Capítulo 3: Desarrollo del sistema

En este apartado se hablará sobre las especificaciones de sistema operativo bajo las que el sistema fue desarrollado, así como de las herramientas tomadas en cuenta para el mismo motivo y el funcionamiento del algoritmo que lleva a cabo el procesamiento.

Para comenzar, el sistema operativo utilizado fue Ubuntu 16.04.4 LTS (Xenial Xerus) basado en la serie 4.4 de Linux; esto debido a que otro de los elementos base del proyecto es Robot Operating System (ROS), el cual tiene una mejor compatibilidad con el sistema de Ubuntu, comparado con sistemas de Windows o Mac OS.

En cuanto a la ROS, se utilizó la versión Kinetic Kame para establecer la comunicación del equipo con el dron Parrot Bebop, utilizado en las etapas de experimentación, para lo cual se utilizaron los drivers de bebop_autonomy, desarrollados en el Laboratorio de Autonomía de la universidad de Simon Fraser, los cuales se encargan de definir en formato de código los mensajes que pueden ser enviados hacia el vehículo para su control.

Robot Operating System es un software de código abierto bajo licencia BSD, una licencia de software permisiva con pocas restricciones y que permite el uso de código para casi cualquier tipo de distribución; el cual tiene un enfoque hacia ayudar a los desarrolladores a crear aplicaciones robóticas a través de sus múltiples herramientas como abstracción de hardware, drivers de dispositivos, librerías, visualizadores, administración de paquetes, entre otros. ROS funciona haciendo uso de nodos para establecer comunicación entre los componentes de un sistema robótico, los cuales no deben de estar ubicados necesariamente en el mismo sistema, o incluso la misma arquitectura computacional; utiliza un nodo principal a través del que los demás nodos se comunican, de esta manera, un nodo que desea enviar un mensaje a otro se convierte en un publicador, el cual envía su mensaje hacia el nodo principal bajo un tópico específico, mientras que el nodo que debe recibir el mensaje, convirtiéndose en el suscriptor, lee continuamente al nodo principal hasta encontrar algún mensaje bajo el tópico que le ha

sido asignado.

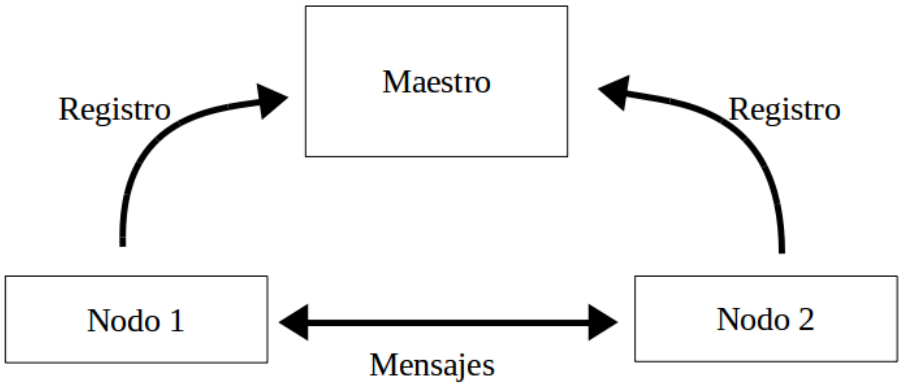

Figura 4: Representación en nivel alto del funcionamiento de ROS.

La interpretación de la Figura 4 implica que cuando un nodo publicador se registra al maestro, éste declara que publicará un tópico específico, del mismo modo, cuando un nodo suscriptor se registra, declara que se suscribirá a un tópico específico.

3.1: Herramientas de código abierto

A continuación se describen las herramientas de desarrollo disponibles como código abierto que fueron utilizadas para el desarrollo de cada etapa de este proyecto.

3.1.1: Open Drone Map

Open Drone Map (ODM) es un kit de herramientas de código abierto para el procesamiento de imágenes aéreas desarrollado por Michael Waechter, Nils Moehrle, y Michael Goesele, capaz de producir productos como nubes de puntos, modelos digitales de superficie, modelos digitales de superficie texturizados, imágenes ortorectificadas, modelos digitales de elevación, entre otros.

