Escuela Politécnica Superior de Jaén
Grado en Ingeniería Informática

Universidad de Jaén
Escuela Politécnica Superior
de Jaén

TRABAJO FIN DE GRADO

PROTOTIPO DE CONTROL AVANZADO DE GRANDES PLANTACIONES MEDIANTE TELEDETECCIÓN

Alumno

Alfonso López Ruiz

Tutor

Carlos Javier Ogayar Anguita
(Departamento de Informática)

Francisco Ramón Feito Higueruela
(Departamento de Informática)

Junio, 2019

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Carlos Javier Ogayar Anguita y Don Francisco Ramón Feito Higueruela, tutores del Trabajo Fin de Grado titulado: '**Prototipo de control avanzado de grandes plantaciones mediante teledetección**', que presenta Don Alfonso López Ruiz, otorgan el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, junio de 2019

El alumno:

LOPEZ RUIZ ALFONSO - 26509386T

Firmado digitalmente por LOPEZ RUIZ ALFONSO - 26509386T
Fecha: 2019.06.20 21:58:24 +02'00'

Alfonso López Ruiz

Los tutores:

FEITO HIGUERUELA FRANCISCO RAMON - 51865598T

Firmado digitalmente por FEITO HIGUERUELA FRANCISCO RAMON - 51865598T
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-51865598T, givenName=FRANCISCO RAMON, sn=FEITO HIGUERUELA, cn=FEITO HIGUERUELA FRANCISCO RAMON - 51865598T
Fecha: 2019.06.20 23:19:41 +02'00'

OGAYAR ANGUITA CARLOS JAVIER - 26039170L

Firmado digitalmente por OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Nombre de reconocimiento (DN): c=ES, serialNumber=26039170L, sn=OGAYAR ANGUITA, givenName=CARLOS JAVIER, cn=OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Fecha: 2019.06.21 08:37:41 +02'00'

Carlos Javier Ogayar Anguita

Francisco Ramón Feito Higueruela

(Página intencionalmente en blanco)

Agradecimientos

En estos parráfos me gustaría expresar mi agradecimiento a todos aquellos que han colaborado de alguna manera en la realización de este trabajo.

En primer lugar, agradezco a mis tutores, Francisco Feito y Carlos Ogayar, así como a Juanma, el tiempo que han empleado generosamente en guiar este proyecto a lo largo de tantas reuniones.

Un agradecimiento especial a mis padres, José Luis y Ana, que siempre han facilitado la realización de mis estudios académicos y se han sacrificado para ello.

Por último, un gran agradecimiento a mi hermano, José Luis, por su apoyo durante todos estos años, que ha utilizado amablemente sus horas de estudio en época de exámenes para revisar este documento (¡siempre me falta una coma!).

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Trabajo Teórico/Experimental
Especialidad (solo TFG)	Tecnologías de la Información
Mención (solo TFG)	Sistemas Gráficos
Idioma	Español
Tipo	Específico
TFT en equipo	No
Fecha de asignación	09/04/2019
Resumen	La tecnología, y por lo tanto la agricultura de precisión, suponen un factor clave para la consecución del aumento de la producción en la agricultura de la forma más óptima posible. Se propone así en este trabajo la utilización de datos de varios sensores (multiespectrales, térmicos, etc) para obtener información útil de un cultivo, no sin antes revertir todos los errores que puedan contener las capturas realizadas, de tal manera que estas se transforman hasta alcanzar un estado que se considera óptimo para el estudio. Se definen pues una serie de algoritmos y operaciones para la consecución de objetivos tales como la extracción de árboles o la georreferenciación de puntos de una imagen. Habrá que probar la viabilidad de dichos algoritmos para su introducción en una futura aplicación de utilidad para el agricultor.
Abstract	Technology, and so the precision agriculture, are a key factor to enable the production increase in the most optimal way in the agriculture. This work proposes the use of data from several sensors, such as multispectral or thermal, to obtain useful information for a crop, and the first step for such an objective is to reverse the errors of the recorded data so we can achieve its optimal state for the research. We propose a set of algorithms and operations to achieve goals such as tree recognition or georeferencing image points. Therefore, we need these processes as correct in order to include them in a final application which helps directly the farmer.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

Contenido

1 - Descripción del trabajo 19

1.1 - Introducción 19

1.2 - Antecedentes y estado del arte 21

1.2.1 - Antecedentes 21
1.2.2 - Estado del arte 24

1.3 - Definiciones y abreviaturas 31

1.3.1 - Abreviaturas 32
1.3.2 - Terminología 33

1.4 - Objetivos del trabajo 35

1.5 - Alcance 37

1.6 - Metología de desarrollo 38

1.7 - Material y métodos 42

1.8 - Normas y referencias 43

1.8.1 - Licencias software 43

1.9 - Planificación temporal 47

1.10 - Presupuesto 52

1.10.1 - Coste hardware 53
1.10.2 - Coste software 55
1.10.3 - Coste de materiales 56
1.10.4 - Coste de personal y servicios 57
1.10.5 - Coste derivado del proyecto 58
1.10.6 - Coste total 58
1.10.7 - Amortización 59

2 - Desarrollo del proyecto 61

2.1 - Tecnologías utilizadas 61

2.2 - Desarrollo iterativo 66

2.2.1 - Primera Iteración y segunda iteración 67
2.2.2 - Tercera iteración 76
2.2.3 - Cuarta iteración 89
2.2.4 - Quinta iteración 96
2.2.5 - Sexta iteración 105
2.2.6 - Séptima iteración 108

2.2.7 - Octava iteración 158
2.2.8 - Novena iteración 160
2.2.9 - Décima iteración 173
2.2.10 - Undécima iteración....181
2.2.11 - Duodécima iteración....183
2.2.12 - Diagrama de clases....207
2.2.13 - Patrones de diseño 232

3 - Experimentación....237

3.1 - Conceptos previos 238

3.1.1 - Coeficiente de correlación 238

3.2 - Extracción de contornos 239

3.3 - Registro de imágenes multiespectrales 242

3.4 - Registro de imágenes RGB y multiespectrales....252

3.5 - Registro de imágenes RGB y térmicas....256

3.6 - Especificación de prototipos realizados 262

4 - Resultados y Discusión....263

5 - Conclusiones y trabajos futuros....267

6 - Apéndices 271

6.1 - Instalación y configuración del sistema....271

6.2 - Manuales de usuario....275

6.2.1 - Controles de teclado y ratón....282

6.3 - Publicaciones Científicas 284

6.4 - Guía original del Trabajo Fin de Título....284

7 - Bibliografía....285

Índice de ilustraciones

Ilustración 1. Cámara multiespectral Parrot Sequoia. 20
Ilustración 2. SARP DJI Matrice 210 empleado por el departamento para obtener la información de entrada de este trabajo. Fuente: (Grupo Acre, 2019). 22
Ilustración 3. Imágenes multiespectrales utilizadas durante el desarrollo del proyecto. Se muestran aquí cuatro capas distintas que representan cuatro intervalos diferentes del espectro. a) GRE b) NIR c) RED d) REG. 23
Ilustración 4. Imagen multiespectral en su formato original, con distorsión fisheye. 24
Ilustración 5. Comparativa entre el modelo de distorsión fisheye radial y fullframe. 25
Ilustración 6. Imágenes multiespectrales superpuestas sin alinear (efecto de image ghosting) en la izquierda, y a la derecha, las mismas imágenes multiespectrales alineadas y superpuestas. 26
Ilustración 7. Distinción de vegetación respecto de su sombra y el suelo, implementado por Micasense (MicaSense, 2018). 28
Ilustración 8. Cosido de imágenes (izquierda) y ortomosaico (derecha) (Pix4D, 2019). 29
Ilustración 9. Dispositivo de captura de imagen RGB y térmica, DJI Zenmuse XT2. Fuente: (Grupo Acre, 2019). 30
Ilustración 10. Imagen RGB (izquierda) e imagen térmica (derecha). 30
Ilustración 11. Esquema de los roles que define la metodología ágil SCRUM. 40
Ilustración 12. Esquema del ciclo de vida de un proyecto de metodología ágil (basado en iteraciones). 42
Ilustración 13. Tareas del proyecto distribuidas a lo largo de las iteraciones junto al tiempo dedicado a cada una de ellas. 50
Ilustración 14. Diagrama de Gantt para la planificación temporal del proyecto. 52
Ilustración 15. Composición donde se muestran todas las imágenes capturadas por el dispositivo Parrot Sequoia para un determinado área de terreno. 68
Ilustración 16. Distancia física en milímetros entre las lentes de Parrot Sequoia (Parrot , 2017). 68
Ilustración 17. Resultado obtenido al suavizar y umbralizar la imagen multiespectral RED. .69
Ilustración 18. Extracción de contornos con un núcleo de emborronamiento de tamaño 10x10. 70
Ilustración 19. Superposición de la imagen multiespectral NIR, con una opacidad del 50%, y la umbralización obtenida. 71
Ilustración 20. Representación del algoritmo de inundación para una imagen cualquiera, donde los desniveles aquí ilustrados vienen dados por la intensidad de la imagen. 72
Ilustración 21. Ejemplo de marcadores de una imagen, zonas de donde se partirá en el algoritmo. 73
Ilustración 22. Obtención de contornos en la imagen multiespectral RED mediante el algoritmo de inundación. 74
Ilustración 23. Storyboard de la aplicación desarrollada hasta la segunda iteración. 75
Ilustración 24. Comparativa de las bandas RED y NIR respectivamente. 76
Ilustración 25. Reflectancia de la vegetación a lo largo del espectro (Humboldt State University, 2018). 77
Ilustración 26. Resultado de la diferencia de imágenes multiespectrales NIR y RED. 78
Ilustración 27. Suavizado de la diferencia entre las imágenes NIR y RED. 79

Ilustración 28. Umbralización de la imagen suavizada de la Ilustración 27. 79
Ilustración 29. Resultado que se obtiene al dibujar los contornos extraídos sobre la imagen multiespectral RED. 81
Ilustración 30. Proceso completo de extracción de contornos a partir de las imágenes multiespectrales RED y NIR. 82
Ilustración 31. Diferencia entre imágenes multiespectrales rectificadas. 83
Ilustración 32. Pseudocódigo del flujo de operaciones necesarias para extraer los contornos de los árboles presentes en la escena. 83
Ilustración 33. Comunicación entre entidades y actores para llevar a cabo la extracción de contornos de imágenes multiespectrales. 84
Ilustración 34. Intercambio de mensajes para llevar a cabo la operación de abrir imágenes multiespectrales. 85
Ilustración 35. Secuencia de vistas necesarias para obtener los contornos a partir de los tres parámetros expuestos anteriormente. 88
Ilustración 36. Envolvente convexa para un conjunto de puntos P. Véase como el segmento $\boldsymbol{pq}$ queda dentro de la envolvente. 90
Ilustración 37. Comparativa entre los contornos originales de la vegetación y las envolventes convexas obtenidas a partir de los polígonos iniciales. 91
Ilustración 38. Comparativa entre los contornos originales de la vegetación y las envolventes hexagonales obtenidas a partir de los polígonos iniciales. 92
Ilustración 39. Representación de centros de masas en contornos originales, junto al detalle de un polígono cuyo centro de masas no se incluye dentro del mismo. 93
Ilustración 40. Centros de masas para el resto de contornos derivados de los originales. 93
Ilustración 41. Diagrama de secuencia para la extracción de contornos y manipulación del resultado para su simplificación y visualización óptima. 94
Ilustración 42. Vista que nos permite seleccionar cómo se renderizan los contornos resultantes y si se incluyen o no los centros de masas. 95
Ilustración 43. Vistas que representan las operaciones necesarias para el almacenamiento de imágenes en el sistema de ficheros. 96
Ilustración 44. Imagen multiespectral RED donde se puede observar la distorsión fisheye que genera la lente. 97
Ilustración 45. Esquema de eliminación de la distorsión de ojo de pez sobre una imagen multiespectral. 97
Ilustración 46. Detalle de una imagen a la que se le aplica una transformación cualquiera, generando un resultado donde se pueden ver píxeles de intensidad con el color por defecto, 0. 99
Ilustración 47. Representación de los triángulos similares que se generan cuando se muestra una posición de la imagen final y el punto de la imagen distorsionada de donde se obtiene su valor de intensidad. 101
Ilustración 48. Imagen sin distorsión fisheye tras aplicar el algoritmo que se acaba de exponer. 102
Ilustración 49. Pseucódigo del algoritmo de eliminación de la distorsión fisheye. 102
Ilustración 50. Diagrama de secuencia para la acción de eliminar la distorsión fisheye de las imágenes multiespectrales. 104
Ilustración 51. Vistas de la aplicación con las que interactúa el usuario para eliminar la distorsión fisheye. 105

Ilustración 52. Esquema de comunicación entre entidades del modelo MVC en nuestra aplicación. 106
Ilustración 53. Modelo MVC jerárquico. 107
Ilustración 54. Caso optimista donde la cámara captura el terreno en posición nadir. 110
Ilustración 55. Combinación de todas las imágenes multiespectrales en una misma imagen, utilizando cierta opacidad para todas las capas (excepto la base). 111
Ilustración 56. Esquema con todos los parámetros necesarios para calcular la cantidad de metros que corresponde a cada píxel. 112
Ilustración 57. Representación de los diferentes tipos de transformaciones que se pueden utilizar al registrar imágenes en OpenCV. 116
Ilustración 58. Pseudocódigo de registro de imágenes multiespectrales. 119
Ilustración 59. Representación del árbol que se forma entre imágenes multiespectrales para efectuar el registro. 121
Ilustración 60. Resultado de ejecutar el algoritmo de registro de imágenes sobre la multiespectral NIR. 121
Ilustración 61. Superposición de las cuatro imágenes multiespectrales registradas con un cierto nivel de opacidad. Desaparece el efecto de image ghosting. 123
Ilustración 62. Comparativa entre una captura GRE y RGB tras aplicar el algoritmo de registro de imágenes. 125
Ilustración 63. Combinación de imágenes GRE y RGB a través de la opacidad, una vez han sido registradas. 125
Ilustración 64. Representación de la salida del dispositivo Parrot Sequoia como un modelo multicapa. 126
Ilustración 65. División del globo terrestre en zonas y cuadrantes. Sistema de referencia UTM. Autor: Jan Krymmel. 130
Ilustración 66. Representación del punto principal de una imagen y otro punto cualquiera del que se desea conocer su localización en el sistema de coordenadas UTM. 135
Ilustración 67. Representación de las tres rotaciones propuestas en función de los ángulos de Tait-Bryan sobre un avión. Fuente: (CHRobotics, s.f.). 136
Ilustración 68. Sistema de coordenadas de la imagen. 137
Ilustración 69. Escena donde el dron alcanza un ángulo pitch relativamente elevado, de tal forma que la escena difiere en gran medida de lo que se obtendría con una orientación nadir. 140
Ilustración 70. Imágenes multiespectrales junto al ángulo yaw presente en sus metadatos. 142
Ilustración 71. Vista aérea de la estructura que aparece en las imágenes multiespectrales de la Ilustración 70, junto a la indicación del norte de la Tierra (Google, s.f.). 142
Ilustración 72. Método de rectificación de los ángulos contenidos en los metadatos de las imágenes multiespectrales. 143
Ilustración 73. Imágenes con ángulos rectificados sin aplicar la corrección de 125º. 144
Ilustración 74. Rectificación de las imágenes con el algoritmo anteriormente presentado. 144
Ilustración 75. Esquema de selección de vecinos y cálculo de distancias en la interpolación bilineal. 147
Ilustración 76. Interpolación bilineal sin comprobación de existencia de vecinos. 147
Ilustración 77. Posicionamiento de las esquinas de una imagen multiespectral en el mapa. 151

Ilustración 78. Imagen rotada de la que se extraen los datos georreferenciados. 151
Ilustración 79. Operaciones hasta alcanzar el registro de imágenes. 153
Ilustración 80. Operaciones a realizar hasta eliminar la rotación de una imagen multiespectral. 153
Ilustración 81. Flujo de operaciones hasta obtener un fichero con los datos de la imagen multiespectral georreferenciados. 154
Ilustración 82. Flujo de operaciones hasta obtener un fichero con información exclusivamente de centros de masas de contornos. 154
Ilustración 83. Modificación de la altura estimada del dron almacenada en la aplicación. ..155
Ilustración 84. Vistas de la aplicación necesarias para llevar a cabo todas las operaciones expuestas en este apartado. 158
Ilustración 85. Integración de varios conjuntos de multiespectrales en la aplicación. 159
Ilustración 86. Barra de progreso durante el proceso de registro de imágenes. 160
Ilustración 87. Proceso de obtención de la línea epipolar de la imagen 2. 161
Ilustración 88. Relación geométrica que se establece entre todos los componentes vistos anteriormente para formar un plano. 162
Ilustración 89. Representación de un sistema estereoscópico ideal, donde los ejes ópticos son claramente paralelos. 163
Ilustración 90. Pseudocódigo del algoritmo de generación del mapa de disparidad entre dos imágenes multiespectrales tras el registro. 165
Ilustración 91. Gradiente que representa la disparidad de los puntos al aplicarse una transformación global. 166
Ilustración 92. Subpixel Stereo Correspondence (Evangelidis, 2005). 166
Ilustración 93. Ortomosaico objetivo de este apartado. 167
Ilustración 94. Pseudocódigo del método principal para la inserción de una nueva imagen en el ortomosaico. 169
Ilustración 95. Pseudocódigo para comprobar si una imagen se puede integrar en el ortomosaico actual. 169
Ilustración 96. Pseudocódigo para actualizar coordenadas UTM del ortomosaico. 170
Ilustración 97. Pseudocódigo para transformar una coordenada UTM en una posición dentro de la imagen del ortomosaico. 170
Ilustración 98. Integración de la nueva imagen en el ortomosaico actual. 171
Ilustración 99. Ortomosaicos generados con 2 y 3 imágenes respectivamente. 171
Ilustración 100. Diagrama de secuencia para la acción de componer el ortomosaico. 172
Ilustración 101. Vistas involucradas en la composición del ortomosaico. 173
Ilustración 102. Representación de la imagen multiespectral sobre un plano. 174
Ilustración 103. Comparativa entre el renderizado en modo malla de triángulos y malla de alambre. 175
Ilustración 104. Pseucódigo para triangular un polígono mediante el algoritmo de Ear Clipping. 177
Ilustración 105. Representación de contornos con sus tapas generadas mediante el algoritmo de Ear Clipping. 177
Ilustración 106. Centros de masas de los contornos, representados como esferas. 178
Ilustración 107. Resultado de la escena tras aplicar cada luz y resultado final. 179
Ilustración 108. Comparativa de aliasing al aplicar un número de samples distinto. A la derecha, la captura por defecto sin buffer secundario. 179

Ilustración 109. Diagrama de secuencia que muestra la interacción que puede llevar a cabo el usuario con el módulo 3D. 180
Ilustración 110. Vistas involucradas en la visualización 3D de los contornos extraidos previamente. 181
Ilustración 111. Información aportada por SIGPAC sobre una parcela. 182
Ilustración 112. Esquema lógico de tipo Entidad-Relación de una base de datos destinada a guardar información estructural de árboles a lo largo del tiempo. 183
Ilustración 113. Imagen RGB y multiespectral capturadas en instantes de tiempo muy cercanos....184
Ilustración 114. Comparativa de imagen RGB y multiespectral sin distorsión fisheye. 185
Ilustración 115. Flujo de operaciones a llevar a cabo para registrar una imagen RGB y una multiespectral. 188
Ilustración 116. Pseudocódigo para el registro de una imagen multiespectral en una RGB. 188
Ilustración 117. Resultados de ejecutar el algoritmo de registro de imágenes RGB-multiespectral. 189
Ilustración 118. Superposición de resultados del registro sobre imágenes RGB originales a través del canal alpha....190
Ilustración 119. Flujo de operaciones para extraer la vegetación de una imagen RGB. 191
Ilustración 120. Imagen RGB con un color de terreno muy diferente de las imágenes anteriores. 192
Ilustración 121. Algoritmo mejorado para la extracción de contornos en la imagen RGB. ..193
Ilustración 122. Resultados del algoritmo de extracción de árboles para dos tipos de terrenos diferentes. 194
Ilustración 123. Comparativa de imagen térmica y RGB. Detalle de huellas donde se aprecia la distorsión de esta segunda. 197
Ilustración 124. Esquema de registro de imagen RGB con una imagen térmica. 199
Ilustración 125. Pseudocódigo para obtener el registro de una imagen térmica y una imagen RGB. 199
Ilustración 126. Registro utilizando un área de mayor tamaño. 200
Ilustración 127. Registro de imágenes donde se muestra sólo el área recortada de la imagen RGB y la térmica superpuesta, eliminando el fondo negro. 200
Ilustración 128. Registro utilizando un área más pequeña que en figuras anteriores. 201
Ilustración 129. Ejemplo de registro del dispositivo ThermalFusion Capture. 202
Ilustración 130. Registro de imágenes multiespectrales y térmicas con estructuras alternativas a la vegetación. 203
Ilustración 131. Diagrama de secuencia para la acción de registrar imagen multiespectral con RGB. 204
Ilustración 132. Diagrama de secuencia para incorporar a la aplicación una imagen térmica o RGB. 204
Ilustración 133. Diagrama de secuencia de la incorporación de las imágenes RGB y térmica a la aplicación, desencadenando su registro. 205
Ilustración 134. Secuencia de vistas necesarias para obtener el registro RGB-térmica y RGB-multiespectral. 207
Ilustración 135. Diagrama de clases para el módulo de la aplicación al completo. 208
Ilustración 136. Diagrama de clases parcial del módulo de aplicación. 208

Ilustración 137. Diagrama de clases de entidades relacionadas con la interfaz del módulo de aplicación. 210
Ilustración 138. Diagrama de clases de espacios de nombres y clases geométricas del módulo de aplicación. 211
Ilustración 139. Esquema de la clase encargada de la traducción de coordenadas de sistemas de referencia diferentes. 212
Ilustración 140. Diagrama de clases del módulo de procesamiento de imágenes al completo. 213
Ilustración 141. Diagrama de clases parcial de la vista (UI) del módulo de procesamiento de imágenes (I). 214
Ilustración 142. Jerarquía de componentes de la interfaz del módulo de procesamiento de imágenes. 214
Ilustración 143. Diagrama de clases parcial de la vista (UI) del módulo de procesamiento de imágenes (II). 216
Ilustración 144. Diagrama de clases parcial de la vista (UI) del módulo de procesamiento de imágenes (III). 217
Ilustración 145. Diagrama de clases del modelo y controlador de este módulo. 219
Ilustración 146. Diagrama de clases de utilidades del módulo de procesamiento de imagen (I). 220
Ilustración 147. Esquema de la clase que representa a una imagen. 222
Ilustración 148. Espacio de nombres que define operaciones con imágenes. 223
Ilustración 149. Diagrama de clases para la imagen y sus metadatos asociados. 224
Ilustración 150. Diagrama de clases del módulo de visualización 3D. 225
Ilustración 151. Diagrama de clases parcial del módulo de visualización 3D (I). 226
Ilustración 152. Diagrama de clases parcial del módulo de visualización 3D (II). 228
Ilustración 153. Diagrama de clases parcial del módulo de visualización 3D (III). 229
Ilustración 154. Diagrama de clases parcial del módulo de visualización 3D (IV). 231
Ilustración 155. Esquema de una clase que aplica el patrón Singleton. 232
Ilustración 156. Ejemplos de entidades asociadas a los objetos del esquema MVC. 234
Ilustración 157. Ejemplo de espacio de nombres que hace las veces de traductor. 235
Ilustración 158. Ejemplo de implementación del patrón Strategy. 236
Ilustración 159. Árboles representados en la imagen como si fueran una única masa de vegetación. 240
Ilustración 160. Representación gráfica de los coeficientes de correlación medios obtenidos para cada configuración. 251
Ilustración 161. Representación gráfica del tiempo de ejecución medio por configuración. 251
Ilustración 162. Imagen RGB más cercana en el tiempo a la multiespectral de la izquierda. 252
Ilustración 163. Representación gráfica del coeficiente de correlación medio para cada configuración propuesta. 255
Ilustración 164. Representación gráfica del tiempo de ejecución medio para cada configuración propuesta. 256
Ilustración 165. Imágenes propuestas como entrada del algoritmo de registro de imágenes RGB y térmicas. 257
Ilustración 166. Imagen térmica alineada (derecha) con el recorte de la imagen RGB (izquierda). 257

Ilustración 167. Resultado de combinar las dos imágenes anteriores.....257
Ilustración 168. Representación gráfica del coeficiente de correlación medio para cada configuración propuesta. 262
Ilustración 169. Representación gráfica del tiempo de ejecución medio para cada configuración propuesta. 262
Ilustración 170. Diagrama de clases del espacio de nombres que nos permite medir el tiempo de ejecución. 263
Ilustración 171. Componentes de Windows elegidos en la instalación de Microsoft Visual Studio.....271
Ilustración 172. Extensiones de Visual Studio, incluyendo la correspondiente a Qt. 272
Ilustración 173. Especificación de la versión de Qt y ruta a utilizar. 273
Ilustración 174. Diálogo para seleccionar el nivel de precisión de la información. 276
Ilustración 175. Visualización en 3D de los contornos extraídos.....278
Ilustración 176. Selección de distancia del dispositivo de captura al suelo, en milímetros. .279
Ilustración 177. Diálogo con los parámetros que podemos modificar para controlar el proceso de extracción de contornos.....280

Índice de tablas

Tabla 1. Planificación temporal del proyecto, donde para cada iteración se muestra un resumen del trabajo realizado. 49
Tabla 2. Coste económico de dispositivos necesarios para obtener la información de entrada de la aplicación....54
Tabla 3. Coste económico del hardware que forma el equipo de desarrollo. 55
Tabla 4. Coste económico del software empleado en este proyecto. 56
Tabla 5. Coste económico de materiales empleados en el proyecto. 56
Tabla 6. Coste económico derivado del personal que forma parte o hace posible el desarrollo del proyecto....58
Tabla 7. Coste económico de otros servicios derivados del proyecto. 58
Tabla 8. Resumen de costes hasta ahora justificados. 59
Tabla 9. Tabla de amortización de bienes expuestos en el coste del proyecto. 60
Tabla 10. Intervalo del espectro en el que cada imagen multiespectral opera (Parrot Sequoia, 2019). 77
Tabla 11. Metadatos necesarios para eliminar la distorsión fisheye. 103
Tabla 12. Metadatos de una imagen multiespectral utilizados durante la totalidad de este apartado. 153
Tabla 13. Coeficientes de correlación entre imágenes multiespectrales (Conjunto de datos 1, IMG_6) y su RGB asociada, todas en escala de grises. 188
Tabla 14. Nuevos metadatos empleados durante esta iteración. 203
Tabla 15. Configuraciones óptimas halladas para la extracción de contornos en imágenes multiespectrales. 242
Tabla 16. Correlación entre imágenes originales sin distorsión (no alineadas). 245
Tabla 17. Resultados obtenidos en el registro de imágenes multiespectrales para un nivel de precisión 1^{-10}, utilizando 30 iteraciones como máximo. 246
Tabla 18. Resultados del registro de imágenes al reducir nivel de precisión. 248
Tabla 19. Resultados obtenidos tras ejecutar el registro de imágenes para unos parámetros intermedios entre las dos pruebas anteriores. 249
Tabla 20. Resultados obtenidos al realizar el registro de imágenes multiespectrales reduciendo previamente el tamaño de las mismas. 250
Tabla 21. Resultados del registro de imágenes RGB y multiespectrales con la configuración inicial. 253
Tabla 22. Resultado del registro de imágenes multiespectrales y RGB al reducir iteraciones y nivel de precisión. 254
Tabla 23. Resultados del registro de imágenes multiespectrales y RGB al aumentar el tamaño de imagen. 255
Tabla 24. Resultados del registro de imágenes térmicas y RGB con la configuración inicial. 259
Tabla 25. Resultados de registro de imágenes RGB y térmicas, conservando el tamaño de estas últimas. 260
Tabla 26. Resultados de registro de imágenes RGB y térmicas, repitiendo la primera prueba con menos precisión e iteraciones....261
Tabla 27. Imágenes de las que depende el registro de cada multiespectral. 281

Tabla 28. Combinaciones de teclado para acceder a las operaciones de tratamiento de imágenes.....283
Tabla 29. Controles de la cámara y la escena 3D. 284

1 - DESCRIPCIÓN DEL TRABAJO

1.1 - Introducción

El objetivo de este trabajo de fin de grado es posibilitar el control exhaustivo de grandes plantaciones a partir de teledetección, lo que implica que se utiliza la información capturada por diversos sensores para extraer información de utilidad sobre la explotación objetivo.

La información de entrada de este proyecto procede de dispositivos de captura de imágenes multiespectrales, térmicas e imágenes de color tomadas a partir del vuelo de un dron sobre un terreno. A partir de todas estas imágenes, suponiendo un estado óptimo en el que todas se hallan sobre un mismo sistema de referencia, se podría extraer información de interés para un determinado punto del terreno a partir del valor de intensidad con el que se representa dicha posición en todas las imágenes de entrada. Si bien es cierto que hay información importante en la propia superficie del terreno, en este trabajo nos centramos en la vegetación, y más concretamente en los árboles presentes en el área capturada. Por ejemplo, se podría conocer el estado de salud de un árbol a partir de la información de las imágenes multiespectrales.

Existen procedimientos bien conocidos para el control de una plantación, como la obtención de índices, que es una técnica que se encuentra actualmente resuelta, por lo que no es ese el objetivo de este trabajo. Sin embargo, sí que sería de interés poder obtener diferentes tipos de información para un determinado punto que sabemos que se corresponde a vegetación, estableciéndose así un modelo multicapa.

Por lo tanto, se acaban de exponer los dos pilares de este proyecto: **distinción de la vegetación para conocer en qué áreas del terreno debemos centrar la atención**, y **asociación de múltiples tipos de información a la vegetación extraída** (datos térmicos, intensidad en una imagen multiespectral, posicionamiento en un sistema de coordenadas representado sobre la superficie terrestre, etc).

Los principales problemas que podemos encontrar no vienen sólo dados por el entrelazado de toda la información citada anteriormente, sino también por el camino que es necesario recorrer hasta que cada capa de información se encuentra en un

estado óptimo para llevar a cabo ese entrelazado. Para ello se debe tener en cuenta la Ilustración 1.

Ilustración 1. Cámara multiespectral Parrot Sequoia.

El dispositivo que aquí se presenta es la fuente de nuestros datos multiespectrales. En base a la disposición de las lentes y sus características físicas se pueden distinguir dos grandes problemas:

- **Distorsión de ojo de pez** derivada de una distancia focal pequeña que tiene como fin capturar un área de terreno mayor.
- **Problemas de alineación entre imágenes,** incluso cuando todas las imágenes se capturan en el mismo instante de tiempo, dado que el sistema de referencia de cada imagen es distinto (diferente posición para cada lente y diferentes ejes ópticos para cada una, derivados de la calibración del fabricante).

Para el caso de las imágenes térmicas el problema es igualmente complejo, dado que hay que registrar una imagen distorsionada (RGB), y una imagen térmica con mucha menos distorsión (de nuevo, como resultado de las distancias focales empleadas).

La justificación de este proyecto radica en la importancia creciente de la agricultura de precisión para la toma de decisiones sobre un cultivo y la falta de soluciones en el ámbito de la integración de múltiples tipos de información para una misma posición del terreno. Se proponen así algoritmos que ya resuelven ciertos productos software, pero se utilizan como un medio más para llegar a nuestro modelo multicapa, por lo que este proyecto de investigación requiere solucionar problemas bien conocidos, aunque se empleen algoritmos alternativos, y otros no tan habituales.

Por otro lado, el uso de un software externo para la resolución de un determinado problema implica en la mayoría de los casos que la obtención de datos a partir de la información capturada debe llevarse a cabo en el laboratorio, una vez ha

terminado el vuelo sobre el terreno. Sin embargo, si se dispone de software adaptado y propio, todo esto se puede integrar en un dispositivo de capacidades restringidas para realizar los cálculos necesarios a medida que se captura la información durante el vuelo. No es este último el objetivo de este proyecto, pero sí que podría ser un trabajo futuro que se planteara a partir de la base que aquí se desarrolla.

1.2 - Antecedentes y estado del arte

En esta sección se analiza el estado del entorno que rodea a este trabajo, de tal forma que nos servirá para comprender la situación de la que se parte, no sólo a nivel de área de trabajo, como es la agricultura de precisión, sino también en cuanto a otras soluciones propuestas para resolver los problemas que se han encontrado.

Se utiliza este apartado como una extensión de la introducción para comentar la problemática, de forma más detallada, asociada al área de trabajo.

1.2.1 - Antecedentes

Las necesidades de la agricultura han ido variando a lo largo del tiempo, siempre partiendo de que el objetivo principal es obtener el máximo rendimiento posible, aumentando así la producción. Si bien la preocupación original de esta situación era la seguridad alimentaria, que se puede suponer ya resuelta en la mayor parte de Europa, otro efecto del objetivo citado son las consecuencias negativas sobre el medio ambiente debido a factores como la contaminación del agua, emisiones de gases de efecto invernadero, etc (Tscharntke, 2012). A este problema se le debe sumar la necesidad de aumentar la producción debido al crecimiento de la población, por lo que el sector agrícola debe encontrar la solución para satisfacer el objetivo de aumentar la producción sin causar los efectos negativos hasta ahora generados en el medio ambiente.

Una solución factible es el uso de la tecnología, por lo que a grandes rasgos podemos hablar ya de una **agricultura de precisión**. En 1997, la Cámara de Representantes de Estados Unidos daba una definición para este término muy similar a lo que acabamos de ver: sistema de información integrada y basado en la producción agrícola para aumentar a largo plazo la eficiencia, productividad y rentabilidad de la producción de un cultivo minimizando el impacto en el entorno (Joint Research Centre

(JRC) of the European Commission, 2014). Otra descripción más reciente sería aplicar el tratamiento correcto en el lugar correcto y en el momento exacto (Adamchuk, 2010).

Se puede extraer por lo tanto de estas definiciones que la agricultura de precisión pretende el uso de la tecnología para aumentar el rendimiento de un cultivo y la rentabilidad de las estrategias habituales de uso de fertilizantes, pesticidas y otros productos necesarios (Capri, 2018). Se comprueba además que la degradación ambiental decae con el uso de la agricultura de precisión debido a la reducción de emisiones de ciertas sustancias nocivas (Joint Research Centre (JRC) of the European Commission, 2014).

Ahora bien, para este trabajo, una vez conocido el propósito y la terminología, podemos indagar sobre la tecnología que se ha venido usando en esta área a lo largo del tiempo para poder observar el estado de un cultivo y responder en base a las medidas y conocimientos adquiridos.

En la actualidad, el área más destacable dentro de la tecnología que posibilita la agricultura de precisión es la teledetección, aunque su avance ha sido muy escaso en este entorno durante décadas. El principal cambio que se ha producido para que la teledetección tome gran relevancia son los desarrollos tecnológicos relativamente recientes, como podría ser la obtención de imágenes satélite de alta resolución o el auge de vehículos aéreos no tripulados, como el empleado en este trabajo (Ilustración 2), que a menudo se conocen como drones o más formalmente SARP (Sistemas Aéreos Remotamente Pilotados). Normalmente, la tecnología por satélite se considera más cara y menos precisa, (2, 10) metros, mientras que mediante el uso de SARP se puede llegar a alcanzar una precisión de (2, 10) centímetros.

Ilustración 2. SARP DJI Matrice 210 empleado por el departamento para obtener la información de entrada de este trabajo. Fuente: (Grupo Acre, 2019).

La implementación de la agricultura de precisión procede principalmente del desarrollo de diferentes sensores junto a procedimientos que permiten relacionar los datos obtenidos con ciertas operaciones de gestión del cultivo: uso de herbicidas, fertilizantes, siembra, etc. Muchos de los sensores que se han venido empleando se basan en el principio de la teledetección, de tal forma que se captura información espacial de resolución variable a través de múltiples longitudes de onda mediante cámaras multiespectrales e hiperespectrales a bordo del dispositivo remoto (ya sea un SARP o satélite). Un ejemplo de información multiespectral se muestra en la Ilustración 3.

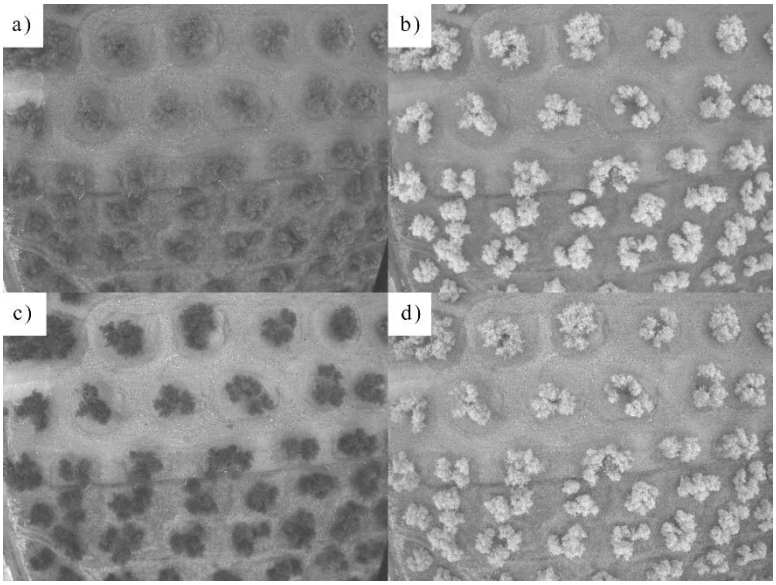


Ilustración 3. Imágenes multiespectrales utilizadas durante el desarrollo del proyecto. Se muestran aquí cuatro capas distintas que representan cuatro intervalos diferentes del espectro. a) GRE b) NIR c) RED d) REG.

A partir de la información multiespectral se puede analizar las propiedades del suelo, el estado de salud de las plantas o el rendimiento del cultivo (Friedl M. , 2018). Hay que tener en cuenta a partir de ahora que ciertos componentes de los que depende precisamente ese rendimiento se podrían medir con este tipo de imágenes, teniendo en cuenta que cada material tiene un rango del espectro donde su reflectancia es mayor, y por lo tanto, se puede detectar o medir con mayor facilidad. Normalmente se aprovecha dicha funcionalidad en combinación con otras técnicas, como la mezcla de intervalos del espectro para la obtención de índices (Huete, 1990). Uno de los problemas que nos encontramos aún en este aspecto es la limitación de la resolución y de las bandas espectrales que se pueden capturar.

Un último aspecto a tener en cuenta en este apartado son los Sistemas de Información Geográficos (SIG), que posibilitan la localización de datos georreferenciados, procedentes de diferentes sensores, sobre un mapa, de tal forma que se puede visualizar el estado de un terreno completo. El fin último será permitir que el agricultor pueda acceder a todos estos datos mediante algún sistema de gestión del cultivo, ya sea a través de productos software, consultas a servicios externos, etc.

1.2.2 - Estado del arte

En esta sección se pretende dar a conocer el estado en el que se encuentra la investigación de los diferentes aspectos a tratar en este trabajo. Dado que son varios los frentes a tener en cuenta, se propone el estado de cada uno de forma individual. En este apartado no se realiza una descripción extensa de los problemas, sino que estos serán expuestos con mayor detalle en el proceso de desarrollo.

El primero de los frentes a tener en cuenta, aunque no es este necesariamente el orden de implementación, corresponde a la eliminación de la **distorsión de ojo de pez o *fisheye*** de las imágenes. Como se ha comentado previamente, esta característica surge al utilizar distancias focales pequeñas que nos permiten capturar áreas de terreno mayores. Por lo tanto, se puede suponer que se trata de un problema frecuente en el ámbito de las imágenes multiespectrales. El problema se puede observar claramente en la Ilustración 4.

