Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Documentación para desarrolladores

Detalle de la estructura y principales funcionalidades presentes en el código de la aplicación

Integrantes: Clemente Henríquez M. clemente.henriquez@ug.uchile.cl
Eric Contreras P. eric.contreras@ing.uchile.cl
Fabián Osses V. ossesvfabian@gmail.com
Ignacio Santos S. insantos@uc.cl
Jorge Cerda V. jorge.cerda@ug.uchile.cl
Magdalena Álvarez P. magdalena.alvarez@ug.uchile.cl
Raúl De la Fuente A. raul_i-d@live.cl
Rodrigo Urrea L. rodrigo.urrea@ug.uchile.cl
Proyecto: Cuantificación de Geología

Fecha de realización: 2 de diciembre de 2022
Santiago, Chile

Resumen

El presente documento contiene información sobre la aplicación del proyecto de Cuantificación de Geología que puede resultar útil para desarrolladores.

En el documento se presenta la estructura actual del proyecto, una descripción de los principales módulos y su función en la aplicación. Luego se da una descripción del flujo que se sigue para utilizar la aplicación y finalmente se comenta sobre las releases automáticas de la aplicación en el repositorio.

Índice de Contenidos

Índice de Figuras

Índice de Códigos

1. Documentación

1.1. Estructura del proyecto

Al clonar el proyecto debería encontrarse con la siguiente organización de carpetas y archivos.

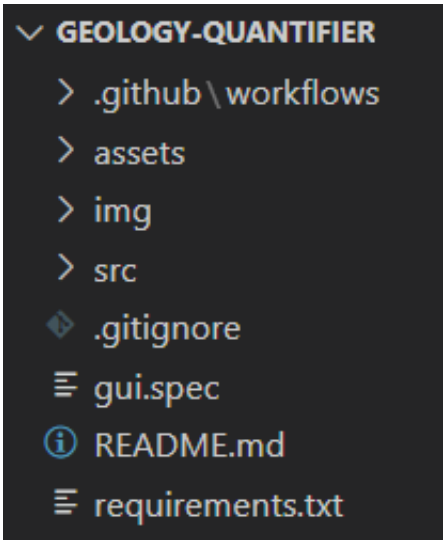


Figura 1: Estructura de carpetas

- *src*: Scripts que encapsulan distintas funcionalidades de la aplicación, con el archivo **gui.py** como el script principal que corre la aplicación.
- *assets*: Contiene los recursos gráficos utilizados por la GUI.
- *img*: Imágenes de muestra para probar la aplicación.
- *.github/workflows*: Workflows para automatizar tareas del repositorio.
- *gui.spec*: Archivo con las especificaciones para empaquetar la aplicación con pyinstaller y generar un ejecutable.

1.2. Módulos

La aplicación tiene distintos módulos (*scripts* de *Python*), los cuales implementan distintas herramientas utilizadas por la interfaz gráfica. Estos tienen como objetivo quitarle complejidad a la interfaz gráfica.

1.2.1. Color Segmentation

Este modulo implementa el método de segmentación por color, el cual a través del algoritmo *k-means* genera máscaras a partir de una imagen, cada una con un rango de colores distintos determinada por el algoritmo antes mencionado.

Estas máscaras permiten separar una imagen por sus colores, lo que tiende a que los minerales se agrupen en cada una de estas máscaras. Sin embargo el algoritmo no es capaz de automáticamente generar las mejores máscaras, por lo que es necesario que el usuario intervenga y genere varias segmentaciones hasta obtener las deseadas.

La función *generate_clusters* es la que replica el comportamiento descrito, esta función recibe como argumentos una **imagen**, la cual debe ser una matriz de *orginal_width* x *original_height* x 3, misma representación que la que entrega la librería *cv2* con su función *imread* y también recibe el **número de mascaras** que se desean generar, el cual es pasado al algoritmo *k-means* para generar el número de mascaras deseado. Es función retorna una lista que contiene cada una de las mascaras generadas por el algoritmo.

1.2.2. Image Managers

Este módulo implementa distintas funciones que permiten abrir y guardar archivos, esto para simplificar este proceso en otros módulos del código.

Las funciones *load_image_from_path* y *save_image_from_path* reciben como argumento el *path* de la imagen que se busca cargar o guardar. La función *save_image_from_path* recibe además la imagen que se busca guardar en el *path* ingresado, la cual debe ser una matriz con el mismo formato que entrega la función *imread* de la libreria *cv2*.

