs(path[:-6] + r'\Results')
915     region_skytemp = np.zeros((rows,cols))
916     lastabsorbancia0 = np.zeros((rows,cols))
917     lastporcen = 0
918     fracSA_old = 0
919     umbr_timedif_old = 0
920     umbr_sumabsor_old = 0
921
922     newskyavailable = 0 #debe empezar con cero, a menos que ponga un codigo que
923     #cargue un archivo de cielo artificial.
924     profiledefined = 0
925     uuu=2#4
926     vvv=3#6
927     for k in range(filenum[0]+uuu,filenum[1]+vvv):
928
```

```
929         cincoNombres=nombrespares[(k-uuu):(k+vvv)]
930         print ('loading 5 pairs of images from data dir...',cincoNombres)
931         if len(cincoNombres)!=(uuu+vvv):
932             print("» » No hay al menos 5 pares de imagenes")
933 
934         if(preloadAvailable == 0):
935 
936             secuencia=estructura(cincoNombres,rows,cols,path,BiasAve310,BiasAve330,VigAve310,VigAve330,offset) #modified on 20161014.1py
937             print("images loaded in structure\t ok")
938 
939             tiempo=secuencia['time'].astype(int)
940             img310=secuencia['m310'].astype(float)
941             img330=secuencia['m330'].astype(float)
942             print ( np.shape(secuencia) )
943             #del secuencia
944 
945         ################################################################
946         #########  Timedif de absorbancia.  ######################
947         ################################################################
948             absorbancia=[]#buscar si hay un append de numpy
949             warnings.filterwarnings('error')
950             for i in range(uuu+vvv):
951                 try:
952                     absorbancia.append(np.log10(img330[i]/img310[i]))
953                 except Warning:
954                     warnings.resetwarnings()
955                     absorbancia.append(np.nan_to_num(np.log10(img330[i]/img310[i])))
956                     warnings.filterwarnings('error')
957                     continue
958             warnings.resetwarnings()
959             sumabsorbancia=0
960             timedifAbso=0
961             imagen330=0
962             for i in range(uuu+vvv):
963                 sumabsorbancia+=absorbancia[i]
964             for i in range(uuu+vvv-1):
965                 timedifAbso+=np.absolute(absorbancia[i]-absorbancia[i+1])
966             timedifAbso = cv2.blur(timedifAbso,(7,7))
967             for i in range(uuu+vvv):
968                 imagen330+=img330[i]
969 
970             #el tiempo entre imagenes sucesivas
971             delta_t = tiempo[2]-tiempo[1]
972 
973         #----Graficar sumI330----#
974         if corre_unaVez == 0:
975             fig = plt.figure(1)
976             ax = fig.add_subplot(111)
977             ax.set_title('Imagen sumI330')
978             ax.set_xlabel('Columna [$j$]')
979             ax.set_ylabel('Fila [$i$]')
980             rr=ax.contourf(imagen330,40);cbar_rr = fig.colorbar(rr)
981             cbar_rr.ax.set_ylabel('Valor de los pixeles, u.a.')
982 
983             plt.savefig(pathTemp+prefix+"sumI330.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
984 
985             ##CARGANDO IMGS TIMEDIFABSO Y ++IMG310
986             timedifAbso.astype('float').tofile(path+'timedifAbso'+str(k-uuu))
```

```
            imagen330.astype('float').tofile(path+'im330-'+str(k-uuu))
            sumabsorbancia.astype('float').tofile(path+'sumabsorb-'+str(k-uuu))
            absorbancia[0].astype('float').tofile(path+'absorbancia0-'+str(k-uuu))
            absorbanc = absorbancia[0]
            absorbancia0 = absorbanc
        else:
            #when preLoad available
            timedifAbso=np.fromfile(path+'timedifAbso'+str(k-uuu),dtype=np.float64).reshape(rows,cols)
            imagen330=np.fromfile(path+'im330-'+str(k-uuu),dtype=np.float64).reshape(rows,cols)
            sumabsorbancia=np.fromfile(path+'sumabsorb-'+str(k-uuu) , dtype=np.float64).reshape(rows,cols)
            absorbancia0 = np.fromfile(path+'absorbancia0-'+str(k-uuu) , dtype=np.float64).reshape(rows,cols)
            #absorbancia0 = absorbanc