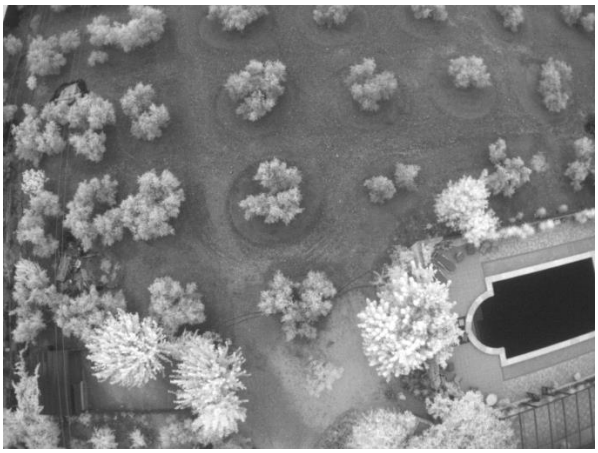


Ilustración 4. Imagen multiespectral en su formato original, con distorsión fisheye.

No obstante, esta problemática no se reduce a las cámaras multiespectrales, sino que ha sido un tema ampliamente tratado. Se pueden distinguir hasta dos tipos

de imágenes con distorsión ojo de pez: circulares o de imagen completa (*fullframe*) (Panotools, 2019), donde nuestros datos se ajustan al segundo modelo.

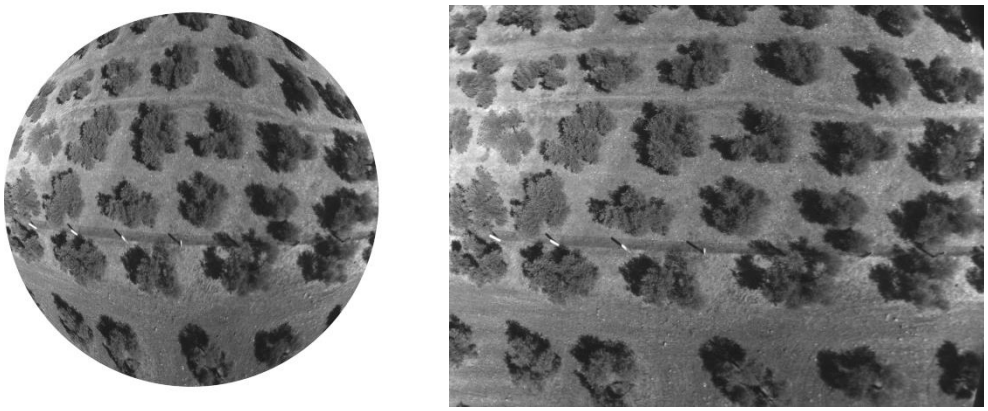


Ilustración 5. Comparativa entre el modelo de distorsión fisheye radial y fullframe.

Algunas soluciones a la eliminación de la distorsión vienen dadas por los parámetros físicos de la cámara junto a heurísticas para determinar constantes que intervienen en el proceso (Dhane, 2012), o bien por operaciones de minimización de la distorsión resultante (Wei, 2011). Sin embargo, el software Pix4D, orientado a trabajar con imágenes de drones y que incluye a Parrot Sequoia como uno de sus productos, nos provee de su modelo de distorsión fisheye (Pix4D, 2018), el cual se limita al uso de los propios parámetros físicos de la cámara, como serán la distancia focal, los coeficientes de distorsión o los coeficientes de la matriz afín. El principal problema al trabajar con algunos de los parámetros que se encuentran en los metadatos es transformarlos al píxel como unidad, algo que se muestra durante el desarrollo del proyecto.

Incluso hay técnicas, aplicadas exclusivamente a las imágenes multiespectrales, que consideran la eliminación de la distorsión *fisheye* y la **falta de alineación entre imágenes** (que será nuestra siguiente parada) en un solo proceso (J. P. Jhan, 2017), con varias desventajas a tener en cuenta:

- Se basan en parámetros obtenidos de los datos de la imagen, que en muchos casos dependen exclusivamente de la calibración de la cámara.
- El uso de datos derivados del sistema de calibración puede inducir a un registro inexacto debido a los errores que pueden existir en dicho sistema. Para solucionar este problema se emplean algoritmos de estimación de modelos matemáticos, como RANSAC (ver Definiciones y abreviaturas).

Esta metología presenta la ventaja de obtener todos los datos necesarios para la transformación de una forma eficiente (un simple acceso a los metadatos), teniendo en cuenta que la otra técnica más habitual es la obtención de la correspondencia entre imágenes, también conocida como *Image Matching* (Xiaoyong Shen, 2014) (véase la Ilustración 6), que puede causar también fallos si se escoge un par de imágenes cuyo valor de intensidad difiere en gran medida entre ellas.

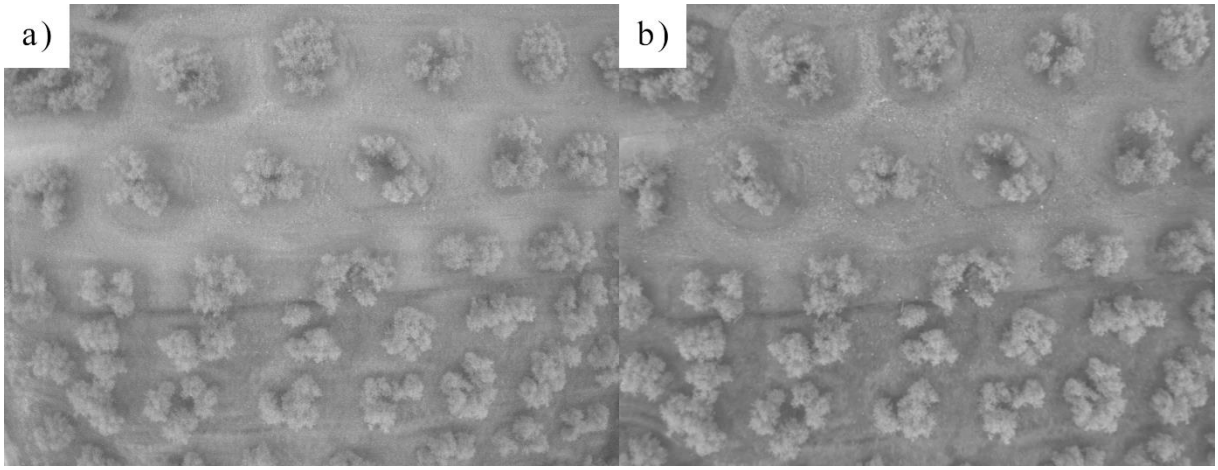


Ilustración 6. Imágenes multiespectrales superpuestas sin alinear (efecto de image ghosting) en la izquierda, y a la derecha, las mismas imágenes multiespectrales alineadas y superpuestas.

Sin embargo, esto último puede no representar un problema si se busca la correspondencia entre aquellos pares de imágenes de intensidad más afín (Yasir R., 2018). No sólo se debe considerar una entrada adaptada, como se acaba de comentar, sino también un algoritmo de obtención de la correspondencia de imágenes capaz de soportar este tipo de distorsiones fotométricas (Georgios D. Evangelidis, 2008). El algoritmo que se acaba de citar busca la transformación global que mejor ajuste las dos imágenes de entrada a través de un coeficiente de correlación. Este algoritmo plantea una serie de ventajas:

- Resuelve un problema muy complejo de forma lineal.
- Es capaz de trabajar a nivel de subpíxel, buscando un ajuste del tamaño que se especifique como entrada del algoritmo. Esto nos permite reducir las dimensiones de las imágenes para trabajar más eficientemente, teniendo en cuenta que ciertas imágenes de este proyecto, como la RGB, tienen una resolución muy grande.

- No busca la correspondencia de cada píxel, sino que transforma la imagen completa y comprueba la calidad del ajuste alcanzado. Así hasta llegar a un número de iteraciones máximo, que también se podrá proveer como entrada del algoritmo.

Llegados a este punto se supone que todas nuestras imágenes representan exactamente el mismo área de terreno, por lo que podemos pasar a la **extracción de los árboles en la imagen**.

A diferencia de nuestro objetivo, que es distinguir claramente cada árbol de un cultivo, obtener su información en varias imágenes y hacer un seguimiento del mismo a lo largo del tiempo, el objetivo más común es distinguir áreas de vegetación y analizar índices derivados de la composición de imágenes multiespectrales a partir de operaciones como la diferencia o la división. En el mejor de los casos, estas áreas se delimitan muy bien para contener sólo una parte del cultivo o el terreno en su totalidad (Sebastian Candiago, 2015); (Esther Salamí, 2014); (Nebiker, 2016); y a partir de ahí estudiar el estado o ciertas propiedades del suelo y la vegetación mediante índices.

Existe una alternativa a Parrot Sequoia en el mercado de las cámaras multiespectrales, MicaSense, que posee su propia librería en Python para el tratamiento de las imágenes capturadas y resuelve, entre otros problemas, la alineación de imágenes para su dispositivo. En la documentación de dicha librería se exponen algunos ejemplos de procesamiento de estas imágenes, entre los que se encuentra la extracción de la vegetación de una imagen (MicaSense, 2018). Aunque obtiene un resultado lejos de lo que se busca en este trabajo, como se muestra en la Ilustración 7, supone una aproximación al algoritmo que se desarrolla más tarde y, de hecho, ya utiliza las dos capas que nosotros usaremos, RED y NIR.

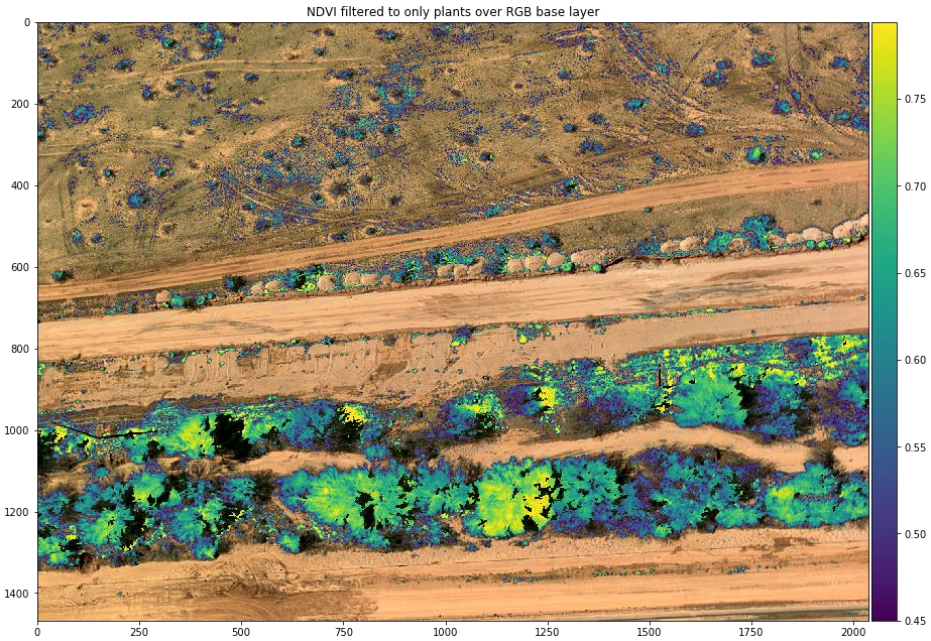


Ilustración 7. Distinción de vegetación respecto de su sombra y el suelo, implementado por Micasense (MicaSense, 2018).

A la vez que se distingue la vegetación del resto del entorno se muestra un determinado índice derivado de la mezcla de imágenes multiespectrales.

Si queremos controlar un único árbol, extraído a partir de la mezcla de capas, habrá que identificarlo de alguna manera. Se puede considerar para cada polígono (árbol) su centro de masas, de tal manera que se puede identificar de forma única a cada individuo mediante una posición en un sistema de coordenadas, como podría ser UTM. Para ello necesitamos principalmente parámetros como la altura de vuelo del dron, los ángulos de rotación de la cámara y la localización geográfica de un punto de la imagen (Hemerly, 2014). Considerando una única captura, no hay muchas más vías para resolver este problema. Nótese como hasta ahora no se ha eliminado la distorsión de perspectiva en la imagen, por lo que para obtener una mayor precisión debería considerarse una vista ortográfica. Sin embargo, para nuestros propósitos de investigación no se implementa esto último, entre otras razones porque los parámetros de la cámara multiespectral para la georreferenciación son lo suficientemente imprecisos tanto en ángulos como en posicionamiento como para no invertir más esfuerzo en esta vía de trabajo.

Una vez fuéramos capaces de identificar la localización geográfica de las cuatro esquinas de la imagen podríamos intentar generar un **ortomosaico**. Para tal fin se necesitan un conjunto de imágenes de entrada que se solapen entre sí. A medida que

aumenta el solape entre imágenes se demuestra que se obtiene más precisión (Mesas-Carrascosa, Generating UAV accurate ortho-mosaicked images using a six-band multispectral camera arrangement, 2014); (Mesas-Carrascosa, Accurate ortho-mosaicked six-band multispectral UAV images as affected by mission planning for precision agriculture proposes, 2017). Si bien, la mayoría de artículos encontrados en cuanto a la generación del ortomosaico tratan de hallar las condiciones y parámetros óptimos para la generación del mismo, pero acuden a software de terceros (Pix4D, Agisoft, etc) para obtener el resultado (Berg-Jürgens, 2015). En los primeros artículos citados se trabaja además con condiciones óptimas, suponiendo que la altura se mantiene estable a lo largo del vuelo, algo que para nuestra información de entrada no se puede suponer. Es necesario por lo tanto formular un algoritmo que no puede ser catalogado como generación de ortomosaico sino como un cosido o pegado de imágenes (*image stitching*), entre las cuales pueden existir diferencias notables, obteniéndose así un resultado que no sería más que un ortomosaico con errores (Pix4D, 2019), como muestra la Ilustración 8.

Ilustración 8. Cosido de imágenes (izquierda) y ortomosaico (derecha) (Pix4D, 2019).

Por lo tanto, el enfoque con el que se realiza el ortomosaico, o más bien el cosido de imágenes, será basándose íntegramente en la información que se ha empleado para asignar una posición geográfica a un punto de la imagen. Dicho de otra forma, nos basaremos en la precisión del sistema de posicionamiento para conocer el punto exacto donde colocar y solapar las imágenes.

La última sección del estado del arte de este trabajo se centra en el **registro de imágenes** tan diferentes como una imagen **de color y una imagen multiespectral**

o una imagen de color y una imagen térmica (ver Ilustración 10). Para el segundo de los registros (RGB-térmica) no se pueden citar estudios que nos ayuden a desarrollar el algoritmo que se implementa, dado que encontramos principalmente investigaciones para solucionar este problema con una determinada disposición física, que por supuesto, no se ajusta a nuestra escena. Por ejemplo, (Palmero, 2016) propone una escena donde la imagen térmica se traslada verticalmente respecto de la RGB, y donde prácticamente el ángulo de visión para ambos dispositivos es el mismo. Sin embargo, con sólo observar nuestro dispositivo de captura DJI XT2 (Ilustración 9) y la Ilustración 10, queda claro que no este no es nuestro caso.

Ilustración 9. Dispositivo de captura de imagen RGB y térmica, DJI Zenmuse XT2. Fuente: (Grupo Acre, 2019).

Aunque el proceso como tal será es complejo, el registro que subyace en el algoritmo que se implemente utiliza el mismo método de maximización de la correlación entre dos imágenes que se citó en el registro de imágenes multiespectrales (Georgios D. Evangelidis, 2008), y algo similar ocurre para el registro RGB-multiespectral.

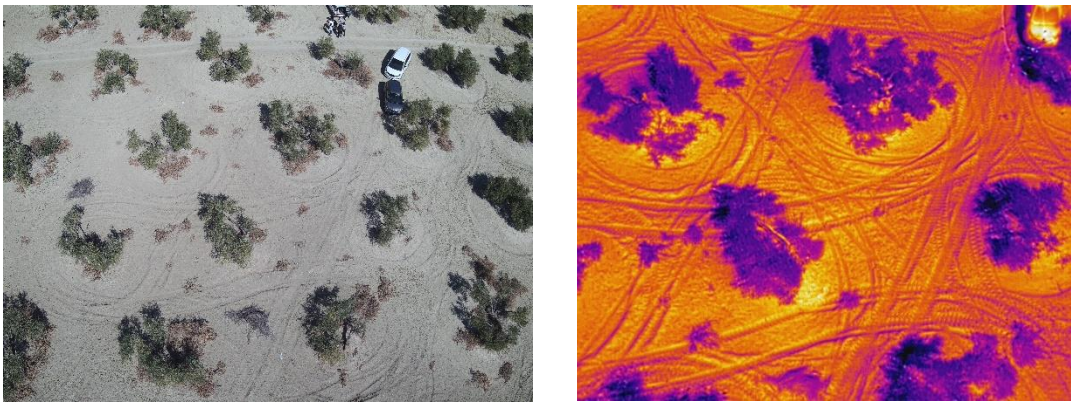


Ilustración 10. Imagen RGB (izquierda) e imagen térmica (derecha).

Para este último registro la situación en cuanto a investigación es idéntica al caso anterior, ya que normalmente se considera el registro de imágenes que capturan escenas muy similares, algo que no se ajusta a nuestra situación. De hecho, para este caso la problemática es mucho mayor que para la imagen térmica, ya que esta última y RGB se capturan en el mismo instante, algo que no sucede para el par RGB-multiespectral, dado que se capturan con dispositivos completamente diferentes. Si bien la cámara multiespectral obtiene también una imagen RGB, esta no es de utilidad para nuestra aplicación dado que posee los mismos problemas que cualquiera de las capas multiespectrales: pobre posicionamiento GPS, ángulos de rotación (*gimbal*) a priori incorrectos, etc.

Se pueda comprobar así que la investigación en el área del registro de imágenes RGB y multiespectrales gira en torno a imágenes cuyas condiciones de captura son muy similares, por lo que el problema se reduce a buscar la correspondencia de ciertos puntos característicos entre alguna imagen multiespectral ((Süsstrunk, 2011) emplea la imagen NIR) y la imagen RGB (Yasir R., 2018).

Dado que son varios los problemas que se deben atajar, primero se puede plantear una búsqueda de imagen RGB más cercana en el tiempo de captura a una multiespectral. Una vez encontradas esas dos imágenes, se puede llevar a cabo un proceso de registro entre una imagen multiespectral (cualquiera de las cuatro disponibles, aunque la GRE es la más apropiada en escala de grises) y una imagen RGB.

1.3 - Definiciones y abreviaturas

En este apartado se presenta y describe aquella terminología específica y abreviaturas que se deben conocer para la correcta comprensión de esta documentación, aunque cuando lleguemos al punto en el que se mencionen estas expresiones se hará una presentación más amplia del término en cuestión o bien se podrá deducir su significado a partir de la temática tratada. Se parte de que el lector de este documento desconoce la tecnología más básica de nuestra área, por lo que cualquier término puede ser objetivo de este apartado.

1.3.1 - Abreviaturas

API	Interfaz de programación de aplicaciones. Supone una capa de abstracción para que un software pueda acceder a ciertas funciones, procedimientos, etc.
ECC	Enhanced Correlation Coefficient. Algoritmo de registro de imágenes (Georgios D. Evangelidis, 2008).
EXIF	Exchangeable Image File. Estándar para formatos de imagen de cámaras digitales (JPEG, TIFF, etc), a los que les añade etiquetas de metadatos. Un etiqueta podría ser, por ejemplo, la fecha y hora de captura de la imagen.
IDE	Integrated Development Environment. Aplicación que facilita al desarrollador la creación de software.
GLM	OpenGL Maths. Librería multiplataforma para la realización de cálculos complejos con vectores, matrices, quaterniones, etc.
GPL	General Public License. Licencia de software libre y código abierto.
GRE	Green. Imagen que captura longitudes de onda en el intervalo [510, 590] nanómetros (Athos Agapiou, 2012) (Parrot Sequoia, 2019).
GSD	Ground Sample Distance. Distancia, medida en el suelo, entre píxeles de una imagen digital.
HSV	Espacio de color, también conocido como HSB. H responde a Hue, matiz, S a Saturation y V a valor.
IBO	Index Buffer Object. Objeto OpenGL que define la topología de una malla de triángulos.
JSON	JavaScript Object Notation. Es un formato de archivo estándar que utiliza texto legible por cualquier lector para transmitir información de objetos compuestos de pares clave-valor y vectores de tipos de datos.
LAB	Puede referirse a dos espacios de color, aunque en este proyecto siempre se hará referencia a CIELAB. La L responde a luminosidad, B al gradiente de azul y A el componente verde-rojo.
MVC	Model-View-Controller. Patrón de diseño orientado a la creación de interfaces de usuario.
NIR	Near Infrared. Imagen que captura longitudes de onda en el intervalo [750, 830] nanómetros (Athos Agapiou, 2012) (Parrot Sequoia, 2019).

OpenCV	Open source Computer Vision. Librería de visión por ordenador y aprendizaje automático.
OpenGL	Open source Graphics Library. Interfaz de programación para la creación de gráficos 2D y 3D.
RANSAC, RAC	Random Sample Consensus o RANSAC. Es un método iterativo para la estimación de parámetros de un modelo matemático que presenta valores atípicos, los cuales no son de interés. Se puede considerar por lo tanto como un método de detección de anomalías.
RED	Red. Imagen que captura longitudes de onda en el intervalo [620, 700] nanómetros (Athos Agapiou, 2012) (Parrot Sequoia, 2019).
REG	RedEdge. Imagen que captura longitudes de onda en el intervalo [725, 745] nanómetros (Athos Agapiou, 2012) (Parrot Sequoia, 2019).
SARP	Sistemas Aéreos Remotamente Pilotados, más comúnmente conocidos como drones.
SIG	Sistema de Información Geográfico que permite almacenar, organizar, analizar y controlar datos espaciales o geográficos.
SIGPAC	Sistema de Información Geográfica de Parcelas Agrícolas.
SQL	Structured Query Language. Es un lenguaje de consulta de bases de datos relacionales.
UTM	Universal Transverse Mercator. Sistema de coordenadas basado en la proyección cartográfica de Mercator.
VAO	Vertex Array Object. Objeto OpenGL que contiene uno o más VBOs y uno o más IBOs.
VBO	Vertex Buffer Object. Objeto OpenGL que contiene la geometría de una malla de triángulos.
XML	Extensible Markup Language. Lenguaje de marcas que codifica la información en un formato legible tanto para un humano como para un ordenador. Utiliza una estructura similar a la de HTML.

1.3.2 - Terminología

Fisheye distortion	Distorsión visual de la imagen originada por lentes súper gran angular (ángulos de visión muy grandes y distancias focales pequeñas).

Image matching	Búsqueda de puntos o características de una imagen (*source*) en otra que será el objetivo (*target*), de tal forma que se puedan representar ambas sobre un mismo sistema de referencia.
Imagen multiespectral	Imagen que captura luz de un determinado rango de longitudes de onda del espectro electromagnético, incluidas aquellas no visibles por el ojo humano. Al incluir sólo cierta parte del espectro habrá fenómenos y estructuras que se puedan analizar con mayor facilidad. Por ejemplo, la imagen NIR se suele utilizar para analizar la vegetación.
Georreferenciación	Posicionamiento de una entidad sobre una localización geográfica única en un sistema de coordenadas y un *datum*. De esta forma, seremos capaces de trasladar el sistema de coordenadas de una imagen a otro sistema de coordenadas como UTM.
Image ghosting	Combinación de imágenes entre las cuales se produce un movimiento de traslación. En este proyecto se genera este efecto al demostrar la falta de alineación entre imágenes multiespectrales.
Mapa de disparidad	Imagen en escala de grises que muestra la profundidad de los objetos de una escena respecto de una cámara, para lo cual será necesario disponer de dos imágenes de entrada, donde ambas deben representar una parte común de la escena. A partir de la correspondencia entre imagen izquierda y derecha y las características físicas de la cámara, se puede establecer la disparidad de la escena común a ambas imágenes.
nadir	En términos de sensores remotos se refiere a la orientación perpendicular a la superficie terrestre.
Ortomosaico	Composición de imágenes sin distorsiones geométricas y con una intensidad de color similar, de tal forma que se reconstruye un área de terreno adheriendo todas las imágenes, para lo cual se deben considerar factores como la localización de cada una en un sistema de coordenadas o puntos característicos entre imágenes.
Polilínea	Secuencia de líneas conectadas que conforman un único objeto.
Registro de imágenes	Eliminación de diferencias geométricas para que un mismo punto o píxel de varias imágenes se correspondan con el mismo espacio del mundo, permitiéndonos así comparar las imágenes con mayor facilidad (Espejo, 2008). Un caso en el que esta operación sería necesaria

	es la obtención de imágenes de un terreno capturadas en instantes de tiempo diferentes, haciendo así que no compartan un mismo sistema de referencia.
Ruido de imagen	Variaciones de brillo o de color en la imagen producidas por el dispositivo de entrada, que comúnmente será una cámara digital. El ruido se presenta con un aspecto de granularidad sobre la propia imagen, y dicho defecto no será tolerable para muchos de los algoritmos de procesamiento de la información visual que aquí se traten.
Teledetección	De acuerdo al Instituto Geográfico Nacional, la teledetección es la técnica de adquisición de datos de la superficie terrestre desde sensores instalados en plataformas espaciales. Si se procesa la información captada por los sensores, se podrá obtener información interpretable de la Tierra.

1.4 - Objetivos del trabajo

A partir de toda la información expuesta en la introducción de este documento, se puede extraer una lista exhaustiva de los objetivos que se pretende obtener:

- Dada un área de terreno, un objetivo principal sería la **segmentación y obtención de aquellas zonas pertenecientes exclusivamente a vegetación de interés**. Estas zonas no serán más que polígonos aislados que aproximan, con mayor o menor grado de acierto, el contorno de aquellos elementos de estudio. Se puede extraer de esta última reflexión que otro objetivo derivado de la extracción de polígonos puede ser la **simplificación de los mismos para reducir la geometría asociada**, y por lo tanto, el coste de las operaciones derivadas.

- El dispositivo de captura de imágenes multiespectrales no proporciona sólo una capa de información sino 4 (en realidad 5 si añadimos la RGB, que tendrá poco interés para nuestra investigación). Un problema común en este ámbito es el **registro de imágenes**, dado que estas pueden no tomarse en un mismo instante de tiempo o se capturan con cámaras cuyo sistema de referencia es diferente del resto (que puede corresponder simplemente a un desplazamiento en alguna dirección). De hecho, el registro de imágenes será un tema recurrente que representa uno de los objetivos principales de

este proyecto. Nótese como se puede realizar el registro entre cualquier par de entradas de las que disponemos: RGB – térmica, RGB – multi, multi (canal A) – multi (canal B)…

- Otro objetivo principal, hasta ahora en segundo plano, sería la **georreferenciación** de las imágenes. Tiene un gran valor el posicionamiento de nuestras imágenes en un sistema de coordenadas sobre la superficie terrestre, dado que nos puede ayudar a componer una vista aérea de la plantación que está siendo objeto de estudio. Son muchas las fases que se deben atravesar desde el registro de imágenes hasta la obtención de un **ortomosaico**, por lo que para seguir explorando este objetivo nos referiremos directamente a la documentación del proceso de desarrollo.
- **Almacenamiento de información de un terreno**. Se trata de un objetivo secundario dado que nuestro proyecto no consideraba en primera instancia la inclusión de esta herramienta. Sin embargo, puede ser de interés proponer un modelo de base de datos para guardar la información asociada a un terreno, teniendo en cuenta que se podría volar varias veces sobre el mismo a lo largo del tiempo, permitiéndonos así analizar su evolución. Hablamos entonces de una base de datos 4D, aunque en realidad trabajamos con localizaciones geográficas, y por lo tanto, se reduce a 3D, donde una de las dimensiones debe ser necesariamente el tiempo.
- **Visualización de resultados en 3D** para facilitar su comprensión. Se trata también de un objetivo secundario que se elabora teniendo en cuenta que el fin último de hallar información de un cultivo es que el agricultor pueda remediar los problemas encontrados. Una visualización algo más flexible se puede plantear para tal fin, de tal forma que se ha incluido un módulo de representación de datos 3D.
- Desde un punto de vista genérico, el objetivo último de este trabajo es conseguir una aplicación que nos posibilite poner en marcha los algoritmos que se propondrán más tarde para solucionar todos y cada uno de los problemas planteados al comienzo de este documento. Además, el objetivo de esta aplicación no es el de servir como un producto final que pudiera

comercializarse, sino **crear una solución que sirva de ayuda al investigador para comprobar la corrección de los algoritmos desarrollados**.

1.5 - Alcance

Una vez expuestos los objetivos de este trabajo, es necesario definir claramente qué se incluye dentro del mismo, y para ello se definirán todos los entregables y resultados esperados, de tal forma que nos podemos basar en la sección de Documentos entregables (ver Guía original del Trabajo Fin de Título) de la propuesta de trabajo fin de grado que hace referencia a este proyecto.

Así, los entregables y resultados del trabajo deben ser necesariamente los siguientes:

- **Código fuente** del software desarrollado, donde se debe destacar la implementación de los algoritmos propuestos a lo largo de este trabajo. Con el fin de aportar al investigador o usuario de la aplicación una visualización cómoda de los resultados de los algoritmos, se desarrolla una interfaz para soportar estos procesos. Por lo tanto, el código fuente no sólo incluye las soluciones a los problemas que se plantean sino también una aplicación de interfaz completa para visualizar la salida de los algoritmos.
- **Ejecutable del prototipo** desarrollado: ejecutable de la aplicación principal cuyas utilidades deben satisfacer los objetivos anteriormente expuestos. Podemos aprovechar por lo tanto esta descripción para mostrar, de forma muy breve, qué herramientas debe incorporar el ejecutable de la aplicación:
    - Debe trabajar con imágenes multiespectrales, RGB y térmicas, permitiendo así su visualización en la aplicación.
    - Extracción de árboles de interés en una imagen.
    - Adaptación y corrección de imágenes multiespectrales: eliminación de distorsiones de captura y registro de imágenes para situar todas sobre un mismo sistema de referencia.

  - Georreferenciación de puntos de una imagen multiespectral, extrayendo así para cada uno su posición sobre un sistema de coordenadas de la Tierra y la información multiespectral asociada.
  - Registro de la imagen RGB con una imagen térmica.
  - Registro de imagen RGB con multiespectral.
  - Visualización de contornos 3D para facilitar la comprensión de la escena obtenida.
  - Generación de un ortomosaico con imágenes de entrada. En esta ocasión se trabajará con imágenes multiespectrales.
  - Definición del modelo de una base de datos que pueda servir para guardar la información generada.
- **Memoria del trabajo desarrollado**, que será este documento, donde se incluyen los correspondientes manuales de uso y de instalación del proyecto, además de la justificación y documentación de la soluciones propuestas.
- **Otros documentos adicionales**, entre los que se incluye el código fuente para la creación de la base de datos o los *scripts* desarrollados para la experimentación.
- **Vídeo de demostración de funcionamiento de la aplicación**.
- **Información de entrada**, que incluye todas las imágenes multiespectrales, RGB y térmicas utilizadas para comprobar la corrección de los algoritmos desarrollados.

1.6 - Metología de desarrollo

Dado que se trata de un proyecto de investigación dentro del departamento de informática, se puede diferenciar claramente una parte interesada (que serían los miembros del departamento involucrados en este trabajo), y un equipo de desarrollo, en el que se integra el único autor de este trabajo. El hecho de que este se ajuste a una situación más real nos permite llevar a cabo, por ejemplo, reuniones cada cierto tiempo.

En estas reuniones se puede comprobar el progreso del proyecto, la usabilidad del sistema, aportar nuevas ideas o simplemente tratar en mayor profundidad los siguientes aspectos a desarrollar.

Por lo tanto, el proyecto se adapta muy bien a una metodología iterativa como podría ser **SCRUM**. Esta metodología se integra dentro de lo que se conocen como **metodologías ágiles**, que comienzan a surgir en el siglo XXI como respuesta ante ciertos problemas que sufrían algunas metodologías tradicionales (Scrum Methodology, 2017); (cprime, 2018):

- Falta de capacidad ante cambios. En nuestro caso, las iteraciones y las revisiones sobre el progreso del proyecto en un momento dado nos permiten fácilmente orientar el trabajo en una determinada dirección.
- Generación de un excesivo volumen de documentación.

Los principios con los que surgen este tipo de desarrollos son los siguientes:

- Mayor prioridad de individuos e interacciones sobre procesos y herramientas.
- Mayor prioridad de un software operativo (que funcione correctamente) frente a una extensa documentación.
- Mayor prioridad a la colaboración del cliente que a la negociación del contrato.
- Mayor prioridad de la respuesta ante cambios frente al seguimiento de un plan estricto.

Una de las principales metodologías ágiles que existen hoy en día es SCRUM, una metodología ágil que define un conjunto de prácticas y roles que serán de utilidad para desarrollar nuestro proyecto.

¿Qué roles considera SCRUM dentro del proyecto? (ver Ilustración 11)

- **Scrum Master**: es el encargado de facilitar la aplicación de la metodología SCRUM, gestionar cambios en el proyecto y organizar reuniones críticas. Debe tener un conocimiento avanzado de SCRUM y estar al tanto del estado del proyecto, de esta forma puede ayudar al equipo a superar los problemas que vayan surgiendo durante el desarrollo. No se puede considerar como un líder debido a que uno de los principios de esta metodología es que el equipo se auto-organiza.

- **Product owner**: es la fuente de los requisitos del proyecto (la única fuente de la verdad). Es el encargado de definir las características del producto, informar de posibles errores, etc, y como ya comentábamos, trabaja muy cerca del equipo (tiene una gran posibilidad de interacción con el equipo de desarrollo).

  Será el encargado de mantener lo que se conoce como *Product backlog*, que define, a alto nivel, las características del proyecto (funcionalidad, requisitos, estimaciones, etc). Es el único que puede modificar dicho documento.
- **Equipo de desarrollo**: está formado por todos aquellos invidividuos que desarrollan el producto. Se auto-organizan: división de tareas, asignación de las mismas, etc. Se considera que deben ser equipos pequeños (3-9 personas).

Ilustración 11. Esquema de los roles que define la metodología ágil SCRUM.

Nótese como estos roles que acabamos de citar hacen más bien referencia a un equipo de una mínima entidad, sin embargo, en este trabajo sólo ha intervenido un desarrollador, que hace las funciones de los roles de Scrum Master y equipo de desarrollo, y un conjunto de personas del departamento involucradas en el proyecto, que podrían tomar el papel de Product Owner debido a que son los que orientan y determinan a grandes rasgos el trabajo a realizar en cada iteración.

Una vez definidos los roles, debemos desarrollar el flujo de trabajo que sigue esta metología:

- Las iteraciones que anteriormente comentábamos se denominan ***sprints***, que no serán más que períodos de tiempo en los que se lleva a cabo un trabajo. El tamaño de los *sprints* puede ser variable aunque es

recomendable que sean constantes. Por ejemplo, una opción es comenzar el proyecto con un *sprint* de una cantidad de tiempo concreta e ir ajustándolo en base a las observaciones y resultados de *sprints* anteriores.

En nuestro caso, los objetivos que se marcan en cada *sprint* difieren mucho, por lo que se ha optado por un tamaño de iteración variable que se puede adaptar a los objetivos marcados. Obviamente, no lleva el mismo tiempo el estudio inicial de las librerías que se van a utilizar en el proyecto que el desarrollo de una gran cantidad de utilidades incluidas en la aplicación.

- Al final de un *sprint* se deben presentar los avances logrados así como el resultado obtenido. En principio, el producto generado en un *sprint* podría constituir ya una entrega al cliente. Lo que se pretende aquí es comprobar el estado del proyecto y conseguir una constante mejora del proceso.

  En nuestro caso, la revisión del final de un *sprint* se corresponde a una reunión con los tutores del proyecto y todas aquellas personas involucradas. A excepción del primer *sprint*, meramente de estudio, en todos los demás se generan productos estables que podrían constituir ya una entrega.
- El comienzo de un *sprint* debe servir como un punto de planificación del mismo. ¿Qué tareas podemos llevar a cabo en este tiempo? ¿Cuánto tiempo necesitamos?
- En proyectos de mayor entidad podrían considerarse reuniones diarias. Obviamente, en nuestro proyecto esto no es posible, especialmente pensando en que el equipo de desarrollo no puede realizar suficientes avances en un período de tiempo tan corto, aunque podría ser útil para resolver dudas que hubieran surgido después de la última reunión.

Cada una de estas iteraciones sigue el flujo que se indica en la Ilustración 12.

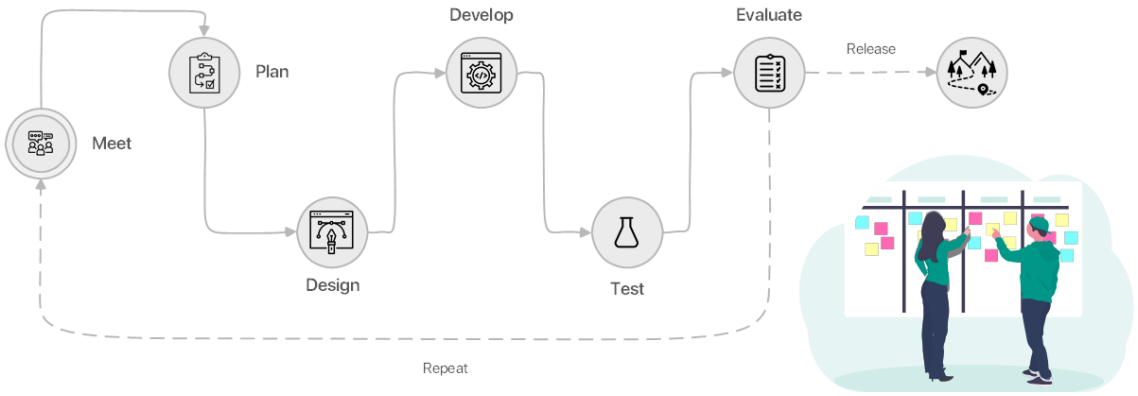


Ilustración 12. Esquema del ciclo de vida de un proyecto de metodología ágil (basado en iteraciones).

En definitiva, los principales beneficios de la metodología SCRUM son los siguientes:

- **Flexibilidad**: en nuestro proyecto hay cambios constantes. Prácticamente en cada reunión pueden surgir necesidades diferentes.
- **Reduce el tiempo que necesita el usuario para poder utilizar el producto**, aunque sea una entrega parcial. Por ejemplo, el cliente podría probar la utilidad de segmentar la vegetación a partir de una imagen multiespectral en una etapa muy temprana del desarrollo, aunque el resto de funcionalidades aún no se hayan desarrollado.
- **Más calidad de software**. Las revisiones al final de cada *sprint* pueden ayudar a lograr precisamente esto, por poner un ejemplo.
- **Más productividad**, derivado del trabajo autónomo del equipo.
- **Predicción de tiempos**: si sabemos que en un *sprint* se debe implementar una funcionalidad, es fácil dar un tiempo de espera hasta que dicha utilidad pueda ser probada por el usuario.
- **Reducción de riesgos**.

1.7 - Material y métodos

Los medios utilizados en el desarrollo de este trabajo se describen necesariamente en el apartado de Presupuesto. Asimismo, se deben incluir aquellos

conjuntos de datos, definidos anteriormente como parte del entregable, que se obtienen a partir de todos los dispositivos listados en dicho apartado.