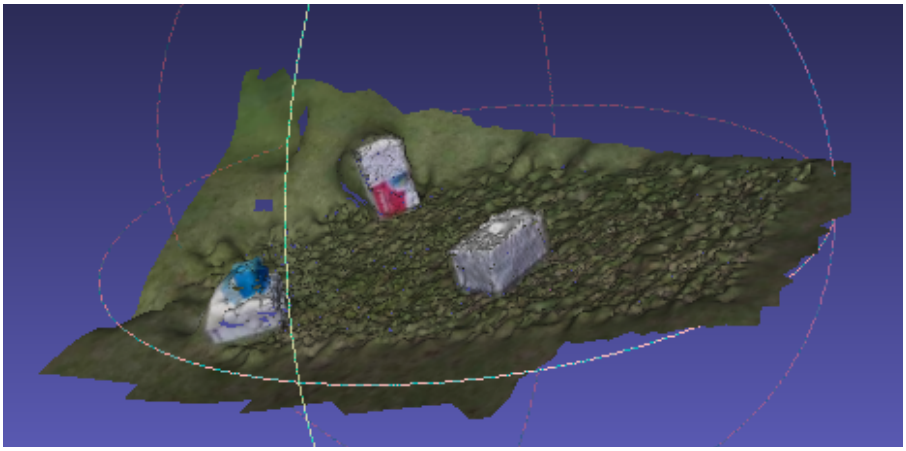

Figura 5: Mapa tridimensional de tres cajas colocadas sobre el pasto generado por Open Drone Map.

Para la generación de nubes de puntos, ODM toma imágenes de un paquete y las procesa de una en una para encontrar puntos clave en cada una de ellas. Después, toma los puntos de cada imagen, denominados descriptores, para buscarlos dentro del resto de imágenes contenidas en la base de datos, lo que le permite computar una triangulación a través de la cual se obtiene una estimación de la posición de cada descriptor en los ejes X, Y, y Z. De esta manera, se genera una nube de puntos en la que cada punto es definido por una matriz que representa su ubicación en X, Y, y Z, así como los valores correspondientes al color de su descriptor como proporciones de rojo, verde, y azul (RGB).

Open Drone Map cuenta además con una función que emplea los datos de GPS contenidos en las imágenes para realizar un mapa tridimensional con referencias geográficas, lo que significa que la malla proyectada se encontrará a una escala y orientación acertada respecto al mundo real, lo que resulta especialmente útil para los motivos de este proyecto, pues si se tiene una escala acertada, es posible realizar mediciones de la región directamente sobre el mapa generado.

Para un funcionamiento más sencillo, ODM requiere que exista una carpeta dedicada a los proyectos de mapeo que este realizará, la cual es definida como parte del proceso de

instalación, indicado en la página oficial, dentro del archivo settings.yaml que se genera dentro de la carpeta de instalación de Open Drone Map. Esto permite que el sistema sea activado con solo especificar el nombre del paquete donde se encuentran las imágenes cuando se envía la instrucción en la terminal, sin tener que especificar la ruta completa. Cabe resaltar que para que esto funcione, dentro del paquete que representa a cada proyecto, debe existir una carpeta nombrada específicamente “images”, en la que se ubican las imágenes que serán procesadas.

```
# Uncomment lines as needed to edit default settings.
# Note this only works for settings with default values. Some commands like --rerun <module>
# or --force-ccd n will have to be set in the command line (if you need to)

# This line is really important to set up properly
project_path: '/home/user/Mapping'
# Example: '/home/user/ODMProjects
```

Figura 6: Fragmento del documento settings.yaml, donde se observa cómo se define la dirección donde se leerán y escribirán los archivos generados por ODM.

3.1.2: Open Source Computer Vision Library

Open Source Computer Vision Library (OpenCV) es una librería enfocada al procesamiento de imagen computacional lanzado bajo licencia BSD, lo que permite que sea usado de forma gratuita tanto para fines académicos como comerciales, y cuenta con interfaces de C++, Python, y Java con soporte para Windows, Linux, Mac OS, iOS, y Android, lo que implica una amplia gama de aplicaciones sobre las que puede ser utilizado.