        hist330, bins330 = np.histogram(imagen330.ravel(),50, density = True)
        bins330 = shiftx(bins330)
        hist330 = np.insert(hist330,0,0)
        bins330 = np.insert(bins330,0,0)
        LML = max_of(hist330)
        LML = LML[ hist330[LML] > 0.0000015 ]
        sss=0
        for ss in range(LML[0],45):#int(len(hist330)*9/10)):
            if hist330[ss] < 0.000001:
                break
            sss = ss
        ttt=0
        for tt in range(LML[-1],5,-1):#int(len(hist330)/10),-1):
            if hist330[tt] < 0.0000007:
                break
            ttt = tt-1
        i_umbral330 = int((sss+ttt)/2)
        umbral330 = imagen330 > bins330[i_umbral330]
        umbral330 = cv2.dilate(np.array( umbral330 ,dtype=np.float64),np.ones((2,2),np.float64),iterations = 2)
        umbral330 = cv2.erode(umbral330,np.ones((3,3),np.uint8),iterations = 1)
        #----Graficar una imagen binaria cielo.volcan usando contourf----#
        if corre_unaVez == 0:
            fig = plt.figure(1)
            fig.set_size_inches(6,4.6)
            ax = fig.add_subplot(111)
            ax.set_title('Imagen binaria cielo volcán')
            ax.set_xlabel('Columna [$j$]')
            ax.set_ylabel('Fila [$i$]')
            ax.contourf(umbral330,1)

            fig.savefig(pathTemp+prefix+"cielo.PNG", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
            print('el umbral obtenido para cielo.volcan',str(bins330[i_umbral330]))

        #-------------- TimeDif -----------#
        timedifAbso = timedifAbso*umbral330
        hTD = Histograma( timedifAbso , 600 , 5 , 2 )
        histTimedif = hTD.hist
        binsTimedif = hTD.binz

        localmax_histTimedif = hTD.MaxList(0.1)

        #----Graficar un histograma de sumI330----#
        if corre_unaVez == 0:
            fig = plt.figure(1)
            fig.set_size_inches(5,4.5)
```

```
            ax = fig.add_subplot(111)
            ax.set_title('Histograma de sumI330')
            ax.grid(True);plt.locator_params(axis='x',nbins=4)
            ax.axis([0.0,400000,0,max(hist330)+0.1*max(hist330)])
            ax.set_xlabel('Intensidad en los pixeles')
            ax.set_ylabel('frecuencia normalizada')#Numero de pix normalizado #Densidad de probabilidad.
            ax.yaxis.set_major_formatter(tck.FormatStrFormatter('%1.1e'))
            ax.plot(bins330,hist330,'g-')#plt.contourf(imagen330,40);plt.colorbar()
            ax.plot(bins330[int((ss+tt)/2)],(hist330[int((ss+tt)/2)]),'ro')
            fig.savefig(pathTemp+prefix+"histSumI330.pdf", bbox_inches='tight', pad_inches=0, transparent=True);plt.clf()

        #----------------SumAbsor ---------#
        sumabsorbancia=sumabsorbancia*umbral330
        hSA = Histograma( sumabsorbancia , 300 , 11 , 2 )
        histSumabsor, binsSumabsor = hSA.hist , hSA.binz

        localmax_histSumabsor_SV = hSA.MaxList(0.08)

        i_uSumAbs = localmax_histSumabsor_SV[0] + 5
        fracSA = np.sum(histSumabsor[:i_uSumAbs]*np.diff(hSA.bins[:i_uSumAbs+1]))

        #Umbrales
        umbr_timedif = binsTimedif[ localmax_histTimedif[0] ]
        umbr_sumabsor = binsSumabsor[ i_uSumAbs ]
        if fracSA < 0.3*fracSA_old:#si es menor a new < 0.3*old
            umbr_sumabsor = binsSumabsor[ localmax_histSumabsor_SV[1] + 5 ]
            i_uSumAbs = localmax_histSumabsor_SV[1] + 5 #pero hay que actualizar fracSA
            fracSA = np.sum(histSumabsor[:i_uSumAbs]*np.diff(hSA.bins[:i_uSumAbs+1]))
        if newskyavailable:
            umbr_timedif = (umbr_timedif + umbr_timedif_old)/2
            umbr_sumabsor = (umbr_sumabsor + umbr_sumabsor_old)/2
        #gc.collect()
        #243 umbr timedif
        pluma_timedif = timedifAbso < umbr_timedif

        #247 umbrsumabsor
        pluma_sumabsor = sumabsorbancia < umbr_sumabsor

        #----Graficar timediffAbso usando contourf----#
        if corre_unaVez == 0:
            fig = plt.figure(1)
            fig.set_size_inches(6,4.6)
            ax = fig.add_subplot(111)
            ax.set_title('Imagen timediffAbso')
            ax.set_xlabel('Columna [$j$]')
            ax.set_ylabel('Fila [$i$]')
            ax.contourf(timedifAbso,40)

            plt.savefig(pathTemp+prefix+"timediffAbso.png", bbox_inches='tight', pad_inches=0, transparent=True);plt.clf()
        #----Graficar un histograma de timediffAbso----#
            fig = plt.figure(1)
            fig.set_size_inches(6,4.6)
            ax = fig.add_subplot(111)
            ax.set_title('Histograma de timediffAbso')
            ax.grid(True);ax.locator_params(axis='x',nbins=4)
            ax.axis([0.0,0.05,0,max(histTimedif)+0.5])
            ax.plot(binsTimedif,histTimedif,color='red')
            #plt.plot(binsTimedif,histTimedif_SF,'b--')
```