En cuanto a la metodología, para el desarrollo del proyecto se ha empleado SCRUM, como se ha propuesto en el apartado anterior. Dentro de cada iteración, la metodología se basa en una fase de estudio de las funcionalidades a desarrollar, implementación de las mismas, reunión al final de la propia iteración y documentación del desarrollo realizado, donde esta última fase no se integra en la planificación como parte de una iteración sino como una tarea global.

1.8 - Normas y referencias

En este apartado se detallan todas aquellas normas y reglamentos que han influenciado alguna etapa de desarrollo del proyecto, como puede ser la correcta elección de software externo que cubra las necesidades de este trabajo.

1.8.1 - Licencias software

El objetivo de este proyecto no es producir un software comercializable sino un prototipo de aplicación con fines de investigación. Sin embargo, teniendo en cuenta situaciones futuras y los posibles usos que se le pueda dar al software derivado de este proyecto, es necesario conocer qué se nos permite hacer con él teniendo en cuenta el software de terceros que se ha incorporado a este trabajo. Lo que a continuación se expone no pretende sólo delimitar qué se puede hacer con el prototipo resultante, sino también conocer si es necesario utilizar un software alternativo con menos restricciones en el caso de que la dirección que tome este proyecto fuera completamente diferente a la actual.

Se tratarán sólo licencias de código abierto ya que se supone que sólo se tiene acceso a este tipo de software durante el desarrollo del trabajo.

1.8.1.1 - Estudio de licencias del proyecto

Una licencia software es un instrumento legal que controla el uso y redistribución de software. De forma general, se definen un conjunto de permisos que un desarrollador le puede otorgar a un usuario para distribuir, usar o modificar el producto bajo una licencia específica.

Algunos términos de interés dentro del área que vamos a introducir son los siguientes (Open Source Initiative, 2019):

- *Copyright*: conjunto de normas jurídicas y principios que afirman los derechos morales y patrimoniales que la ley concede a los autores por el simple hecho de la creación de una obra literaria, artística, musical, científica o didáctica, publicada o no.
- Software libre y de código abierto (FOSS o FLOSS, Free and Open Source Software): los usuarios pueden estudiar, modificar y mejorar su diseño mediante la disponibilidad del código fuente. El software libre y el software de código abierto presentan ciertas diferencias en su filosofía. De hecho, para que un software se considere software libre debe cumplir con los requisitos anteriormente comentados, y para que sea de código abierto debe reunir también los siguientes:
    - Libre distribución.
    - Libre acceso al código fuente.
    - La redistribución de modificaciones debe estar permitida.
    - Integridad del código fuente original: según la licencia, las modificaciones podrían ser redistribuidas sólo como parches.
    - Nadie puede quedarse fuera: no debe existir discriminación de personas o grupos.
    - Los usuarios comerciales no pueden ser excluidos (áreas de iniciativa).
    - Deben aplicarse los mismos derechos a todo el que reciba el producto software.
    - El programa no puede licenciarse sólo como parte de una distribución mayor.
    - No debe restringir a otro software, es decir, no puede obligar a que algún otro software que se distribuya junto a este deba pertenecer a la misma categoría.

- No debe requerir la aceptación de la licencia por medio de un acceso (click de ratón, etc).

Teniendo en cuenta el software que utiliza el producto final de este trabajo (que se tratará más tarde), las licencias de interés para conocer nuestras restricciones son las siguientes (Github, 2019):

- Licencias permisivas: se trata de licencias de software libre, y por lo tanto, permite su uso para cualquier propósito: distribución, modificación, compartir modificaciones, etc.

  Al ser permisiva, las obras derivadas del proyecto base no tienen por qué distribuirse con la misma licencia. Ni siquiera están obligadas a ser *open source*. Sin embargo, aquellas partes no modificadas deben mantener la licencia original. Licencias de interés:

  - **BSD 3-Clause** "New" or "Revised" License. Utilizada por OpenCV. Permisos:

    - Posibilidad de uso comercial: tanto el proyecto original como modificaciones.
    - Libre distribución.
    - Libre modificación.
    - Concesión de derechos de patente por parte de las personas que han contribuido en el proyecto.
    - Se puede usar y modificar en privado.

    Condiciones:

    - El software debe incluir una copia de la licencia y del *copyright* establecido.

- Licencias *copyleft*: a diferencia de las licencias permisivas, todas las obras derivadas del proyecto original se deben licenciar bajos los mismos términos y condiciones de la licencia original. Licencias de interés:

  - **GNU General Public License v2.0**. Licencia de Exiv2. Permisos:

    - Posibilidad de uso comercial: tanto el proyecto original como modificaciones.

- Libre distribución.
- Libre modificación.
- Se puede usar y modificar en privado.

Condiciones:

- El código fuente debe estar disponible cuando el software se distribuya.
- El software debe incluir una copia de la licencia y del *copyright* establecido.
- Las modificaciones deben mantener la misma licencia. En algunas condiciones se pueden utilizar licencias relacionadas o similares.
- Los cambios realizados al proyecto se deben documentar.

- **GNU Lesser General Public License v3.0**. Utilizada por Qt. Diferencias respecto de la licencia GNU General Public License v2.0:
  - La condición de mantener la misma licencia para las modificaciones no se aplica en caso de que el proyecto se utilice como una librería.
  - Concesión de los derechos de patente por parte de las personas que han contribuido en el proyecto.
  - Se especifica que trabajos mayores, definiendo trabajos mayores (*larger works*) como aquellos proyectos que combinan el proyecto original con otro material, se podrían distribuir bajo licencias diferentes. Nótese que esa licencia diferente debe ser compatible con esta licencia que estamos definiendo, entendiendo por compatibilidad la no contradicción de los fundamentos por los que ambas se rigen.

1.8.1.2 - Conclusiones sobre licencias

En base a las licencias que se incluyen en este proyecto, que acaban de ser presentadas, se debe fijar nuestro interés en este apartado de conclusiones en las

licencias empleadas por Qt y Exiv2, ya que las licencias (L)GPL son más restrictivas, mientras que BSD (OpenCV) ó MIT son justamente lo contrario.

El principal obstáculo de cara a una comercialización está en la primera de las restricciones que ambas licencias GPL consideran: código fuente disponible al distribuirse el software. A estas alturas debemos plantearnos también qué se considera como distribución, dado que no será lo mismo una distribución interna (uso privado) que de cara al público. Nótese además como no se nos permite modificar la licencia, por lo que se debe redistribuir bajo el mismo formato. Se extraen por lo tanto las siguientes conclusiones:

- Con un objetivo no comercial, y por lo tanto utilizándose en un ámbito privado, no es necesario permitir el acceso al código fuente. Si bien, dado que este trabajo tiene un carácter exclusivo de investigación, parece contraproducente no permitir el acceso a dicha información, al menos en el ámbito privado de la organización.
- En el caso de un uso comercial sería necesario buscar alternativas a las librerías que emplean estas licencias restrictivas (o bien utilizar una licencia comercial y no GPL, si la tuvieran, que no es el caso de Exiv2), dado que cualquier usuario tendría acceso al código fuente y podría redistribuirlo de forma independiente.

1.9 - Planificación temporal

Para elaborar la planificación temporal se debe tener en cuenta que este proyecto forma parte de un trabajo de fin de grado, por lo que el número de horas de de desarrollo asciende hasta 300. Dada la naturaleza iterativa del proyecto, el plazo de elaboración debe dividirse necesariamente en iteraciones que no tienen por qué ser independientes, sino que se apoyan en la base formada por una etapa previa.

A modo de introducción, se presenta un resumen del trabajo realizado en cada uno de los *sprints* o iteraciones en la Tabla 1.

Iteración y fecha de comienzo	Descripción
Iteración 1 (29 de octubre)	Comienzo del proyecto. Revisión de OpenCV y Qt.

Iteración 2 (7 de noviembre)	Proyecto de segmentación de imágenes para detección de vegetación. Desarrollar aplicación y probar algoritmos.
Iteración 3 (23 de noviembre)	Un paso más en la segmentación: eliminar sombras de la vegetación. Estudiar posibles algoritmos para su implementación.
Iteración 4 (3 de diciembre)	Mostrar información geométrica sobre los contornos extraídos: simplificar contornos mediante varios tipos de envolventes.
Iteración 5 (10 de diciembre)	Transformación de imágenes. Primer paso: extraer metadatos de las mismas. Estudio de librerías que trabajen con estos datos. Eliminar distorsión de ojo de pez de las imágenes.
Iteración 6 (20 de diciembre)	Revisión al completo de la aplicación: esquema Modelo-Vista-Controlador y mejoras sobre los procesos de la aplicación (guardado de imágenes, carga, etc).
Iteración 7 (23 de enero)	Registro y georreferenciación de imágenes.
Iteración 8 (26 de febrero)	Integración de varias imágenes multiespectrales a la vez. Modificación de la aplicación completa. Barra de progreso en procesos pesados.
Iteración 9 (25 de marzo)	Mejora de alineación de imágenes con el norte. Otras funcionalidades: disparidad y ortomosaico.
Iteración 10 (9 de abril)	Estudio de otras representaciones: 3D.
Iteración 11 (18 de abril)	Estudio de una base de datos temporal: creación de la misma.
Iteración 12 (24 de abril)	Registro de una imagen RGB con una imagen multiespectral y térmica.

Subproyecto sin iteración (29 de octubre)	Documentación del proyecto.

Tabla 1. Planificación temporal del proyecto, donde para cada iteración se muestra un resumen del trabajo realizado.

Además de cada una de las iteraciones, se presenta un desarrollo paralelo a todas ellas que integra el proceso de documentación de este trabajo desde el día de comienzo del mismo, y que debe terminar cuando todos los entregables, incluida esa documentación, se den por finalizados.

Otra forma de presentar esta información es mediante un **diagrama de Gantt**, que será mucho más gráfico e informativo a nivel de cronología que la información que se acaba de mostrar. A continuación se presentan algunos detalles importantes antes de pasar a visualizar dicho diagrama:

- Cada iteración se considera como un subproyecto que integra a su vez una serie de tareas descritas a alto nivel.
- En nuestro proyecto se ha hecho distinción de hasta tres tipos de tareas, que se presentan de colores diferentes:
  - **Tareas de estudio**: sirven para iniciarnos en un determinado tema hasta ese momento desconocido, o bien actualizar la información y explorar nuevas vías de resolver un problema.

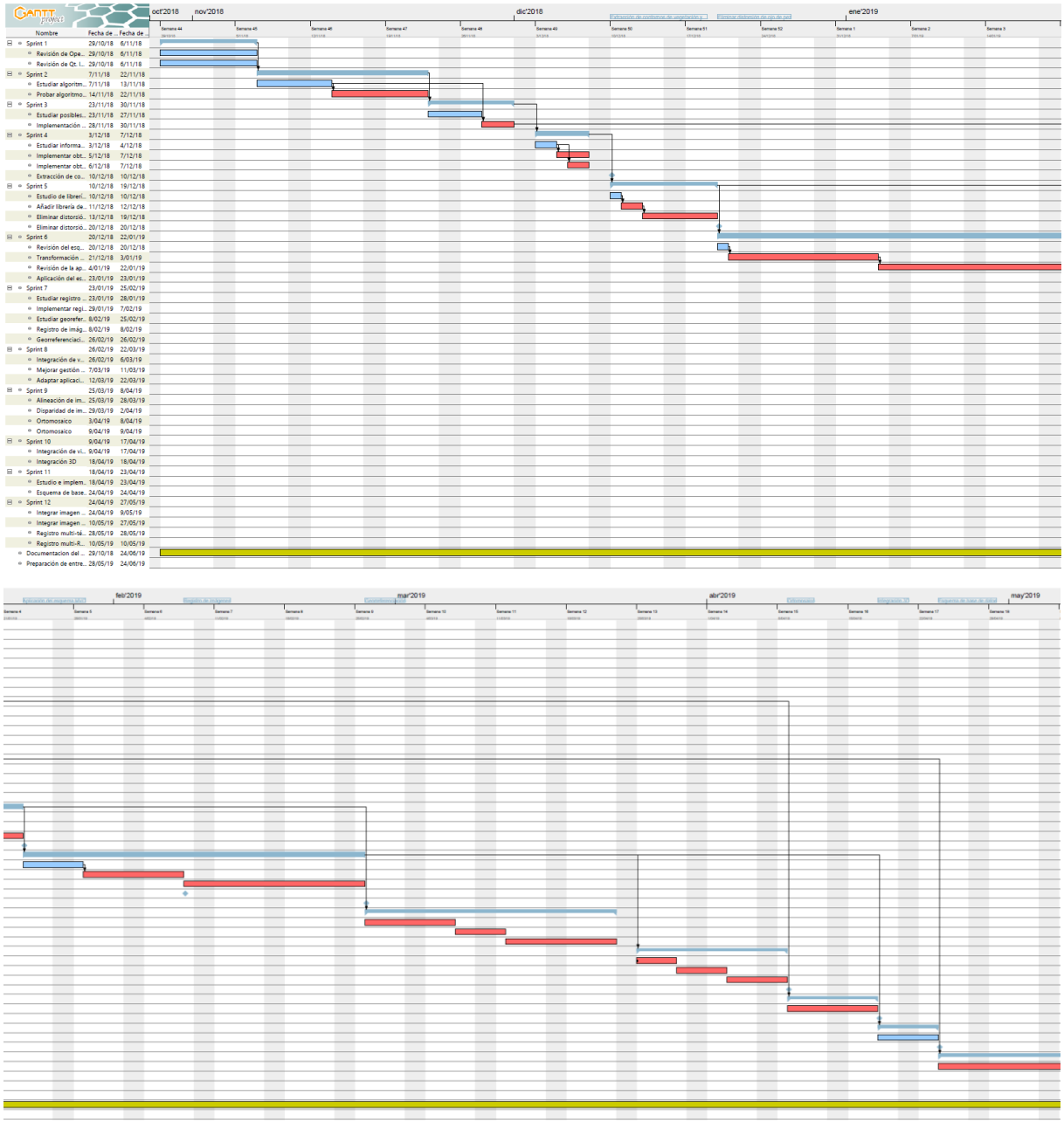
    Color:
  - **Implementación**: proyecto de programación que nos conducirá necesariamente a la solución final.

    Color:
  - **Documentación y preparación de resultados**. Color:
- Dentro del proyecto se podrán distinguir una serie de tareas que no son tal sino **hitos**, que representan el logro o la finalización de un aspecto importante del proyecto. Se integran como tareas de un solo día que en el esquema vendrán representadas en forma de rombos.

Dada la entidad del esquema final, como se puede apreciar en la Ilustración 13 y la Ilustración 14, se presenta toda esta información en intervalos de tiempo reducidos:

Nombre	Fecha de inicio	Fecha de fin
Sprint 1	29/10/18	6/11/18
Revisión de OpenCV. Integración con Visual Studio	29/10/18	6/11/18
Revisión de Qt. Integración con Visual Studio y primeros pasos	29/10/18	6/11/18
Sprint 2	7/11/18	22/11/18
Estudiar algoritmos de segmentación y su implementación en OpenCV	7/11/18	13/11/18
Probar algoritmos estudiados y probar relación resultado / latencia	14/11/18	22/11/18
Sprint 3	23/11/18	30/11/18
Estudiar posibles algoritmos para eliminar sombras de contornos	23/11/18	27/11/18
Implementación de algoritmos de eliminación de sombras	28/11/18	30/11/18
Sprint 4	3/12/18	7/12/18
Estudiar información sobre extracción de los contornos propuestos: hexágonos y envolventes convexas	3/12/18	4/12/18
Implementar obtención de contornos	5/12/18	7/12/18
Implementar obtención de centros de gravedad	6/12/18	7/12/18
Extracción de contornos de vegetación y centros de gravedad	10/12/18	10/12/18
Sprint 5	10/12/18	19/12/18
Estudio de librerías que nos permitan extraer metadatos	10/12/18	10/12/18
Añadir librería de metadatos al proyecto: Exiv2	11/12/18	12/12/18
Eliminar distorsión de ojo de pez. Obtener datos de ficheros e implementación	13/12/18	19/12/18
Eliminar distorsión de ojo de pez	20/12/18	20/12/18
Sprint 6	20/12/18	22/01/19
Revisión del esquema Modelo-Vista-Controlador	20/12/18	20/12/18
Transformación de la aplicación para adaptarse al esquema MVC	21/12/18	3/01/19
Revisión de la aplicación. Mejora de los procesos existentes y creación de otras comandos básicos (cargar, guardar...)	4/01/19	22/01/19
Aplicación del esquema MVC	23/01/19	23/01/19
Sprint 7	23/01/19	25/02/19
Estudiar registro de imágenes	23/01/19	28/01/19
Implementar registro de imágenes	29/01/19	7/02/19
Estudiar georeferenciación de imágenes y extraer dicha información de las imágenes	8/02/19	25/02/19
Registro de imágenes	8/02/19	8/02/19
Georreferenciación	26/02/19	26/02/19
Sprint 8	26/02/19	22/03/19
Integración de varias multiespectrales a la vez en la aplicación	26/02/19	6/03/19
Mejorar gestión de procesos pesados en la aplicación	7/03/19	11/03/19
Adaptar aplicación a cambios	12/03/19	22/03/19
Sprint 9	25/03/19	8/04/19
Alineación de imágenes con el norte	25/03/19	28/03/19
Disparidad de imágenes	29/03/19	2/04/19
Ortomosaico	3/04/19	8/04/19
Ortomosaico	9/04/19	9/04/19
Sprint 10	9/04/19	17/04/19
Integración de visualización 3D en la extracción de contornos	9/04/19	17/04/19
Integración 3D	18/04/19	18/04/19
Sprint 11	18/04/19	23/04/19
Estudio e implementación de base de datos temporal	18/04/19	23/04/19
Esquema de base de datos	24/04/19	24/04/19
Sprint 12	24/04/19	27/05/19
Integrar imagen multiespectral con imagen térmica	24/04/19	9/05/19
Integrar imagen multiespectral con imagen RGB	10/05/19	27/05/19
Registro multi-térmica	28/05/19	28/05/19
Registro multi-RGB	10/05/19	10/05/19
Documentacion del proyecto	29/10/18	24/06/19
Preparación de entregables	28/05/19	24/06/19

Ilustración 13. Tareas del proyecto distribuidas a lo largo de las iteraciones junto al tiempo dedicado a cada una de ellas.


oct'2018
nov'2018
dic'2018
ene'2019
feb'2019
mar'2019
abr'2019
may'2019

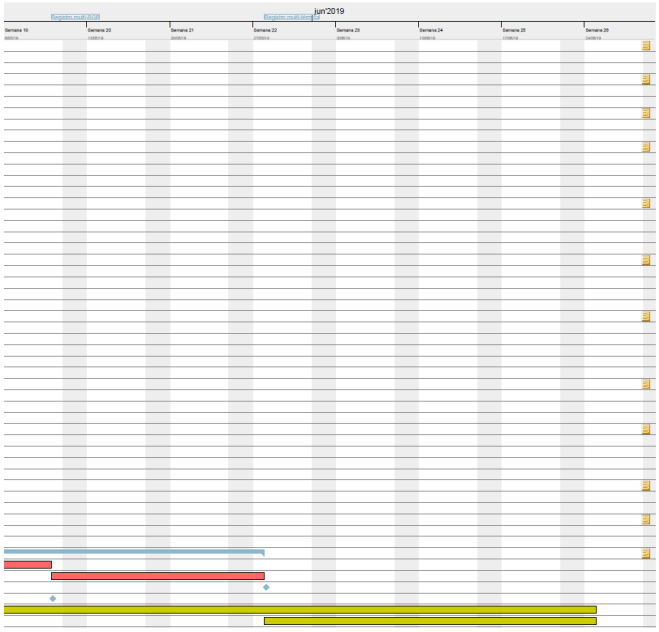


Ilustración 14. Diagrama de Gantt para la planificación temporal del proyecto.

1.10 - Presupuesto

El objetivo de la elaboración del presupuesto es conocer el **coste económico que conlleva la ejecución del proyecto de investigación que aquí se trata**. Para la elaboración y justificación del presupuesto se listarán un conjunto de materiales y medios necesarios junto a su coste económico. Se tenderá siempre a escoger el coste del proveedor oficial, por lo que el presupuesto que aquí se presenta podría verse modificado, especialmente reduciéndose, si se obtiene todo lo necesario de fuentes alternativas.

Se pueden clasificar las necesidades de este proyecto en las siguientes categorías:

- **Hardware**: no sólo incluye los componentes del ordenador que nos permite desarrollar este proyecto, sino también aquellos dispositivos a través de los cuales se obtiene la información que se utiliza en este trabajo.
- **Software**: no todo el software que aquí se emplea es gratuito, y aunque existan versiones que sí lo sean para la comunidad de estudiantes, se consideran versiones ampliadas y de pago (si las hubiera), como probablemente suceda en una empresa. Un ejemplo de esto que aquí se expone es Visual Studio, que tendría la versión gratuita, Visual Studio

Community, y la versión profesional. El objetivo, como siempre, es situarnos en el peor de los casos a la hora de elaborar el presupuesto.

- **Materiales cotidianos de trabajo**.
- **Personal y servicios**: coste económico asociado a la actividad de investigación y el soporte a la misma. Por ejemplo, una parte importante de este proyecto es la captura de información, de la que no se encarga el investigador sino otro tipo de personal.
- **Costes indirectos** derivados del uso de instalaciones, facturas de servicios, etc.

Se procede por lo tanto a desglosar el coste de cada una de las categorías que se acaban de citar.

1.10.1 - Coste hardware

Comenzaremos este apartado detallando el coste asociado a los dispositivos que se emplean durante el vuelo del dron para capturar la información de entrada necesaria en este proyecto, así como aquellos dispositivos que permiten transportar la información capturada al equipo de desarrollo (Tabla 2), aunque nuestra labor no sea llevar a cabo todas estas operaciones de obtención de los datos.

Dispositivo	Descripción	Coste estimado
Parrot Sequoia	Cámara multiespectral	3.200€
Parrot Sequoia: diana de calibración	Diana para calibrar la cámara multiespectral de forma autónoma	123,88€
DJI Zenmuse XT2 (640, 13/19 mm)	Cámara termográfica (también proporciona imágenes RGB)	9.990€
DJI Matrice 210	Dron para la captura de información y todos sus componentes asociados	9.998€
Disco duro externo Toshiba Canvio Basics 2.5'' 1TB USB 3.0	Almacenamiento de información capturada en el vuelo	46,99€

Total	23.358,87€

Tabla 2. Coste económico de dispositivos necesarios para obtener la información de entrada de la aplicación.

Para los dispositivos DJI su precio se ha calculado a partir de fuentes externas como *ThermalCapture*[1] o *cnet*[2], dado que la página oficial (y otras tantas fuentes) no proporcionan precios en muchos casos sino sólo las especificaciones de los productos.

Además del coste de estos dispositivos que se acaban de presentar, se debe añadir el hardware que forma parte del equipo de desarrollo. Para tal fin se exponen a continuación (Tabla 3) los componentes del ordenador empleado durante el trabajo autónomo, extraído de su correspondiente factura de una tienda de la franquicia APP Informática.

Componente	Descripción	Coste estimado
Intel core i7.6700 3.4GHz 8M LGA1151 14NM	Procesador	324,1€
MSI 1151 Z170 A PRO	Placa base	117,4€
W. Digital 1TB 3.5 WD10EZEX SATA3 7200 64MB W	Disco duro	54,9€
Kingston HyperX Savage DDR4 16GB 2666 CL13 K	Memoria RAM	109,5€
Torre ATX NOX COOLBAY SX Negra Iluminación L	Carcasa de la torre	47,1€
Cooler Master Refrigeración. CPU HYPER 212X INTEL	Ventiladores	37,4€
GIGABYTE VGA NVDIA GTX 1070 WF2 OC 8GB DDR5	Tarjeta gráfica	532,2€
Asus Grabadora 24X Sata Energy DR W24F1ST	Grabadora	14,8€
Samsung 2.5 256GB SATA3 SERIE 850 PRO	Disco duro SSD	148€

[1] https://thermalcapture.com/
[2] https://www.cnet.com/products/dji-matrice-m210-drone/

NOX URANO VX 650W ATX	Fuente de alimentación	54,6€
ASUS VS247NR 23.6" FullHD Negro Reacondicionado	Monitor	98€
Teclado mecánico CM Storm	Teclado	119,9€
Keep Out X4	Ratón	29€
Total		1.686,9€

Tabla 3. Coste económico del hardware que forma el equipo de desarrollo.

Nótese como dicha factura corresponde al coste del ordenador utilizado para el desarrollo de este proyecto en un instante de tiempo no necesariamente actual.

1.10.2 - Coste software

Dentro de esta sección se incluye todo aquel software no gratuito necesario para algunas de las tareas que forman parte del trabajo. Asimismo, se omite el software que sí lo es en cualquiera de sus versiones, dado que no representa un coste económico. Cabe destacar que algunos de los productos que se citan a continuación (Tabla 4) no han generado ningún gasto durante la elaboración de este proyecto, pero sí podrían representarlo en una empresa.

Software	Descripción	Coste estimado
Visual Studio 2019 Professional	Desarrollo de la aplicación principal	40,2€ para la tarifa mensual (x8) = 321,6€
Adobe Photoshop CC	Prototipado de soluciones mediante manipulación de imágenes	214,32€
PyCharm Professional	Prototipado de algoritmos	199€ el primer año
Licencia de Windows 10 Pro	Sistema operativo del ordenador de desarrollo	199,99€

Office 365 Personal	Prototipado de interfaces gráficas, documentación	69€ / mes (x8) = 552€
Visual Paradigm Professional	Gestión y análisis del proyecto	35€ / mes (x8) = 280€
Total		1.776,91€

Tabla 4. Coste económico del software empleado en este proyecto.

La mayoría de productos aquí listados tienen varios tipos de planes, aunque se han elegido los planes mensuales dado el carácter temporal del proyecto. En cualquiera de los casos, los planes anuales o permanentes son más económicos si se desea utilizar el producto con continuidad, por lo que de nuevo se ha acotado el precio por encima.

1.10.3 - Coste de materiales

Los materiales que se incluyen en la Tabla 5 son materiales de oficina que nos ayudan durante el proceso de elaboración del proyecto. Para simplificar la búsqueda de costes de cada material se ha utilizado la web de comercio electrónico Amazon[3].

Material	Descripción	Coste estimado
Pack de 5 cuadernos espiral de tapa dura, A5	Prototipos manuales, desarrollo de soluciones, planteamiento de problemas…	7,56€
Casio FX-82MS, calculadora científica	Comprobación de soluciones obtenidas y otros cálculos	10,45€
5 unidades de bolígrafos BIC azules	Prototipos manuales, desarrollo de soluciones, planteamiento de problemas…	2,4€
Total		20,41€

Tabla 5. Coste económico de materiales empleados en el proyecto.

[3] https://www.amazon.es/

1.10.4 - Coste de personal y servicios

Para estimar el coste asociado al personal será aún más importante acudir a las fuentes correctas, y es por ello que nos hemos basado en el **XVII Convenio colectivo estatal de empresas de consultoría** (Ministerio de Empleo y Seguridad Social, 2018), y estudios de mercados y de la opinión pública, publicado en el BOE.

Buena parte del peso de este trabajo recae en la investigación, aunque de cara a un producto final, la gestión, el análisis y el control de un proyecto podrían ser fundamentales. Asimismo, el desarrollador principal del trabajo no debe ser necesariamente quien obtenga los datos de entrada a partir del vuelo de un dron. De hecho, en nuestro proyecto esto no es así. Parece obvio por lo tanto la contratación de personal para tal fin. Igualmente, se podría contemplar personal para el montaje de dispositivos tales como el ordenador de desarrollo. Para este último caso no se acude al convenio citado sino a la factura que anteriormente se presentaba, donde también se muestra el coste económico por tiempo de montaje y puesta en marcha del sistema.

Personal	Descripción	Coste estimado
Analista	Análisis y gestión del proyecto	25.189,02€ / año, 18.891,75€ / 8 meses
Programador senior	Desarrollo del proyecto de programación	17.715,65€ / año, 13.286,73€ / 8 meses
Personal de montaje del equipo	Profesional a cargo del montaje de los componentes	39,5€
Personal para volar el dron	Salario del personal[4]	13.503,97€ / año, 10.127,97€ / 8 meses
	Curso de piloto avanzado de drones (hasta 25 kg)[5]	1.290€

[4] Se ha considerado el perfil de Operador de periféricos del convenio, grupo E II.

[5] En base a un curso ofertado por la empresa TOPGUN, certificada por la Agencia Estatal de Seguridad Aérea (AESA): http://topgun.com.es/piloto-avanzado-de-drones-rpas/

	Seguro de responsabilidad civil (< 20 kg)[6]	200€ / año
Total		43.835,22€

Tabla 6. Coste económico derivado del personal que forma parte o hace posible el desarrollo del proyecto.

1.10.5 - Coste derivado del proyecto

En esta última sección de desglose de elementos necesarios se incluyen aquellos gastos derivados del propio proceso de desarrollo: luz, agua, conexión a internet, hipoteca/alquiler, limpieza, etc.

Si bien es cierto que podríamos detallar el coste de ciertos servicios como la conexión a internet, existen otros tantos como el gasto de luz durante el tiempo de ejecución del proyecto que se desconocen por completo. Por lo tanto, haremos dos diferenciaciones: servicios de coste conocido, que serán la base de este apartado, y otros tantos servicios que se cubrirán con un sobrecoste de un porcentaje del valor de ejecución del proyecto. Se deberá calcular pues este segundo una vez que se conocozca la sumatoria de todo lo anteriormente listado.

Servicios	Descripción	Coste estimado
Ono Vodafone Fibra 600 Mb y llamadas ilimitadas	Conexión a internet para la consulta de documentación	24,84€ / mes (x8) = 198,72€
Sobrecoste para cubrir otros servicios	Sobrecoste para cubrir servicios básicos como agua o luz	0.1 x 70.678,31€ = 7.067,83€
Total		7.266,55€

Tabla 7. Coste económico de otros servicios derivados del proyecto.

1.10.6 - Coste total

Se pueden resumir todos los costes hasta considerados en la Tabla 8.

[6] https://aeracoop.net/guia-seguros-drones/

Origen	Coste estimado
Coste hardware	25.045,77€
Coste software	1.776,91€
Coste de materiales	20,41€
Coste de personal y servicios	43.835,22€
Coste derivado del proyecto	7.266,55€
Total	77.944,86€

Tabla 8. Resumen de costes hasta ahora justificados.

A partir de la sumatoria de todos los costes que se han establecido con anterioridad se obtiene la cifra de **77.944,86€** para la puesta en marcha de este proyecto partiendo de cero. Como ya se ha comentado durante este apartado, dicha cifra deberá adaptarse a las necesidades del grupo que desarrolle este trabajo, dependiendo de los medios de los que ya disponga.

1.10.7 - Amortización

Además del coste de cada uno de los dispositivos y materiales aquí empleados, es necesario calcular la amortización parcial de cierto conjunto de los elementos expuestos con anterioridad.

Se aplica la amortización a los bienes materiales, especialmente los relacionados con el hardware de este proyecto. Por ejemplo, el software de terceros que se contrata no tendrá una vida útil como tal sino un intervalo de tiempo hasta que el software pueda quedar obsoleto, algo que en muchos casos no sucede ya que se ve sometido a actualizaciones constantes. Quizás esto último fuera necesario en el caso de un sistema operativo, que pone a la venta nuevas versiones cada cierto tiempo y en el que no se puede suponer que podamos actualizar a la última versión de forma gratuita. En cualquiera de los casos, se considerará el coste del bloque de **software que no se utiliza temporalmente** (licencia permanente), y se le aplicarán los coeficientes que puedan aproximar su amortización.

Para todos los bienes que se exponen se considera una **amortización lineal** para reducir la complejidad del problema, de tal forma que se define una base

amortizable a partir del coste inicial y el valor residual, además del tiempo de vida del bien en cuestión (o bien un coeficiente de amortización para aquellos bienes que se supone que no tendrán vida útil propiamente dicha).

La vida útil del hardware aquí empleado se considera de 10 años, y la del software permanente de 6 años, de acuerdo a los coeficientes de amortización lineal que considera la Agencia Tributaria[7] (Agencia Tributaria, 2017). Asimismo, se parte de que el valor residual del hardware es del 30% de su valor inicial, y de un 50% para el software.

La amortización lineal propuesta se debe calcular como sigue:

$$Base\ AMT = Precio\ adquisición - Valor\ residual \quad (1)$$

$$AMT\ anual = \frac{Base\ AMT}{Número\ de\ años} = Base\ \ AMT * Coef.\ de\ AMT \quad (2)$$

Se obtiene así la Tabla 9 con los resultados de aplicar las anteriores fórmulas a cada uno de los bienes amortizables de este proyecto.

Tipo de elemento	Valor inicial	Valor residual	Vida útil	Amortización anual
Hardware (conjunto)	25.045,77€	7.513,73€	10 años	2.429,4€
Software permanente (conjunto)	434,31€	207,15€	6 años	37.86€
Total amortizado				2.532,26€

Tabla 9. Tabla de amortización de bienes expuestos en el coste del proyecto.

Dado que el desarrollo del proyecto conlleva 8 meses, se considera que el valor amortizado es 2.532,26€ / 12 * 8 = **1.688,17€**.

[7] Tabla de coeficientes de amortización lineal a partir del 1 de enero de 2017 (última actualización). Fuente: Agencia Tributaria.

2 - DESARROLLO DEL PROYECTO

En este apartado se detallan todos los aspectos relacionados con la implementación de este trabajo, lo que incluye no sólo la explicación de qué soluciones se han propuesto y su razonamiento, sino también una fase de introducción del lector en el tema que se va a tratar.

A continuación se describen aquellas tecnologías y servicios que han posibilitado la implementación de la solución final.

2.1 - Tecnologías utilizadas

El objetivo de este apartado es describir brevemente las tecnologías que se han utilizado para ayudar, de una manera u otra, al desarrollo del proyecto.

- **Lenguajes de programación**: la elección del lenguaje C++ viene influenciada por las librerías que a continuación se describen. Si bien es cierto que el apartado de procesamiento de imágenes se puede cubrir con otros lenguajes como Python, la representación tridimensional que nos sirve de soporte en ciertas etapas ha decidido implementarse sobre C++, dado que ya partimos de cierta experiencia previa con OpenGL (aunque de nuevo, podría utilizarse a través de *bindings* (PyOpenGL 3.x, s.f.)).

  Sin embargo, para las operaciones de experimentación que se plantean sí que se ha utilizado Python para desarrollar algoritmos a modo de prototipado por su facilidad de uso.

  Una mínima parte del proyecto va destinada a la generación de una base de datos temporal, por lo que es necesario algún lenguaje acorde a dicho objetivo. Dado que se ha optado por crear la base de datos con Oracle Database y SQL Developer, el lenguaje de consulta elegido es SQL. En cualquier caso, el objetivo de este pequeño apartado es crear un modelo de base de datos que podría fácilmente extrapolarse a una base de datos no relacional convirtiendo los atributos en claves de una entidad.

- **Librerías**: múltiples librerías son necesarias para resolver ciertos problemas que se nos plantean y también para posibilitar la extracción de información.

- **OpenCV**: es una librería de visión por ordenador y aprendizaje automático de tipo *open source* (ya se ha entrado en detalle en el apartado de Licencias software). Contiene una gran cantidad de algoritmos optimizados en este área, tanto clásicos como más recientes: detección y reconocimiento de caras, identificación de objetos, seguimiento de objetos en la escena, seguimiento del movimiento de la cámara, extracción de nubes de puntos 3D, etc (OpenCV, 2019).

  Buena parte de nuestro proyecto se fundamenta sobre el procesamiento de imágenes, y el uso de una librería tan robusta y con tal cantidad de algoritmos nos permite dirigir nuestros esfuerzos hacia otras tareas que no sean la implementación de algoritmos básicos ya asentados (filtros gaussianos, segmentación, conversión a un espacio de color diferente, etc).

- **Exiv2** (Exiv2, 2019): el procesamiento de la imagen no se restringe a extraer información gráfica y realizar un conjunto de operaciones acorde a ella, sino también a extraer información física (distancia focal, coeficientes de distorsión, etc) y actuar en consecuencia.

  Exiv2 es una librería implementada en C++ para el acceso a un amplio conjunto de metadatos que pueden estar incluidos en la imagen. Una de las ventajas que tiene sobre otras alternativas es que puede trabajar con un gran número de formatos de imagen, además de los múltiples esquemas de metadatos que es capaz de reconocer. En concreto, nosotros trabajamos con esquemas tan comunes como XMP o el estándar EXIF, pero dado que se puede añadir cualquier otro dispositivo al conjunto de utilidades para el control de la plantación, el soporte a esa gran cantidad de metadatos es algo a valorar positivamente.

- **Application Programming Interface o API**, entendiendo como tal aquellas interfaces de programación de aplicaciones.

  - **Open Graphics Library (OpenGL)**. Se trata de una interfaz de programación multiplataforma y multilenguaje (a través de *bindings*)

que posibilita la obtención de productos gráficos 2D y 3D (OpenGL, 2019). En nuestro proyecto se relega a un papel secundario ya que tiene más peso el procesamiento de imágenes, aunque todo esto nos permite abrir una puerta de cara a un producto comercial donde el usuario pueda observar los resultados desde otro punto de vista.

- ***Framework* para la creación de aplicaciones gráficas**. Una parte muy importante de este proyecto es la interfaz gráfica, ya que nos ayuda a comprobar la corrección de los resultados obtenidos a partir de los algoritmos propuestos. Si el producto se desarrollara como una herramienta más del dispositivo de vuelo para la extracción de información en tiempo real, entonces no sería necesario este componente y debería eliminarse para mejorar la eficiencia del producto.

  El *framework* seleccionado para tal fin es **Qt** (Qt, 2019), que se enmarca como un *framework* multiplataforma orientado a objetos para el desarrollo de aplicaciones con interfaces gráficas. Se define además como software libre y de código abierto (GNU Lesser General Public License v3) por la compañía Qt Company, aunque esto no siempre es cierto ya que hay al menos dos formas de uso de este *framework*: uso comercial, que se distribuye totalmente bajo la fórmula anteriormente comentada, y de forma gratuita, que limita ciertas funcionalidades a GPL y elimina otras tantas herramientas respecto de la versión de pago.

  Se debe destacar que Qt soporta varios compiladores, incluyendo la *suite* Visual Studio, con la que se integra fácilmente a través de un *plugin*, de tal forma que como se ha hecho en nuestro caso, se puede crear directamente un proyecto de C++ que ya integra Qt.

  Otras funcionalidades no menos importantes para nuestro proyecto son las siguientes:

  - Incluye acceso a bases de datos: JSON, XML, SQL, etc. Podría ser necesario para el apartado de creación de una base de datos.
  - Incluye manejo de hilos (QThread). Útil para procesos pesados, de tal forma que no se bloquea la interfaz.

- Incluye soporte de acceso a la red. Posible conexión con un sistema de información geográfico o SIG en un futuro, por ejemplo, para obtener la altura de un punto geolocalizado.
- Control de eventos basado en el patrón observador, a través de lo que se conocen como señales (*signals*) y ranuras (*slots*). De esta forma, se definen receptores de eventos para un elemento de la interfaz. Por ejemplo, se podría definir un botón cuyo evento de *click* tenga como destinatario la ventana principal.
- Incluye su propia librería matemática: trabaja con vectores, matrices o quaterniones como lo haría, por ejemplo, GLM. Nos sirve para todas las operaciones de rotación de la imagen (matrices), visualización 3D, etc.