Las especificaciones para su funcionamiento dependen del tipo de uso que al que se va a enfocar; si va a ser utilizado para procesar imágenes almacenadas, basta con especificar la dirección del archivo sobre el que se desea trabajar, por otro lado, en caso de ser utilizado para procesar imágenes obtenidas en tiempo real a través de un sensor, es necesario indicar el puerto a través del que este será leído, siendo que en algunos casos, como en este proyecto, se debe utilizar una herramienta de “traducción” de formatos para que OpenCV sea capaz de leer y procesar la imagen recibida. En ambos casos, la librería es capaz de encargarse de

generar una ventana sobre la que se muestre ya sea el procesamiento realizado, o el que se está llevando a cabo en tiempo real para que el usuario pueda monitorear los resultados.

OpenCV cuenta con una amplia gama de ejemplos que se descargan durante el proceso de instalación, los cuales ayudan a comprender el uso de sus herramientas más básicas hasta las más complicadas, permitiendo que un usuario sin experiencia en procesamiento de imagen logre realizar procesamientos de imagen de complejidad media en poco tiempo, o bien, que se enfoque en aprender a utilizar una herramienta específica para emplearla en un sistema de desarrollo. Una muestra de estos ejemplos, se puede observar en la Figura 7, donde se generó una interfaz para que el usuario sea capaz de definir la tonalidad que el programa buscará dentro de la imagen recibida para detectar y resaltar el centro de figuras circulares del color definido.

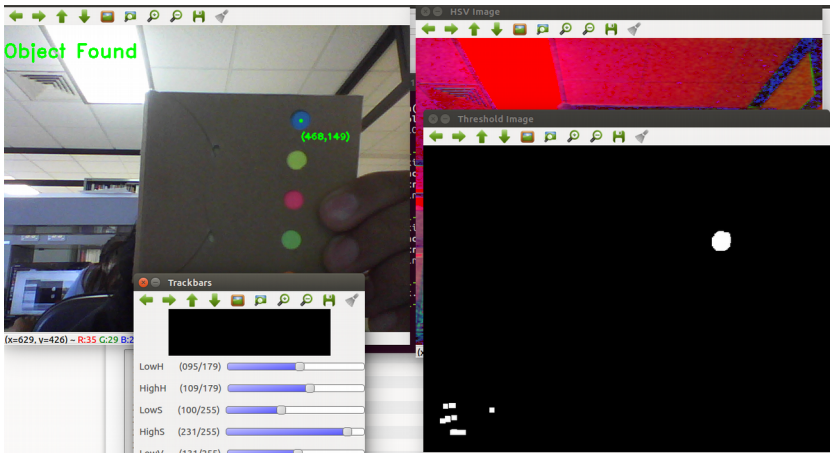

Figura 7: Detección de círculos de tonalidad variable en cámara web usando un código de ejemplo de OpenCV.

3.1.3: Point Cloud Library

Point Cloud Library (PCL) es un proyecto abierto a gran escala para procesamiento de imágenes y nubes de puntos en dos y tres dimensiones lanzado bajo licencia BSD, que contiene distintas aplicaciones y metodologías para algoritmos de filtrado, extracción de características, reconstrucción de superficies, segmentación de modelos, entre otros, que

pueden ser utilizados para filtrar datos de ruido, unir nubes de puntos tridimensionales, extraer puntos clave o encontrar descriptores correspondientes a un objeto en concreto, entre muchas otras aplicaciones.