```
            ax.plot(binsTimedif[localmax_histTimedif[0]],histTimedif[localmax_histTimedif[0]],'mo',label="umbralTD")
            ax.set_xlabel('Intensidad en los pixeles')
            ax.set_ylabel('frecuencia normalizada')#Numero de pix normalizado #Densidad de probabilidad.
            #ax.yaxis.set_major_formatter(tck.FormatStrFormatter('%1.1e'))
            fig.savefig(pathTemp+prefix+"hist-timediffAbso.pdf", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
        #----Graficar una imagen binaria generada por timediffAbso usando contourf----#
            fig = plt.figure(1)
            fig.set_size_inches(6,4.6)
            ax = fig.add_subplot(111)
            ax.set_title('Imagen binaria obtenida con timediffAbso')
            ax.set_xlabel('Columna [$j$]')
            ax.set_ylabel('Fila [$i$]')
            ax.contourf(pluma_timedif,1)
            fig.savefig(pathTemp+prefix+"cielo-timediffAbso.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
            #print('el umbral obtenido fue',str(bins330[i_umbral330]))

        #----Graficar sumAbsor usando contourf----#
        if corre_unaVez == 0:
            fig = plt.figure(1)
            fig.set_size_inches(6,4.6)
            ax = fig.add_subplot(111)
            ax.set_title('Imagen sumAbsor')
            ax.set_xlabel('Columna [$j$]')
            ax.set_ylabel('Fila [$i$]')
            ax.contourf(sumabsorbancia,40)

            plt.savefig(pathTemp+prefix+"sumAbsor.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
        #----Graficar un histograma de sumAbsor----#
            fig = plt.figure(1)
            fig.set_size_inches(6,4.6)
            ax = fig.add_subplot(111)
            ax.set_title('Histograma de sumAbsor')
            ax.set_xlabel('Columna [$j$]')
            ax.set_ylabel('Fila [$i$]')
            ax.grid(True);ax.locator_params(axis='x',nbins=4)
            ax.plot(binsSumabsor,histSumabsor,color='red')
            ax.plot(binsSumabsor[i_uSumAbs],histSumabsor[i_uSumAbs],'mo',label="umbralSO")
            ax.set_xlabel('Intensidad en los pixeles')
            ax.set_ylabel('frecuencia normalizada')#Numero de pix normalizado #Densidad de probabilidad.
            #ax.yaxis.set_major_formatter(tck.FormatStrFormatter('%1.1e'))
            fig.savefig(pathTemp+prefix+"hist-sumAbsor.pdf", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
        #----Graficar una imagen binaria generada por sumAbsor usando contourf----#
            fig = plt.figure(1)
            fig.set_size_inches(6,4.6)
            ax = fig.add_subplot(111)
            ax.set_title('Imagen binaria obtenida con sumAbsor')
            ax.set_xlabel('Columna [$j$]')
            ax.set_ylabel('Fila [$i$]')
            ax.contourf(pluma_sumabsor,1)
            fig.savefig(pathTemp+prefix+"cielo-sumAbsor.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
            #print('el umbral obtenido fue',str(bins330[i_umbral330]))
        pluma_sumabsor = cv2.erode(np.array(pluma_sumabsor,dtype=np.float64),np.ones((3,3),np.float64),iterations = 2)
        pluma_timedif = cv2.erode(np.array(pluma_timedif,dtype=np.float64),np.ones((3,3),np.float64),iterations = 1)

        mask = ( timedifAbso > 2.2*umbr_timedif )*umbral330

        regionSky = (pluma_timedif*pluma_sumabsor*umbral330)
        #Indices de la region de cielo
```