Las funciones *load_image_from_window* y *save_image_from_window* tienen el mismo funcionamiento que las antes mencionadas, con la salvedad de que no reciben un *path* si no que usan el explorador de archivos para determinarlo.

1.2.3. Image Tree

Este módulo contiene la estructura de datos que organiza las imágenes que se suben y crean en la aplicación. Es un árbol en el cual el padre es la *imagen original* y los hijos son las distintas mascaras generadas a partir de él. La estructura tiene los siguiente parámetros:

- *parent* es una referencia al padre del nodo actual del árbol (*null* si es la raíz) y permite la navegación bidireccional por el árbol.
- *image* es la imagen asociada al nodo, puede ser la imagen subida por el usuario o una máscara de dicha imagen.
- *name* es el nombre de la imagen, sirve para mostrarle dicho nombre al usuario y que este pueda tener una mejor noción de cada imagen.
- *childs* es una lista que contiene a los hijos del nodo actual, los cuales son a su vez nodos del árbol.

Esta estructura de datos contiene métodos para permitir al usuario el modificar las mascaras generadas por la segmentación por colores y así mejorar sus resultados (es decir, obtener una mejor segmentación de los minerales), dichos métodos son:

- *delete* este método permite eliminar hijos del nodo actual, lo que repercute en la imagen del nodo eliminando dichos colores y los reemplaza con el color base de la imagen (negro). Esta eliminación se propaga por el árbol eliminando dichos píxeles de cada foto.

- *merge* este método permite juntar 2 o más hijos del nodo actual, lo que mezcla los píxeles presentes en cada una de sus imágenes y genera un nuevo nodo con solo dicha imagen.
- *split* este método permite volver a segmentar por colores la imagen del nodo, lo que elimina los hijos del nodo y genera otros nuevos.

1.2.4. Percent

Este módulo se encarga de calcular la cantidad de píxeles presentes en una imagen, ignorando ciertos colores considerados como color base (negro).

La función *percent* recibe una imagen (en el mismo formato que el retornado por el método *imread* de la librería *cv2*) y retorna el porcentaje de píxeles con color distinto al color base (negro).

1.2.5. Sample Extraction

Este módulo se encarga de implementar las diferentes formas de recortar el testigo en la GUI, retornando finalmente la forma del recorte con la que se procederá a analizar. La idea de este módulo es que el usuario pueda escoger los puntos a partir de los que se creará el recorte de forma interactiva, ajustando dichos puntos según lo que el usuario percibe de forma visual.

En particular, existen 3 formas de recortar la imagen:

- ***Modo rectangular:*** Este modo mantiene los 4 puntos de tal forma de que siempre el resultado del corte sea rectangular, y no realiza ningún algoritmo de corrección de perspectiva, por lo que el resultado final será el recorte mismo sin procesar.
- ***Modo unwrapping:*** Este modo le pide al usuario que mueva 6 puntos, y realiza una corrección de perspectivas llamando a las funciones del módulo *Unwrapping*, que se encarga de corregir la distorsión de la imagen producida al sacarle una fotografía a un cilindro.
- ***Modo libre:*** Este modo permite mover los 4 puntos de forma libre al usuario, de tal forma que al mover un punto, no afecte la posición de los demás. Al realizar el corte, este modo utiliza los métodos de *OpenCV getPerspectiveTransform* y *WarpPerspective*, que realizan una corrección de perspectiva de tal forma de distorsionarla para dejarlo en forma rectangular.

Cada modo posee una clase que simula sus métodos. Todas las clases heredan los métodos de *AbstractExtraction*. El método *draw_circles_and_lines* se encarga de dibujar los círculos que ilustran los puntos junto con las líneas entre dichos puntos en la imagen, según los puntos recibidos. *_margin_conditions* permite que los puntos no se salgan de los márgenes de la imagen.

- La clase *FreeExtraction* se encarga de los métodos del Modo Libre. El método *reset_vertices* reinicia las posiciones de los puntos a los definidos por defecto, que están centrados en la imagen. *move_vertex* se encarga de mover el punto correspondiente comprobando que se cumplan ciertas condiciones para ello, como que se cumpla *_margin_conditions* y que los puntos no traspasen una línea dibujada. Terminando el flujo, *cut* permite cortar la imagen y finalmente aplicar *getPerspectiveTransform* y *WarpPerspective* sobre los puntos guardados en ese momento, corrigiendo la perspectiva de la imagen. El método *to_corners* mueve los puntos a las esquinas de la imagen.