Como detalle adicional, cabe destacar que Qt es uno de los principales *frameworks* de creación de interfaces gráficas hoy en día, siendo utilizada por múltiples compañías de gran valor (AMD, Blizzard Entertainment, European Space Agency, Philips, Samsung, etc) y por aplicaciones bien conocidas (Google Earth, Adobe Photoshop Elements, Autodesk Maya, Mathematica, QGIS, TeamViewer, etc). Aunque esto último parece un detalle más, tiene su valor dado que uno de los principales aspectos a considerar en una librería o un *framework* debe ser su mantenimiento y soporte. El hecho de que múltiples compañías de peso lo utilicen parece un buen indicador en estos términos.

- **Entornos de desarrollo**: serán aquellas aplicaciones que nos faciliten el desarrollo software con un determinado lenguaje. Nuestro principal IDE o entorno de desarrollo será **Visual Studio**, que integra C++ e incluso nos facilita la creación de proyectos de Qt a través de un *plugin*, como se ha comentado con anterioridad. Por otro lado, para la fase de experimentación, donde se utiliza Python, es necesario un IDE como **PyCharm**, integrado dentro de la *suite* de productos de JetBrains. En cualquiera de los casos, ambos entornos de desarrollo tienen una versión gratuita (*Community*).

  También cabe destacar **SQL Developer**, que trabaja sobre nuestra base de datos de Oracle y que nos permite ejecutar sentencias SQL. No es sólo un

entorno de desarrollo sino más bien un gestor de base de datos que nos permite modificarla también mediante la propia interfaz.

- **Sistema de gestión de base de datos**: para tal fin se ha utilizado el gestor objeto-relacional **Oracle Database**, cuya última versión es la 19c. No trabajamos directamente con este sistema sino sobre él con otras herramientas como SQL Developer.
- **Software**: otra parte muy importante del desarrollo de este proyecto son las soluciones software que no se incluyen en apartados previos y que han servido de soporte al trabajo en múltiples etapas. Si bien, se deben distinguir dos categorías principales:
    - **Software de soporte al desarrollo**: se incluyen aquí herramientas utilizadas como soporte para la obtención de los algoritmos que se proponen en apartados posteriores.
        - **Adobe Photoshop**: se trata de un editor de gráficos rasterizados utilizado principalmente para la manipulación de imágenes. Nos permite comprender los problemas que se presentan entre imágenes y constituye una alternativa razonable a la programación, debido a que incluye filtros (que se podrían programar o se encuentran disponibles en OpenCV) que, seleccionados correctamente, nos pueden guiar hacia la solución de un determinado problema.
    - **Software de control de desarrollo de proyecto:** constituyen un conjunto de herramientas que no colaboran directamente en la obtención del resultado final pero nos permiten llegar hasta él de una forma más controlada.
        - **Git Bash:** herramienta de control de versiones de un proyecto a través de consola. Garantiza la integridad de la solución y se puede relacionar fácilmente una rama de trabajo con cada una de las iteraciones que anteriormente se han propuesto.
        - **Gantt Project:** herramienta para la gestión de tiempo en el proyecto. Permite estructurar el tiempo de desarrollo y

documentación del que se dispone mediante hitos, iteraciones y tareas a realizar o ya completadas.

- **Otras herramientas auxiliares**: se consideran en este apartado otras herramientas complementarias que han sido de ayuda durante el desarrollo del proyecto, pero que no se pueden enmarcar dentro de las secciones anteriores. Generalmente hablaremos de servicios web, como los que siguen:
    - **Utilidades para extraer los metadatos de una imagen**. Bien se podría utilizar una herramienta por línea de comandos como Exiv2 o una utilidad web para tal fin (Friedl J. , s.f.). En el caso particular del visualizador que aquí se referencia también se añade la localización de la imagen en el mapa.
    - **Conversión de coordenadas del tipo Latitud, Longitud a UTM**, con el fin de comprobar la corrección del método de conversión entre sistemas de coordenadas de la superficie terrestre que más tarde se muestra. Un ejemplo de esta utilidad es (Montana State University, s.f.), que nos permite una conversión bidireccional y la selección de diferentes tipos de *datum* (WGS84, ED50, etc).
    - **Visualización de puntos sobre un mapa.** Dada la localización geográfica de la cámara o de las cuatro esquinas de una imagen, sería de interés posicionar dichos puntos sobre un plano. Entre otras cosas, nos sirve para comprobar la efectividad de la georreferenciación llevada a cabo. Un ejemplo de utilidad para tal fin es Google Maps en la vista de satélite (Google, s.f.).

2.2 - Desarrollo iterativo

En este apartado se documentan todas las soluciones que se proponen para los problemas existentes en el control de grandes plantaciones a través de diversos tipos de sensores. Será algo más que un mero proceso de documentación de la solución, dado que se incluye para cada nuevo problema una introducción al mismo. Ciertamente se podría haber incluido dicha información en un apartado de conceptos previos, pero dada la gran variedad de la problemática que se plantea, puede resultar

más cómodo entrar en situación justo antes de proponer los algoritmos implementados.

Asimismo, dada la naturaleza iterativa del proceso de desarrollo, se divide esta sección en tantos apartados como iteraciones existan, si bien no todas ellas incluyen los mismos subapartados, sino que depende de la magnitud de cada uno.

Ciertos subapartados de las iteraciones no muestran el carácter iterativo de este proyecto dado que, por ejemplo, un *storyboard* integra, en el momento en el que se redacta este documento, la solución completa. Por lo tanto, ciertas operaciones que se pueden mostrar no tenían por qué estar necesariamente implementadas en el momento de finalización de una iteración. Otros tipos de información, como puede ser el diagrama de clases, se expondrán cuando completemos la implementación de este trabajo. Asimismo, para cada funcionalidad añadida se mostrará su respectivo diagrama de secuencia.

Un apartado que se podría incluir por cada iteración es el de Pruebas de usabilidad del sistema. No obstante, el sistema que se ha implementado no tiene como objetivo el de servir como producto final, sino actuar como una herramienta más para probar algoritmos que nos permitien controlar grandes plantaciones a partir de cierta información de entrada. De hecho, una de las posibles derivaciones de este proyecto sería realizar las operaciones necesarias durante un vuelo y no después del mismo en el laboratorio, en cuyo caso debiera de prescindirse de la interfaz desarrollada. Es por ello que no se incluyen en este documento las respectivas pruebas de usabilidad, aunque sí se han incluido, por ejemplo, controles, para facilitar al investigador el uso de esta aplicación (Controles de teclado y ratón).

2.2.1 - Primera Iteración y segunda iteración

En la planificación se describe la primera iteración como un simple estudio de las características de Qt y OpenCV asociadas a la problemática de partida, que es la extracción de contornos de la vegetación. Es por ello que se incluyen aquí las dos primeras iteraciones, de las cuales el resultado será una **aplicación funcional que extraiga los contornos**, aunque no se correspondan con el resultado deseado.

La información de entrada para esta primera parte del proyecto serán las cinco imágenes que nos permite obtener la cámara multiespectral Parrot Sequoia: cuatro correspondientes a diferentes intervalos aislados del espectro, y una imagen RGB, tal y como muestra la Ilustración 15.

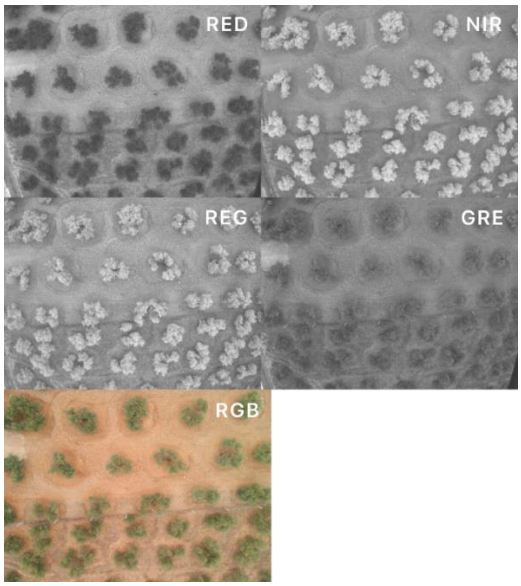


Ilustración 15. Composición donde se muestran todas las imágenes capturadas por el dispositivo Parrot Sequoia para un determinado área de terreno.

Aunque no sea este nuestro objetivo por ahora, se pueden ver claramente ciertas diferencias entre las imágenes, que se traducen en traslaciones horizontales y verticales, debido a la disposición de las lentes en el dispositivo. Por lo tanto, se trabaja por ahora con imágenes ciertamente no alineadas, aunque la diferencia es tan pequeña que el algoritmo que se describe en la siguiente iteración funciona incluso en estas condiciones. En esta iteración sólo se trabaja con una capa.

Las traslaciones entre imágenes vienen descritas por el modelo de distancias entre lentes que nos proporciona Parrot Sequoia (Ilustración 16). Como se puede ver, no se especifica la distancia de la imagen RGB al resto porque normalmente no entra dentro del proceso de registro de imágenes ni es de utilidad alguna, siempre y cuando se disponga de otra imagen RGB de mejores características.

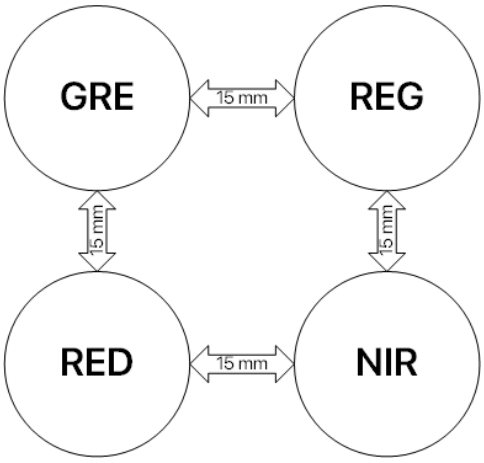


Ilustración 16. Distancia física en milímetros entre las lentes de Parrot Sequoia (Parrot , 2017).

En el caso que nos ocupa, que es la extracción de contornos, una primera aproximación a la solución del problema puede surgir al fijarnos en el constraste de las imágenes, donde la multiespectral RED será visualmente la más valiosa en este aspecto. Se puede formular así un algoritmo básico de extracción de árboles, donde algunas de las operaciones que ahora se proponen forman parte del algoritmo final:

- **Suavizado de la imagen para eliminar el ruido y detalles irrelevantes**. Podemos utilizar cualquiera de las funciones que nos da la librería OpenCV: *blur*, *GaussianBlur*, *medianBlur*, etc. En nuestro caso no tenemos ningún condicionante que haga que nos decantemos por un método u otro. Se elige así el filtro gaussiano.
- **Umbralización de la imagen**. Se fija un umbral de intensidad y todos los píxeles cuyo valor esté por encima de dicho umbral (suelo) se pueden pintar de color negro. Lo contrario sucede con aquellos píxeles cuyo valor está por debajo (corresponden a vegetación y sombra).

Se obtiene así resultados como el de la Ilustración 17.

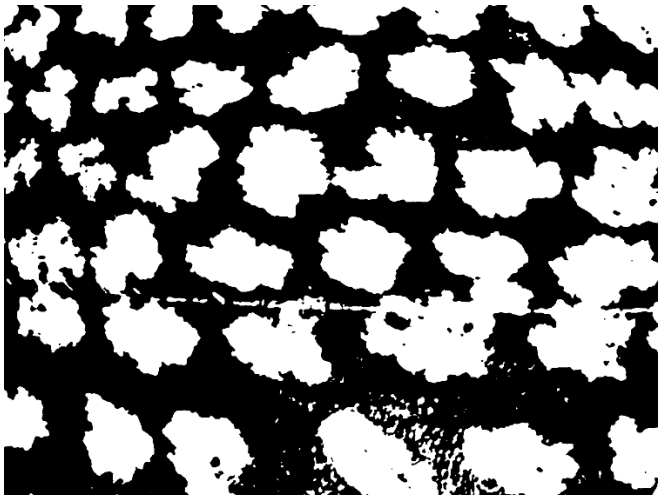


Ilustración 17. Resultado obtenido al suavizar y umbralizar la imagen multiespectral RED.

La solución que se alcanza deja mucho que desear: se entremezclan los árboles (no se distinguen individuos) y se detectan ciertos desniveles del terreno (de color oscuro en la imagen original) como si también fuera vegetación. Otros problemas, como pueden ser las manchas (*blobs*) tan pequeñas que aparecen, se pueden eliminar utilizando un umbral de área para los contornos detectados, de tal forma que todos aquellos que se encuentren por debajo de dicho umbral se eliminan

(ayuda a filtrar vegetación que cualquier terreno puede contener y que no es necesariamente de interés).

Por otro lado, el problema se presentaba realmente como una extracción de contornos y no de polígonos, por lo que a partir de una imagen binaria deberían obtenerse las polilíneas que definen los contornos de los árboles, como se muestra en la Ilustración 18. Esta operación ya viene implementada en la librería OpenCV mediante la operación *findContours*, donde habrá que tener en cuenta dos parámetros importantes:

- **Modo de obtención de contornos**: a la hora de devolver el resultado se puede establecer una jerarquía (*RETR_TREE*), obtener sólo los contornos exteriores (*RETR_EXTERNAL*), no establecer jerarquía ninguna (*RETR_LIST*), etc. Nótese como en la figura anterior se pueden apreciar huecos dentro de las propias áreas de vegetación detectadas, que a su vez suponen otros contornos. Por lo tanto, parece necesario descartar aquellos que tienen un padre en la jerarquía.
- **Densidad de puntos de los contornos**: se puede obtener una lista de puntos dada sólo por las esquinas donde se producen cambios de orientación (*CHAIN_APPROX_SIMPLE*) o bien una lista densa de puntos que especifica posiciones incluso donde no hay cambios de dirección respecto del punto anterior y siguiente (*CHAIN_APPROX_NONE*). Obviamente, el primero de ellos se ajusta mejor a nuestras necesidades, ahorrando además memoria.

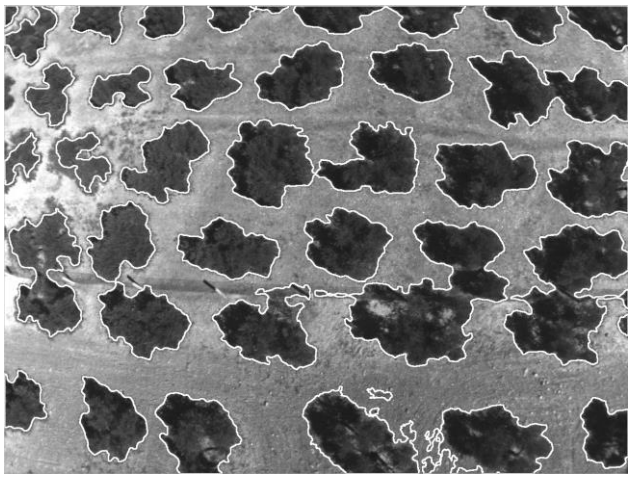


Ilustración 18. Extracción de contornos con un núcleo de emborronamiento de tamaño 10x10.

Un último problema del algoritmo desarrollado hasta ahora se puede ver en la Ilustración 19.

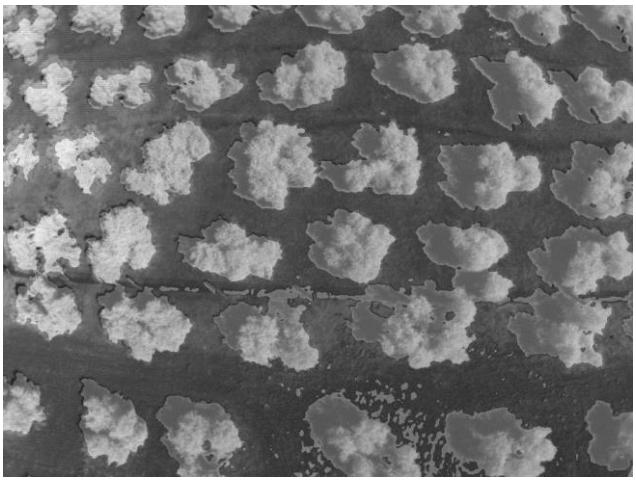


Ilustración 19. Superposición de la imagen multiespectral NIR, con una opacidad del 50%, y la umbralización obtenida.

Como ya se adelantaba, los polígonos extraídos contienen tanto la propia vegetación como la sombra de la misma. Es un problema que aparece en aquellas imágenes donde la sombra tiene un tamaño considerable y donde esta sigue una dirección que propicia su representación en la imagen. Por lo tanto, dependiendo del ángulo de captura y de la posición del Sol podría haber imágenes donde la sombra captada fuera prácticamente nula, en cuyo caso funcionaría este algoritmo. Por supuesto, esas condiciones son difíciles de conseguir y no óptimas, porque el objetivo al capturar las imágenes es que el eje Z de la cámara sea perpendicular al terreno.

Además de este algoritmo aquí expuesto, se propone otro método en esta iteración, que sería la **inundación de la imagen** (*watershed*), uno de los algoritmos más comunes en el ámbito de la segmentación. Sin embargo, su implementación es más compleja y su tiempo de respuesta es mayor.

En este proceso se considera una representación tridimensional de la imagen, donde el valor de intensidad de cada píxel representa la altura. Se distinguen hasta tres tipos de puntos:

- **Mínimo local**.
- **Cuenca**: será un punto donde si de deja caer una gota de agua esta llegaría hasta un mínimo local.
- **Cresta** o línea divisoria: puntos donde si se dejara caer una gota de agua, esta podría llegar a más de un mínimo local.

El objetivo del algoritmo es el de obtener los bordes de los objetos (*edge*) y no segmentar como tal. Por lo tanto, lo que se intenta buscar son las crestas o líneas divisorias.

El algoritmo consiste en inundar las cuencas de manera uniforme, de tal forma que si el agua de dos cuencas se va a mezclar se coloca una **presa** para evitar que se mezcle. Una vez inundada la imagen lo único que nos quedan son esas presas que representan los contornos de los objetos presentes en la imagen. Se puede ver este esquema en la Ilustración 20.

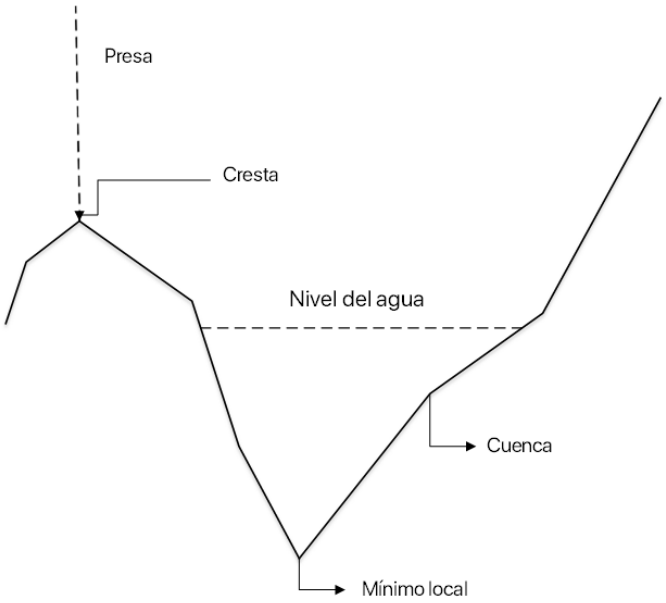


Ilustración 20. Representación del algoritmo de inundación para una imagen cualquiera, donde los desniveles aquí ilustrados vienen dados por la intensidad de la imagen.

En realidad, el algoritmo se complica mucho más debido a que las imágenes, incluidas las que se utilizan en este proyecto, suelen presentar ruido, generando así lo que se conoce como **sobresegmentación**: se extraen una cantidad excesiva de contornos que no representan en absoluto los bordes de los objetos que se buscan.

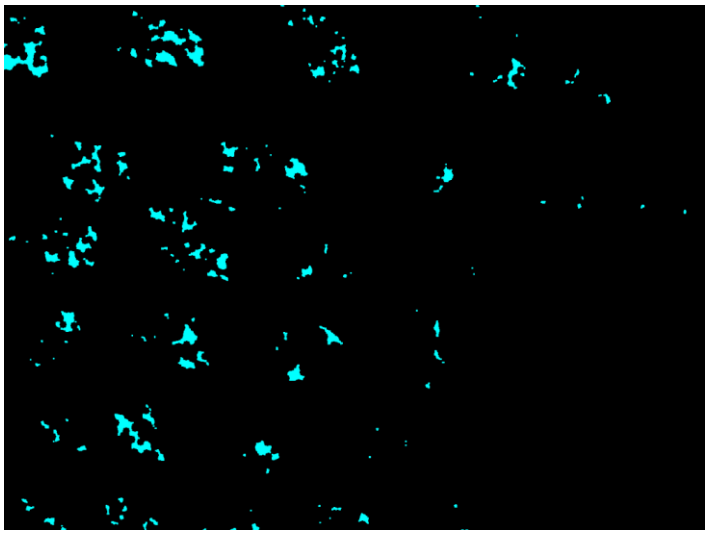


Ilustración 21. Ejemplo de marcadores de una imagen, zonas de donde se partirá en el algoritmo.

La solución será suavizar y utilizar **marcadores** (ver Ilustración 21), regiones rodeadas de puntos más altos donde podemos comenzar a inundar, en lugar de hacerlo en toda la imagen. Normalmente cuando se habla de marcadores también se relacionan con el *foreground*, que son aquellas zonas que estamos seguros que forman parte de objetos de interés. El *background* será justo lo contrario, zonas que sabemos que no pertenecen a nuestros objetos.

La implementación de todo lo que hemos descrito queda poco acotada, por lo que son muchas las formas que podemos encontrar de desarrollar este algoritmo. La secuencia de pasos que nosotros hemos establecido es la siguiente:

- Suavizado de la imagen para eliminar el ruido.
- Utilización de un filtro Laplaciano para resaltar los bordes de la imagen.
- Umbralización para calcular marcadores.
- Cálculo de la distancia de cada punto blanco (sería un objeto) respecto del fondo (negro). Recordemos que hemos umbralizado previamente.
- Se vuelve a umbralizar la imagen y se dilata para aumentar un poco el área de los marcadores.
- Se aplica el algoritmo de inundación sobre la imagen suavizada utilizando los marcadores anteriormente calculados. Se obtienen así los bordes o presas de la imagen.

En nuestro caso, al aplicar el algoritmo de inundación siguiendo la anterior secuencia de pasos, se obtienen resultados como el de la Ilustración 22.

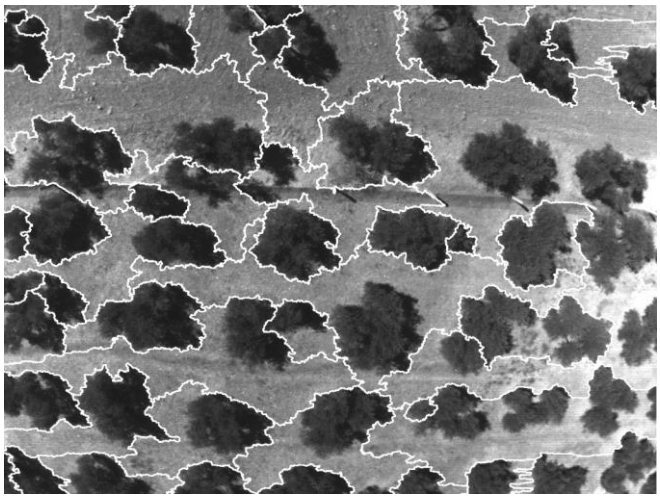


Ilustración 22. Obtención de contornos en la imagen multiespectral RED mediante el algoritmo de inundación.

Obviamente, el resultado deja mucho que desear, al igual que sucedía con el método alternativo a este, por lo que necesitamos otra metodología. Se formula así en la siguiente iteración un algoritmo para solventar toda la problemática encontrada hasta ahora.

2.2.1.1 - Storyboards

A través de un *storyboard* se muestra el flujo básico de operaciones a llevar a cabo, desde el punto de vista del usuario, hasta obtener un determinado resultado a través de las vistas de una aplicación simplificada. Nótese como el resultado de este proyecto es un prototipo, y por lo tanto la interfaz básica aquí desarrollada no dista en gran medida del resultado final.

El esquema de aplicación que se propone en la Ilustración 23 contiene sólo la función de extracción de contornos, dado que en este punto se desconocían algunos de los objetivos finales de este proyecto debido a que aún no se disponía de parte de la información de entrada finalmente considerada en este trabajo, como son las imágenes térmicas, por lo que se supuso que la interfaz que se incluye a continuación podría contener todas las operaciones necesarias.

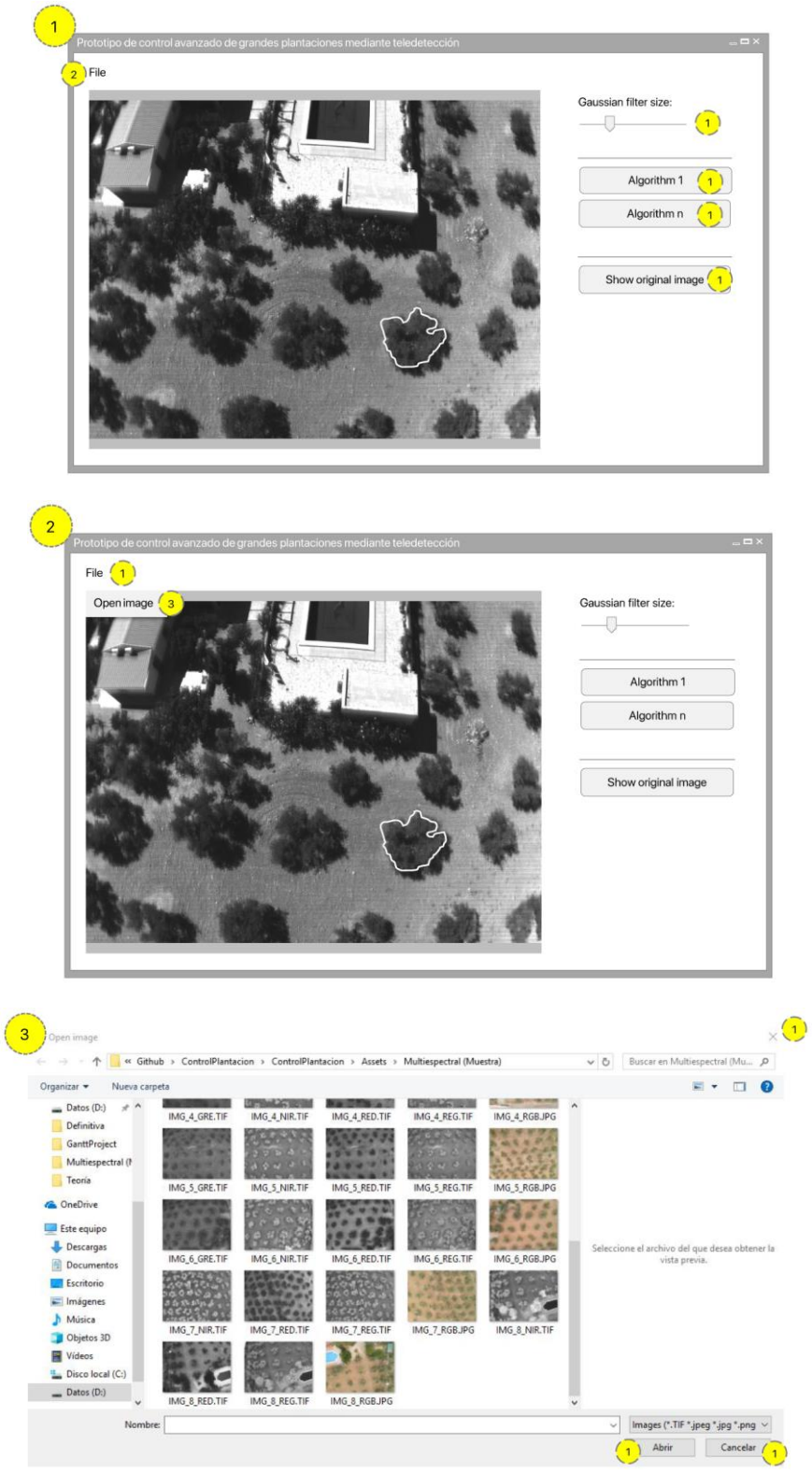


Ilustración 23. Storyboard de la aplicación desarrollada hasta la segunda iteración.

Nótese como múltiples controles conducen a la misma vista, aunque no necesariamente todos generan el mismo resultado.

2.2.2 - Tercera iteración

En esta iteración se propone el algoritmo final de la extracción de contornos, para lo cual nos podemos fijar en cada una de las imágenes multiespectrales anteriormente presentadas (ver Ilustración 3), especialmente en dos:

- **RED**: la vegetación tiene color oscuro, tanto el propio árbol como sus sombras, mientras que el suelo tiene un color más claro (mayor nivel de gris).
- **NIR**: la vegetación tiene un color muy similar al blanco, mientras que las sombras mantienen un color muy oscuro y el suelo también posee un color más claro que las sombras (como ya sucedía antes).

Todo esto que se acaba de comentar se corresponde realmente a la **función de reflectancia de la vegetación a lo largo del espectro**, $f(\delta)$, que se muestra en la Ilustración 25, donde dicha reflectancia alcanza uno de sus puntos más altos en la imagen NIR, y un mínimo local en la imagen RED, en torno a 0.7 µm. Se sabe a qué imagen pertenecen estos puntos porque se conoce el intervalo del espectro que captura cada imagen multiespectral, como bien muestra la Tabla 10.

Cabe destacar también que existen puntos más apropiados para estas tareas. Por ejemplo, la imagen NIR no contiene el máximo global de reflectancia, sin embargo, es la imagen que integra el máximo valor a nuestro alcance.

La intensidad que se da para cada tipo de elemento en las dos imágenes multiespectrales citadas se puede ver con claridad en la Ilustración 24.

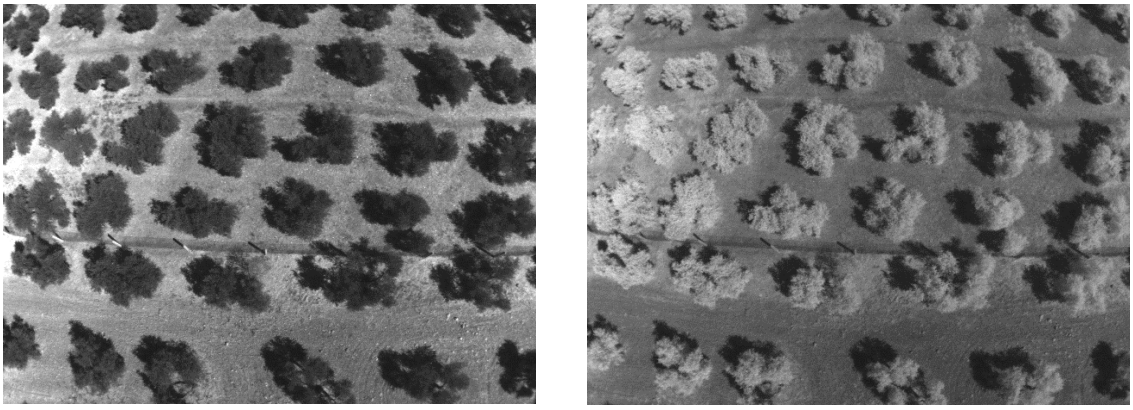


Ilustración 24. Comparativa de las bandas RED y NIR respectivamente.

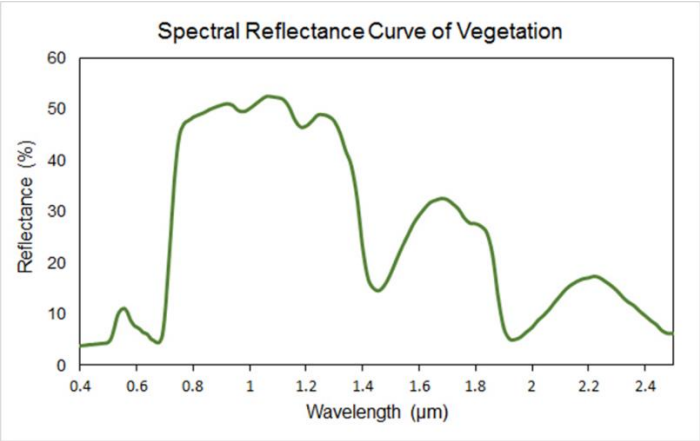


Ilustración 25. Reflectancia de la vegetación a lo largo del espectro (Humboldt State University, 2018).

Imagen multiespectral	Intervalo del espectro
Verde (GRE)	[510, 590] nm
Rojo (RED)	[620, 700] nm
Borde rojo (REG)	[725, 745] nm
Infrarrojo cercano (NIR)	[750, 830] nm

Tabla 10. Intervalo del espectro en el que cada imagen multiespectral opera (Parrot Sequoia, 2019).

Dadas las dos imágenes citadas, se propone una operación de diferencia entre ellas. Fijémonos en lo que pasa al realizar la operación NIR – RED con valores de intensidad aproximados:

- **Vegetación**: NIR (~255) – RED (~30).
- **Sombras**: NIR (~0) – RED (~0).
- **Suelo**: NIR (Nivel de gris intermedio) – RED (Nivel de gris intermedio similar al de NIR).

Lo que sucede es que se obtiene una imagen muy similar al resultado de una umbralización, donde **los árboles tendrán un color muy cercano al blanco y el resto de la imagen tendrá color negro**. Tiene además una ventaja: no sería necesario suavizar para eliminar el ruido de cierta parte de la imagen, dado que se elimina por completo tanto suelo como sombra. Aún así, se aplicará un pequeño

suavizado con el fin de hacer más uniforme el color de la vegetación. Asimismo, se debe aplicar un filtro no demasiado grande de este último tipo, dado que lo contrario podría provocar que se perdieran detalles de fronteras entre árboles, lo que resultaría en un polígono que integrara a varios de los ellos (no se distinguen individuos).

Si aplicamos la diferencia de imágenes se obtiene un resultado como el de la Ilustración 26.

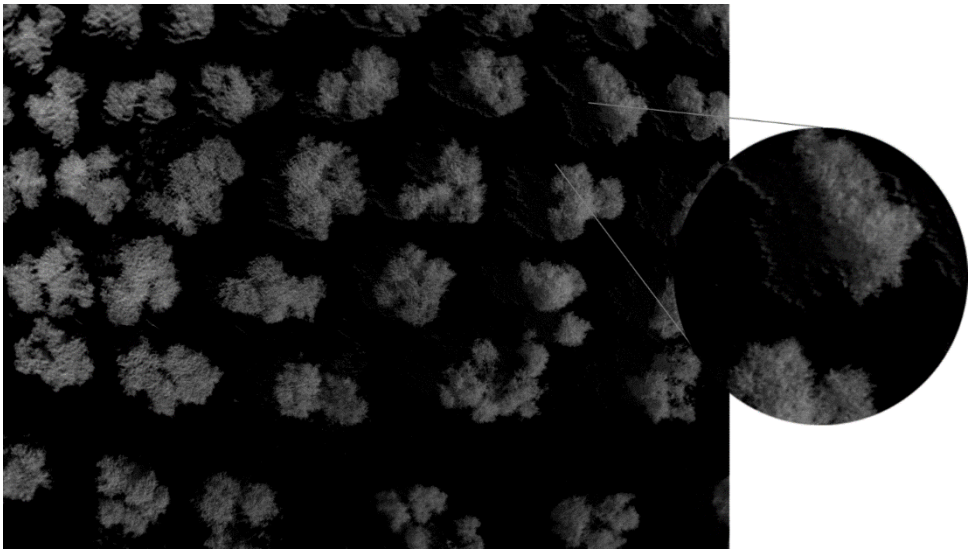


Ilustración 26. Resultado de la diferencia de imágenes multiespectrales NIR y RED.

Aunque el resultado obtenido tiene una calidad mayor que los resultados de la iteración anterior, se pueden ver finas líneas que trazan la sombra, y que se corresponden al hecho de que las imágenes multiespectrales no están registradas. De hecho, aún en esta iteración no se propone una solución para ese problema, por lo que todos los resultados posteriores se obtienen teniendo en cuenta este inconveniente. Si bien, la imagen diferencia que aquí se muestra será procesada hasta extraer los contornos, por lo que esos errores que se muestran en el detalle de la figura anterior desaparecen.

A partir de esta diferencia de imágenes se propone el algoritmo completo, que contiene las siguientes puntos:

1. **Operación de diferencia entre las bandas NIR y RED**, que nos devuelve una imagen en escala de grises como la que se acaba de mostrar.
2. **Suavizado de la imagen**: de esta manera, el color de los árboles se hace más uniforme (aún hay ruido dentro de la vegetación) y se eliminan

prácticamente por completo los errores derivados de la falta de alineación entre imágenes, como se muestra en la Ilustración 27.

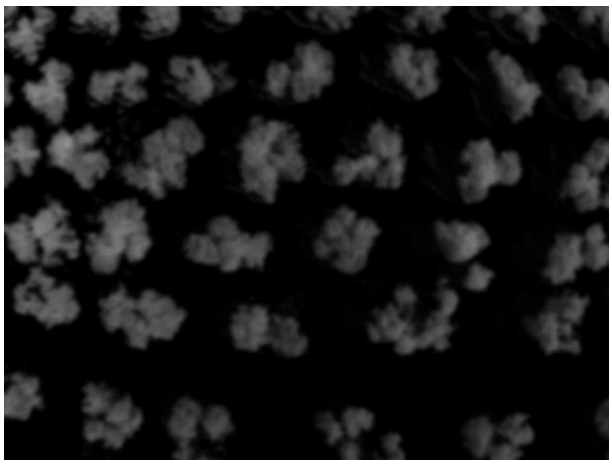


Ilustración 27. Suavizado de la diferencia entre las imágenes NIR y RED.

3. **Umbralización de la imagen** para distinguir aún más claramente los árboles del suelo. Es cierto que en la imagen anterior ya se distingue bastante bien un elemento de otro, pero la extracción de contornos posterior funciona mucho mejor si la imagen es binaria (blanco y negro). Lo que se obtiene por lo tanto es una imagen segmentada como la de la Ilustración 28.

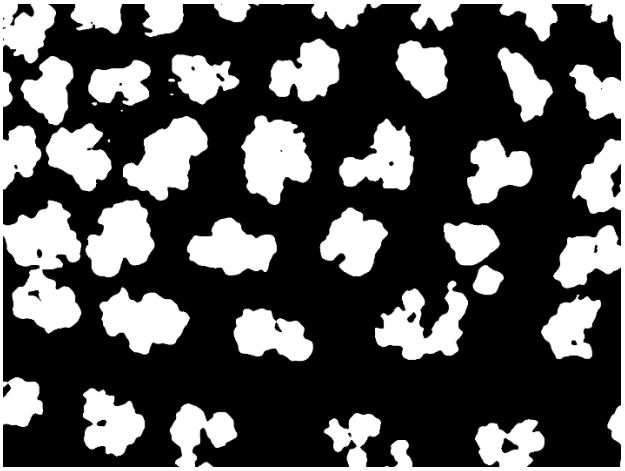


Ilustración 28. Umbralización de la imagen suavizada de la Ilustración 27.

4. **Extracción de contornos de la imagen binaria.** Como ya se comentó en la iteración anterior, este paso nos lo da ya implemetado OpenCV, de tal forma que nos devuelve dos elementos: un vector de contornos (que a su vez serán vectores de puntos) y un vector con cuatro valores para cada contorno. Entre estos cuatro valores, dependiendo de los *flags* que se

utilicen al calcular el contorno, se encuentra el índice del contorno padre en la jerarquía. Aquellos contornos que poseen padre en la jerarquía deben descartarse dado que se corresponden a huecos dentro de áreas de vegetación, que en ningún caso son de interés.

También podemos **filtrar contornos en función del área mínima**, como ya se proponía en el algoritmo inicial. Nótese que una imagen NIR registra la vegetación, ya sea un árbol o un simple arbusto, con intensidad similar, por lo que el resultado no siempre podría darnos sólo árboles al incluirse otros tipos de vegetación en la imagen multiespectral.