```
1161        indices_regionSky = np.where( regionSky == 1 )
1162
1163        #un histograma de region, para conocer el porcentaje de pixeles sky
1164        H_sky, binsH_sky = np.histogram(regionSky[ regionSky != 0 ].ravel(),5)
1165        skyR = np.sum(H_sky)
1166        porcentaje = skyR/rows/cols
1167        print("PORCENTAJE es", porcentaje)
1168        difp = (porcentaje - lastporcen)
1169
1170        if porcentaje > 0.02:
1171            newskyavailable = k-uuu+1
1172            region_skytemp = regionSky
1173            lastabsorbancia0 = absorbancia0
1174
1175        if newskyavailable:#True despues de que encuentre la primera region de cielo aceptable.
1176            lastporcen = porcentaje
1177            fracSA_old = fracSA
1178            umbr_timedif_old = umbr_timedif
1179            umbr_sumabsor_old = umbr_sumabsor
1180            cheesySky = 0
1181
1182            if porcentaje <= 0.02:
1183                cheesySky = ajuste2d( region_skytemp, lastabsorbancia0, rows, cols )
1184            else:
1185                cheesySky = ajuste2d( regionSky, absorbancia0, rows, cols )
1186
1187                if corre_unaVez == 0:
1188                    #----Graficar absorbancia en region de cielo extrapolada----#
1189                    fig = plt.figure(1)# si no uso arg = 1, sucede que aparecen figuras extras durante la ejecucion.
1190                    ax = fig.add_subplot(111)
1191                    ax.set_title('Extrapolación de absorbancia')
1192                    ax.set_xlabel('Columna [$j$]')
1193                    ax.set_ylabel('Fila [$i$]')
1194                    rr = ax.contourf(cheesySky,40);cbar_rr = fig.colorbar(rr)
1195                    cbar_rr.ax.set_ylabel(r'Absorbancia')
1196
1197                    fig.savefig(pathTemp+prefix+"absCieloExtrap.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
1198            end=time.clock()
1199            print( localdir + "__" +str("%03d" % (k-uuu)) ,"Tiempo: ",end-start)
1200
1201            angle = datosVolcan['ang FOV-plume']
1202            distancia = datosVolcan['distance cam-plume']
1203            dpix = np.zeros((rows,cols))
1204            lpix = np.zeros((rows,cols))
1205            for i in range(cols):
1206                for j in range(rows):
1207                    dpix[j,i] = distancia * ( math.cos( math.radians( (float(i)-(cols/2.))*23./512 ) ) + \
1208                    ( math.sin( math.radians( (float(i)-(cols/2.)) * 23./512 )) / math.tan( math.radians( angle - (float(i)-(cols/2)) *23
1209            lpix = dpix*math.radians(23.0)/512
1210            if corre_unaVez == 0:
1211                #----Graficar una imagen I310uncorr e I330uncorr usando contourf----#
1212                fig = plt.figure(1)
1213                ax = fig.add_subplot(111)
1214                ax.set_title('distancias')
1215                ax.set_xlabel('Columna [$j$]')
1216                ax.set_ylabel('Fila [$i$]')
1217                rr = ax.contourf(dpix,40);cbar_rr = fig.colorbar(rr)
1218                cbar_rr.ax.set_ylabel(r'Distancias [m]')
```

```
1219
1220                fig.savefig(pathTemp+prefix+"distancias.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
1221
1222            print('newskyavail =',newskyavailable)
1223            coefScattering = [-0.000192,-0.000169]#[-0.00006,-0.000025]#[-0.00004634,-0.00003702]#
1224            use_fixedcoefScattering = False #True si ya se conocen los coeficientes de dispersion.
1225
1226            if use_fixedcoefScattering and profiledefined == 0:
1227                    profiledefined = 1
1228                    xpoints , ypoints = coords_points(img310[0],6)#coordenadas de la curva Gamma
1229
1230            if newskyavailable and profiledefined == 0:
1231                    SKY = [np.sum(img310[0][indices_regionSky])/len(indices_regionSky[0]),np.sum(img330[0][indices_regionSky])/len(indice
1232                    print('sky',SKY, 'un pixel de cielo',indices_regionSky[0][0],indices_regionSky[1][0])
1233
1234                    x,y,length_profile, D0, D1 = profilecoords(img310[0])#coordenadas perfil de intensidades
1235                    profileI310 = img310[0][y, x]   #i,j = rows,cols;
1236                                                    #then coords x iterates over all j cols indexes and coord y iterates over i rows.
1237                    profileI330 = img330[0][y, x]
1238                    distancias_profile = D0 + np.arange(length_profile, dtype=float) / length_profile * ( D1 - D0 )
1239
1240                    xpoints , ypoints = coords_points(img310[0],6)#coordenadas de la curva Gamma
1241
1242                    A = np.vstack([distancias_profile, np.ones(len(distancias_profile))]).T
1243                    m, b = np.linalg.lstsq(A, profileI310)[0]
1244                    p0 = [b,-1.0*10**-4]
1245                    print('una intensidad de profile310',profileI310[2],'I0 guess',p0[0])
1246                    #AJUSTE 310
1247                    def ajuste_perfil310( d , I0 , sigma ):
1248                            return I0 * np.exp( sigma * d ) + SKY[0] * ( 1 - np.exp( sigma * d ) )
1249
1250                    popt310, __ = curve_fit( ajuste_perfil310, distancias_profile, profileI310, p0 )
1251                    #AJUSTE 330
1252                    def ajuste_perfil330( d , I0 , sigma ):
1253                            return I0 * np.exp( sigma * d ) + SKY[1] * ( 1 - np.exp( sigma * d ) )
1254
1255                    popt330, __ = curve_fit( ajuste_perfil330, distancias_profile, profileI330, p0 )
1256
1257                    #----Graficar ajustes para obtener coef de dispersion----#
1258                    if corre_unaVez == 0:
1259                        fig = plt.figure()
1260                        fig.set_size_inches(6.47,4)
1261                        ax = fig.add_subplot(111)
1262                        ax.set_title(r'Ajustes para obtener $\sigma_{310}$ y $\sigma_{330}$')
1263                        ax.grid(True)
1264                        ax.set_xlabel('Distancia [m]')
1265                        ax.set_ylabel('Intensidad en los pixeles')
1266                        ax.plot(distancias_profile,profileI310, 'r+')
1267                        ax.plot(distancias_profile, ajuste_perfil310(distancias_profile ,popt310[0], popt310[1]),'r-',label='310')
1268
1269                        ax.plot(distancias_profile,profileI330, 'b+')
1270                        ymin, ymax = ax.get_ylim()
1271                        __ = (ymax-ymin)/10
1272                        ax.plot(distancias_profile, ajuste_perfil330(distancias_profile ,popt330[0], popt330[1]),'b-',label='330')
1273                        ax.text(x=distancias_profile[0]+20,y= ymin+1*__,s="$-\sigma_{310}=$"+str( "%.6f" % popt310[1] ) , color='black' ,
1274                        ax.text(x=distancias_profile[0]+20,y=ymin+2*__,s="$-\sigma_{330}=$"+str( "%.6f" % popt330[1] ) , color='black' ,
1275                        ax.text(x=distancias_profile[0]+20,y = ymin+3*__,s="$I_0(310)=$"+str( "%.1f" % popt310[0] ) , color='black' , for
1276                        ax.text(x=distancias_profile[0]+20,y = ymin+4*__,s="$I_0(330)$="+str( "%.1f" % popt330[0] ) , color='black' , for
```