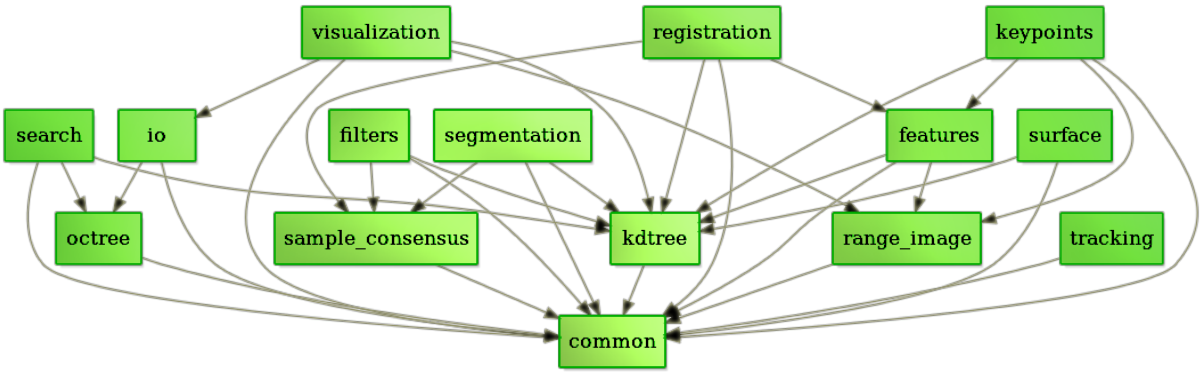

Figura 8: Gráfica de dependencia de Point Cloud Library (http://pointclouds.org/about/).

Es evidente que PCL resulta en una herramienta muy poderosa para el procesamiento de imágenes tridimensionales, sin embargo, y debido a su gran optimización sobre este aspecto, en el marco de este proyecto, Point Cloud Library fue explotado únicamente para una de sus aplicaciones más sencillas: la proyección de nubes de puntos ejecutada a través de la terminal. Esto último resultó en el factor determinante por el cual se usó esta herramienta en lugar de alguna otra, pues a diferencia de PCL, las demás no son capaces de realizar la proyección con al ejecutar una línea de comando desde la terminal.

3.1.4: Meshlab

Meshlab es un sistema de código abierto para el procesamiento y la edición de mallas tridimensionales, a través del cual es posible realizar diversas tareas de postprocesamiento tanto de nubes de puntos, como de mallas texturizadas. Realizado con el objetivo de ser una herramienta sustentable, eficiente, y fácil de utilizar, por lo que resulta en “una aplicación de visualización de mallas, donde un objeto tridimensional, almacenado en una variedad de formatos puede ser cargado e inspeccionado interactivamente de forma sencilla, simplemente

arrastrando y dando clic en la misma malla” (Cignoni et al., 2008).

Es por esta simplicidad, que se optó por utilizar esta herramienta, dado que el uso que se le dió a lo largo de este proyecto fue únicamente para visualizar e inspeccionar las mallas tridimensionales producidas por Open Drone Map, sin explotar sus herramientas de post procesamiento controlado de forma manual.

3.2: Algoritmo

El sistema está distribuido en tres etapas principales: selección de imágenes en tiempo real a partir de una transmisión de video, inicialización de mapeo, y visualización de resultados. Dependiendo del tipo de aplicación, el código desarrollado puede ser alterado a modo de ajustarse con los resultados esperados; en su formato original, los mapas son creados cada vez que se recupera un “n” número de imágenes, siendo “n” un parámetro ajustable. De este modo, el mapa visualizado cada vez que termina el proceso de mapeo actual representa la información de las imágenes captadas hasta el modelo previo junto con las nuevas imágenes recuperadas. Por otro lado, también es posible configurar el sistema para que el proceso de mapeo no inicie sino hasta que se ha finalizado por completo la etapa de filtrado de imágenes, es decir, hasta que se considera que durante el vuelo se ha captado información suficiente del área a mapear, de esta manera es posible decrementar el tiempo de procesamiento, pues se podría prescindir de las dos etapas consecuentes hasta el momento en que el vuelo ha finalizado, permitiendo así que, cuando se inicia el mapeo, todos los recursos del procesador sean dedicados exclusivamente a ello.