De acuerdo a nuestro objetivo, que es identificar individuos dentro de un conjunto de árboles representados en una imagen, queremos encontrar contornos de árboles completos, por lo que otro filtro que se aplica es el de eliminar aquellos polígonos que se encuentran seccionados. Esto es fácil de comprobar, ya que se buscan dentro del contorno segmentos que se correspondan al vector normalizado $\hat{v}\,(1,0)$ ó $\hat{u}\,(0,1)$, considerando siempre que se recorre un contorno en sentido antihorario y que estos vectores se corresponden a la diferencia entre puntos adyacentes situados en los extremos de la imagen.

Se establece por lo tanto que un contorno debe ser eliminado si alguno de sus puntos cumple la siguiente condición:

$$x_i - x_{i-1} = 0 \text{ ó } y_i - y_{i-1} = 0,$$

$$siempre\ que\ x_i \epsilon \{0, width - 1\} \vee y_i \epsilon \{0, height - 1\}$$

Si dibujamos los contornos resultantes sobre alguna de las bandas originales, como puede ser la RED, se obtiene la Ilustración 29.

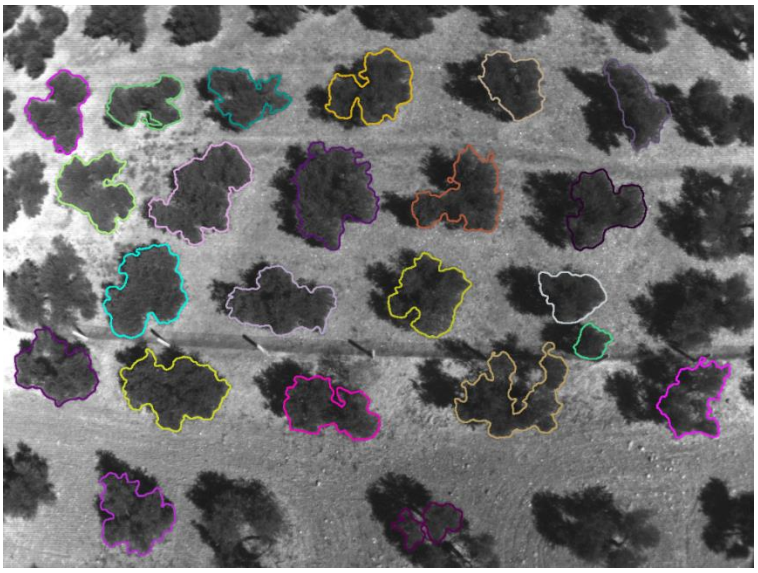


Ilustración 29. Resultado que se obtiene al dibujar los contornos extraídos sobre la imagen multiespectral RED.

En base a lo que hemos visto, el usuario tendrá hasta tres **grados de libertad para ajustar la extracción de contornos:**

- **Tamaño del núcleo que se utiliza al emborronar**: cuanto más grande más se difumina la imagen, y es posible que árboles que estén muy cercanos se confundan. Conforme se suaviza, el contorno de los árboles también se va simplificando, por lo tanto, cuanto más pequeño sea el núcleo (máscara), más puntos tendrán los contornos y normalmente serán más complejos. Aunque se le da al usuario la libertad de escoger el tamaño del núcleo de emborronamiento, se pueden configurar fácilmente como un valor estático en el intervalo [2, 4].
- **Área mínima para el filtrado de contornos**: es útil para eliminar vegetación detectada que no se corresponde necesariamente a árboles. De nuevo, considerando que el dron vuela a una altura similar durante todo el recorrido, se puede fijar fácilmente un área mínima teniendo en cuenta el tamaño de la imagen multiespectral. Dado que el área de dicha imagen es 1280x960 = 1228800 u^2, se puede establecer entonces un porcentaje de esta área como el mínimo a considerar para extraer los contornos. Por ejemplo, un valor de área mínima razonable (en función de los resultados obtenidos al utilizar los *datasets* disponibles) podría ser 4000u^2 (0.32%). En cualquiera de los casos, parece sensato también modificar este valor acorde

al cultivo que se está tratando, por ejemplo, podría ser un terreno con árboles de corta edad y por lo tanto, de menor área.

Se debe tener también en cuenta que el problema del cálculo de área de un polígono no es inmediato, pero de igual manera, ya nos lo da OpenCV implementado.

- **Valor de umbralización (*threshold*)**: es un factor clave en el resultado. Nótese como el color de la vegetación debe estar más cerca del blanco (255), por lo tanto, cuanto más se acerque el umbral al valor 0, se filtran menos puntos y por lo tanto aparecen más zonas coloreadas como si fueran vegetación. A diferencia de los dos valores anteriores, este otro no es tan automático debido a que depende de la intensidad de color resultante de la diferencia entre las imágenes NIR y RED, teniendo en cuenta que en muchas ocasiones los límites entre árboles cercanos pueden ser muy difusos, y una forma de diferenciarlos correctamente parte de este método.

En definitiva, todo el proceso que se sigue para obtener los contornos de los árboles a partir del conjunto de imágenes multiespectrales queda plasmado en la Ilustración 30.

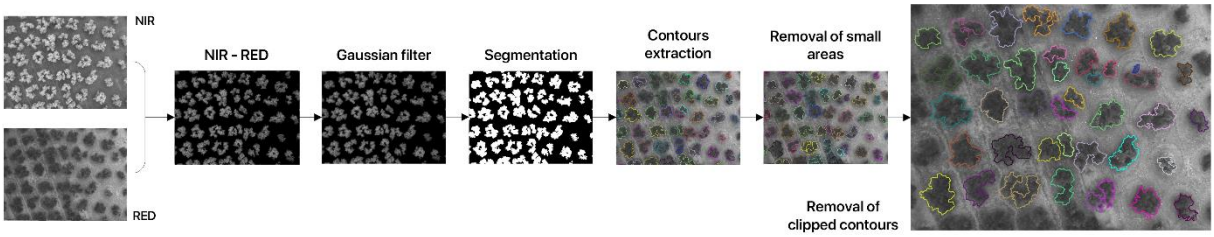


Ilustración 30. Proceso completo de extracción de contornos a partir de las imágenes multiespectrales RED y NIR.

Para este flujo aquí representado sí que se ha eliminado la distorsión y la falta de alineación entre imágenes, dado que se ha extraído la imagen completa del artículo creado a partir de este trabajo. Así, la diferencia entre las dos imágenes rectificadas ya no debería contener esos pequeños errores que sí aparecían en el proceso ilustrado durante esta iteración (ver Ilustración 31).

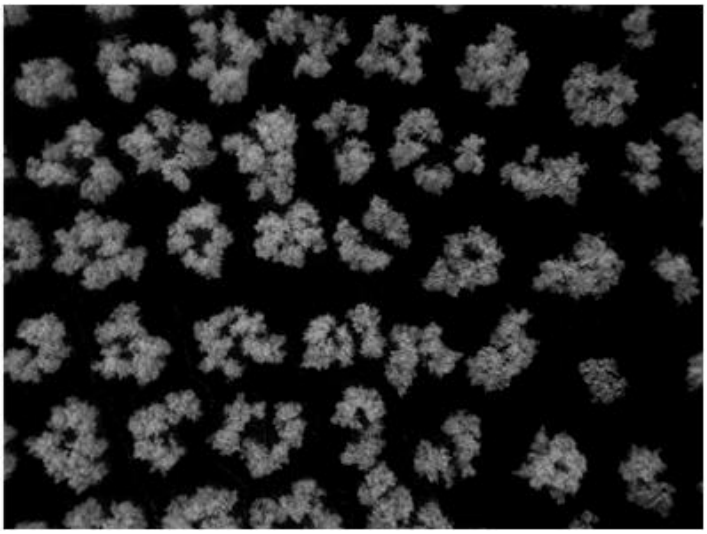


Ilustración 31. Diferencia entre imágenes multiespectrales rectificadas.

En forma de pseudocódigo, en lugar de representación gráfica, el algoritmo seguiría el flujo de operaciones que se muestra en la Ilustración 32.

Algorithm 1: Vegetation extraction from RED and NIR multispectral images

```
1  function ExtractVegetation
     (redImage, nirImage, blurKernelSize, threshold);
   Input  : two multispectral images, where both of them must be
            necessarily the RED and NIR channels, as well as a kernel size
            to blur the image, a threshold value in the range [0, 255] for the
            gray intensity and a minimum area value
   Output: a set of point arrays which form the vegetation contours
2  if redImage.size ≠ nirImage.size then
3  |  return null;
4  end
5  redImageGray ← grayscale(redImage);
6  nirImageGray ← grayscale(nirImage);
7  difference ← subtract(redImageGray, nirImageGray);
8  if blurKernelSize > 0 then
9  |  smoothDifference ← smoothImage(difference, blurKernelSize);
10 else
11 |  smoothDifference ← difference;
12 end
13 segmentation ← segmentateImage(smoothDifference, threshold);
14 contours ← findContours(hierarchy, segmentation);
15 while contour < contours.size do
16 |  if contours[contour].father ≠ null ∨ contours[contour].area ≤
   |   minArea then
17 |  |  contours.remove(contour − −);
18 |  end
19 end
20 contours ← deleteClippedContours(contours);
```

Ilustración 32. Pseudocódigo del flujo de operaciones necesarias para extraer los contornos de los árboles presentes en la escena.

2.2.2.1 - Diagramas de secuencia

Se exponen aquí los objetivos de un diagrama de secuencia por ser la primera iteración en la que se muestra uno. Dicho esto, el objetivo que perseguimos con estos esquemas es mostrar de forma gráfica la interacción entre entidades de la aplicación en un determinado orden. No se pretende documentar de forma exhaustiva todas las clases que intervienen en el proceso sino simplemente mostrar aquellos objetos principales que se comunican entre sí para alcanzar un resultado.

En primer lugar, se propone el flujo de operaciones a tener en cuenta a la hora de extraer los contornos a partir de las imágenes multiespectrales NIR y RED (Ilustración 33).

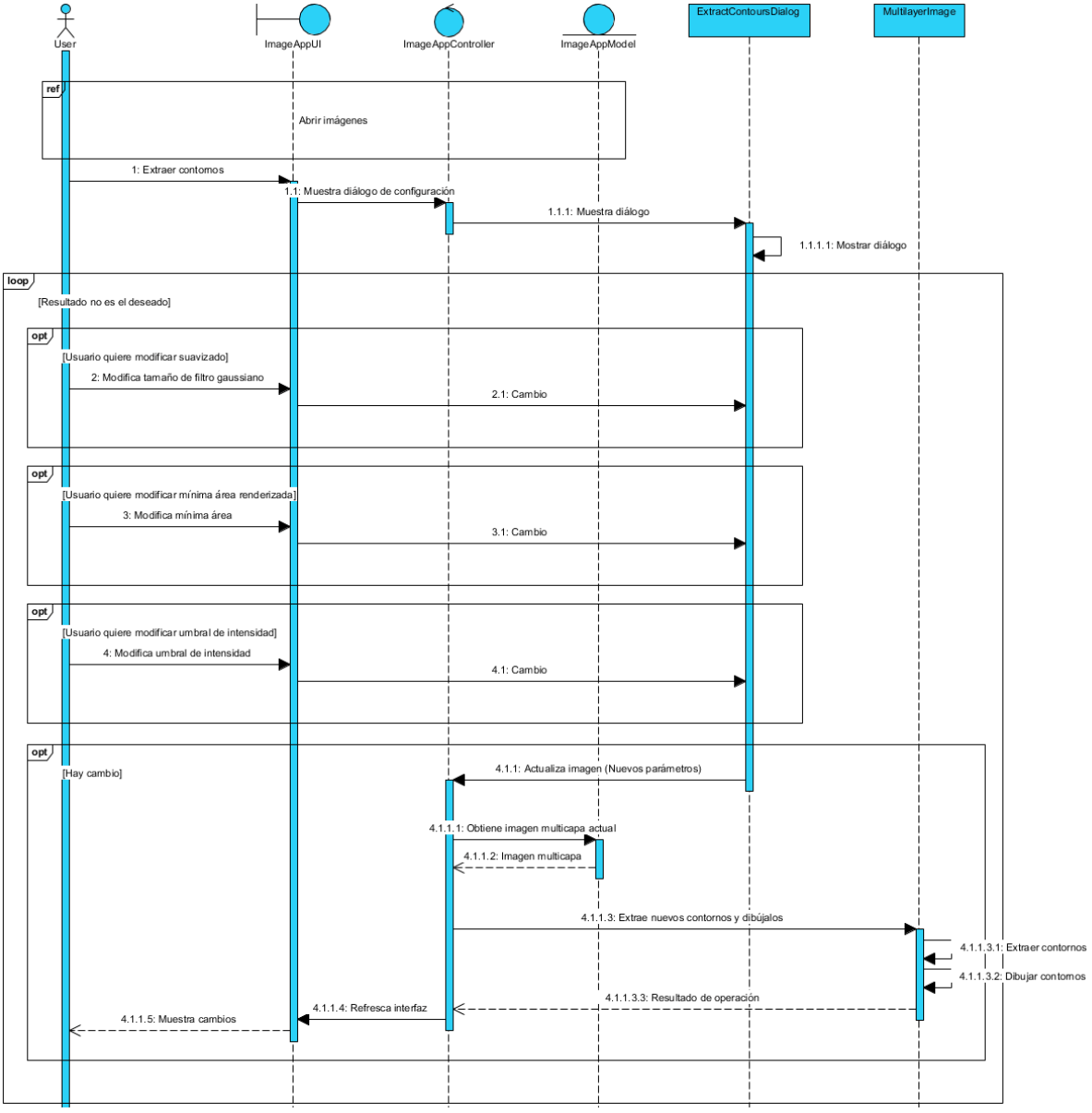


Ilustración 33. Comunicación entre entidades y actores para llevar a cabo la extracción de contornos de imágenes multiespectrales.

A la hora de extrapolar todo esto al proyecto de programación se debe tener en cuenta que no todo el procesamiento de la extracción de contornos se realiza en la clase *MultilayerImage*, sino que buena parte de la implementación recae en la propia entidad imagen, que no se muestra aquí por simplificación.

También se puede observar en este punto un conjunto de clases que muestran la base del patrón Modelo-Vista-Controlador, que se incorpora en iteraciones posteriores, aunque se representa aquí porque los diagramas de secuencia responden a la solución final.

Hasta ahora no se había citado el modelo multicapa (que se corresponde con la clase *MultilayerImage*) porque sólo hemos trabajado con dos imágenes, pero cuando se alineen y se rectifiquen todas sí que se podrá hablar de dicho modelo de formá más extendida. Por ahora sólo nos interesa saber que engloba todas las imágenes que captura el dispositivo Parrot Sequoia.

Por otro lado, el flujo de operaciones necesario para abrir imágenes multiespectrales es el que se representa en la Ilustración 34.

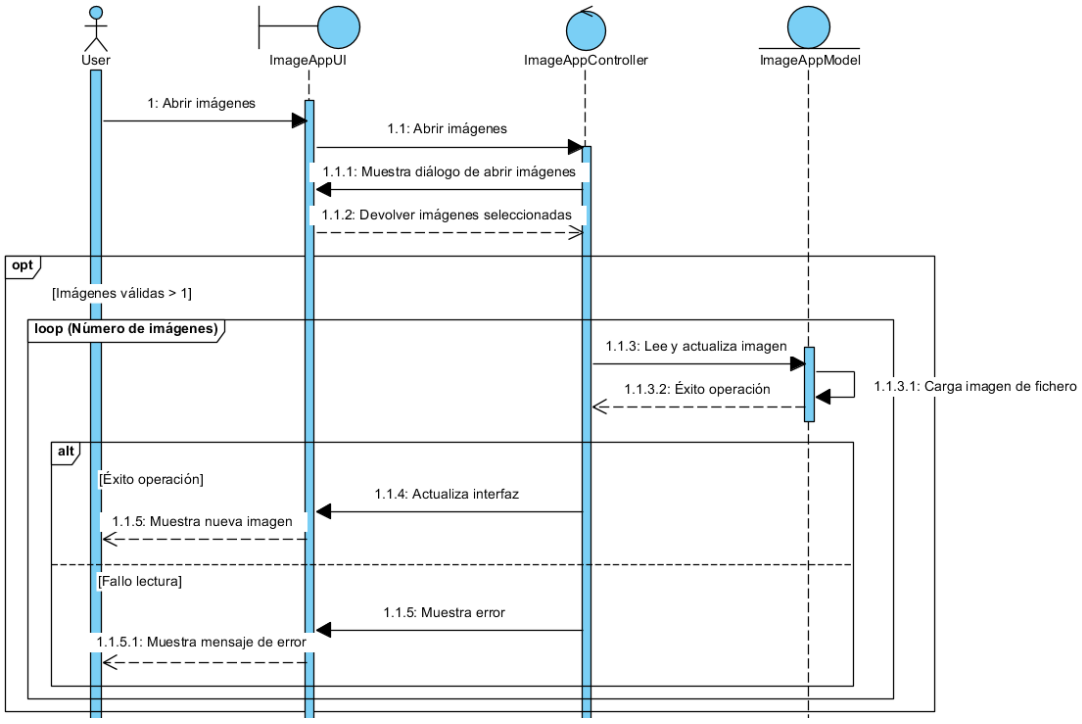


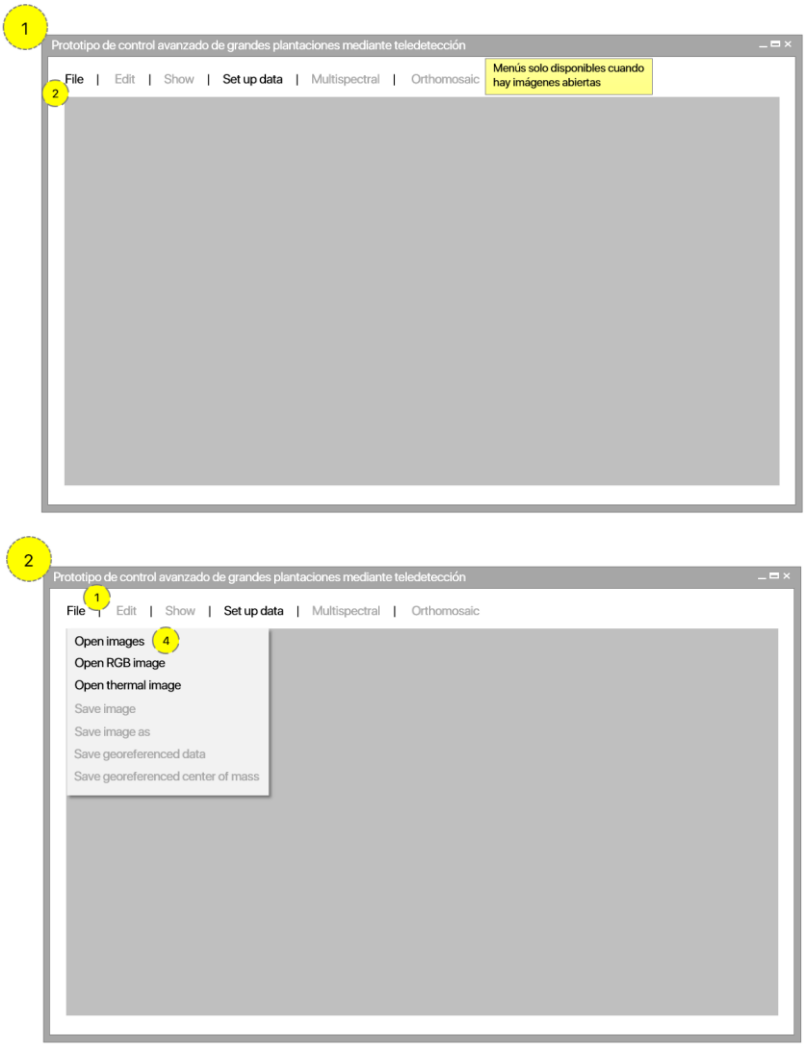
Ilustración 34. Intercambio de mensajes para llevar a cabo la operación de abrir imágenes multiespectrales.

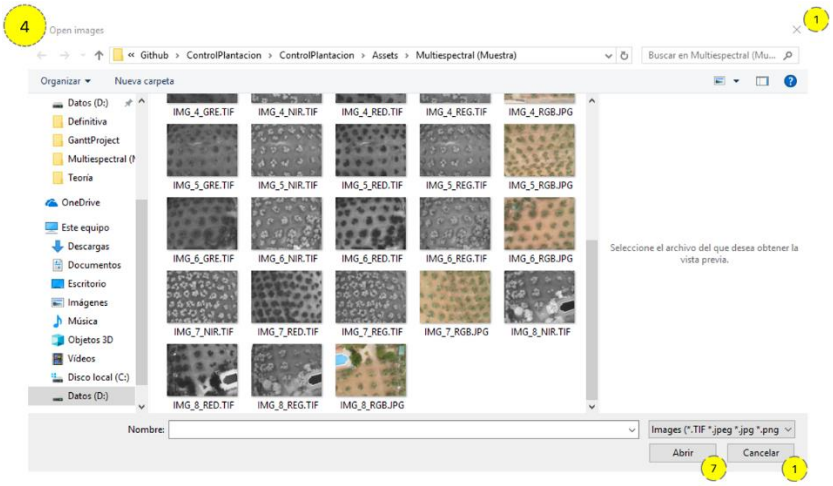
Nótese como en la Ilustración 33 se daba siempre por válida la extracción de contornos después de abrir imágenes, sin embargo, es posible que la salida de este último proceso fuera un número o tipo de imágenes insuficiente para llevar a cabo

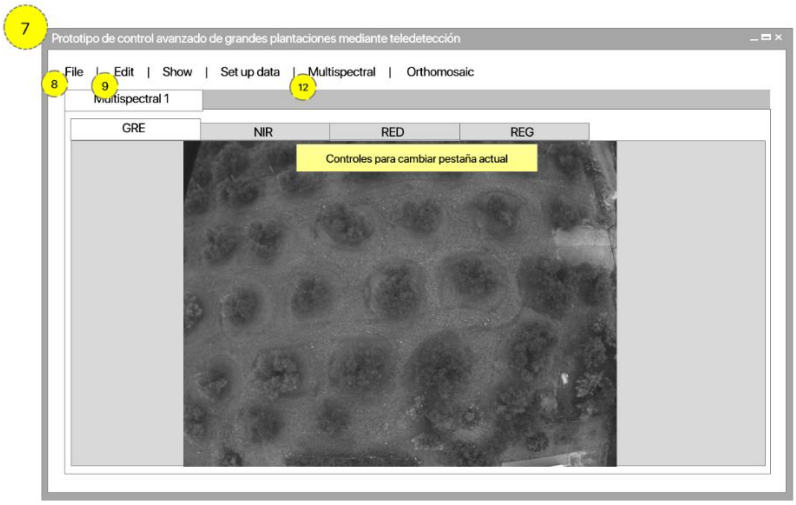
dicho proceso. En cualquiera de los casos, de forma poco intuitiva sí que se incluye esta comprobación, ya que al actualizar la interfaz de la aplicación se bloquearán ciertos controles del menú en base al estado de la base de datos y las imágenes almacenadas en la misma. Por ejemplo, si no hubiera imágenes suficientes para la extracción de contornos se bloquearía el control oportuno.

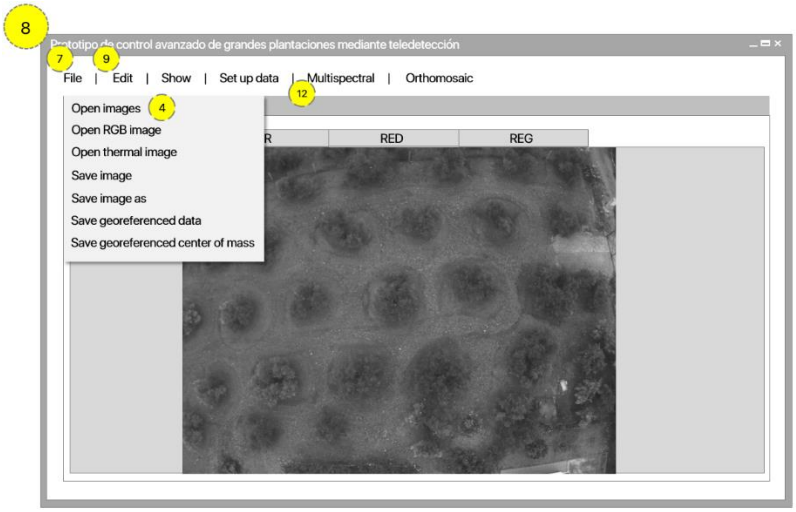
2.2.2.2 - Storyboards

Se muestran en este apartado, teniendo en cuenta el *storyboard* de la aplicación final, aquellas vistas necesarias para obtener el resultado de esta iteración, que vendría a ser la extracción de contornos de árboles. Se incluyen así las vistas que permiten abrir imágenes y visualizar el resultado en la Ilustración 35.




4
Open images
1
« Github › ControlPlantacion › ControlPlantacion › Assets › Multiespectral (Muestra)
Buscar en Multiespectral (Mu...
Organizar
Nueva carpeta
Datos (D:)
Definitiva
GanttProject
Multiespectral (
Teoria
OneDrive
Este equipo
Descargas
Documentos
Escritorio
Imágenes
Música
Objetos 3D
Vídeos
Disco local (C:)
Datos (D:)
IMG_4_GRE.TIF
IMG_4_NIR.TIF
IMG_4_RED.TIF
IMG_4_REG.TIF
IMG_4_RGB.JPG
IMG_5_GRE.TIF
IMG_5_NIR.TIF
IMG_5_RED.TIF
IMG_5_REG.TIF
IMG_5_RGB.JPG
IMG_6_GRE.TIF
IMG_6_NIR.TIF
IMG_6_RED.TIF
IMG_6_REG.TIF
IMG_6_RGB.JPG
IMG_7_NIR.TIF
IMG_7_RED.TIF
IMG_7_REG.TIF
IMG_7_RGB.JPG
IMG_8_NIR.TIF
IMG_8_RED.TIF
IMG_8_REG.TIF
IMG_8_RGB.JPG
Seleccione el archivo del que desea obtener la vista previa.
Nombre:
Images (*.TIF *.jpeg *.jpg *.png
Abrir
Cancelar
7
1


7
Prototipo de control avanzado de grandes plantaciones mediante teledetección
File
Edit
Show
Set up data
Multispectral
Orthomosaic
8
9
12
Multispectral 1
GRE
NIR
RED
REG
Controles para cambiar pestaña actual


8
Prototipo de control avanzado de grandes plantaciones mediante teledetección
7
9
File
Edit
Show
Set up data
Multispectral
Orthomosaic
12
Open images
4
Open RGB image
Open thermal image
Save image
Save image as
Save georeferenced data
Save georeferenced center of mass
RED
REG

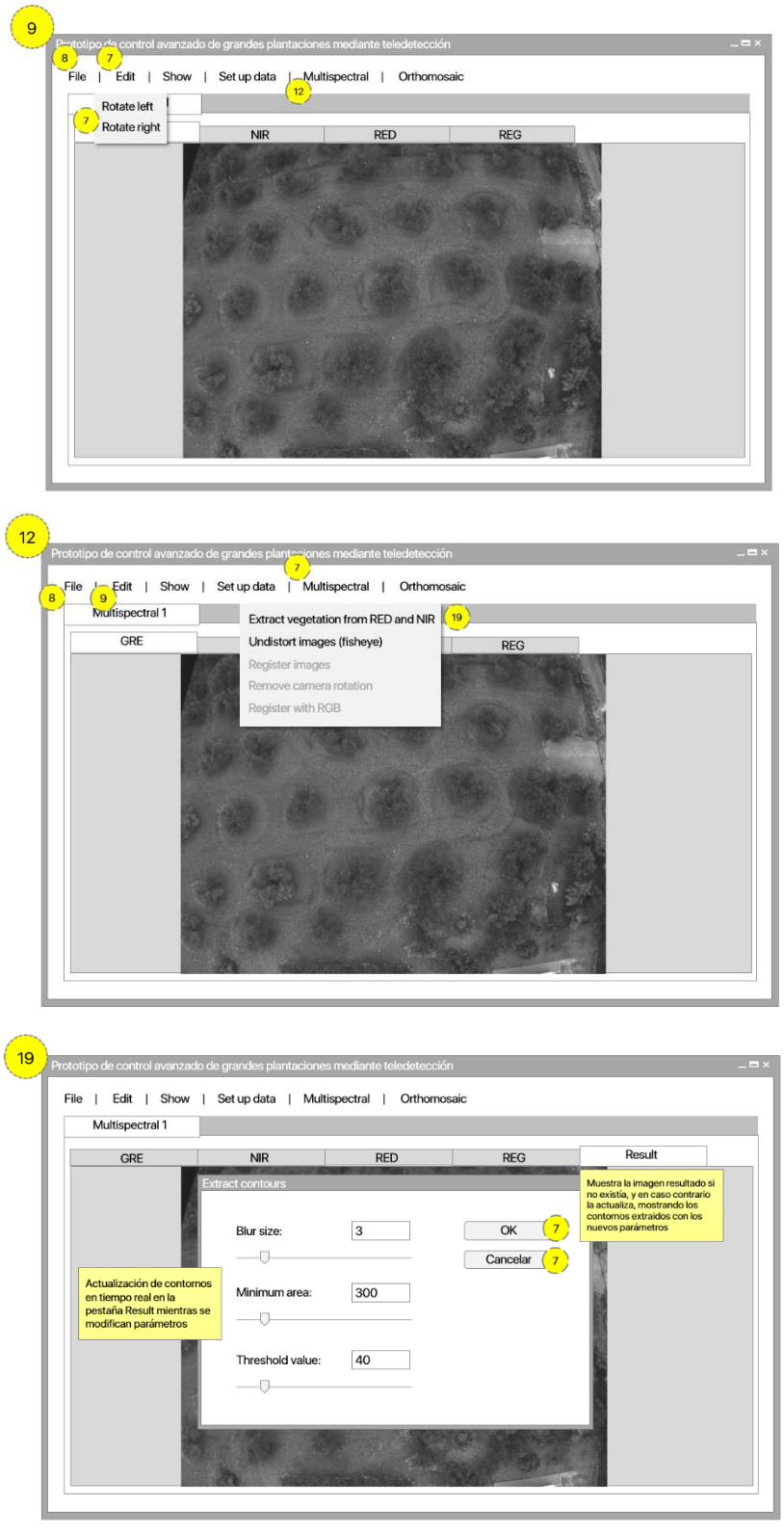


Ilustración 35. Secuencia de vistas necesarias para obtener los contornos a partir de los tres parámetros expuestos anteriormente.

2.2.3 - Cuarta iteración

Una vez se extraen los contornos que representan los árboles de una plantación, podemos optar por simplificar la polilínea obtenida. Nótese como a la hora de guardar los contornos extraídos en una base de datos, una política como esta nos puede ayudar a reducir el espacio que ocupa el conjunto de árboles, conservando aún una forma identificativa del árbol en cuestión. En cualquiera de los casos, se debe tener en cuenta que los polígonos resultantes no conservan la forma original, incrementando su área en todos los casos.

En iteraciones posteriores se siguen utilizando estos contornos, por lo que su simplificación reduce el tiempo de respuesta de los algoritmos, aunque podría introducir también un mayor error.

El primer tipo de simplificación considerado para un polígono es la **envolvente convexa** a partir de un conjunto de puntos P, que no son más que los vértices de las polilíneas que forman los contornos. La envolvente convexa se define como el polígono convexo más pequeño que contiene dichos puntos (Erickson, Convex Hulls, 2008):

- **Polígono**: región del plano delimitada por un conjunto de segmentos o aristas que forman un ciclo (es una forma cerrada). Un **vértice** será el punto donde se encuentren dos aristas.
- **Convexo**: un polígono se define como convexo si todos sus vértices lo son. A su vez, un vértice es convexo si el siguiente vértice, p_{i+1} (sentido antihorario), queda a la izquierda del segmento $p_{i-1}p_i$, y por lo tanto, el ángulo entre los segmentos $p_{i-1}p_i \; y \; p_i \; p_{i+1}$ resulta menor que π. Se deduce pues que para cualquier par de puntos p y q dentro del polígono, el segmento pq que forman queda dentro de dicho polígono (ver Ilustración 36).
- **Más pequeño**: cualquier subconjunto de la envolvente convexa deja fuera al menos a un punto del conjunto inicial P.

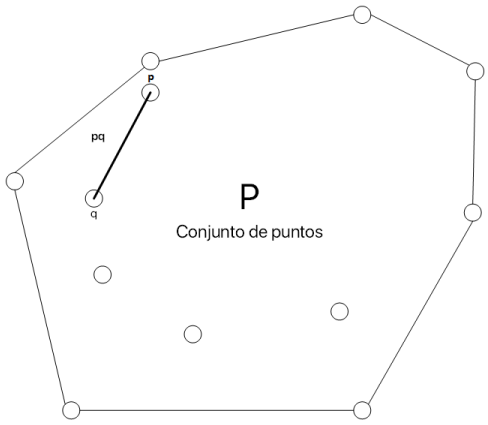


Ilustración 36. Envolvente convexa para un conjunto de puntos P. Véase como el segmento pq queda dentro de la envolvente.

Al igual que en nuestro trabajo, en el ámbito gráfico la envolvente convexa se utiliza para obtener representaciones aproximadas de un conjunto de puntos, de tal manera que nos puede ayudar a acelerar un proceso. También nos podría servir para agrupar objetos geométricos.

El problema de hallar las envolventes convexas es un problema común en geometría, y existen multitud de algoritmos para hallar la solución, entre los que se encuentran los siguientes:

- Jarvis march: es el más simple, de complejidad $O(nh)$, sea h el número de puntos de la envolvente convexa.
- Método de Graham: de complejidad computacional $O(n\ log\ n)$.
- Quickhull: su complejidad computacional es $O(n\ log\ n)$, aunque puede llegar a $O(n^2)$ en el peor de los casos.
- Divide y vencerás: también de complejidad $O(n\ log\ n)$.
- **Sklansky** (Sklansky, 1982), en concreto, el algoritmo de 1982 (había una versión previa con ciertos errores). Es el algoritmo que implementa OpenCV y el que se incorpora a este proyecto, sin embargo, se ha probado que el algoritmo es incorrecto para ciertos casos extremos.

El resultado de aplicar dicho algoritmo se puede ver en la Ilustración 37:

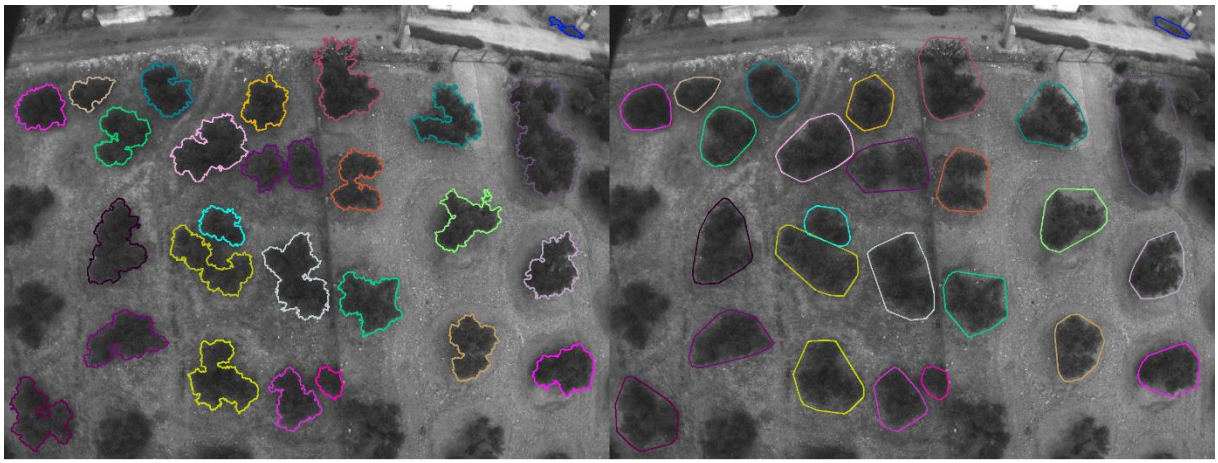


Ilustración 37. Comparativa entre los contornos originales de la vegetación y las envolventes convexas obtenidas a partir de los polígonos iniciales.

Nótese como el área de los contornos crece al simplificarlos, aunque no será un aumento tan grande como el del siguiente polígono aproximado: **hexágonos regulares**. Para esta simplificación del polígono se debe encontrar un centro que describa una circunferencia, de tal forma que la creación del hexágono se limita a la obtención de 6 puntos que se encuentran necesariamente en dicha circunferencia y que se obtienen a partir de la división del espacio completo, 360º, entre 6. Se debe hacer entonces un barrido desde α = 0º hasta α = 300º (inclusive), incrementando α en 60º cada iteración. Asimismo, el centro de la circunferencia se describe como el centro exacto de la caja que envuelve el polígono y que viene delimitada por las coordenadas mínimas y máximas en X e Y. Por lo tanto:

$$Centro_{Circunferencia} = (\frac{min_x + max_x}{2}, \frac{min_y + max_y}{2}) \quad (3)$$

Se obtiene así la Ilustración 38 a partir de los mismos parámetros de la escena previa.

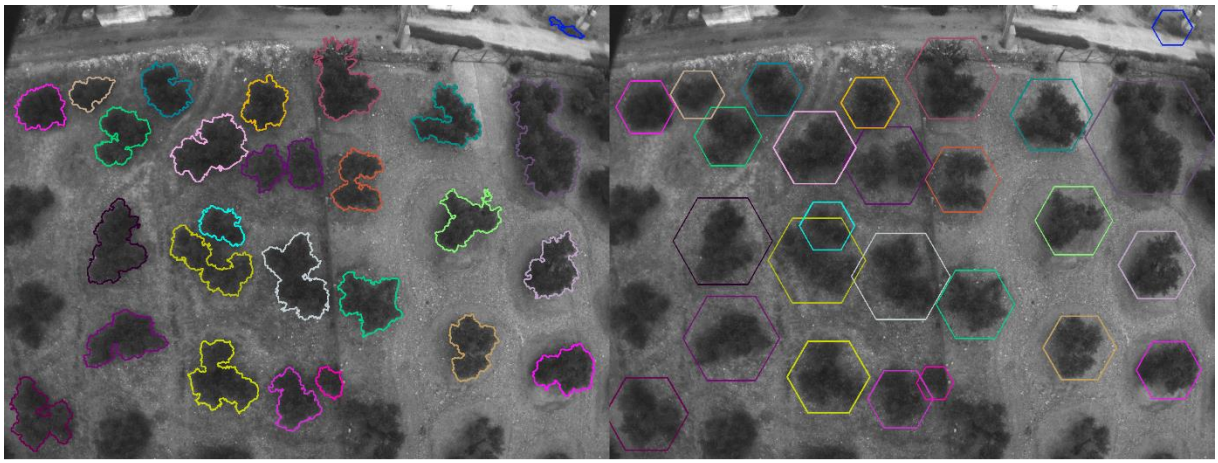


Ilustración 38. Comparativa entre los contornos originales de la vegetación y las envolventes hexagonales obtenidas a partir de los polígonos iniciales.

El último paso de esta iteración es obtener para cada uno de los contornos anteriores el **centro de masas**, que en este caso se corresponde también con el centroide si tenemos en cuenta que la masa del polígono se distribuye uniformemente en los vértices y no en la superficie completa. Al fin y al cabo, el objetivo de este algoritmo es obtener un punto representativo del árbol, por lo que se ha considerado la definición más simple del centroide, que es interpretar el polígono como una lista de puntos y no como una superficie.