```
                    ax.legend(loc='best',framealpha=0.6)
                    fig.tight_layout()
                    fig.savefig(pathTemp+prefix+"ajusteCoefDispersion.pdf", bbox_inches='tight');plt.clf()

                print('dispers310',popt310[1])
                print('dispers330',popt330[1])
                print('I0 310',popt310[0])
                print('I0 330',popt330[0])

                coefScattering[0] = popt310[1]
                coefScattering[1] = popt330[1]

                profiledefined = 1

        if newskyavailable and profiledefined == 1:
                SKY = [np.sum(img310[0][indices_regionSky])/len(indices_regionSky[0]),np.sum(img330[0][indices_regionSky])/len(indice
                print('sky',SKY, 'Un pixel de cielo',indices_regionSky[0][0],indices_regionSky[1][0])

                img310_0 = img310[0].copy()
                img310_1 = img310[1].copy()
                regionPlumaCielo = umbral330
                regionCielo = regionSky

                if corre_unaVez == 0:
                    #----Graficar una imagen I310uncorr e I330uncorr usando contourf----#
                    fig = plt.figure(1)# si no uso arg = 1, sucede que aparecen figuras extras durante la ejecucion.
                    ax = fig.add_subplot(111)
                    ax.set_title('Imagen I310 con dilución')
                    ax.set_xlabel('Columna [$j$]')
                    ax.set_ylabel('Fila [$i$]')
                    rr = ax.contourf(img310[0],40);cbar_rr = fig.colorbar(rr)
                    cbar_rr.ax.set_ylabel('Intensidad en los pixeles, u.a.')
                    fig.savefig(pathTemp+prefix+"I310uncorr.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()

                    fig = plt.figure(1)
                    ax = fig.add_subplot(111)
                    ax.set_title('Imagen I330 con dilución')
                    ax.set_xlabel('Columna [$j$]')
                    ax.set_ylabel('Fila [$i$]')
                    rr = ax.contourf(img330[0],40);cbar_rr = fig.colorbar(rr)
                    cbar_rr.ax.set_ylabel('Intensidad en los pixeles, u.a.')
                    fig.savefig(pathTemp+prefix+"I330uncorr.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()


                AAsincorre = numpy.log10(img330[0]/img310[0])

                img310[0] = corr_plume_intensity( img310[0] , SKY[0] , coefScattering[0], dpix )#popt310[1] , dpix )# popt310[1] es:
                img330[0] = corr_plume_intensity( img330[0] , SKY[1] , coefScattering[1], dpix )#popt330[1] , dpix )

                AAcorr = numpy.log10(img330[0]/img310[0])  - cheesySky #Despues de aplicar la correccion de
                                                                                                                    #dilucion, se


                AAcorr.astype('float').tofile(path+'AAcorr-'+str(k-uuu))
                if corre_unaVez == 0:
                    #----Graficar una imagen I310corr e I330corr corregida usando contourf----#
                    fig = plt.figure(1)# si no uso arg = 1, aparecen figuras extras durante la ejecucion.
                    ax = fig.add_subplot(111)
```