- La clase *UnwrappingExtraction* se encarga de los métodos del Modo Unwrapping. Comparte varios métodos con el mismo nombre que *FreeExtraction*, pero cambiando sus mecanismos. El método *move_vertex* mueve los puntos, definiendo a su vez un sub-método de *move_vertex* por cada punto (entrando en ella al mover dicho punto, llamadas en *_process_scale_mov*), definiendo en este cómo se debe mover cada punto por punto seleccionado. El método *reset_vertices* mueve a los seis puntos a su posición por defecto. Finalmente, *cut* se encarga de cortar la imagen según los puntos que se tienen en ese momento, aplicando el método *unwrapping* del módulo *Unwrapping* para aplicar la función de aplanamiento de imagen para figuras cilíndricas.
- La clase *RectangleExtraction* se encarga de los métodos del Modo Rectangular. Esta clase hereda todos los métodos de *FreeExtraction*. La única diferencia en la práctica con este es por el funcionamiento del método *move_vertex*, donde el movimiento de los puntos es tal que se mantiene una forma rectangular entre dichos puntos.
- La clase *SampleExtractor* comunica al módulo con la GUI. Se introduce la imagen en el módulo con *set_image*, los métodos *to_rectangle*, *to_unwrapping* y *to_free* cambian el modo del módulo a su nombre respectivo, mientras que *init_extractor* cambia el modo según el guardado en el módulo. El resto de métodos conectan los métodos internos del mismo nombre de las otras clases con la GUI, o permiten obtener datos del módulo.

1.2.6. Shape Detection

En este módulo se implementa el método de detección de formas, el cual consiste en 2 etapas:

- **Detección de formas** Este paso se lleva a cabo por la función *contour_segmentation* y consiste en la detección de todas las figuras aisladas en la imagen, para ello se ocupa el método *findContours* de la librería *cv2*, dicho método agrupa los píxeles con un color distinto al base (negro) que se encuentran en la imagen.
- **Categorización de formas** Este paso se lleva a cabo por la función *contour_agrupation*, la cuál busca agrupar las formas obtenidas anteriormente en distintos grupos (representando los distintos minerales encontrados en la foto), para ello actualmente se generan 3 grupos dependiendo del *aspect ratio* de la figura (ancho dividido en largo).

En este módulo también se encuentra la estructura de datos *ContourData* la cual encapsula una forma y permite acceder a métricas de la misma, esta clase tiene los siendo campos:

- *contour* es el contorno de la figura, el cuál tiene el mismo formato que el entregado por el método *findContours* de la librería *cv2*.
- *r* es el rectángulo inscrito en la figura.
- *group* indica el grupo al cual pertenece la figura.

Esta estructura de datos tiene diversos métodos que permite obtener información cuantitativa de la misma, todas ellas son llamadas finalmente por el método *get_all_statistics* el cual además de recuperar la información le aplica una conversión a milímetros, para mejorar la usabilidad de dichos datos.

Este módulo también contiene funciones para dibujar los rectángulos inscritos en cada figura (*cluster_segmentation*, obtener información cuantitativa de cada figura (*generate_results*) y generar una imagen por cada grupo segmentado (*image_agrupation*).

1.2.7. Test

Este módulo se encarga de validar los datos que se obtienen de cada figura (lo que se obtiene con los métodos presentes en la clase *ContourData*) comparándolos con los resultados que se deberían obtener realmente (mediante el uso de fórmulas matemáticas). Para ello se crean imágenes artificiales con polígonos de distintas formas, luego estos polígonos son procesados tanto por la clase *ContourData* como con las fórmulas matemáticas y luego estos valores se comparan y en caso de que la diferencia no supere un cierto margen se sigue comparando, en caso contrario se falla con un *AssertionError*.

Los métodos *math_** son aquellos que representan a las fórmulas matemáticas para obtener las distintas estadísticas de los testigos de roca. La función *testing* es la encargada de ejecutar las pruebas, toma como argumentos la función matemática, el método de *ContourData* y un margen de error y falla con un *AssertionError* en caso de que los valores obtenidos por ambos métodos tengan una diferencia mayor al margen del error ingresado.

En general la aplicación no presenta un margen de error mayor al 5 %.

1.2.8. Tube

Este módulo se encarga de generar una visualización 3D de la imagen ingresada, se recomienda usar este herramienta solo cuando la imagen ingresada sea una representación panorámica de un testigo de roca, pues de otra forma la imagen se verá distorsionada y no se generara una buena visualización.