El centroide de un polígono se define como su centro de masas (Stephens, 2014), y el centro de masas se define como la posición relativa donde se concentra toda la masa del sistema (desde un punto de vista inercial) (Shore, 2008). Se puede ver por lo tanto como ambas definiciones coinciden para una lista de vértices donde la masa o el peso de todos es el mismo (p. ej. m_i = 1), como es nuestro caso:

$$Centroide = \frac{\sum_{i=0}^{n} p_i}{n} \tag{4}$$

$$Centro\ de\ masas = \frac{\sum_{i=0}^{n} m_i * p_i}{M} \tag{5}$$

Sea M la suma total de las masas de todos los vértices y m_i el peso de un vértice en concreto.

Nótese como el centro de masas o el centroide no deben estar necesariamente dentro del polígono, por lo que habrá que tener esto en cuenta en etapas posteriores donde se intente triangular un polígono. Algunos ejemplos de obtención de estos puntos característicos son la Ilustración 39 y la Ilustración 40.

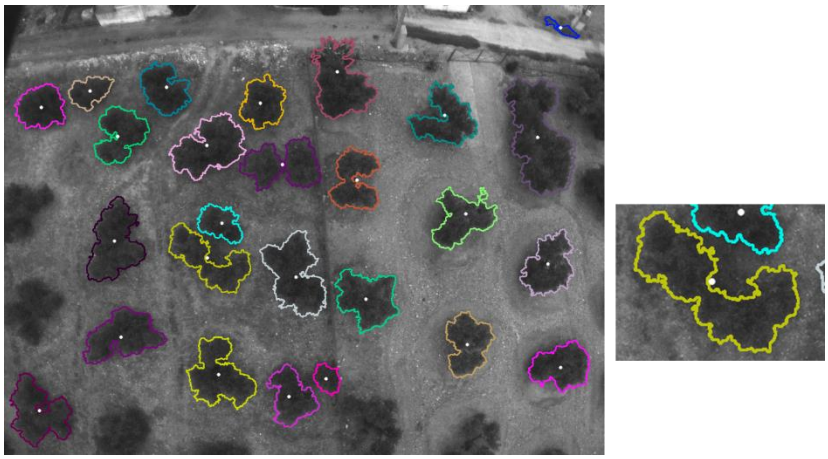


Ilustración 39. Representación de centros de masas en contornos originales, junto al detalle de un polígono cuyo centro de masas no se incluye dentro del mismo.

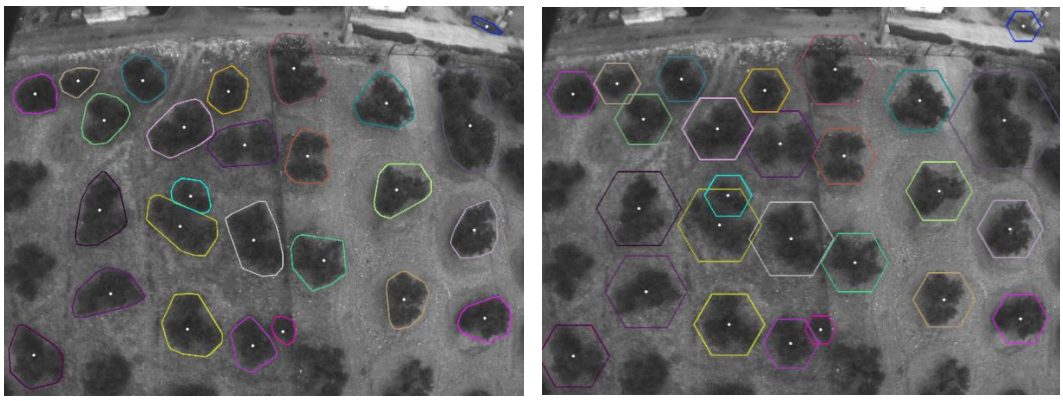


Ilustración 40. Centros de masas para el resto de contornos derivados de los originales.

2.2.3.1 - Diagramas de secuencia

El diagrama de secuencia parte del diagrama de la iteración anterior, añadiendo las posibles decisiones del usuario de modificar la visualización de los contornos obtenidos en cada ciclo, de tal forma que podríamos tener que recalcular el resultado (contornos de un tipo concreto y centros de masas). Se establece así un bucle mientras que el usuario desee modificar el resultado, de tal forma que la flexibilidad de la interacción radica en dos variables: qué tipo de contorno se muestra, ya sea simplificado o no, y si se deben incluir los centros de masas de los polígonos resultantes. Se especifica así el flujo de comunicación entre entidades en la Ilustración 41.

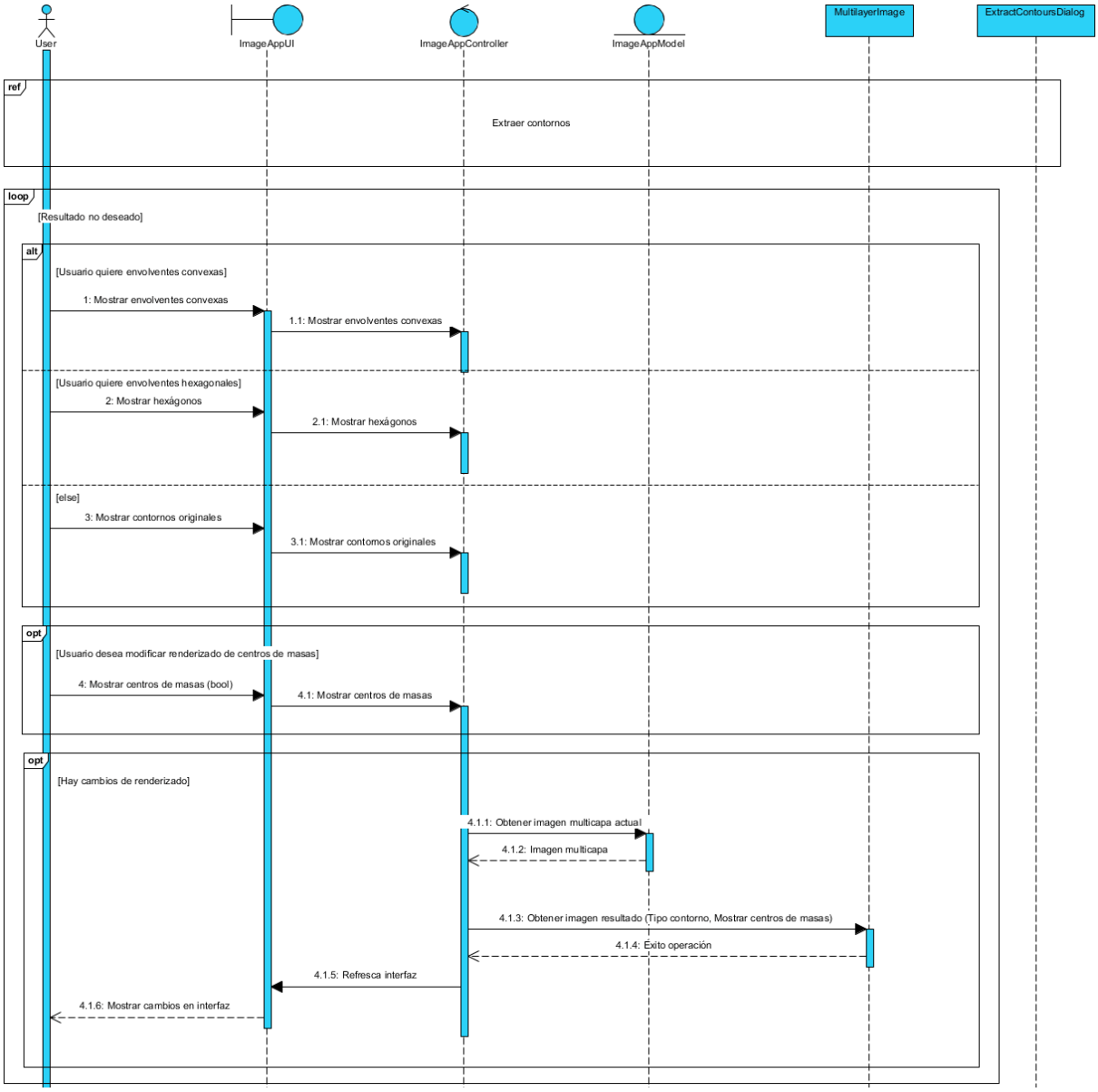


Ilustración 41. Diagrama de secuencia para la extracción de contornos y manipulación del resultado para su simplificación y visualización óptima.

2.2.3.2 - Storyboards

De nuevo, sobre la base de la aplicación que anteriormente se mostraba, sólo habrá que añadir aquella vista en la que se nos permita escoger el tipo de contorno y si se deben mostrar los centros de masas de cada forma, como se puede observar en la Ilustración 42.

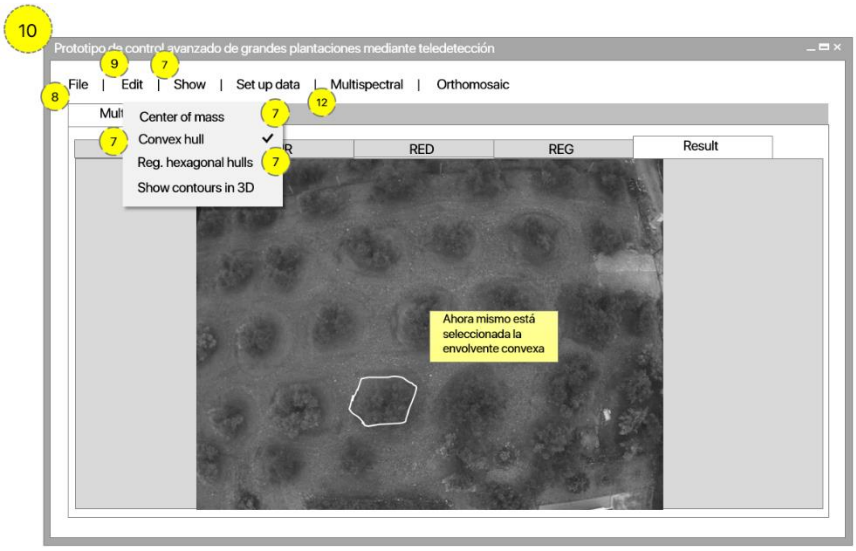
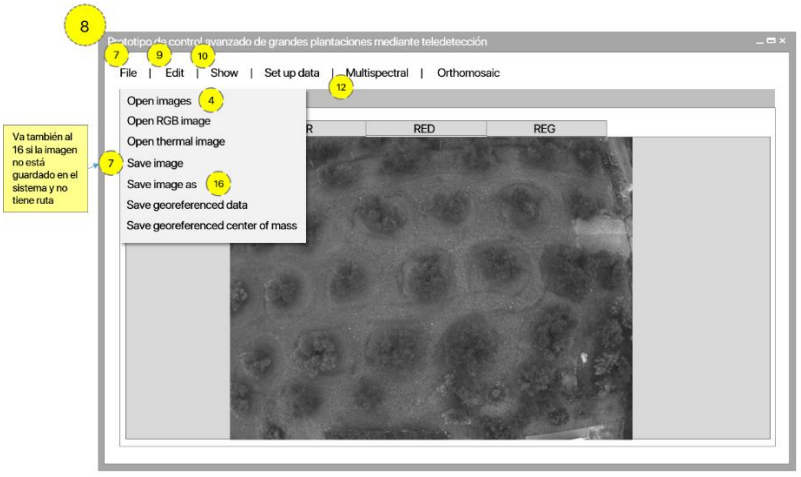


Ilustración 42. Vista que nos permite seleccionar cómo se renderizan los contornos resultantes y si se incluyen o no los centros de masas.

Para esta vista en concreto, se representa una escena donde la envolvente convexa está seleccionada y prueba de ello es que el *checkbox* de *Convex hull* está marcado. De esta forma, si se desmarcara dicha opción se volvería al estado inicial y se dibujarían los contornos originales. Nótese como sólo uno de los tipos de polígonos simplificados puede estar seleccionado a la vez.

Dado que ahora disponemos de una imagen resultado además de las multiespectrales iniciales, todas las operaciones de guardar imágenes en el sistema de ficheros cobran sentido (ver Ilustración 43).



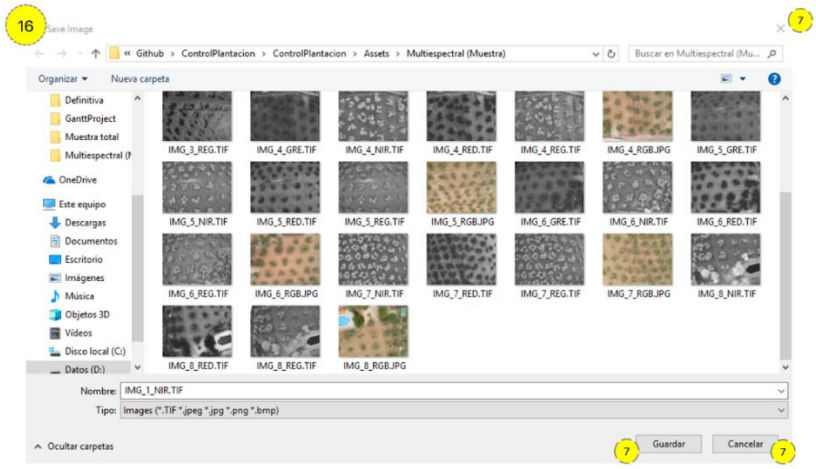


Ilustración 43. Vistas que representan las operaciones necesarias para el almacenamiento de imágenes en el sistema de ficheros.

2.2.4 - Quinta iteración

El objetivo principal de esta iteración es **eliminar la distorsión *fisheye*** que presentan las imágenes multiespectrales como resultado de las condiciones de captura. Sin embargo, para llevar a cabo esta operación necesitamos una serie de parámetros que se obtienen de los **metadatos de la imagen**, por lo que otro objetivo principal, que también nos ayudará en iteraciones posteriores, debe ser necesariamente el acceso a esos metadatos.

La integración de la librería de acceso a los metadatos pertenece más bien al apartado de Instalación y configuración del sistema, y el principal interés de la clase que accede a estos datos, *ImageMetadata*, se encuentra en el diagrama de clases que se presenta al final de todo este proceso de desarrollo.

Como bien sabemos, la cámara multiespectral que se ha empleado captura hasta 5 imágenes: 4 imágenes multiespectrales (RED, GRE, NIR Y REG) y una imagen RGB. A diferencia de esta última, que sigue un modelo de distorsión de perspectiva, las otras cuatro cámaras siguen un modelo de distorsión *fisheye* u ojo de pez. Este tipo de distorsión se genera debido al ángulo de visión de la lente, es cual es bastante elevado (61.9º x 2 = 123.8º), como resultado de una distancia focal muy pequeña (aunque también influyen las dimensiones del sensor) (Panotools, 2019). Nótese como al aumentar la distancia focal, el ángulo de visión se reduce, de tal forma que cada vez se captura un área menor pero también se eliminan este tipo de distorsiones (ver Ilustración 45).

Una forma muy sencilla de comprobar esta característica es observar imágenes donde aparezcan construcciones, ya que las líneas rectas pasan a deformarse, siguiendo una curva. Este fenómeno tiene especial intensidad en los extremos de la imagen más que en el centro (al fin y al cabo la estructura que se sigue es una circunferencia), y es por ello que en la Ilustración 44 se representa una línea claramente deformada en la parte izquierda, mientras que los muros situados en el centro se pueden seguir con una línea recta con algo menos de dificultad.

Ilustración 44. Imagen multiespectral RED donde se puede observar la distorsión fisheye que genera la lente.

Nótese como esta distorsión puede plantear un problema en la georreferenciación de los puntos de la imagen, donde se desea que la localización geográfica obtenida para un punto realmente se corresponda a su situación en el mundo real. El punto de partida para comenzar con el proceso de eliminar la distorsión de ojo de pez será la Ilustración 45.

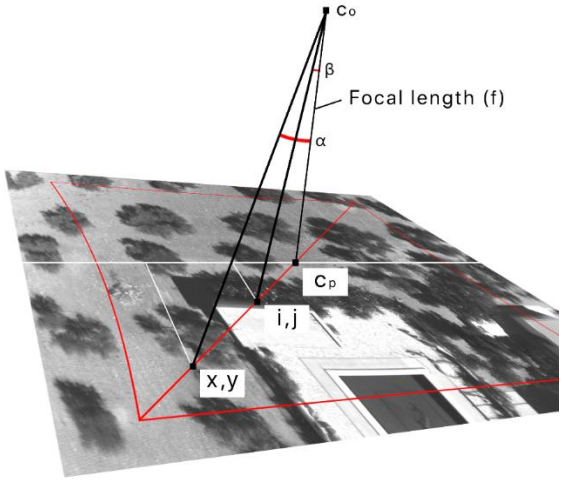


Ilustración 45. Esquema de eliminación de la distorsión de ojo de pez sobre una imagen multiespectral.

Se parte de la **distancia focal** (f), que es la distancia entre el centro óptico (c_o) y el punto principal de la imagen (c_p), cuyo valor para las lentes del dispositivo Parrot Sequoia que poseen un modelo de distorsión de ojo de pez es de 3.8 mm. Para cada punto $p_{x,y}$ de la imagen resultante habrá que calcular de qué punto de la imagen original se obtiene el nivel de intensidad. Para ello serán necesarias dos funciones, $f(x)$ y $f(y)$, que se expresan de forma conjunta en la siguiente matriz M:

$$M = \begin{pmatrix} f(0,0) & f(1,1) & \dots & f(0,w-1) \\ f(1,0) & f(1,1) & \dots & f(1,w-1) \\ \dots & \dots & \dots & \dots \\ f(h-1,0) & f(h-1,1) & \dots & f(h-1,w-1) \end{pmatrix}, \quad (6)$$

sea h la altura de la imagen y w la anchura de la misma.

La posición de la cual se obtiene el valor de intensidad en $p_{x,y}$ se nota en la Ilustración 45 como (i,j), donde estos dos valores representan una posición cualquiera dentro de un píxel de la imagen original, teniendo en cuenta que $i,j \in \mathbb{R}$.

Toda la gestión del nuevo *mapeo* a través de una matriz se puede llevar a cabo mediante la función de OpenCV *remap* (OpenCV, 2015), la cual tiene varios argumentos destacables:

- **Imagen inicial.** Es un objeto *cv::Mat*.
- **Imagen destino**. Es la matriz donde se guarda el resultado de aplicar la transformación M a la cada píxel de la imagen resultante.
- **Matrices de transformación en X y en Y**. Esto es importante ya que OpenCV divide lo que nosotros hemos llamado M en dos matrices, M_x y M_y.

$$M_x = \begin{pmatrix} f(0) & f(1) & \dots & f(w-1) \\ f(0) & f(1) & \dots & f(w-1) \\ \dots & \dots & \dots & \dots \\ f(0) & f(1) & \dots & f(w-1) \end{pmatrix}, M_y = \begin{pmatrix} f(0) & f(1) & \dots & f(0) \\ f(1) & f(1) & \dots & f(1) \\ \dots & \dots & \dots & \dots \\ f(h-1) & f(h-1) & \dots & f(h-1) \end{pmatrix}, \quad (7)$$

  de nuevo, sea h la altura de la imagen inicial y w la anchura de dicha imagen.
- **Método de interpolación**. Nótese como una posición $p_{x,y}$ de la imagen resultante toma el valor de intensidad de (i,j) en la imagen distorsionada, que no tiene por qué corresponderse con el centro de un píxel (valores enteros), sino que puede encontrarse en cualquier posición. Los valores de la imagen original se asocian a los píxeles, por lo que si se quiere hallar la intensidad de una posición cualquiera se debe utilizar alguna técnica de interpolación como la bilineal.

Es importante demostrar como este método de *mapeo* debe ser la solución y no una transformación para cada píxel de la imagen distorsionada. Si se hiciera esto último, cada píxel $p_{a,b}$ pasaría a ser $p_{c,d}$ en la imagen final, donde $c, d \in \mathbb{R}$, de tal forma que habría que redondear estos últimos valores. Por lo tanto, quedarían posiciones con valores de intensidad nulos, generando el patrón que se muestra en la Ilustración 46, que en el mejor de los casos se podría eliminar mediante alguna técnica de interpolación o un filtro gaussiano.

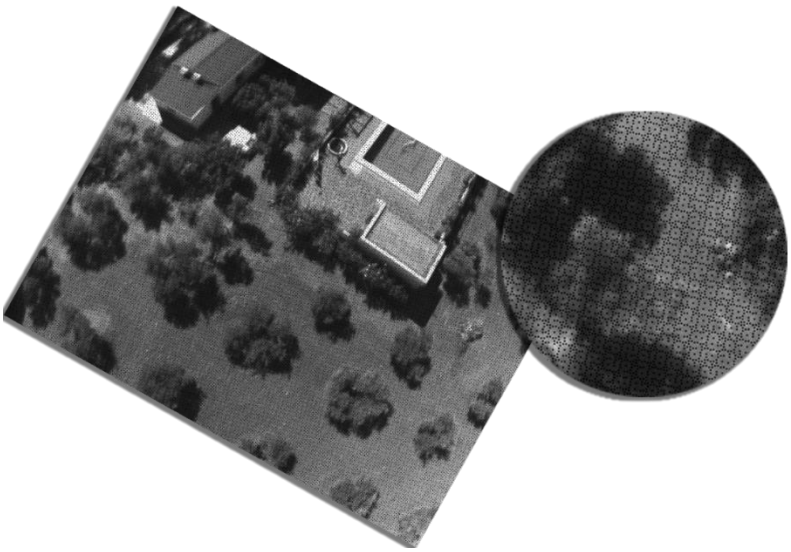


Ilustración 46. Detalle de una imagen a la que se le aplica una transformación cualquiera, generando un resultado donde se pueden ver píxeles de intensidad con el color por defecto, 0.

Para conseguir las dos funciones propuestas anteriormente, $f(x)$ y $f(y)$ (o bien M), además de la distancia focal y el punto principal de la imagen, necesitamos lo que se conocen como coeficientes de distorsión ($k_0, k_1, k_2\ y\ k_3$). Todos estos parámetros se obtienen de los metadatos de la imagen, por lo tanto, el módulo de inclusión de la librería Exiv2 es una parte fundamental de este proyecto, ya que a excepción de la primera fase de extracción de contornos, todo lo demás se fundamenta en estos datos.

Nuestro primer paso para eliminar la distorsion es extraer el ángulo α existente entre el segmento $c_o c_p$ y $c_o p_{x,y}$:

$$r\,(x,y) = \sqrt[2]{(x - c_{p_x})^2 + \,(y - c_{p_y})^2} \tag{8}$$

$$\alpha \;=\; 2 * \frac{1}{\pi} * \tan^{-1}\left(\frac{r}{f}\right) \tag{9}$$

Llegados a este punto tenemos tenemos el radio r, que iría desde el punto principal de la imagen hasta nuestro punto $p_{x,y}$. A partir de dicho módulo y la distancia focal f se puede hallar el ángulo entre los segmentos anteriormente citados. Sin embargo, ahora mismo tendríamos un ángulo que se mueve en el intervalo $[0, \frac{\pi}{2}]$, que se corresponde a su vez con el intervalo $[0, 90]$ grados. Por lo tanto, se debe multiplicar el ángulo hasta ahora obtenido por $\frac{2}{\pi}$ para normalizarlo y que se traslade al intervalo $[0, 1]$ (Pix4D, 2018).

El nuevo ángulo, que nos lleva hasta la posición (i, j), se puede calcular de la siguiente manera:

$$\beta = k_1 + k_2 * \alpha + k_3 * \alpha^2 + k_4 * \alpha^3 \quad (10)$$

De cara a conseguir esa posición en la imagen distorsionada, se deben tener en cuenta diversos aspectos:

- Es necesario desnormalizar el ángulo β obtenido al igual que normalizamos el ángulo original, α.
- Conocemos el módulo del nuevo radio, ya que $\tan\beta = r_2 \frac{1}{f}$, de tal forma que este nuevo radio será igual a $\tan\beta * f$.
- Entre las dos posiciones comentadas, (x, y) e (i, j), el punto principal, c_p, y la proyección en el eje X (también valdría para Y) de los dos primeros puntos citados, (x, a) e (i, b), se establece una relación de triángulos similares que nos conduce hasta la siguiente formulación:

$$\frac{x - c_{p_x}}{r_1} = \frac{i - c_{p_x}}{r_2} \qquad \frac{y - c_{p_y}}{r_1} = \frac{j - c_{p_y}}{r_2} \quad (11)$$

Los triángulos similares mantienen la forma, y por lo tanto las proporciones, pero la escala cambia entre ellos. Todas estas relaciones se pueden ver en la Ilustración 47.

Ilustración 47. Representación de los triángulos similares que se generan cuando se muestra una posición de la imagen final y el punto de la imagen distorsionada de donde se obtiene su valor de intensidad.

A modo de ejemplo, se ilustra mediante un rectángulo deformado (estirado a partir de las esquinas) el área que se incluye en la imagen final.

Finalmente, si lo que desea hallar es i nos basta con aislar dicha variable en las fórmulas antes expuestas y sustituir r_2 por $\tan\beta * f$. Esto mismo es aplicable a la altura del punto resultante, j, lo que nos lleva hasta las siguientes ecuaciones:

$$i = \frac{\pi * f * \tan\beta * (x - c_{p_x})}{r_1 * 2} + c_{p_x} \quad (12)$$

$$j = \frac{\pi * f * \tan\beta * (y - c_{p_y})}{r_1 * 2} + c_{p_y} \quad (13)$$

La matriz M que anteriormente hemos visto toma estos valores para cada punto de la imagen, y como ya hemos comentado, se puede aplicar directamente con la función *cv::remap*. El resultado de este proceso de eliminación de la distorsión sobre aquella primera imagen donde la estructura de la casa se deformaba se puede ver en la Ilustración 48.

Ilustración 48. Imagen sin distorsión fisheye tras aplicar el algoritmo que se acaba de exponer.

Otra forma de seguir este algoritmo de eliminación de la distorsión *fisheye* es mediante el pseudocódigo de la Ilustración 49.

Algorithm 2: Undistortion algorithm to remove this effect from an image captured by a camera that follows the fisheye model

```
function Undistort (image);
  Input  : any multispectral image
  Output: undistorted image with the same dimensions than the original
          one
if removedDistortion then
    return null;
end
if !metadataAvailable then
    return null;
end
if !metadataAvailable.followsFisheyeModel then
    return null;
end

f ← metadata.FocalLength;
c_x, c_y ← metadata.PrincipalPoint;
distCoeff ← metadata.DistortionCoefficients;
transformation_x, transformation_y ←
    Matrix(image.width, image.height);

while col < image.width do
    while row < image.col do
        cellRemap ← getUndistortValue(col, row, c_x, c_y, f, distCoeff);
        transformation_x[col, image.height − 1 − row] ← cellRemap.x;
        transformation_y[col, image.height − 1 − row] ← cellRemap.y;
    end
end

undistorted ← remap(image, transformation_x, transformation_y);
```

```
function getUndistortionValue (col, row, c_x, c_y, f, distCoeff);
x ← x − c_x;
y ← y − c_y;
radius ← √(x² + y²);
alpha ← 2 ∗ arctan(radius/f)/π;
beta ← distCoeff[0] + distCoeff[1] ∗ alpha + distCoeff[2] ∗ alpha² +
    distCoeff[3] ∗ alpha³;
new_x ← π ∗ f ∗ x ∗ tan(β)/(radius ∗ 2);
new_y ← π ∗ f ∗ y ∗ tan(β)/(radius ∗ 2);

return (new_x + c_x + new_y + c_y);
```

Ilustración 49. Pseucódigo del algoritmo de eliminación de la distorsión fisheye.

Aprovecharemos para cerrar este apartado de eliminación de la distorsión *fisheye* con la Tabla 11, donde se listan todos aquellos metadatos necesarios para solventar este problema, con sus respectivas etiquetas de acceso (independientes de la librería propuesta) y ciertas transformaciones necesarias para poder utilizarlos en las ecuaciones antes citadas:

Tipo de metadato	Etiqueta de acceso	Descripción
Distancia focal	Exif.Photo.FocalLength	Distancia focal en milímetros.
Coeficientes de distorsión	Xmp.Camera.FisheyePolynomial	Cadena con cuatro valores flotantes separados por una coma.
Coeficientes de matriz afín	Xmp.Camera.FisheyeAffineMatrix	Cadena con cuatro valores flotantes separados por una coma. Nos permite hallar la distancia focal en píxeles: $Focal\ length = \frac{2 * affine[0]}{\pi}$
Punto principal	Xmp.Camera.PrincipalPoint	Cadena con dos valores flotantes separados por una coma.
Modelo de distorsión	Xmp.Camera.ModelType	Cadena que indica el modelo de distorsión de la imagen: *fisheye*, *projection*, etc. Si no sigue el modelo *fisheye*, no será necesario eliminar su distorsión como hemos visto.

Tabla 11. Metadatos necesarios para eliminar la distorsión fisheye.

2.2.4.1 - Diagramas de secuencia

El diagrama de secuencia de la Ilustración 50 refleja la comunicación necesaria entre entidades para solventar el problema propuesto en esta iteración:

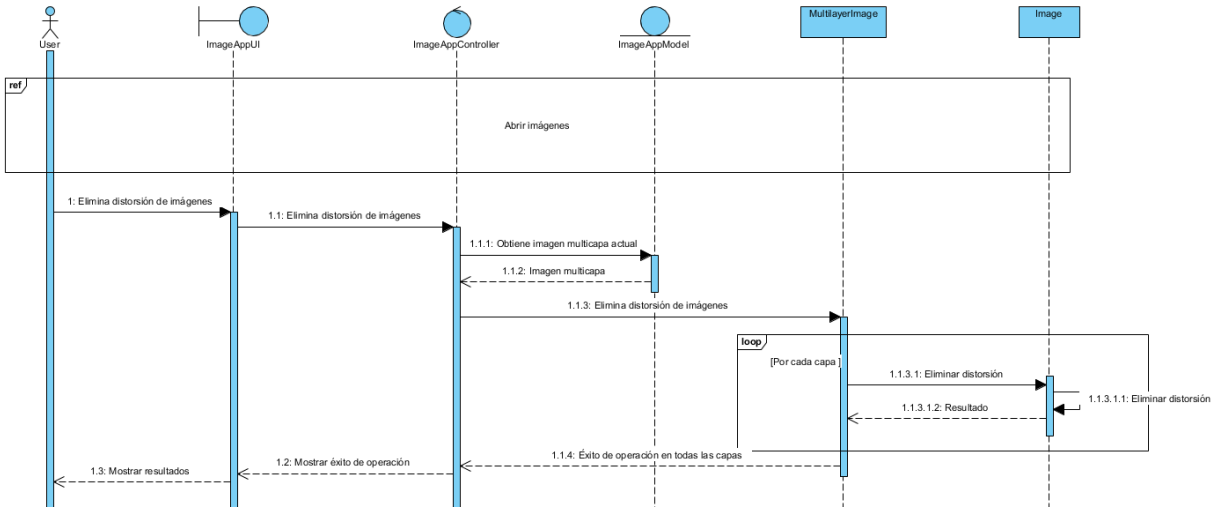
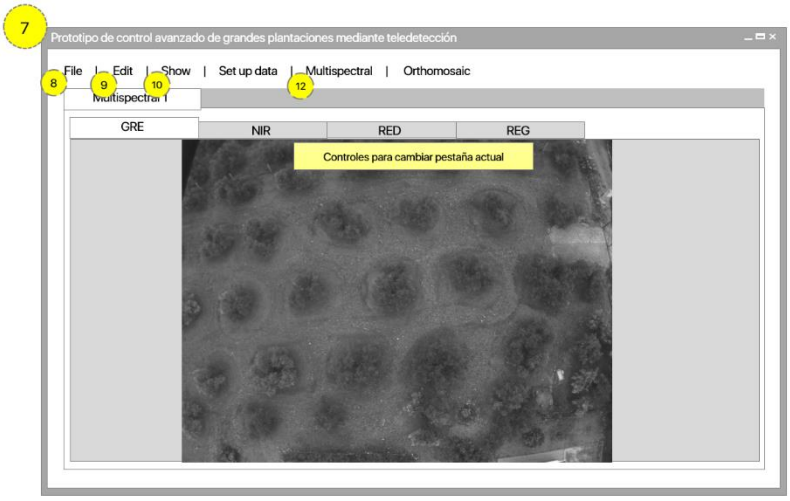


Ilustración 50. Diagrama de secuencia para la acción de eliminar la distorsión fisheye de las imágenes multiespectrales.

Nótese como el usuario no tendrá la opción de eliminar la distorsión de una imagen concreta sino que debe aplicar dicha operación al modelo multicapa en su totalidad. También se introduce aquí por primera vez la clase *Image*, que ya intervenía en la implementación de operaciones de iteraciones anteriores, aunque no se mostraba para simplificar el diagrama a tratar.

2.2.4.2 - Storyboards

Desde el punto de vista del boceto de la aplicación, la eliminación de la distorsión se incluye como una operación más dentro de un conjunto de opciones asociadas a las imágenes multiespectrales (menú *Multispectral*). Dado que el *storyboard* desarrollado representa la aplicación final, en la Ilustración 51 se muestran controles que pertenecen a operaciones que aún no se han expuesto.



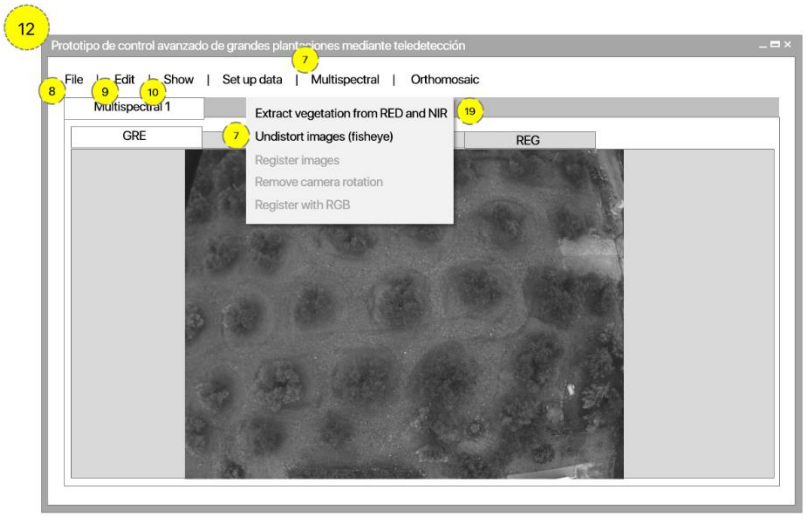


Ilustración 51. Vistas de la aplicación con las que interactúa el usuario para eliminar la distorsión fisheye.

2.2.5 - Sexta iteración

El objetivo de la sexta iteración, una vez se ha añadido un número aceptable de funcionalidades a la aplicación, es comprobar el esquema de clases desarrollado hasta ese punto. Más concretamente, se utiliza esta iteración para ceñirse al patrón **Modelo-Vista-Controlador**, y por lo tanto, los cambios que se produzcan aquí se verán más bien reflejados en el diagrama de clases que se muestre tras completar todas las iteraciones.

En cualquiera de los casos, se hará un breve repaso del patrón y se propondrá un esquema adaptado a esta aplicación.

De forma básica, el patrón Modelo-Vista-Controlador delimita qué tipo de entidades podemos encontrar en la aplicación (Modelo, Vista y Controlador) y cómo interactúan entre ellas. Este segundo concepto de interacción normalmente se adecúa a la aplicación que se va a desarrollar, y como prueba de ello es que pueden encontrarse múltiples esquemas de comunicación: entidades modelo que se comunican con vistas directamente (Eric Freeman, 2009), esquemas en los que esto no sucede (Apple Developer, 2018), etc.

El modelo de comunicación que se ha seguido al desarrollar esta aplicación es el que se representa en la Ilustración 52.

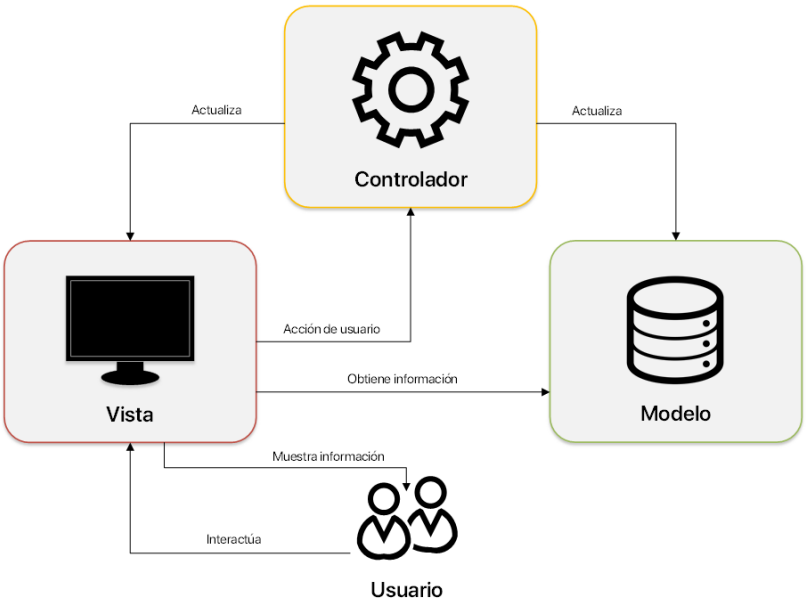


Ilustración 52. Esquema de comunicación entre entidades del modelo MVC en nuestra aplicación.

El papel de las tres entidades consideradas es el siguiente:

- **Vista**: se incluyen aquí tanto las clases de Qt relacionadas con la visualización como aquellas clases propias más elaboradas que integran a su vez una serie de controles y otros elementos tales como mensajes de texto. Normalmente, estas clases propias también extienden el comportamiento (herencia) de alguna clase básica de Qt.

- **Modelo**: instancia cuya principal función es almacenar elementos necesarios en la aplicación y garantizar el acceso a los mismos. Por ejemplo, podrían ser imágenes multiespectrales o variables básicas para otros procesos que el usuario puede modificar en la interfaz. Nunca se comunica por iniciativa propia con alguna de las otras entidades, simplemente espera peticiones: actualiza imagen, devuelve imagen, etc.

- **Controlador**: recibe los eventos de usuario y realiza las operaciones oportunas a partir de los datos que se almacenan en el modelo. Los eventos se notifican al controlador mediante el mecanismo de *signal-slots* que implementa Qt, y que claramente hace partícipe a la interfaz en esta operación (las señales proceden de los controles). Una vez se recibe un evento, el flujo básico será pedir información a entidades de la aplicación tales como el modelo o incluso la vista (p. ej. índice de pestaña abierta),

realizar las operaciones necesarias con dichos datos y actualizar de nuevo la interfaz para mostrar los cambios que se han producido al usuario.

Además de estas tres entidades, hay otras tantas clases que difícilmente se podrían integrar dentro de alguna de estas. Por ejemplo, una clase *MultispectralImage* que almacena imágenes y que opera con ellas. En el mejor de los casos se podrían introducir dentro del modelo, ya que al fin y al cabo soportan su trabajo.

Todo esto que aquí se ha comentado nos conduce a implementar una estructura de clases que ya se puede observar en apartados anteriores a través de los diagramas de secuencia. Un último detalle de este esquema MVC se asocia al tamaño y a los diversos módulos que integra esta aplicación. Desde un comienzo se hace una clara distinción entre aplicación relativa al procesamiento de imágenes y procesamiento de información 3D. Es por ello que se ha propuesto un **MVC jerárquico**, donde cada módulo tendrá su propio esquema MVC (ver Ilustración 53).