```
1335                    ax.set_title('Imagen I310 sin dilución')
1336                    ax.set_xlabel(r'$n$ columnas')
1337                    ax.set_ylabel(r'$m$ filas')
1338                    rr = ax.contourf(img310[0],40);cbar_rr = fig.colorbar(rr)
1339                    cbar_rr.ax.set_ylabel('Intensidad en los pixeles, u.a.')
1340                    fig.savefig(pathTemp+prefix+"I310corr.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
1341
1342                    fig = plt.figure(1)
1343                    ax = fig.add_subplot(111)
1344                    ax.set_title('Imagen I330 sin dilución')
1345                    ax.set_xlabel('Columna [$j$]')
1346                    ax.set_ylabel('Fila [$i$]')
1347                    rr = ax.contourf(img330[0],40);cbar_rr = fig.colorbar(rr)
1348                    cbar_rr.ax.set_ylabel('Intensidad en los pixeles, u.a.')
1349                    fig.savefig(pathTemp+prefix+"I330corr.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
1350
1351                    #----Graficar AA sincorreccion de dil usando contourf----#
1352                    fig = plt.figure(1)
1353                    ax = fig.add_subplot(111)
1354                    ax.set_title('Absorbancia aparente AA sin corregir dil.', fontsize=12)
1355                    ax.set_xlabel('Columna [$j$]')
1356                    ax.set_ylabel('Fila [$i$]')
1357                    rr = ax.contourf(umbral330*AAsincorre,40);cbar_rr = fig.colorbar(rr)
1358                    cbar_rr.ax.set_ylabel('Absorbancia Aparente')
1359                    fig.savefig(pathTemp+prefix+"AAsinCorrDil.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
1360
1361
1362                    #----Graficar AA corregida usando contourf----#
1363                    fig = plt.figure(1)
1364                    ax = fig.add_subplot(111)
1365                    ax.set_title('Absorbancia aparente AA corregida')
1366                    ax.set_xlabel('Columna [$j$]')
1367                    ax.set_ylabel('Fila [$i$]')
1368                    rr = ax.contourf(umbral330*numpy.nan_to_num(AAcorr),40);cbar_rr = fig.colorbar(rr)
1369                    cbar_rr.ax.set_ylabel('Absorbancia Aparente')
1370                    fig.savefig(pathTemp+prefix+"AAconCorrDil.png", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf()
1371
1372                    #----Graficar AA/k_a corregida usando contourf----#
1373                    fig = plt.figure(1)# si no uso arg = 1, sucede que aparecen figuras extras durante la ejecucion.
1374                    ax = fig.add_subplot(111)
1375                    ax.set_title(r'AA/$k_\alpha$')
1376                    ax.set_xlabel('Columna [$j$]')
1377                    ax.set_ylabel('Fila [$i$]')
1378                    rr = ax.contourf(umbral330*numpy.nan_to_num(AAcorr)/(1.5*10**(-4)),40);cbar_rr = fig.colorbar(rr)
1379                    cbar_rr.ax.set_ylabel('Densidad de columna [ppmm]')
1380                    fig.savefig(pathTemp+prefix+"SCAdistributionSO2.PNG", bbox_inches='tight', pad_inches=0,transparent=True);plt.clf
1381                    corre_unaVez = 1
1382
1383                img310_0 = numpy.rot90(img310_0,-1).T
1384                AAcorr = numpy.rot90(AAcorr,-1).T
1385                img310_1 = numpy.rot90(img310_1,-1).T
1386                regionPlumaCielo = numpy.rot90(regionPlumaCielo,-1).T
1387                AA = regionPlumaCielo*numpy.nan_to_num(AAcorr)/(1.5*10**(-4))
1388
1389                points = [list(xpoints),list(ypoints)]
1390                flux_engine = create_flux_engine(points)
1391                motion_engine = MotionEngine(configof)
1392
```