3.2.1: Filtrado de imágenes

El algoritmo de filtrado de imágenes está basado en el procedimiento Feature Matching de la librería de OpenCV (Open Source Computer Vision Library) (OpenCV library, 2018), que

analiza una imagen para encontrar puntos clave, denominados descriptores, los cuales son regularmente esquinas, dado que son los elementos más fáciles de detectar geométricamente. A través del método de comparación de fuerza bruta, el sistema extrae los descriptores de la primera imagen de referencia, la cual es obtenida inmediatamente al recibir imagen de video, es decir, en cuanto el sistema ha sido activado. Esta imagen es almacenada dentro de la base de datos para realizar los mapeos, y los descriptores que contiene son comparados con los que se encuentran procesando en tiempo real la señal de video recibida; de esta manera, una nueva imagen de referencia es obtenida y almacenada en la base de datos cuando la cantidad de puntos clave encontrados en la imagen en vivo, comparada con la referencia actual, es menor a un porcentaje de umbral definido en el código. Esto se hace con el objetivo de reducir la cantidad de imágenes que se analizan durante la siguiente etapa, lo que resulta en un decremento del tiempo total de procesamiento.

Es importante destacar que la imagen recibida por el dron a través de ROS se encuentra en un formato diferente al que OpenCV utiliza para realizar el procesamiento, por lo que se debe de utilizar una herramienta de ROS denominada CvBridge, la cual es capaz de traducir el mensaje de imagen recibido directamente del sensor de cámara del dron al formato cv::Mat que requiere OpenCV para realizar su funcionamiento. De esta manera, se generó un nodo que monitorea al tópico ‘/bebop/image_raw’ para buscar las imágenes que se espera sean publicadas por el dron, o por un publicador de video que emula el envío de mensajes del sensor fotográfico de un Bebop, lo que permite que los mensajes sean procesados en el código por la herramienta de CvBridge para convertirlos al formato en el que las herramientas de OpenCV pueden procesarlos. Además, durante este proceso, se crea una ventana dentro de la que se proyectarán las imágenes recibidas y la imagen de referencia.

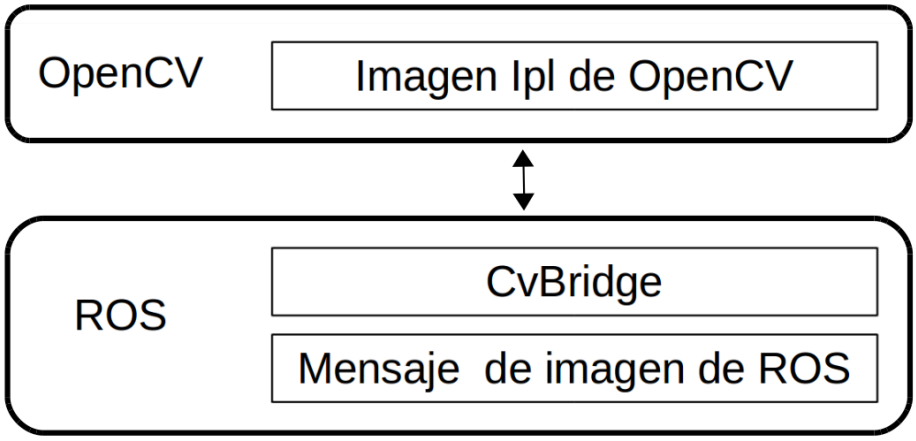

Figura 9: Representación gráfica del funcionamiento de ROS con OpenCV, a través de CvBridge.

En cuanto el video ha sido convertido al formato apropiado, se definen los arreglos que contienen a los descriptores tanto de la imagen en vivo, como de la imagen de referencia,al igual que la cantidad máxima de descriptores que se buscarán en cada imagen. Este último parámetro es especialmente importante, debido a que la cantidad de descriptores contenidos en la imagen determinan la eficacia del umbral de coincidencia que se usa para decidir el momento en el que se almacena un nuevo cuadro; una cantidad baja de descriptores provoca que el umbral sea vencido con facilidad, lo que resulta en un almacenamiento excesivo de imágenes, por lo que se encontró que una búsqueda de un máximo de 2200 descriptores es lo ideal, pues a partir de este umbral la cantidad de descriptores encontrados se mantiene prácticamente constante. Las Figuras 10 y 11 muestran la diferencia entre una búsqueda de 10 y 2200 descriptores, así como las Figuras 11 y 12 muestran la similitud en la cantidad de puntos encontrados entre una búsqueda de 2200 y 3000 descriptores.