La función *fill_tube* recibe una imagen (en el mismo formato que entrega el método *imread* de la librería *cv2*) y genera una visualización 3D de dicha imagen, para ello utiliza el método *read* de la librería *pyvista*, con la cual carga una malla de un cilindro (archivo *tubo.obj* presente en la carpeta *assets*) y luego ocupa el método *plot* de la librería *pyvista* para cargar la imagen como una textura del cilindro antes mencionado.

1.2.9. Unwrapping

Este módulo se encarga de recortar y realizar la corrección de perspectiva sobre el área de imagen seleccionada. Para esto se utilizan 6 puntos, dónde la parte más importante es que los puntos medios superior e inferior se ocupan para arreglar la distorsión curva del testigo seleccionado.

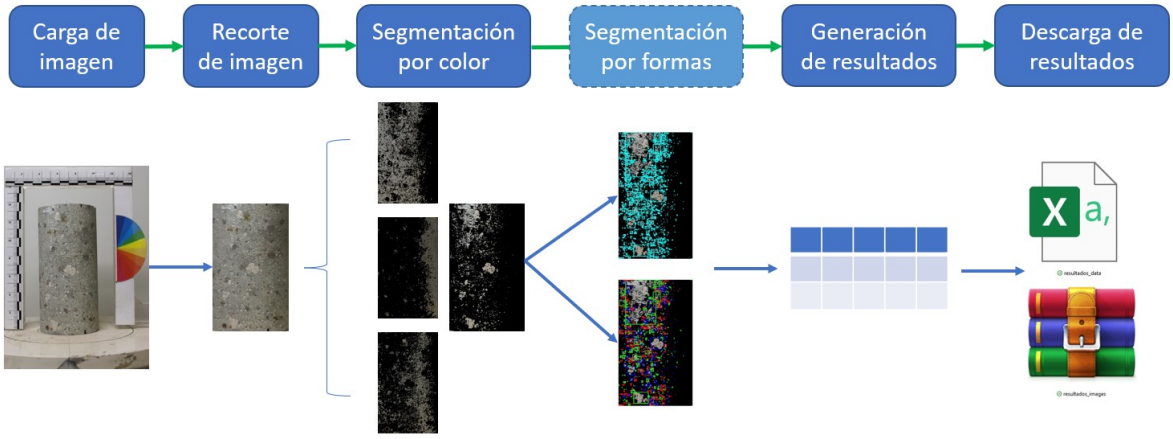
Las funciones aquí ocupadas pertenecen a *gui.py*, *sample_extraction.py* e *unwrapper.py* . En *gui.py* se ocupan las funciones *to_unwrapping* y *cut*, para pasar a la clase *SampleExtractor* en *sample_extraction.py*, la cual dirige el flujo a la clase *UnwrapperExtraction* del mismo archivo usando la función *to_unwrapping* y *cut*. Ya en la clase *sample_extraction*, la selección de los puntos se realiza con la función *cut*, para finalmente llegar al último punto del proceso, en la función *unwrapping* y la clase *LabelUnwrapper*. En esta última se realiza el proceso de corrección de perspectiva y se entrega de vuelta finalmente la imagen a la función *preview* y el proceso de análisis del testigo.

1.2.10. Utils

Este módulo contiene diversos constructores de elementos de la interfaz gráfica, como botones, *toogle* y *hovers*. También están las funciones *get_results_filepath* y *get_file_filepath* que permiten extraer el *path* de un archivo. La función *generate_zip* que genera un *zip* con los archivos que ingresan como input.

1.3. Pipeline GUI

La **GUI** sigue el siguiente pipeline:



- **Carga de la imagen**, esto se hace a través del botón *Seleccionar imagen*.
- **Recorte de la imagen**, para extraer el área relevante donde se encuentra el testigo. En este paso se le pide al usuario que ajuste el área de recorte, seleccione la altura del recorte (por defecto 20) y finalmente presione el botón *Recortar*.
- **Segmentación por color**, en este paso se le solicita al usuario la cantidad de minerales distintos presentes en el testigo para así generar esa misma cantidad de mascaras de color, luego de esto el usuario puede ajustar estas mascaras combinándolas, eliminándolas y separándolas, esto usando los botones *Combinar*, *Eliminar* y *Separar* respectivamente.
- **Segmentación por formas**, luego de aplicar una segmentación por color el usuario puede ejecutar una segunda segmentación en al cual puede agrupar las figuras presentes en la mascara seleccionada por la relación de aspecto de las mismas. Para ello el usuario debe activar el *toogle* que se encuentra debajo del botón *Analizar*.
- **Generación de resultados**, en este paso se analizan las formas encontrada en la imagen seleccionado y se calculan métricas sobre estas para obtener información cuantitativa sobre ellas. Esto se gatilla al presionar el botón *Analizar*, luego de presionar dicho botón se genera una tabla que muestra los resultados agrupados (promedio de cada métrica), divido por cada uno de los minerales presentes en la roca.
- **Descarga de resultados**, finalmente el usuario puede descargar los datos obtenidos por el software, esto lo hace al presionar el botón *Descargar* que se encuentra debajo de la tabla de resultados, al hacer esto el usuario podrá guardar un archivo *csv* con los resultados individuales de cada figura, y también un archivo *zip* que contiene una imagen de las formas relativas a cada uno de los minerales.