```
1393                    flujo,suerror = procesaunpar( img310_0 * regionPlumaCielo , img310_1 * regionPlumaCielo , motion_engine , flux_engine
1394                    print('flujo,suerror',flujo,suerror)
1395                    with open(prefix+'flujos.txt', 'a') as the_file:
1396                       the_file.write(str(flujo)+','+str(suerror)+'\n')
1397
1398            plt.figure(1)
1399            plt.plot(bins330,hist330,'g-')
1400            plt.savefig(path[:-6]+"/Results/"+localdir+"_"+str("%03d" % (k-uuu))+"hist330.png", bbox_inches='tight')
1401
1402            plt.figure(1)
1403            plt.subplot(251)#
1404            plt.plot(bins330,hist330,'g-')
1405            plt.plot(bins330[int((ss+tt)/2)],(hist330[int((ss+tt)/2)]),'ro')
1406            plt.xticks(()); plt.yticks(())
1407
1408            plt.subplot(256)#
1409            plt.contourf(umbral330,2)
1410            plt.xticks(()); plt.yticks(())
1411
1412            plt.suptitle( localdir + "__" +str("%03d" % (k-uuu)) )
1413            plt.subplot(252)#
1414            plt.title("timedifAbso" , fontsize = 9)
1415            plt.contourf(timedifAbso,18)
1416            plt.xticks(()); plt.yticks(())
1417
1418            plt.subplot(257)#
1419            plt.title("sumabsorbancia" , fontsize = 9)
1420            plt.contourf(sumabsorbancia,18)
1421            plt.xticks(()); plt.yticks(())
1422
1423            plt.subplot(253)#
1424            plt.title("hist timedif" , fontsize = 9)
1425            plt.grid(True);plt.locator_params(axis='x',nbins=4)
1426            plt.axis([0.0,0.05,0,max(histTimedif)+0.5])
1427            plt.plot(binsTimedif,histTimedif,color='red')
1428            #plt.plot(binsTimedif,histTimedif_SF,'b--')
1429            plt.plot(binsTimedif[localmax_histTimedif[0]],histTimedif[localmax_histTimedif[0]],'mo',label="umbralTD")
1430
1431            plt.subplot(258)#
1432            plt.title("hist sumabsor" , fontsize = 9)
1433            plt.grid(True);plt.locator_params(axis='x',nbins=4)
1434            plt.plot(binsSumabsor,histSumabsor,color='red')
1435            plt.plot(binsSumabsor[i_uSumAbs],histSumabsor[i_uSumAbs],'mo',label="umbralSO")
1436
1437            plt.subplot(254)#
1438            plt.text(x=110,y=100,s="< "+str("%.4f" % umbr_timedif) , color='white' , fontsize = 7)
1439            plt.title("umbr timedif" , fontsize = 9)
1440            plt.contourf(pluma_timedif , 2)
1441            plt.xticks(()); plt.yticks(())
1442
1443            plt.subplot(259)#
1444            plt.text(x=110,y=100,s="< "+str("%.4f" % umbr_sumabsor) +'\nf=' + str("%.4f" % fracSA) ,\
1445                     color='white' , fontsize = 7)
1446            plt.title("umbr sumabsor" , fontsize = 9)
1447            plt.contourf(pluma_sumabsor , 2)
1448            plt.xticks(()); plt.yticks(())
1449
1450            plt.subplot(255)#
```

```
            plt.text(x=15,y=25,s="c%"+str("%.4f" % porcentaje)+" csKy="+str((k-uuu)-(newskyavailable-1))+" \ndp="+str("%.4f" % difp) , co
            plt.title("ut*us*u330*abs[0]" , fontsize = 9)
            if porcentaje <= 0.02:
                plt.contourf(region_skytemp*lastabsorbancia0,20)
            else:
                plt.contourf(regionSky*absorbancia0,8)

            plt.subplot(2,5,10)#
            plt.title("Background $\log_{10} B$" , fontsize = 9)#{ img330 | volcan = 0 , cielo = 1 }*absorbancia[0]
            plt.contourf(cheesySky,20)
            plt.xticks(());plt.yticks(()); plt.colorbar()

            plt.savefig(path[:-6]+"/Results/"+localdir+"__"+str("%03d" % (k-uuu))+"_.png",  bbox_inches='tight', pad_inches=0,transparent
            if(pltshow==1):
                    plt.show()
            else:
                    plt.clf()

        print("~~~~~ Secuencia S( "+str(k-uuu)+" ) procesada.\n")
        print("             [].[].[].[].[].[].[].[].[].[].[]  ")
    ## Creando GIF
    if makegif == 1:
        from subprocess import Popen
        #requiere instalacion de ImageMagick para crear GIF
        program = 'C:\Program Files\ImageMagick-7.0.3-Q16\convert.exe'
        image = [program, '-delay', '12','-loop','1',path[:-6]+'/Results/'+localdir+'*.png','-scale','600x450',path[:-6]+'/Results/'+loca
        Popen(image,creationflags=0x08000000)