Figura 10: Búsqueda de un máximo de 10 descriptores sobre la imagen.

Figura 11: Búsqueda de un máximo de 2200 descriptores sobre la imagen.

Figura 12: Búsqueda de un máximo de 3000 descriptores sobre la imagen.

Después de definir los elementos que contienen a los descriptores, es necesario definir los vectores que contienen tanto las posibles coincidencias como las coincidencias confirmadas, a partir de lo cual se inicia el procedimiento de comparación de descriptores de OpenCV, lo que implica la publicación de las imágenes dentro de la ventana creada previamente, una junto a la otra, sobre las cuales se dibujan círculos de múltiples colores encerrando a todos los descriptores encontrados en ambos lados de la ventana, los cuales son, a su vez, unidos por una línea con el descriptor que les corresponde sobre la imagen contraria a la que se encuentran ubicados, como se puede observar en las Figuras 13 y 14.

Figura 13: (Arriba) Muestra una comparación de imágenes con una búsqueda de 10 descriptores (se observa también que un mismo descriptor de la imagen almacenada fue unido erróneamente con dos descriptores de la imagen actual). (Abajo) Muestra una comparación de imágenes con una búsqueda de 100 descriptores.

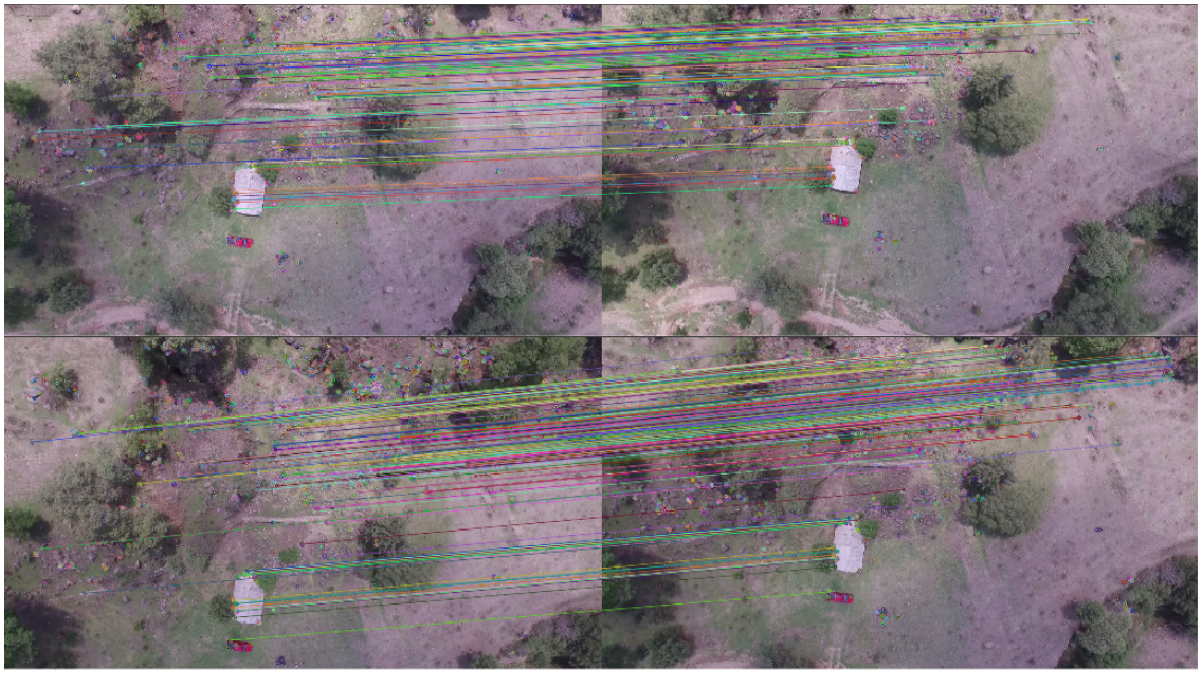

Figura 14: (Arriba) Muestra una comparación de imágenes con una búsqueda de 1000

descriptores. (Abajo) Muestra una comparación de imágenes con una búsqueda de 2200 descriptores, se observa una mayor cantidad tanto de descriptores dispersos, como de descriptores emparejados.