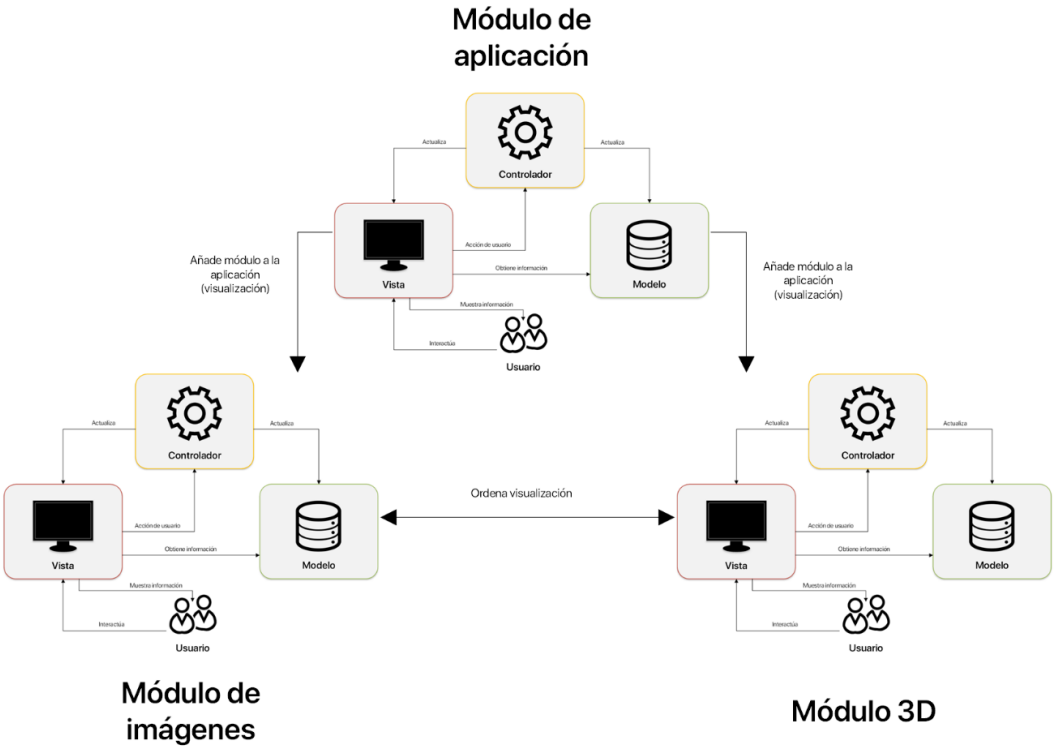


Ilustración 53. Modelo MVC jerárquico.

La vista del módulo de aplicación integra aquellos componentes de utilidad para todas las funcionalidades disponibles: barras de progreso, diálogo base, etc. La comunicación entre el módulo de la aplicación y el resto se produce necesariamente al comienzo de la aplicación, cuando se inicializa uno de ellos (que en nuestro caso

será el módulo de imágenes). Durante el resto de la ejecución, su función es escasa ya que los módulos de imágenes y 3D no son completamente independientes (se accede a uno a través de otro: extracción de contornos => visualización 3D). De hecho, las funciones por ahora del módulo de mayor entidad se reducen a crear la aplicación (preparación de ventana, inicialización de módulos, etc.), asignación a la interfaz del módulo inicial, y destrucción de la aplicación. Por lo tanto, el modelo que aquí se propone sería esencialmente una preparación de cara a una aplicación de mucha más complejidad.

No se introducen en este apartado más detalles, como diagramas de secuencia o *storyboards*, dado que aquí no se ha incluido una funcionalidad como tal. No obstante, el modelo MVC tiene su influencia en todos los esquemas que hemos visto y que nos quedan por presentar:

- En el diagrama de secuencia siempre se presentan tres entidades básicas: modelo, vista y controlador.
- La aplicación que se ha venido visualizando en forma de boceto ya incorpora este modelo, aunque se ha tratado tan sólo el módulo de procesamiento de imágenes hasta ahora.

2.2.6 - Séptima iteración

Para completar el proceso de corrección y adecuación de todas las imágenes multiespectrales obtenidas por el dispositivo Parrot Sequoia sólo nos queda registrarlas; se ha revertido la distorsión *fisheye* de cada imagen, pero aún hay cierta disparidad entre todas ellas que se ajusta al modelo de distancia entre lentes planteado en la Ilustración 16.

Por **registro de imágenes** se entiende el proceso de **integrar varios conjuntos de imágenes en un mismo sistema de coordenadas**. Pueden ser imágenes capturadas desde diferente altura, en un instante diferente, desde puntos de vista dispares, etc. Es un proceso especialmente importante para comparar, ya que para ello necesitamos establecer la correspondencia entre píxeles de diferentes imágenes.

Por ejemplo, si tuviéramos dos imágenes que debiéramos alinear, una sería la **referencia** o base, e intentaríamos alinear la segunda imagen con esta referencia. De hecho, esto sucede en nuestro conjunto de imágenes. Existen múltiples formas de

resolver este problema: una opción clara parece encontrar puntos similares entre dos capturas, y a partir de ahí hallar la función que nos permita transformar una imagen para alinearla con la referencia.

En general, podemos clasificar los algoritmos de registro de imágenes en los siguientes tipos (J. Michael Fitzpatrick, 2000):

- **Basados en la intensidad de la imagen o en características**: se pueden alinear imágenes intentando encontrar patrones de intensidad comunes en dos de ellas (sea una la referencia y otra la que debemos alinear), o bien encontrando estructuras que aparezcan en ambas imágenes, como pudiera ser un punto, una línea o un contorno.
- **Según el modelo de transformación**: para poder entender este apartado es necesario comprender los tipos de transformaciones que se pueden llevar a cabo. Se presentan sólo las transformaciones 2D para simplificar el problema y porque son las que se podrían aplicar a nuestra situación:
    - **Transformaciones lineales**, entre las que se encuentran la rotación, el escalado o la traslación, así como otras transformaciones afines. El problema que tienen este tipo de transformaciones es que se aplican a la imagen al completo, y por lo tanto, si los cambios entre dos capturas se producen en determinadas regiones este tipo de transformaciones difícilmente son capaces de alinear ambos sistemas de coordenadas.
    - **Transformaciones no lineales**: al contrario de las transformaciones anteriores, ahora se pueden deformar las imágenes mediantes transformaciones no rígidas.
- **Según el espacio en el que trabajen**: el algoritmo podría utilizar el propio espacio de la imagen, o bien trabajar en un espacio transformado, por ejemplo, a partir de la transformada de Fourier.
- **Según la interacción que necesiten del usuario**: lo ideal sería que el algoritmo fuera capaz de relacionar dos imágenes de forma autónoma, sin embargo, en ciertas ocasiones se puede hacer partícipe al usuario en dicho proceso, ya sea en mayor o menor grado.

Las diferencias entre imágenes multiespectrales, dejando a un lado el nivel de intensidad de cada banda, se corresponden como mínimo a una traslación. Esto es así porque las imágenes no se capturan desde una misma cámara, sino desde cuatro distribuidas físicamente en zonas dispares. Dicha traslación viene dada por la documentación de Parrot (Parrot , 2017), aunque también podría calcularse manualmente (corresponde a la distancia física entre los centros de las lentes).

La **situación más optimista** de captura tiene las siguientes características (ver Ilustración 54):

- El eje Z de la cámara es *nadir* (perpendicular a la superficie del terreno).
- La calibración del fabricante establece el punto principal de la imagen exactamente en el centro de la misma $(width\ /\ 2, height\ /\ 2)$.
- No existe traslación entre imágenes porque todas se capturan en el mismo instante de tiempo y desde el mismo punto físico.

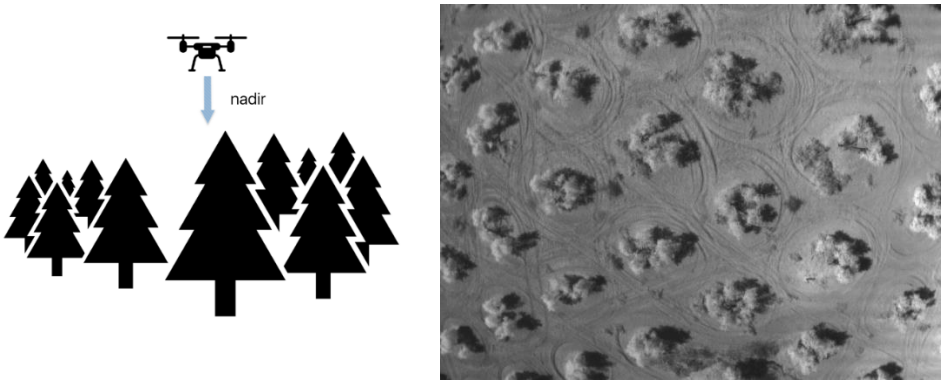


Ilustración 54. Caso optimista donde la cámara captura el terreno en posición nadir.

La realidad dista mucho del escenario que acabamos de presentar, y esto se puede ver al combinar todas las imágenes multiespectrales debido a que aparece lo que se conoce como *image ghosting*, derivado de la traslación entre ellas (ver Ilustración 55).

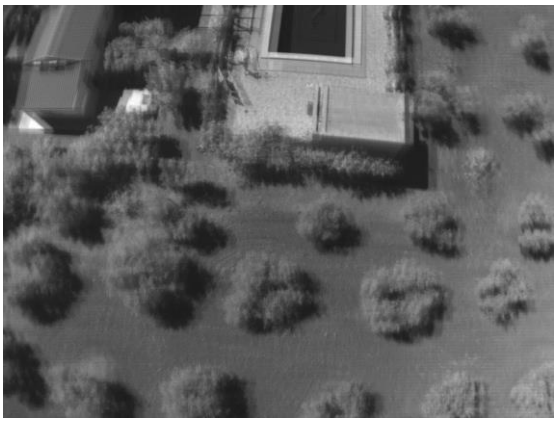


Ilustración 55. Combinación de todas las imágenes multiespectrales en una misma imagen, utilizando cierta opacidad para todas las capas (excepto la base).

¿Cuál es la realidad?

- Las lentes vienen calibradas por el fabricante y determinan que **el punto principal no se encuentra exactamente en** $(\boldsymbol{width}\ /\ \mathbf{2}, \boldsymbol{height}\ /\ \mathbf{2})$, de tal forma que cada lenta tiene su propio punto principal.
- Existe siempre una traslación entre imágenes debido a la distribución física de las lentes. Nótese que conocemos la distancia en mm (15) entre ellas, pero no a cuánto corresponde dicha distancia en píxeles de la imagen.

En principio, para extraer a cuántos metros se corresponde un píxel de la imagen nos es suficiente con la distancia focal (f) y la altura del dron sobre la superficie (d), que es un dato que no tenemos (sólo tendríamos la altura sobre el nivel de mar). A todo esto hay que añadir que la proyección existente en las capturas es una perspectiva, por lo que es necesario establecer ciertas relaciones de semejanza en el plano del suelo y en el plano de la imagen. La Ilustración 56 muestra los parámetros necesarios para obtener toda esta información asociada a un píxel.

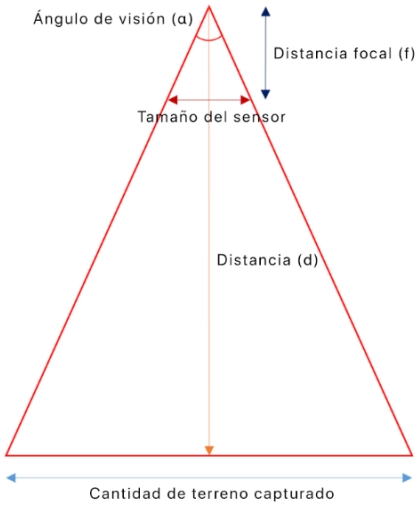


Ilustración 56. Esquema con todos los parámetros necesarios para calcular la cantidad de metros que corresponde a cada píxel.

Tampoco se conoce el ángulo de apertura α de inicio, pero tenemos la anchura de la imagen (1280 píxeles en todas las bandas), la altura (960 píxeles en todas las bandas) y la resolución en *píxeles / mm* para el eje X y el Y (en ambos casos es 266.66 *dots / mm*), por lo que se extrae lo siguiente:

$$Anchura\ del\ sensor\ (mm) = \frac{Anchura\ de\ la\ imagen\ (píxeles)}{Resolución\ en\ X\ (dots\ per\ mm)} \quad (14)$$

Como resultado, se extrae que la anchura del sensor es de 4.8 mm. Una vez que se conoce este dato, podemos calcular el ángulo de apertura $\frac{\alpha}{2}$ a partir de f y la anchura que acabamos de calcular:

$$\alpha = 2 * \tan^{-1}\left(\frac{Anchura\ del\ sensor\ (mm)}{Distancia\ focal\ (mm)}\right) \quad (15)$$

Si conociéramos la distancia d al suelo podríamos calcular a cuánto corresponde en metros la mitad del terreno capturado (en anchura) a partir de α:

$$\frac{Anchura\ del\ terreno\ (m)}{2} = \tan\alpha * d \quad (16)$$

No sería necesario calcular la anchura total del terreno, ya que también podríamos dividir la anchura de la imagen en píxeles entre dos, y obtendríamos la misma información:

$$Metros\ por\ píxel\ \left(\frac{m}{píxeles}\right) = \frac{2}{Anchura\ de\ la\ imagen\ (píxeles)} * \frac{Anchura\ del\ terreno\ (m)}{2} \quad (17)$$

Nótese como en todo momento se ha supuesto que el eje Z de la cámara es *nadir*. En cualquiera de los casos, si esta no fuera la situación se podría modificar la imagen en lugar de toda la formulación expuesta.

Por otro lado, hemos realizado todos estos cálculos a partir de la anchura del terreno, la anchura del sensor, etc. Se podría obtener el mismo resultado si se hiciera con altura en lugar de anchura, y esto sucede así porque la resolución (medida en *dots / mm*) es la misma para los ejes X e Y de la cámara (siempre es 266.66 *dots / mm* para el dispositivo Parrot Sequoia).

Toda esta información que hemos aprovechado para comentar en este apartado nos será de gran utilidad tanto en descripciones próximas como en futuras iteraciones.

Durante el desarrollo de este proyecto se exploran diferentes métodos de registro de imágenes, los cuales se presentan a continuación:

2.2.6.1.1 - Registro de imágenes mediante metadatos

Para este apartado nos basaremos completamente en la información que nos da Parrot Sequoia, así como una de sus plataformas recomendadas para el tratamiento de imágenes, Pix4D (Pix4D, 2018). Para entender todo lo que viene a continuación es necesario definir claramente dos aspectos:

- **Distribución física de las cámaras en el dispositivo**. Además de la distancia entre lentes, ya bien conocida, se debe añadir la distancia adicional debido a que los puntos principales de las imágenes no están en el centro de la imagen (si no fuera así, siempre habría 15 mm de distancia horizontal y/o vertical).
- **Entre las cámaras se forma lo que se conoce como un anillo**. Esta información se puede encontrar en metadatos como *Rig Relatives* y *Rig Index*. Un anillo no es más que un conjunto de cámaras conectadas, entre las que se establecen relaciones geométricas. Para capturar una escena, se lanza un comando que permite que todas las cámaras obtengan su propia imagen simultáneamente (la diferencia es de milisegundos).

  Un anillo tiene las siguientes características:

- Una cámara, que en nuestro caso es la de la banda GRE, se establece como referencia. Esta cámara tendrá una posición T_m y una rotación R_m en el mundo.
- El resto de cámaras son secundarias, y tienen también posición T_s y rotación R_s en el mundo.
- Para cada cámara secundaria, tanto la traslación T_{rel} como la rotación R_{rel} respecto de la cámara principal es conocida. En nuestro caso, la traslación viene dada por esa cantidad de milímetros entre cámaras que ya hemos comentado, y la rotación viene dada por el metadato *Rig Relatives*, que serían tres ángulos de rotación en grados respecto de X, Y y Z.
- A partir de todo esto, se extraen las siguientes relaciones:

$$T_s = T_M + R_M * T_{REL} \quad (18)$$

$$R_S = R_M * R_{REL} \quad (19)$$

$$X'_M = {R_M}^T (X - T_M) \quad (20)$$

$$X'_S = {R_{REL}}^T [{R_M}^T (X - T_M) - T_{REL}] \quad (21)$$

sea X' una posición 3D en el sistema de coordenadas de una cámara principal o secundaria.

Si partimos de la consideración de que las cámaras secundarias definen su posición y traslación respecto de una cámara principal, podemos suponer que la traslación en el mundo para la cámara principal es 0 (T_M = 0). Del mismo modo, al ser esta cámara la referencia del anillo se puede suponer que no ha sufrido rotación alguna ($R_M\ es$ la identidad, I).

Podemos simplificar entonces la fórmula anteriormente presentada:

$$X'_S = {R_{REL}}^T [X - T_{REL}] \quad (22)$$

Conocemos tanto la rotación como la traslación de una cámara secundaria respecto de la principal, pero en el caso de la traslación viene expresada en milímetros y no en píxeles, que es la unidad en la que siempre trabajamos a nivel de imagen. Nótese como para transformar entre dichas unidades necesitamos conocer a cuántos milímetros corresponde un píxel, y esta relación depende, entre otros parámetros, de la distancia d al suelo (la cual desconocemos). El problema encontrado se puede expresar de la siguiente manera:

- La altura de la cámara respecto del suelo es desconocida, dado que sólo disponemos de la altura de vuelo sobre el nivel del mar. En cualquiera de los casos, dada una localización geográfica se podría obtener la altitud del terreno en dicho punto mediante algún sistema SIG externo que nos permita acceder a d mediante la siguiente operación:

$$d = Altura\ de\ vuelo - Altura\ del\ terreno \quad (23)$$

- Se puede suponer que hay un tanto por ciento de desplazamiento de imágenes, lo que involucra hallar estos valores en el eje horizontal y vertical de forma manual para cada imagen. No parece una solución cómoda ni genérica para cualquier tipo de imágenes multiespectrales (no sólo del dispositivo Parrot Sequoia).

En vista de los problemas presentados se busca una alternativa para el registro de imágenes, que será nuestro siguiente apartado.

2.2.6.1.2 - Registro de imágenes mediante ECC (Enhanced Correlation Coefficient)

Como bien sabemos, en nuestro proyecto hemos venido usando la librería OpenCV, la cual incluye algoritmos para el análisis de movimiento. En nuestro caso, las diferencias entre imágenes se corresponden a un movimiento de la escena derivado de la distancia física entre lentes (además de las diferencias de orientación que puedan existir entre ellas).

Uno de esos algoritmos incluidos en la librería es *Enhanced Correlation Coefficient*, ECC (Georgios D. Evangelidis, 2008), donde se describe un método basado en un coeficiente de correlación cuyas principales características son dos:

- Los algoritmos tradicionales comprueban la diferencia de intensidad entre píxeles para hallar similitudes. Sin embargo, este algoritmo es invariante a

distorsiones fotométricas en contraste y brillo. De hecho, esta es una característica que encaja perfectamente en nuestro conjunto de imágenes, debido a que como bien sabemos, cada imagen presenta una apariencia muy diferente a la del resto (Mallick, 2015).

- Resuelve un problema computacionalmente muy complejo en tiempo lineal ($O(n)$).

En definitiva, de lo que trata este algoritmo es de optimizar una serie de parámetros que definen la transformación que se debe producir para alinear ambas imágenes. La cantidad de parámetros a estimar y el tipo viene determinada por el modelo de transformación elegido, de lo cual ya se hablado brevemente en anteriores apartados (transformaciones lineales, no lineales, etc).

La Ilustración 57 resume las transformaciones que OpenCV nos permite incorporar a cualquier algoritmo de estimación de parámetros (*motion estimation*, *region tracking, etc*):

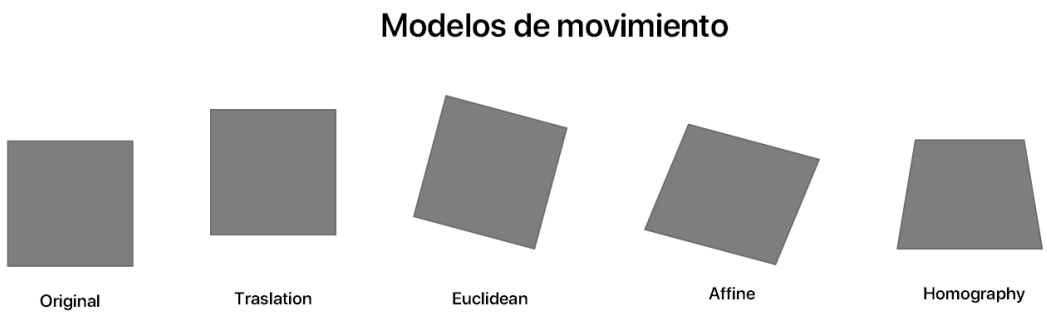


Ilustración 57. Representación de los diferentes tipos de transformaciones que se pueden utilizar al registrar imágenes en OpenCV.

¿Qué diferencias existen entre todas estas transformaciones?

- **Traslación**: la diferencia entre dos imágenes viene simplemente dada por un desplazamiento (x, y), y por lo tanto, sólo habrá que optimizar dos parámetros.
- **Euclídea**: la transformación viene definida por una rotación y un desplazamiento, por lo que se incorpora un parámetro más, un ángulo α. Se debe destacar que la imagen mantiene su tamaño así como los ángulos y el paralelismo entre sus líneas.
- **Afín**: el número de parámetros crece notablemente porque la transformación ahora viene expresada en términos de traslación, rotación,

escalado y *shear* (algo similar a empujar un objeto sólo por uno de los extremos). Necesitamos optimizar hasta 6 parámetros. Las líneas permanecen paralelas, pero los ángulos no tienen por qué mantenerse iguales.

Nótese además que la transformación afín puede representar también la translación y la transformación euclídea. Por lo tanto, la matriz que las define es igual en cuanto a tamaño (2x3).

Nos lleva también a pensar lo siguiente: si sabemos que a la imagen sólo se le aplica traslación y/o rotación, podemos aplicar alguna de las dos primeras transformaciones. Si por el contrario se desconoce qué función alinea una imagen con la referencia se puede optar por la transformación afín, que engloba también las dos anteriores pero nos permite ir un poco más allá.

- **Homografía**: se trata de una transformación proyectiva que consta de 8 parámetros. Ya no podemos utilizar una matriz de 2x3 (hasta 6 parámetros), sino que crece hasta 3x3. Si se le aplica esta transformación a un cuadrado puede resultar en cualquier tipo de cuadrilátero.

Nos queda por lo tanto reflexionar qué tipo de transformación queremos aplicar a una imagen multiespectral para conseguir alinearla con la referencia:

- La **traslación** es una transformación de la que estamos completamente seguros que se debe aplicar. Además, sabemos que existe una distancia física entre lentes que debe representarse mediante esta transformación.
- Sabemos también que existe una **rotación** del sistema de coordenadas de una imagen multiespectral respecto de la referencia. El principal problema es que la rotación real no viene expresada sólo en términos 2D (nótese que la cámara tiene una orientación en el mundo real, expresada en 3D, que nos permite capturar la escena), y sin embargo, la transformación euclídea sólo considera un ángulo de rotación respecto del eje Z de la cámara (modifica x e y en la imagen).

  Necesitamos por lo tanto alguna transformación más compleja que nos permita alinear pares de imágenes, y en este caso hemos optado por

una transformación afín que funciona muy bien para nuestro conjunto de datos.

Sólo nos queda ya comprender cómo funciona el algoritmo, que como ya sabemos, viene dado en la librería OpenCV. La mejor forma de iniciarse es conocer los parámetros que necesita (Mallick, 2015):

- Un elemento obvio será la **matriz resultado** donde el algoritmo almacena la transformación a realizar. Su tamaño es variable en función del modelo de movimiento escogido.
- La matriz no le proporciona al algoritmo la suficiente información para saber el tipo de **transformación** que se pide: afín, homografía, euclídea, etc. Habrá que utilizar alguna de las constantes que define OpenCV (*MOTION_AFFINE*, *MOTION_EUCLIDEAN*, *MOTION_HOMOGRAPHY* ó *MOTION_TRANSLATION*).
- Definir un criterio de comportamiento del algoritmo, integrado en la clase *TermCriteria*. Este elemento es muy importante para comprender el funcionamiento, y viene dado por dos variables principalmente:
    - **Número de iteraciones**: decíamos que el algoritmo resolvía el problema de forma iterativa, por lo que se debe indicar el número de iteraciones máximo de este proceso. Obviamente, cuanto mayor sea este valor es más probable que la solución que se alcance sea mejor, pero también necesita más tiempo para completarse. Se debe por lo tanto alcanzar un equilibrio entre el tiempo empleado y la completitud de la solución obtenida.
    - **Precisión**: se especifica el nivel de precisión que debe buscar el algoritmo. Los valores que se especifican para este parámetro son cercanos al cero, por ejemplo, 1^{-10}, de tal forma que valores más grandes implican que se obtendrá menos precisión. Obviamente, cuanto mayor sea este número (menos precisión) más diferencias existirán entre las imágenes resultantes.

Es importante saber que el algoritmo no siempre puede converger a dicho nivel de precisión, por ejemplo, si el número de iteraciones es demasiado pequeño. En

definitiva, el flujo necesario para alcanzar la alineación de dos imágenes se puede ver en el pseudocódigo de la Ilustración 58.

Algorithm 3: Registers the multispectral layers respect to the reference image, GRE, by using the Enhanced Correlation Coefficient method

```
function RegisterImages (multispectral);
  Input  : A multilayer image that should contains at least the reference
           layer. As the reference layer is already registered it doesn't
           make sense to pass only that layer
  Output: Boolean value with the success of the operation. The aligned
           images replace the previous images and are not returned
if multispectral.alreadyAligned then
    return false;
end
if !multispectral.referenceLayer then
    return false;
end
multispectral.unifyLayersSize();
while imageIndex ≤ RGB do
    if imageIndex ≠ GRE & multispectral.layer[imageIndex] then
        mostSimilarImage ← mostSimilarImage[imageIndex];
        if multispectral.layer[mostSimilarImage] &
         mostSimilarImage ≠ GRE then
            if
             image[imageIndex].alignImage(multispectral.layer[mostSimilarImage])
             then
                Update layer with aligned image
            end
            else
                Unsuccessful registering for this layer
            end
        end
    end
end
while imageIndex ≤ RGB do
    imageRealIndex ←
     multispectral.buildingAlignmentOrder[imageIndex];
    if
     imageRealIndex ≠ GRE & multispectral.layer[imageRealIndex]
     then
        mostSimilarImage ← mostSimilarImage[imageRealIndex];
        if multispectral.layer[mostSimilarImage] then
            if mostSimilarImage == GRE then
                Align imageRealIndex layer with GRE layer
            end
            else
                Align imageRealIndex layer with mostSimilarImage
                through its alignment matrix
            end
        end
    end
end
multispectral.cropImages();
return success;
```

Ilustración 58. Pseudocódigo de registro de imágenes multiespectrales.

Hasta ahora se ha considerado que el registro de imágenes siempre partía de la imagen GRE como la referencia y que el resto se alineaban respecto de esta, sin embargo, el pseudocódigo que acabamos de ver no hace esto. Aunque esa solución puede funcionar, en algunas imágenes multiespectrales falla porque la diferencia entre la intensidad que representan es demasiado grande.

La solución propuesta para evitar este problema es formar un árbol entre las cuatro imágenes, donde la raíz sería la imagen de referencia (es un registro jerárquico). De esta forma, si alguna capa presenta más problemas al alinearla con la referencia, se puede alinear respecto de otra imagen que se asemeje más (Yasir R.,

2018). Esta otra imagen se alinea a su vez respecto de una tercera, que en nuestro caso siempre es la imagen de referencia (GRE).

Por lo tanto, ahora se deben generar varias pasadas, tantas como niveles tenga el árbol (si no consideramos la raíz como un nivel más). En nuestro caso tendremos hasta dos pasadas:

- En la primera pasada se registran aquellas imágenes que necesitan de algún nodo intermedio.
- En la segunda pasada se alinean todas las imágenes respecto de la referencia, sin embargo, aquellas que previamente dependían de otra capa deben registrarse aplicándoles directamente una matriz de transformación en lugar de ejecutar otra vez el algoritmo de registro. Dicha matriz se corresponde al resultado obtenido de alinear el nodo intermedio con la referencia.

Por ejemplo, la capa NIR era la que más dificultades nos planteaba a la hora de ejecutar el registro de imágenes, por lo que se alinea respecto de otra capa cuya intensidad es más similar.

Un posible flujo de registro de imágenes sería el siguiente:

- Primera pasada: se registra la capa NIR respecto de REG (ejecuta algoritmo de registro).
- Segunda pasada: REG y RED se alinean respecto de GRE. NIR se alinea aplicándole la matriz de transformación obtenida por REG (la debe almacenar).

En el esquema de la Ilustración 59 se presenta un árbol de registro de imágenes donde GRE, como en todos los casos, es la raíz. NIR y RED son hojas, mientras que REG actúa de nodo intermedio.

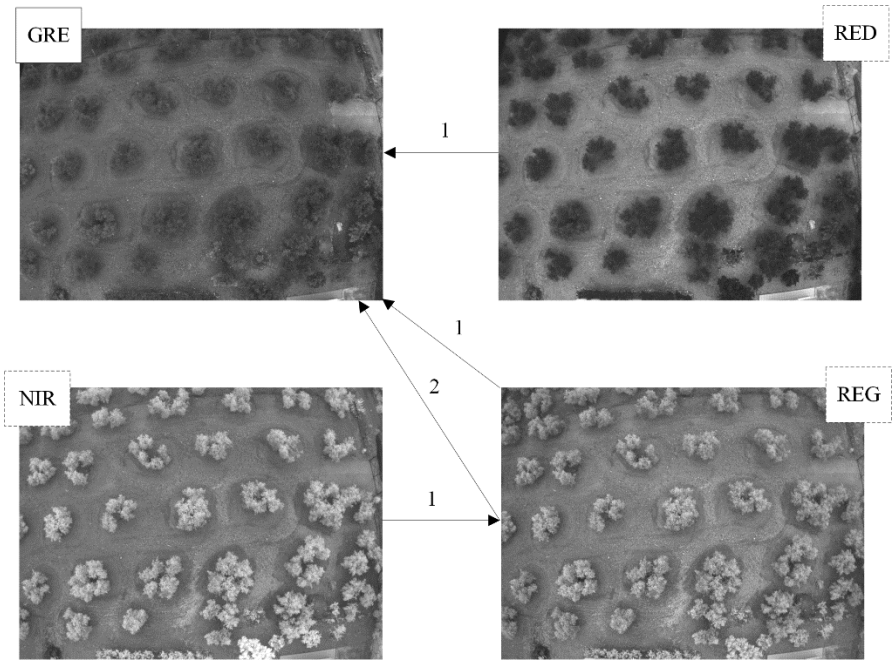


Ilustración 59. Representación del árbol que se forma entre imágenes multiespectrales para efectuar el registro.

Aún nos queda un último detalle por definir, teniendo que en cuenta que la transformación que se aplica a las imágenes contiene al menos dos operaciones básicas como son la traslación y la rotación. Esto implica que a excepción de la imagen de referencia, en el resto se generan bordes sin color alguno que no necesariamente forman un rectángulo (debido a la rotación). En la Ilustración 60 se muestra el resultado de ejecutar el algoritmo de registro sobre la multiespectral NIR.

Ilustración 60. Resultado de ejecutar el algoritmo de registro de imágenes sobre la multiespectral NIR.

Mantener los bordes generados presenta dos problemas:

- Cuando extraíamos los contornos a partir de las capas RED y NIR se desechaban aquellos árboles que no se mostraban por completo en la imagen (estaban cortados), para lo cual hacíamos un búsqueda de puntos colineales en los ejes X e Y en los extremos. Este algoritmo ya no nos serviría por dos razones:
    - Podríamos aumentar el tamaño de los bordes donde se busca esta situación (un *offset* respecto de los extremos), pero esto podría eliminar contornos válidos.
    - Debido a la rotación, los bordes de color nulo generados no tienen por qué ser paralelos a los ejes X o Y, y de hecho, en muchos casos están rotados. Ya no nos valdría sólo con buscar puntos colineales exactamente en dichos ejes.
- En secciones posteriores se trabaja con estas imágenes registradas para hallar de cada píxel o un punto concreto su localización geográfica, pero también su color en cada multiespectral. Por lo tanto, no parece correcto incluir puntos de los que no se dispone de información de nivel de intensidad (aunque podrían ignorarse).

La solución a este problema se consigue fácilmente a partir de la implementación previa del registro de imágenes, ya que todas las capas almacenaban su transformación afín después de alinearlas, debido a que alguna otra capa podría necesitar dicha matriz.

Dadas las cuatro imágenes multiespectrales, GRE (que no se modifica), RED, REG y NIR, se puede extraer el nuevo tamaño de todas las imágenes de la siguiente forma:

- Para cada capa, a excepción de GRE, se puede calcular la nueva posición de sus cuatro esquinas tras aplicarle la transformación afín correspondiente.
- Se pueden obtener los mínimos y/o máximos para cada esquina, de tal manera que al final del algoritmo se extraen cuatro puntos que delimitan el nuevo área para las cuatro imágenes.

$$min_x = max_x(M_i * [0,0,1]^T,\ M_i * [0,h-1,1]^T, min_x)\ \forall i \in [0,3) \quad (24)$$

$$min_y = max_y(M_i * [0,0,1]^T,\ M_i * [w-1,0,1]^T, min_y)\ \forall i \in [0,3) \quad (25)$$

$$max_x = min_x(M_i * [w - 1, 0, 1]^T, M_i * [w - 1, h - 1, 1]^T, max_x) \; \forall i \in [0, 3) \quad (26)$$

$$max_y = min_y(M_i * [0, h - 1, 1]^T, M_i * [w - 1, h - 1, 1]^T, max_y) \; \forall i \in [0, 3) \quad (27)$$

Por ejemplo, para hallar la coordenada de la esquina inferior izquierda se debe aplicar M_i en todas las capas al punto (0, 0), y escoger la máxima x e y de todos los resultados. Si nos quedáramos con el mínimo aún habría capas con valores nulos, cuando lo que desea es hallar el área intersección de todas las imágenes donde sus valores no son nulos.

Por lo tanto, una vez se alinean las imágenes y se eliminan los bordes generados, se puede obtener el resultado de la Ilustración 61 tras combinar las cuatro imágenes multiespectrales con una cierta opacidad:

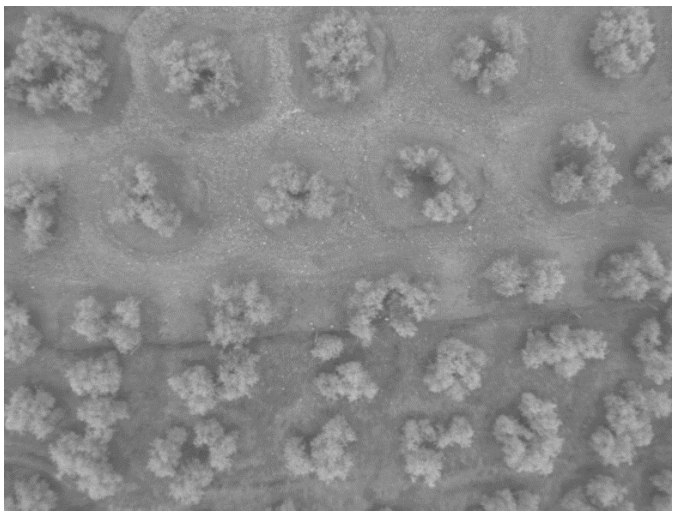


Ilustración 61. Superposición de las cuatro imágenes multiespectrales registradas con un cierto nivel de opacidad. Desaparece el efecto de image ghosting.

Nótese que hasta ahora se ha considerado que las imágenes se toman a una distancia del suelo prudente, ya que de otra forma habría que aumentar el número de iteraciones para que consiga alinear imágenes entre las cuales la distancia es mayor (si la distancia al suelo es muy pequeña la distancia entre imágenes crece), o bien disminuir el nivel de precisión.

Hasta ahora hemos trabajado con el registro de cuatro imágenes multiespectrales, sin embargo, sabemos que disponíamos en realidad de cinco imágenes, donde una de ellas era la RGB. Si bien, para esta última se desconoce a priori el desplazamiento respecto del resto de lentes. Se ha venido dejando a un lado esta otra captura porque no aporta nada al proyecto: la localización geográfica es igual

de pobre que para el resto, y de cara a acudir a una imagen RGB se prefiere la obtenida por otro dispositivo más preciso.

En cualquiera de los casos, las diferencias entre las imágenes multiespectrales y RGB capturadas por Parrot Sequoia se presentan a continuación:

- **Resolución**: la imagen RGB tiene muchas más resolución (4608x3456) que la multiespectral (1280x960). En principio esto no representa un problema dado que ambas presentan el mismo ratio de aspecto (4:3), y por lo tanto, se puede disminuir el tamaño de la imagen de RGB, de tal forma que la información de varios píxeles de esta se almacenen en un único píxel de la imagen resultado (de menor tamaño). De hecho, para el registro de imágenes sería necesaria esta operación dado que la imagen de referencia y la imagen a registrar deben tener las mismas dimensiones.
- **Distancia focal**: determina el ángulo de apertura de la lente, y en el caso de la cámara RGB la distancia focal es mayor (4.9 mm) que en las imágenes multiespectrales (4 mm aproximadamente). Esto también implica que el ángulo de apertura de una y otra es diferente, de hecho, al ser mayor la distancia focal disminuye el ángulo (de 61.9 grados a 51.5 aproximadamente).
- **Modelo de distorsión**: en apartados previos hemos visto como nuestras imágenes multiespectrales presentaban una distorsión de ojo de pez, pero la imagen RGB sólo contiene una distorsión de perspectiva, o eso se puede suponer de sus metadatos. Sin embargo, cuando se observa la imagen sí que hay evidencias de un modelo de distorsión de *fisheye* que se puede comprobar fácilmente al registrarla mediante el proceso ya definido. Incluso conociendo esa distorsión, no podríamos revertirla dado que se desconocen los coeficientes que afectan a la imagen.

En resumen, las diferencias entre los dos tipos de imágenes son muy amplias, y algunas de ellas son las que acabamos de ver (aunque no todas). Si aplicáramos el algoritmo de registro de imágenes sobre una imagen de referencia y una imagen RGB de un conjunto de datos cualquiera, el resultado sería algo similar a lo que se representa en la Ilustración 62 y la Ilustración 63.

Ilustración 62. Comparativa entre una captura GRE y RGB tras aplicar el algoritmo de registro de imágenes.

Ilustración 63. Combinación de imágenes GRE y RGB a través de la opacidad, una vez han sido registradas.

Nótese como el tipo de transformación que se busca en este caso no debe ser una transformación afín sino una homografía, dado que las diferencias son demasiado grandes entre ambas imágenes. Antes de llevar a cabo el proceso también es necesario eliminar la distorsión de la imagen multiespectral, dado que ambas imágenes presentan originariamente un modelo de distorsión diferente.

Como puede observarse, el resultado en este caso es de menor calidad que en el registro de imágenes multiespectrales. De hecho, las imágenes se aproximan pero no llegan a coincidir totalmente. Si bien es cierto que existen diferencias en cuanto al ángulo de visión en ambas imágenes, el principal problema se encuentra precisamente en la distorsión de la imagen RGB, ya que se desconocen totalmente los parámetros que podrían eliminar esa situación (nótese como una transformación

de homografía puede transformar nuestra imagen en cualquier cuadrilátero, pero no en un modelo con forma de circunferencia, como sucede en la distorsión *fisheye*). De hecho, en la última figura se puede ver como el registro es relativamente exacto en la parte inferior de la imagen, pero no así en la parte superior, donde la distorsión es más evidente.