if __name__ == '__main__':
    main()
```

Apéndice C

Obtener imágenes de calibración

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob
import fileinput
import time
#ULTIMO CAMBIO
'''
Correccion de tipo de dato de float al que debe ser: uint16
'''

path='C:\\Users\\salvador\\Documents\\newDownloads\\flatstromboli\\'#'C:\\Users\\salvador\\Documents\\UNAM\\Dropbox\\2016-1\\gral\\Vignet

#lista de archivos, aun tienen pegado imagenesVig\\, asi que hay que hacer split
filelist=glob.glob(path+'*.raw')
filenames=[]
for sfile in filelist:
    filenames.append(sfile.split('\\')[-1][:19])#ULTIMA SECCION, 19 CARACTERES.
#filenames es una lista de los nombres de imagenes.
num = int(2*len(filenames)/3)
print(num,len(filenames))
#codigo siguiente es para cargar imagenes impares (de cam310 que saca *-1.raw)en matrices
M330=0
for i in range(1,num,2):#len({1,3,5,...,139})=70
    print(i)
    M330=M330+np.fromfile(path+filenames[i],dtype=np.uint16).reshape(512,512).astype(float)
#con el siguiente codigo obtengo 70 matrices que se van sumando entrada a entrada.
#aunque solo tomo los archivos que terminan en *-2.waw, es decir los pares.
#es por eso que i va de 2 en 2 desde 0 a 140. len({0,2,4,...,138})=70
M310=0
for i in range(0,num,2):
    print(i)
    M310=M310+np.fromfile(path+filenames[i],dtype=np.uint16).reshape(512,512).astype(float)

#puesto que son 70 sumandos, el promedio es:
M310=M310/10
M330=M330/10

#Luego hago rotar las imagenes, para que queden 'derechas' y no 'de cabeza'.
M310 = np.rot90(M310)
M310 = M310[1:507]
M330 = np.rot90(M330,-1)
M330 = M330[5:511]


VigAve310=M310
max310=np.amax(VigAve310)
VigAve310=VigAve310/max310

VigAve330=M330
max330=np.amax(VigAve330)
VigAve330=VigAve330/max330

#VignettingAverage{310,330} son las imagenes del promedio de Vignetting.
#ya estan rotados correctamente, y tambien se les ha eliminado lineas de pixeles
#quedando con shape (506,512)
VigAve310.astype('float64').tofile('VignettingAverage310')
VigAve330.astype('float64').tofile('VignettingAverage330')
```

```
59
60 recupera310=np.fromfile('VignettingAverage310',dtype=np.float64).reshape(506,512)
61 recupera330=np.fromfile('VignettingAverage330',dtype=np.float64).reshape(506,512)
62 #esto ultimo es solo para visualizar.
63 plt.subplot(2,2,1)
64 plt.contourf(recupera310,30)
65 plt.colorbar()
66 plt.subplot(2,2,2)
67 plt.contourf(recupera330,30)
68 plt.colorbar()
69 Mm310=np.fromfile(path+filenames[0],dtype=np.uint16).reshape(512,512).astype(float)
70 plt.subplot(2,2,3)
71 plt.title('una img310')
72 Mm310 = np.rot90(Mm310)
73 Mm310 = Mm310[1:507]
74 #Mm310 = Mm310.T
75 #Mm310 = Mm310[5:511]
76 plt.contourf(Mm310,30)
77 plt.colorbar()
78 plt.subplot(2,2,4)
79 plt.title('la img310 corregida')
80 plt.contourf(Mm310/recupera310,30)
81 plt.colorbar()
82 plt.show()
83
84 def visualcontrf(A):
85     return np.rot90(A.T)
86 path2 = "C:/Users/salvador/Documents/UNAM/2016-2_4/2016JULAGO/Seq3Ubinas2015_07_08/data/1436383510.16-1.raw"#'C:/Users/salvador/Documents
87 #path2 = 'C:/Users/salvador/Documents/UNAM/2016-2_4/2016JULAGO/Seq3Pacaya2016_01_16/data/1452955486.41-1.raw'
88 orig = np.fromfile(path2,dtype=np.uint16).reshape(512,512)
89 #orig = orig.T
90 orig = np.rot90(orig)
91 orig = orig[1:507]
92 plt.figure(1)
93 plt.subplot(2,2,1)
94 plt.title('original')
95 cmap = plt.cm.get_cmap("cool")
96 cmap.set_under("pink")
97 cmap.set_over("green")
98 plt.contourf(visualcontrf(orig/recupera310),20,cmap=cmap)
99 plt.colorbar()
100 plt.subplot(2,2,2)
101 plt.title('original imshow')
102 plt.imshow(orig/recupera310,'gnuplot2')
103 plt.colorbar()
104
105 plt.subplot(2,2,3)
106 plt.title('original sin correc vign')
107 plt.imshow(orig,'gnuplot2')
108 plt.colorbar()
109 plt.show()
```

Notación y acrónimos

τ Profundidad óptica = densidad óptica.

σ Coeficiente de esparcimiento.

α Coeficiente de absorción.

N Concentración, moléculas o átomos por unidad de volumen [Molec / cm^3].

c Velocidad de la luz.

h Constante de Planck.

λ Longitud de onda.

s Distancia o camino óptico.

S Densidad de columna.

k Sección transversal de extinción, asociada a la concentración N, es la suma de la sección transversal de absorción k_α y la sección transversal de esparcimiento k_σ.

ln, $\log_{10}$ Logaritmo natural (base e) y logaritmo en base 10.

β Coeficiente de extinción, β_e es el coeficiente de extinción de aerosoles y es la suma de β_a (absorción) y β_σ (esparcimiento).

I_λ Espectro de intensidades o radiancia.

E Energía.

ν Frecuencia.

θ Ángulo polar.

ASC Ángulo Solar Cenital.

CV Campo de Visión.

COSPEC Espectrómetro de correlación (COrrelation SPECtrometer).

DOAS Espectroscopia de absorción óptica diferencial (Differential Optical Absorption Spectroscopy).

$\mathbf{A}$ Matriz o imágen de absorbancias. La notación de una matriz $\mathbf{A}$ es equivalente a $(A_{i,j})$.

`d[i,j]` Matriz de distancias $\mathbf{d} = (d_{i,j})$ representada en el código del programa en PYTHON.