Finalmente, ya que el proceso de comparación de imágenes ha sido establecido, es necesario crear un procedimiento para garantizar que, una vez que el umbral ha sido superado, la imagen que se actualiza como la nueva imagen de referencia sea almacenada en el paquete indicado para realizar el mapeo. Para esto, dentro del código se especifican las instrucciones necesarias para crear uno o más nuevos directorios, según sea el formato de uso del sistema, y para crear dentro de estos el paquete “images” antes mencionado, en el que se deben almacenar las imágenes. Dado que el proceso se ejecuta desde la terminal, esto se logra utilizando usando la instrucción ‘mkdir’, que se encarga de crear paquetes en ubicaciones específicas con el nombre que se le indica; a su vez, para ejecutar la instrucción desde el código sobre la terminal de ejecución, esta es enviada usando la función ‘system()’, que toma como argumento una cadena que contiene la instrucción o serie de instrucciones que se desean ejecutar desde la terminal donde el código ha sido inicializado. La dirección sobre la que los paquetes serán creados, depende de la locación que el usuario haya seleccionado para la ubicación de los proyectos de ODM, en este caso, los paquetes deben ubicarse dentro del directorio “Mapping” en el folder personal, como se observa en la Figura 6.

3.2.2: Mapeo

La etapa de mapeo se realiza utilizando el sistema de postprocesamiento OpenDroneMap que, como se mencionó previamente, es capaz de entregar mapas tridimensionales texturizados, sin embargo, a modo de reducir el tiempo de procesamiento, en la mayoría de los casos se optó por detener la computación en cuanto se genera la nube de puntos; no

obstante, es posible permitir al sistema realizar el procesamiento completo de ser necesario.

En caso de que se desee que esta etapa se realice de manera paralela con el proceso de filtrado de imágenes, es decir, que se produzcan mapas cada cierto tiempo (medido en base a la cantidad de imágenes capturadas) durante el vuelo, es necesario que el mapeo sea inicializado en una terminal diferente de la que está llevando a cabo la primera etapa, pues de otra manera esta sería interrumpida hasta que el mapeo haya finalizado, lo que implica la pérdida de todas las imágenes que serían captadas durante este proceso. Para lograr esto, se hace uso nuevamente de la función 'system()', esta vez usando como primera parte del argumento la instrucción "gnome-terminal -x sh -c ' ' ", que indica la creación de una terminal externa, sobre la que es posible ejecutar nuevas instrucciones (colocadas dentro de los apóstrofes, y separadas por un punto y coma para indicar una instrucción diferente), por lo que es a partir de ella que se inicializa el proceso de Open Drone Map, con la posibilidad de realizar el mapeo completo, o bien, usar alguno de los parámetros de ejecución con los que cuenta; por ejemplo '--end-with ' seguido del paquete generado por ODM con el que se desea marcar el final del procesamiento (se usa 'opensfm' para detener el proceso cuando la nube de puntos ha sido generada).

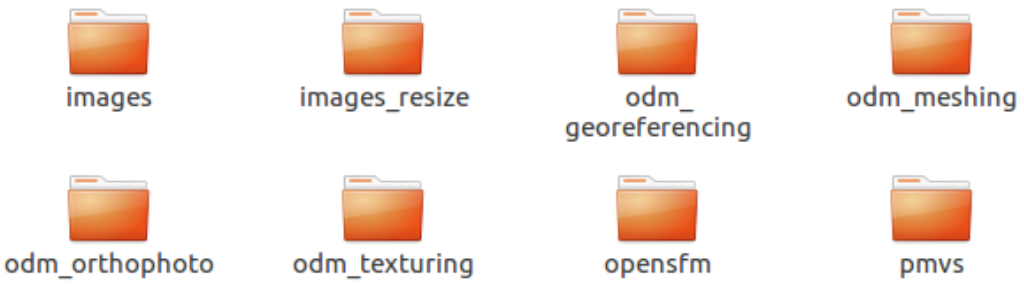

Figura 15: Paquetes creados a partir del procesamiento de ODM; la nube de puntos se encuentra en el directorio /opensfm/depthmaps/merged.ply, mientras que la malla se encuentra en /odm_texturing/odm_textured_model.obj.