1.4. Releases automáticas

Actualmente el repositorio cuenta con un sistema basado en GitHub Actions que produce releases automáticamente cada vez que se realiza Push/Pull Request a la rama principal o la rama que se ocupa para desarrollo.

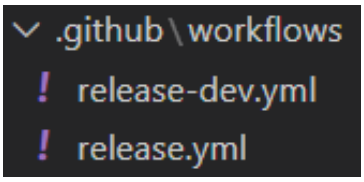


Figura 2: Workflows

Para esto se tienen dos workflows: *release.yml* para la versión estable de la aplicación que se sube a la rama Main y *release-dev.yml* para la versión en la rama de desarrollo de la aplicación.

Un workflow trabaja sobre una maquina virtual de GitHub y genera una version ejecutable del proyecto, para luego subirla a releases junto con el codigo actualizado. Para esto utiliza las siguientes GitHub Actions:

- Checkout: Para tener la repo en la maquina virtual en la que se realizan las acciones.
- Pyinstaller Windows: Para generar el ejecutable de la aplicación con pyinstaller. Para esto se utiliza una versión custom de la acción que se encuentra en esta repo, esto pues se requiere una DockerImage con una version superior de python para la correcta generación del ejecutable.
- Upload artifact: Para que el artefacto creado de la generación del ejecutable quede subida.
- Create/Update tag: Para actualizar el tag que indica la versión de la aplicación, sea *stable* o *test* con la ultima versión del código correspondiente.
- Download artifact: Para descargar el ejecutable generado y poder subirlo a Releases.
- Create Release: Para generar la release con el ejecutable y el código actualizado correspondiente al tag.

Si la acción resulta exitosa debería actualizarse la release con la nueva versión del ejecutable y el código fuente con la que se produjo. Para evitar fallos se recomienda mantener la estructura actual de carpetas y mantener los requirements actualizados, pero en caso de fallos los principales problemas ocurren a la hora de generar el ejecutable y pueden ser por directorios incorrectos, requisitos incompatibles con la versión de python usada en la imagen de Docker para la acción o derechamente un problema con la acción de github usada, por lo mismo es recomendable revisar las paginas de cada una y asegurarse de siempre estar utilizando la ultima versión.

En caso de querer cambiar el como se genera el ejecutable, se puede editar directamente *gui.spec* y ocupar el comando reducido, o cambiar el comando pyinstaller. Actualmente el comando que se utiliza es:

Código 1: Comando pyinstaller

```
pyinstaller.exe --noconsole --windowed --onefile --icon=assets\icon.ico --hidden-import=
  vtkmodules --hidden-import=vtkmodules.all  --hidden-import=vtkmodules.qt.
  QVTKRenderWindowInteractor  --hidden-import=vtkmodules.util  --hidden-import=vtkmodules
  .util.numpy_support  --hidden-import=vtkmodules.numpy_interface.dataset_adapter --add-data
   "assets\*;assets\." --collect-data sv_ttk ".\src\gui.py"
```

Ejecutandolo en la maquina local generara el ejecutable en un carpeta *dist*, ademas de esto se genera la carpeta *build* y el archivo gui.spec. Importante notar que para que el comando funcione se debe estar en el ambiente virtual que cuente con la version de pyinstaller recomendada en *requirements.txt*.

Para subsecuentes compilaciones se puede ejecutar simplemente el comando

Código 2: Comando pyinstaller reducido

```
pyinstaller.exe gui.spec
```

2. Notas

2.1. Alertas de Seguridad

Al descargar e iniciar la aplicación, es probable que windows envié una alerta sobre confianza de la aplicación.

2.2. Convenciones del software

Al momento de subir una imagen a la aplicación, esta no debe contener caracteres especiales en el nombre (por ejemplo "^{o}"), ya que si los contiene la aplicación será incapaz de abrir la imagen.