Además, antes de inicializar el mapeo, se accede a otra terminal para tomar algunas de las imágenes del paquete actual y copiarlas hacia el siguiente utilizando el comando ‘cp -b’, seguido de dos argumentos que representan el paquete donde existe el archivo, y el paquete hacia el que será copiado. Esto es realizado para garantizar que existan puntos coincidentes entre cada mapa y su posterior, de modo que se abra la posibilidad de computar una unión de nubes de puntos o mapas texturizados en post procesamiento.

3.2.3: Visualización

Finalmente, para la etapa de visualización de resultados, se generaron dos archivos ejecutables, uno para la proyección de nube de puntos, y uno para la corrección del archivo que los contiene, pues se encontró que para que la nube pueda ser proyectada, el archivo “merged.ply”, generado por Open Drone Map en el paquete “opensfm/depthmaps” dentro del paquete del proyecto, debe ser abierto y guardado, pues de otro modo el visualizador arroja un error de formato.

Para realizar estas dos acciones, se crearon archivos ejecutables a partir de un código en C++, el primero simplemente accede al archivo especificando la ruta completa de la ubicación del archivo de formato poligonal como argumento al ejecutar el archivo. El segundo es un corto código que hace uso de la librería conocida como Point Cloud Library para activar un visualizador sobre el que se proyecta la nube de puntos contenida en el archivo PLY afectado por el archivo ejecutable previo. Ambos archivos son ejecutados desde la terminal externa de la misma manera que se ejecuta ODM desde el código.

Es recomendable que estos archivos ejecutables sean colocados dentro del paquete de Open Drone Map, pues por simplicidad es desde este directorio que se comienza todo el sistema, debido a que el proceso de mapeo debe ser iniciado específicamente desde ese directorio, por lo que, al incluir los ejecutables de guardado y lectura de archivos poligonales, se reduce la

cantidad de instrucciones enviadas sobre la terminal externa, pues de este modo no es necesario realizar cambios de directorio. Además, se destaca que el proceso de visualización dentro del código funciona únicamente para nubes de puntos, si se desea visualizar el mapa tridimensional texturizado, se utiliza el visualizador de Meshlab, posterior al procesamiento.

Al ser un sistema basado en tres etapas, en la que las dos primeras son bastante robustas, el sistema puede resultar pesado para algunos equipos; especialmente si el control remoto del UAV se realiza a través de la misma computadora, lo que podría resultar en la pérdida de control del vehículo. Por esto, es necesario hacer énfasis en que es posible reducir las demandas de procesamiento con algunos simples ajustes en el código. Por ejemplo, de ser necesario reducir al máximo el consumo de recursos computacionales para garantizar el funcionamiento estable del equipo, el código puede simplificarse hasta contar únicamente con el filtrado de imágenes, de modo que, durante el vuelo, el sistema se dedique únicamente a recopilar las imágenes que caen bajo el umbral cuando son comparadas con la imagen previa; con esto, el proceso de mapeo puede ser realizado fuera de línea en cualquier momento haciendo uso de la base de datos que se generó con la primera etapa. En cuanto al consumo de recursos gráficos, el único punto del sistema que demanda esta clase de recursos es la visualización de resultados, de modo que esta etapa puede ser omitida por completo si se desea que el mapa o la nube de puntos generada sean visualizados en algún momento posterior al procesamiento. Por otra parte, la cantidad de mapas que se generan durante el vuelo puede verse reducida al incrementar la cantidad de cuadros que son capturados por paquete o reduciendo aún más el umbral de selección de imágenes (esto último puede afectar directamente la resolución y precisión del mapa tridimensional), provocando que el cambio de paquete se prolongue, lo que resulta en un retraso en el inicio del procesamiento del siguiente mapa actualizado.