2.2.6.1.3 - Modelo multicapa

Aprovechamos este apartado para recuperar un concepto que ya surgió en iteraciones anteriores al consultar el diagrama de secuencia: el **modelo multicapa**. Ahora que nuestras imágenes se hallan todas sobre un mismo sistema de coordenadas esto toma mucho más sentido, de tal forma que podríamos tener hasta 5 imágenes del mismo tamaño que representan un mismo área (a excepción de la imagen RGB debido a esa pequeña distorsión que sufre). Podríamos lanzar ahora consultas sobre un píxel y obtener varios valores de intensidad que se corresponden realmente al mismo punto de la escena (y de esto nos aprovechamos más tarde).

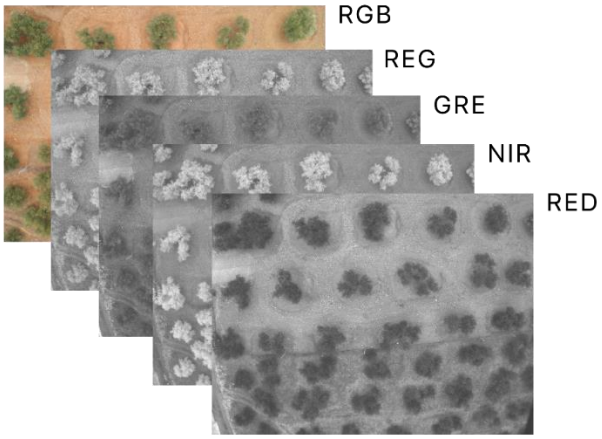


Ilustración 64. Representación de la salida del dispositivo Parrot Sequoia como un modelo multicapa.

La utilización del modelo propuesto en la Ilustración 64 nos permite asimismo abstraer ciertas operaciones que se deben realizar necesariamente con varias imágenes multiespectrales.

2.2.6.1.4 - Georreferenciación

Una vez que nuestras imágenes multiespectrales se encuentran completamente alineadas, el siguiente paso puede ser identificar de forma única cada punto de una imagen multiespectral (al que tenemos acceso en todas las imágenes) a través de la georreferenciación. De esta forma, podríamos guardar en una base de datos información como el nivel de gris en cada imagen para un punto con un identificador basado en su localización geográfica.

El **proceso de georreferenciación** implica que **somos capaces de relacionar el sistema de coordenadas de una imagen con un sistema de coordenadas situado sobre la Tierra**. De esa forma, podemos saber la localización exacta en un espacio físico de cualquier elemento que aparezca en la imagen. En concreto, nos interesa geolocalizar la vegetación que se presenta en la imagen, que en nuestro caso serán árboles, tanto sus centros de masas como sus contornos si así fuese necesario.

Las localizaciones geográficas se representan normalmente utilizando un sistema de coordenadas de referencia, que se puede relacionar a su vez con un sistema de referencia geodésico (*datum*) como puede ser WGS 84. De hecho, cuando comprobemos si nuestros resultados son correctos, utilizaremos una fuente externa que nos permite elegir el sistema de referencia geodésico que queremos (Montana State University, s.f.).

Conocer la localización exacta en el mundo de un objeto supone tener un gran control sobre el mismo. Por ejemplo, nos podemos trasladar a la extracción de contornos, que era uno de nuestros primeros apartados. Dicha operación nos permite conocer el tamaño de un árbol para un instante de tiempo concreto. Como bien sabemos, a partir de dicho contorno se puede extraer el centro de masas, y si consiguiéramos su localización geográfica podríamos hablar de tener un identificador único para cada árbol.

El objetivo podría ser capturar el mismo terreno en varias ocasiones y en instantes de tiempo diferentes. Se pueden encontrar situaciones, por ejemplo, de crecimiento de árboles, que es una de las características que se podría analizar a partir de este proyecto (estamos calculando contornos, a partir de los cuales se puede extraer el área de un polígono). Como es de suponer, si un árbol crece modifica su centro de masas, ya georreferenciado. Se podría establecer entonces un umbral por debajo del cual se considera que sería el mismo punto en diferentes instantes de tiempo.

En definitiva, en nuestro proyecto la georreferenciación es vital para controlar la evolución de un árbol a nivel de volumen o área, enfermedades (si se analizaran), etc.

Se consideran a partir de aquí las etapas necesarias para georreferenciar un punto en el sistema geoespacial del mundo. A modo de introducción, se analiza el sistema de referencia que se emplea para geolocalizar un punto de la imagen así como su implementación, y continuaremos con la transformación de imágenes que se debe llevar a cabo hasta poder volcar todos los datos en un fichero de texto.

Sistema de referencia utilizado: UTM

En secciones previas de este documento se ha comentado ya la inexactitud de la localización geográfica que nos aportan las imágenes multiespectrales. Más concretamente, los metadatos de estas imágenes incluyen la localización geográfica de la cámara en el momento de captura en forma de **latitud y longitud**.

La **latitud** de un punto se define como el ángulo entre el plano ecuatorial y una línea cuya dirección viene dada por el centro de la Tierra y la posición del punto en el globo. Por lo tanto, ambos polos de la Tierra se encuentran a 90º, aunque se diferencian dos hemisferios: norte y sur (Usery, 2010); (Ordnance Survey, 2018).

De forma análoga se representa la **longitud**, con la diferencia de que el plano respecto del cual se compara el ángulo ahora se conoce como meridiano. No existe un único meridiano sino que se puede considerar uno diferente para cada latitud, definiéndose un meridiano como un semicírculo que iría desde el polo norte hasta el polo sur. Se consideran así varios meridianos de referencia, de tal forma que la longitud de un punto en la tierra se pueda obtener como el ángulo entre un meridiano de referencia y otro que pasa exactamente por el punto que se debe localizar.

En nuestras imágenes tenemos el valor de latitud y longitud expresado de una forma no tan común: viene dado como valores enteros para grados, minutos y segundos, y tras ellos, separados por una barra (/), aparece un divisor que nos permite extraer el valor flotante. A partir de ese formato se podrían obtener dos representaciones habituales para una coordenada geográfica de este tipo:

- **Grados, minutos y segundos expresados como tres valores distintos**, donde los segundos pueden venir dados por un valor flotante, ya que es la parte restante del identificador de una localización. Ejemplo: 4º 14’ 14.5’’.

- **Grados, minutos y segundos expresados como un único valor flotante**. A partir de aquí sería fácilmente expresable en el formato justamente anterior. Tiene la ventaja de que no es necesario mantener tres valores distintos (es más compacto).

En cualquiera de los casos, la transformación necesaria para trasladar una coordenada geográfica de latitud y longitud, tal y como viene expresada en los metadatos, hacia este último formato, sería la siguiente:

$$LatLongValue = \frac{Degrees}{DegreesDivisor} + \frac{\frac{Minutes}{MinutesDivisor}}{60} + \frac{\frac{Seconds}{SecondsDivisor}}{3600} \quad (28)$$

$$if\ ((LatLongValue > 0)\ \&\ Direction\ is\ 'W'\ or\ 'S')\ LatLongValue *= -1 \quad (29)$$

Cuando representamos una coordenada en el formato de tres valores siempre se puede añadir otra cadena que nos indique la localización del punto respecto de los dos planos de referencia de la Tierra. Sin embargo, cuando trabajamos con valores flotantes se simplifica mucho más dado que una localización se considera que se encuentra en el intervalo [-90, 90], donde un valor negativo implica que un punto está por debajo de cierto plano de referencia. Se sustituye así una cadena que nos indica la situación del punto respecto de un plano por un signo negativo o positivo. Obviamente, se siguen manteniendo dos valores, latitud y longitud, pero como números reales.

El principal problema que ahora tenemos, utilizando este sistema de referencia basado en latitud y longitud, es que el perímetro de los discos que se consideran en el globo no es el mismo a lo largo de todo él. Obviamente, un disco horizontal en un polo no tendrá el mismo radio que uno situado justamente en el Ecuador de la tierra. Se decide así trabajar con otro sistema de referencia como es **UTM o sistema de coordenadas universal transversal de Mercator**.

El sistema UTM se define tal y como lo hace el sistema anteriormente visto: se basa en una geolocalización de puntos a lo largo de la Tierra en función de dos parámetros, x e y, que nos posibilitan la representación de la posición horizontal independientemente de la altitud.

A diferencia del sistema de latitud y longitud, aquí las distancias entre puntos se pueden obtener de forma mucho más fácil, ya que la unidad en la que se expresa este sistema son los metros. Para ello se debe identificar un punto a través de dos

valores, x e y, y una zona del globo. Para entender este nuevo sistema de referencia podemos imaginarnos una textura rectangular (escala 2:1, 360:180) que englobe a toda la superficie terrestre (ver Ilustración 65). Considerando esta imagen, las zonas se asignan como sigue: se divide el globo terrestre en 60 zonas (fragmentos verticales) de 6º cada una (1-60). Estas zonas abarcan las longitudes [0, 360], pero en altitud excluyen los polos (10º en el sur y 6º en el norte). Cada zona se vuelve a dividir a su vez en vertical generándose 20 cuadrantes por zona, cada uno identificado con una letra del alfabeto (C-X, excluyendo O e I por su similitud a los números 0 y 1 respectivamente). Esta división del globo en cuadrantes es uniforme, a excepción de ciertas zonas como las ya comentadas (polos) (Snyder, Map projections: A working manual, 1935); (Ordnance Survey, 2018).

Se obtiene así una asignación de zonas a lo largo del globo como la siguiente:

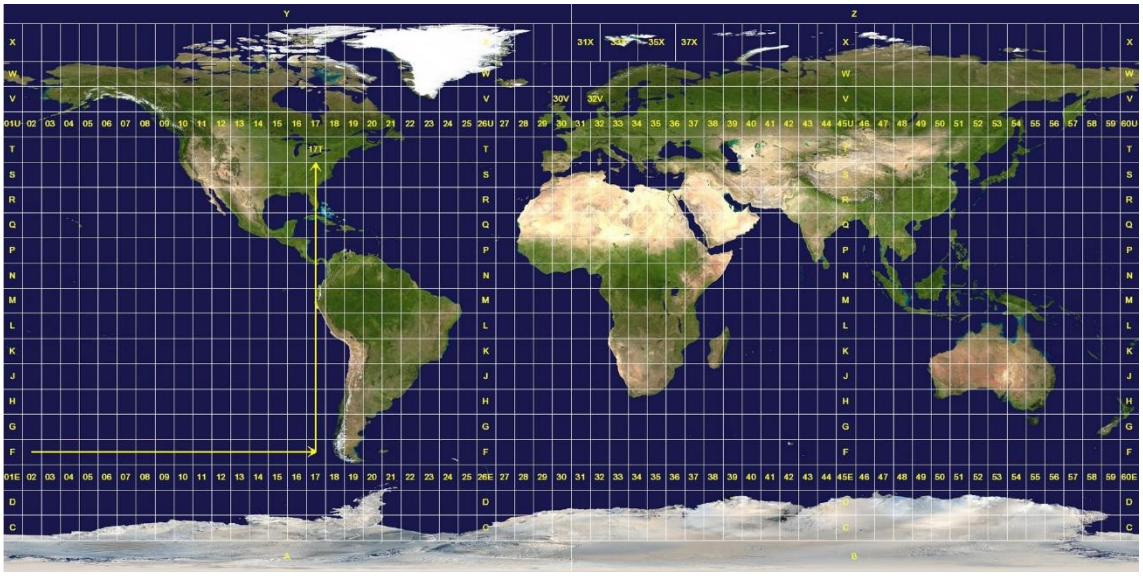


Ilustración 65. División del globo terrestre en zonas y cuadrantes. Sistema de referencia UTM. Autor: Jan Krymmel.

El principal problema que se plantea en este apartado es la **conversión de coordenadas en forma de latitud y longitud hacia el sistema de coordenadas UTM**, que puede llegar a ser un proceso laborioso. Sin embargo, es fácilmente observable que nos ahorra cálculos en un futuro si consideramos que sabemos la distancia en metros entre los píxeles de la imagen. Todo este tema se trata anteriormente, y como ya sabemos, se define una distancia al suelo, d (aproximada), que nos permite averiguar precisamente esa distancia en metros entre dos píxeles cualquiera.

Las operaciones que se deben llevar a cabo para lograr esta conversión son las que se presentan a continuación, teniendo en cuenta que el *datum* empleado es WGS 84, que define una serie de parámetros de entrada que son necesarios para obtener el resultado del algoritmo:

- El radio en el ecuador, α, es de 6378.137 km ($WGS84_A$).
- El factor de compresión del globo, f, es de 1 / 298.257223563.
- N_0, que en el hemisferio norte es 0 km.
- N_0, que en el hemisferio sur es de 10000 km.
- El factor de escala k_0 es 0.9996.
- E_0 es 500 km.
- La primera excentricidad, $WGS84_E$, se define como 0.818191908.

Antes de calcular otros valores iniciales a partir de las variables ya definidas, podemos obtener la zona UTM donde se halla un punto de la Tierra. Teniendo en cuenta las consideraciones ya expuestas, se puede hacer como sigue:

- Primero se debe hallar la zona donde nos encontramos (1-60):

$$Zona = \frac{Latitud + 180}{6} + 1, \tag{30}$$

considerando que la latitud se encuentra entre los valores -180 y 180.

Se deben añadir comprobaciones explícitas para aquellas zonas que no encajan en el esquema uniforme planteado anteriormente. De la misma forma, se puede hallar la cuadrícula donde se encuentra un punto añadiendo a la zona calculada la letra correspondiente a la latitud del punto.

- Para hallar la letra se debe considerar un vector de 20 caracteres que debemos especificar a mano dadas las restricciones expuestas.

$$Letra = char[int\left(\frac{(Latitud + 80)}{8}\right)] \tag{31}$$

Dicha formulación es válida para valores entre -80 y 80. Hay que hacer otra comprobación explícita para valores entre 80º y 84º, que también reciben la última letra. Por supuesto, no se debe devolver una letra para

valores de latitud menores de -80º y mayores de 84º, dado que generaría un error en el acceso al vector anteriormente definido (de tamaño 20).

El resultado de este proceso será una cadena de texto tal como *30S*, donde se indica tanto la zona como la letra correspondiente a la posición en el globo.

Una vez hecho esto podemos hacer los cálculos iniciales para comenzar el proceso duro de conversión, y para ello se necesitan otras tantas variables de entrada del sistema de referencia WGS 84 (Ordnance Survey, 2018):

$$Excentricidad = \frac{WGS84_E^2}{1 - WGS84_E^2} \tag{32}$$

$$N = \frac{WGS84_A}{\sqrt{1 - WGS84_E^2 * \sin(LatitudRad)^2}} \tag{33}$$

$$T = \tan(LatitudRad)^2 \tag{34}$$

$$C = Excentricidad * \cos(LatitudRad)^2 \tag{35}$$

$$Longitud\ en\ origen\ de\ zona = (Zona - 1) * 6 - 180 + 3 * \frac{\pi}{180} \tag{36}$$

$$A = \cos(LatitudRad) * (LongRad - Latitud\ en\ origen\ de\ zona) \tag{37}$$

$$\begin{aligned} M = WGS84_A * (& \left(1 - \frac{WGS84_E^2}{4} - 3 * \frac{WGS84_E^4}{64} - 5 * \frac{WGS84_E^8}{256}\right) \\ & * LatitudRad \\ & - \left(3 * \frac{WGS84_E^2}{8} + 3 * \frac{WGS84_E^4}{32} + 45 * \frac{WGS84_E^8}{1024}\right) \\ & * \sin(2 * LatitudRad) \\ & + \left(15 * \frac{WGS_{84_E^4}}{256} + 45 * \frac{WGS84_E^8}{1024}\right) \\ & * \sin(4 * LatitudRad) - \left(35 * \frac{WGS84_E^8}{3072}\right) * \sin(6 \\ & * LatitudRad)) \end{aligned} \tag{38}$$

De esta forma, una vez calculadas todas estas variables podemos pasar a obtener las dos coordenadas, norte y este, que junto a la zona anteriormente calculada representan una posición única en el sistema de referencia UTM (Snyder, Map

projections: A working manual, 1935); (Snyder, Map Projections Used by the U.S. Geological Survey, 1986):

$$Easting = k_0 * N \\ * \left(A + (1 - T + C) * \frac{A^3}{6} \right. \\ \left. + (5 - 18 * T + T^2 + 72 * C - 58 * Excentricidad) * \frac{A^5}{120} \right) + E_0 \quad (39)$$

$$Northing = k_0 * (M + N * \tan(LatitudRad) \\ * \left(\frac{A^2}{2} + (5 - T + 9 * C + 4 * C^2) * \frac{A^4}{24} \right. \\ \left. + (61 - 58 * T + T^2 + 600 * C - 330 * WGS84_E^2) * \frac{A^6}{720} \right)) \quad (40)$$

$$Northing += N_0 \; si \; Lat < 0 \; (hemisferio \; sur) \quad (41)$$

Así, con este complejo proceso se podría obtener la localización de un punto en el mundo del que ya conocíamos su latitud y longitud. Es decir, lo que que aquí hemos visto es meramente un proceso de traducción a un sistema de referencia distinto. Se puede comprobar su corrección hallando la posición UTM para la tupla $(latitud, longitud)$ a través de este algoritmo y un conversor como (Montana State University, s.f.), de tal forma que ambas fuentes deberían devolver el mismo resultado.

Sin embargo, dada una imagen sólo se conoce la latitud y longitud de un punto de la misma, que se supone que pertenece al punto principal de la imagen rotada (revierte la rotación de la cámara en el momento de captura). Para el resto de puntos hay que hallar la forma de encontrar su posición en el sistema UTM.

Como ya sabemos, la diferencia entre puntos UTM se expresa en metros, por lo que si conocemos la distancia, también en metros, entre dos puntos de la imagen, podemos hallar su posición UTM. Nótese como este proceso sería mucho más complejo si trabajáramos con valores de latitud y longitud, debido a que cada una de las líneas que representa la latitud tiene un diámetro diferente, y por lo tanto, no se puede formular una equivalencia directa entre grados y metros uniforme a lo largo de todo el globo.

Sin embargo, dada una coordenada UTM y la distancia en metros que se estima en apartados anteriores, se puede hallar una nueva posición UTM de la siguiente manera:

$$Utm_{Easting} = Utm_{Easting_{Princ.p.}} + \frac{Meters}{Pixel} * (Pixel_x - Principal\ point_x) \quad (42)$$

$$Utm_{Northing} = Utm_{Northing_{Princ.p.}} + \frac{Meters}{Pixel} * \left(Pixel_y - Principal\ point_y\right) \quad (43)$$

Para poder hacer uso de esta fórmula es necesario que la imagen esté dispuesta tal que el norte de la tierra quede justo en la parte central y superior de la imagen y el este se halle justo a la derecha y en el centro vertical de la imagen. Se ha propuesto esta formulación teniendo en cuenta que el origen del sistema de coordenadas de la imagen se encuentra justamente en la esquina inferior izquierda desde la posición del observador, algo que no necesariamente es así.

Nótese como pueden existir casos degenerados que no se contemplan en la aplicación por simplificación, que se producen si el terreno examinado se encuentra justamente entre dos zonas UTM completamente diferentes. Se notarían todos los puntos de una imagen como si estuvieran en la misma zona cuando realmente no sería así. También cabe destacar que la casuística que produce este error es bastante inverosímil, ya que no se trata de que la parcela o terreno a examinar se encuentre entre dos zonas diferentes, sino que una misma imagen incluya porciones de terreno de dos zonas diferentes. Esto es así porque para cada imagen sólo se traducen los valores de latitud y longitud del centro de la misma, y por lo tanto, si dos imágenes están en zonas completamente diferentes el resultado seguiría siendo válido.

Llegados a este punto se ha obtenido una herramienta fundamental para la georreferenciación, pero nos quedan otras muchas variables por examinar para conseguir hallar el resultado esperado. En realidad, ya disponemos de tres herramientas de las cinco que podemos distinguir en todo este proceso:

- Estimación de la distancia en metros que representa cada píxel.
- Eliminación de la distorsión de ojo de pez de la imagen para evitar una georreferenciación incorrecta en los extremos de la misma.
- Obtención de coordenadas UTM (proceso de traducción entre los dos sistemas de referencia ya conocidos).
- Estimación del vector de desplazamiento *forward* en la imagen, que nos permitirá alinear el punto central superior de la imagen (desde la perspectiva del usuario) con el norte de la Tierra.

- Rotación de la imagen para eliminar las condiciones de captura de la misma sobre el dron. La herramienta anterior no es más que un subconjunto de este proceso, dado que la alineación con el norte vendrá dada por uno de los tres ángulos que trataremos más tarde.

Posicionamiento del norte y este de la tierra en la imagen

El primer párrafo de este apartado se destina, brevemente, a distinguir los diferentes tipos de norte que se pueden considerar en la Tierra, para futuras referencias. Cuando hablemos de norte en este proyecto nos referiremos siempre al polo norte geográfico (Magnetic North vs Geographic (True) North Pole, 2017), que se halla justo en el punto norte en el que convergen los meridianos de la tierra. También podemos encontrar el norte magnético, que es hacia donde señalaría una brújula (su posición varía a lo largo del tiempo), y el norte geomagnético. Por supuesto, las coordenadas geográficas calculadas por el GPS de nuestra cámara multiespectral toman el verdadero norte.

Por otro lado, se desea conocer **dónde se sitúa el norte de la Tierra en la imagen porque se debe conocer la dirección en la que se debe avanzar** (tanto norte como este) para calcular las coordenadas UTM del resto de puntos en la imagen, tal y como se muestra en la Ilustración 66.

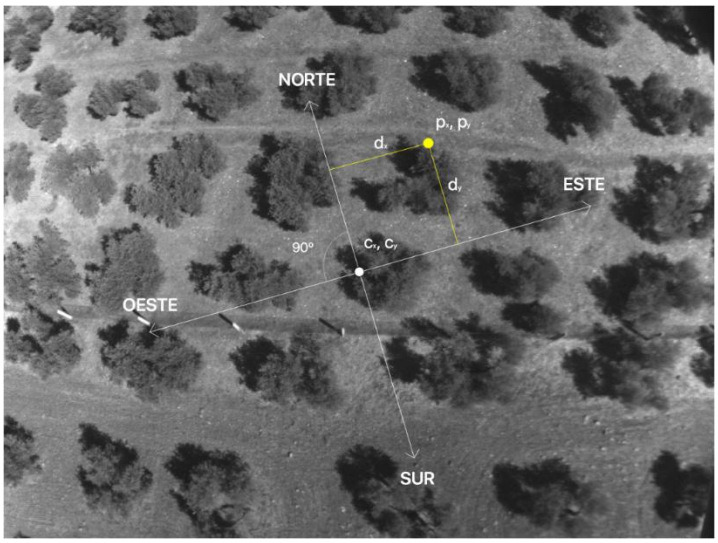


Ilustración 66. Representación del punto principal de una imagen y otro punto cualquiera del que se desea conocer su localización en el sistema de coordenadas UTM.

Se ejemplifica así en esta imagen lo que podríamos llamar vectores *forward*, tanto norte como este. Dados estos dos vectores y el punto principal de la imagen, se origina un nuevo sistema de referencia a partir del cual se puede obtener una posición

relativa respecto del origen de dicho sistema (punto principal) para aquellos píxeles de interés de la imagen. En este caso se representa el centro del sistema de referencia en el centro de la imagen, pero no tiene por qué ser así necesariamente.

Por otro lado, la problemática que en la imagen se plantea, donde se debe hallar d_x y d_y, no se da en nuestra aplicación, dado que para evitar complicar los cálculos se intenta revertir la rotación de la imagen, de tal forma que el vector $Forward_{NORTE}$ pasa a ser $(0,1)$ y $Forward_{ESTE}$ será $(1,0)$.

Una vez visto el resultado que se debería hallar, ¿cómo se obtienen estos dos vectores? La imagen presenta en sus metadatos tres ángulos, *yaw*, *roll* y *pitch*, normalmente empleados en el área de la aviación y naútica, y en última instancia, en el área de los drones. Dichos ángulos son también conocidos como ángulos de **Tait-Bryan**, y normalmente se asocian con los ángulos de Euler, aunque estos últimos utilizan el mismo eje para la primera y tercera rotación básica.

¿Qué tipo de rotación representa cada ángulo? Para explicar todo esto utilizaremos el ejemplo más común: un avión que controla sus rotaciones a partir de las alas y la cola.

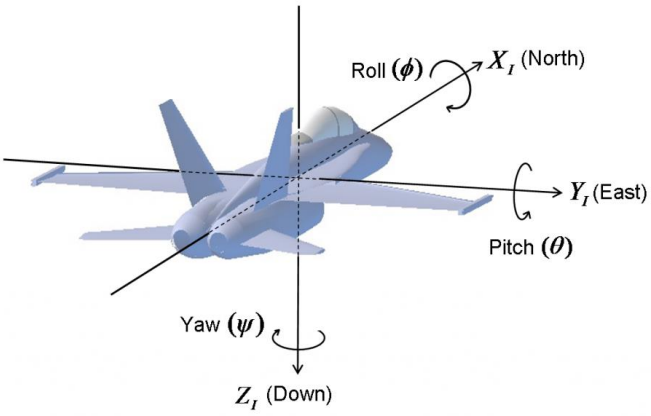


Ilustración 67. Representación de las tres rotaciones propuestas en función de los ángulos de Tait-Bryan sobre un avión. Fuente: (CHRobotics, s.f.).

Nótese como hay que adaptar este sistema de coordenadas del dron al de la imagen, teniendo en cuenta que nosotros trabajamos en un espacio 2D, X e Y. Se propone por lo tanto un sistema de coordenadas para la imagen como el de la Ilustración 68.

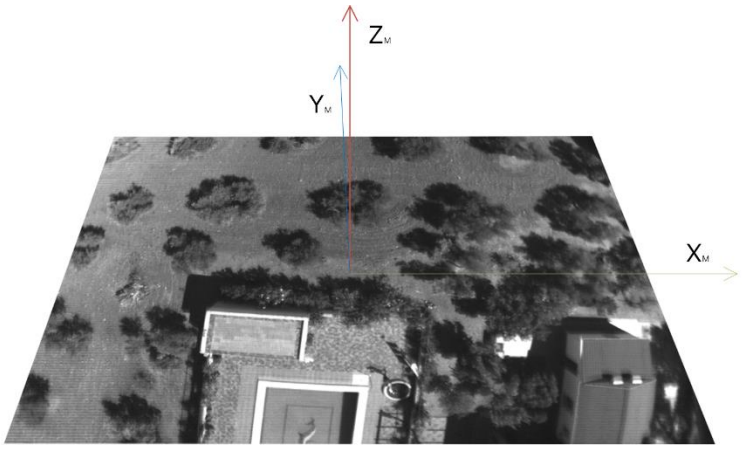


Ilustración 68. Sistema de coordenadas de la imagen.

- El ángulo ***yaw*** es el que más nos interesa para nuestra aplicación, ya que representa la rotación sobre el eje Z del sistema de la imagen. De esta forma, será la principal transformación que nos permita alinear la imagen con el norte de la Tierra.

  En el ejemplo propuesto del avión, sería un movimiento de lado a lado respecto de su centro de gravedad. Justo al contrario que con los ángulos con los que hemos trabajado hasta ahora, una rotación que utiliza un ángulo positivo implica una rotación en sentido horario. Podríamos hablar entonces de un desplazamiento respecto del eje –Z en la imagen (ver Ilustración 68).
- El ángulo ***pitch*** es la rotación que se produce alrededor del eje Y propuesto en la Ilustración 67, generando un movimiento hacia arriba o hacia abajo del avión aquí representado. Un ángulo positivo levantaría la parte delantera del avión y lo contrario sucedería con la cola. En este caso se trata de una rotación en sentido antihorario.

  Una aplicación de este tipo de movimiento es la de ayudar al propio despegue o aterrizaje del modelo en cuestión, ya sea un avión o un dron.
- El último de los ángulos, ***roll***, representa una rotación respecto del eje X del modelo representado anteriormente, en sentido horario (y por lo tanto, podríamos hablar de –Y en el sistema de la imagen). El movimiento vendría a ser una inclinación lateral que en el ejemplo del avión está influenciado por las alas.

Normalmente este ángulo suele ser muy bajo en el dron, ya que se utilizan a menudo lo que se conocen como *gimbal* o estabilizadores para las cámaras (Pix4D, 2018).

En la propia web del software de fotogrametría Pix4D se muestran una serie de casuísticas y la orientación que tendría la cámara en esos casos (Pix4D, 2018). Algunos ejemplos:

- *Yaw* = *roll* = *pitch* = 0. La cámara está justamente mirando al suelo de forma perpendicular a este, con el norte en la parte superior de la imagen. Cuando la cámara mira al suelo de esta manera se suele hablar de *nadir*. Situación idónea.
- *Yaw* = 90, *roll* = *pitch* = 0. La cámara es *nadir* pero en la parte superior de la imagen se encuentra el este.

Se destacan estos ejemplos porque tienen ángulos *roll* y *pitch* nulos, tal y como sería nuestra situación ideal. Sin embargo, en la realidad esto no es así, y por pequeño que sea el ángulo (aunque en determinados casos alcanza unos cuantos grados), se debe tener en cuenta a la hora de rectificar la rotación de la imagen.

Otro detalle que debe tenerse en cuenta respecto de estos ángulos es tanto el signo como el rango en el que se mueven. En principio sí que se definen los rangos por convención (*yaw*: 2π, *roll*: π, *pitch*: 2π), pero no sucede lo mismo para los signos, por lo que habrá que examinarlos con cautela.

Una vez que comprendemos toda la terminología y conceptos relacionados a estos ángulos, ¿cómo obtenemos los vectores que más arriba se proponen? Como comentábamos, hemos utilizado esos vectores con el fin de comprender la problemática, pero todo se resuelve mucho más fácil si en lugar de calcular dichos vectores se rota la escena representada para situar el norte y el este justo donde queremos en la imagen.

Se obtienen así 3 matrices, una por cada tipo de rotación, de las que comentaremos algún detalle:

- *Yaw*: sabemos que se trata de la rotación que se produce respecto del eje Z del sistema de la imagen, y que se produce en sentido horario, por lo que la matriz de transformación que invierte dicha rotación es la siguiente:

$$R(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (44)$$

- *Pitch*: como ya hemos comentado anteriormente, es una rotación respecto del eje X que sigue justamente el sentido de las agujas del reloj. Por lo tanto, para aplicar la transformación inversa se define la siguiente matriz de transformación:

$$R(\beta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\beta) & -\sin(-\beta) & 0 \\ 0 & \sin(-\beta) & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (45)$$

- *Roll*: a partir de este ángulo se obtiene prácticamente la misma situación que para el ángulo *yaw*, con la excepción de que el eje respecto del cual se rota es el eje Y del sistema de la imagen.

$$R(\gamma) = \begin{pmatrix} \cos\gamma & 0 & \sin\gamma & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\gamma & 0 & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (46)$$

Desde que se comenzó a hablar de estos ángulos, se trataba el primero de ellos, *yaw*, como el más importante, dado que es el que más cambios genera en la imagen, y esto se debe a que afecta tanto a los valores de x como a los de y. Sin embargo, también se aplican otras transformaciones que afectan a la profundidad de la imagen, a priori desconocida. ¿Es esto un problema?

Sabemos perfectamente que los ángulos *pitch* y *roll* deben ser en la mayoría de casos valores muy pequeños, debido a que están intímamente relacionados con la estabilidad del dron (a priori controlada con *gimbal*) y con el proceso de despegue / aterrizaje. Es más, el valor *roll* debería ser 0. Sin embargo, el valor *pitch* no debe ser ignorado porque, aunque pequeño, puede alcanzar varios grados y puede generar cierta rotación en la imagen que se debe eliminar para elaborar una georreferenciación precisa.

Ahora bien, el principal problema de aplicar estos ángulos a la imagen viene cuando sus valores son excesivamente grandes para lo que se considera habitual. Por ejemplo, podemos considerar imágenes con bastante inclinación respecto del eje

X. Esto genera transformaciones bastantes extrañas en la imagen (aunque correctas), hasta el punto de escalarse por un factor bastante pequeño en algún eje (siempre menor que 1, por supuesto). Una escena de este tipo se muestra en la Ilustración 69.

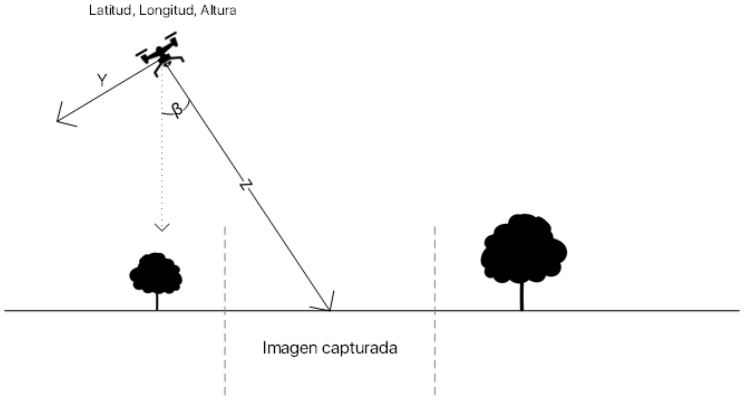


Ilustración 69. Escena donde el dron alcanza un ángulo pitch relativamente elevado, de tal forma que la escena difiere en gran medida de lo que se obtendría con una orientación nadir.

Cuando rotamos la escena capturada, lo que esperamos es ver justamente lo que se obtendría si el dron tuviera en sus tres ángulos el valor 0, es decir, mira exactamente hacia el suelo de forma perpendicular a este (*nadir*) y apunta hacia el norte, que quedaría en la parte superior de la imagen. Si la escena degenera en algo como lo que se representa un poco más arriba, entonces la imagen resultante se deformaría en gran medida (escalado de factor menor que 1).

Es por ello que cuando se dan estas situaciones el resultado obtenido es poco útil pero representa realmente lo que se quiere obtener. Entre otras cosas, nos sirve porque nos está diciendo que la latitud y longitud del dron está muy alejada de la posición de la escena.

Por otro lado, situaciones así son prácticamente nulas en nuestros conjuntos de datos, y si se diera esta situación tampoco se está realizando ninguna operación especial, dado que en este proyecto lo que se pretende es desarrollar y poner a prueba ciertos algoritmos relacionados con el control de una plantación y no elaborar una aplicación final que descarte todo tipo de situaciones extremas.

Otro tema importante que se debe tratar es ¿respecto a qué punto se aplica la rotación? Por simplificación, se podría considerar el centro exacto de la imagen. Sin embargo, sabemos que una lente tendrá asociada un eje óptico a partir del cual se puede obtener el punto principal de una imagen, (c_{p_x}, c_{p_y}).

Se ha considerado por lo tanto que deba obtenerse para cada píxel su posición relativa respecto de dicho punto principal y ejecutar entonces la transformación de rotación. Una vez hecho esto, se deben añadir de nuevo las coordenadas del punto principal. Podemos definir así el resto de transformaciones que nos quedan para completar el proceso:

$$T_1(-c_p) = \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{47}$$

$$T_2(c_p) = \begin{pmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{48}$$

De esta forma, si se combinan todas las transformaciones geométricas hasta ahora vistas se obtiene la siguiente matriz de transformación compuesta:

$$\begin{aligned} &Rotación\ (p_i) = T_2(c_p) * R(\alpha) * R(\beta) * R(\gamma) * T_1(-c_p) * p_i \\ &= \begin{pmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} \cos\alpha * \cos\gamma - \sin\alpha * \sin(-\beta) * \sin\gamma & -\sin\alpha * \cos(-\beta) & \cos\alpha * \sin\gamma + \sin\alpha * \sin(-\beta) * \cos\gamma & 0 \\ \cos\alpha * \cos\gamma - \sin\alpha * \sin(-\beta) * \sin\gamma & \cos\alpha * \cos(-\beta) & \sin\alpha * \sin\gamma + \cos\alpha * \sin(-\beta) * \cos\gamma & 0 \\ -\sin\gamma * \cos(-\beta) & \sin(-\beta) & \cos(-\beta) * \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &* \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ 0 \\ 1 \end{pmatrix} \end{aligned} \tag{49}$$

En esta transformación, como ya se ha anotado antes, se tiene en cuenta una tercera coordenada, z, que no está presente en la imagen y que en cualquiera de los casos nos es indiferente, dado que se especifica un valor cualquiera, ignorándose el resultado para dicha dimensión.

El proceso visto hasta ahora sería un proceso genérico para cualquier tipo de imagen que disponga de estos tres ángulos que hemos explicado. En el siguiente apartado trataremos el caso más específico de las imágenes multiespectrales, que forman parte de los datos de entrada de esta aplicación.

Rectificación de rotación en imágenes multiespectrales

Todo el apartado anterior, aunque genérico, se elabora pensando en este tipo de imágenes con el que hemos estado trabajando hasta ahora, que como ya sabemos, disponen en sus metadatos de los tres ángulos que acabamos de ver. Sin embargo, se puede comprobar fácilmente que estos ángulos son incorrectos para prácticamente todas las imágenes. Dejando a un lado los ángulos asociados a la estabilidad, nos centraremos en *yaw*, relacionado con la alineación de la imagen con el norte de la Tierra, que es la propiedad que más nos interesa.

Para comprobar el error de dicho ángulo se exponen las imágenes multiespectrales de la Ilustración 70 junto al valor *yaw* contenido en los metadatos.

Ilustración 70. Imágenes multiespectrales junto al ángulo yaw presente en sus metadatos.

La problemática que acabamos de presentar no se podría terminar de comprender sin señalar claramente dónde se encuentra el norte dentro de las imágenes. Nuestro conjunto de datos tiene una ventaja para examinar esta situación, y es que además de la propia vegetación se representan ciertas estructuras como el edificio que aquí aparece. A diferencia de esa vegetación, que varía a lo largo del tiempo y es bastante difícil de seguir entre imágenes, estructuras como esta podemos seguirlas fácilmente entre capturas y también sobre un mapa. En concreto, este conjunto de datos es especialmente útil para comprobar todo esto que estamos tratando, debido a que la estructura que se representa se alinea completamente con el norte, como muestra la Ilustración 71.

Ilustración 71. Vista aérea de la estructura que aparece en las imágenes multiespectrales de la Ilustración 70, junto a la indicación del norte de la Tierra (Google, s.f.).

Se puede ver claramente como los ángulos proporcionados por los metadatos no se corresponden a la situación aquí presentada, teniendo en cuenta que el norte lo indica la brújula en la parte superior derecha. Por ejemplo, la segunda de las imágenes multiespectrales coincide prácticamente con la vista aérea que acabamos de presentar, y, sin embargo, se proporciona un ángulo mucho mayor que 0º (recordemos que la rotación *yaw* gira en sentido horario, aunque en ninguno de los casos funcionaría el ángulo aportado).

Se observa además que existe cierta relación entre los ángulos originales y los ángulos que podrían ser la solución al problema de la rotación. De esta manera, se formula un algoritmo (ver Ilustración 72) que rectifica el ángulo original y nos permite eliminar la rotación de una imagen multiespectral de una forma aproximada:

Algorithm 4: Algorithm that rectifies the yaw angle in degrees according to a certain calibration

```
1 function GetRectifiedAngle (angle);
  Input  : angle in degrees as a floating value
  Output: rectified angle in degrees
2 newAngle ← 0
3 if angle < 0 then
4 |   newAngle ← 180 − angle
5 |   if abs(angle/90) >= 1 then
6 |   |   newAngle ← newAngle + 90
7 |   end
8 end
9 return − (newAngle + 125)
```

Ilustración 72. Método de rectificación de los ángulos contenidos en los metadatos de las imágenes multiespectrales.

Para llegar hasta este algoritmo se debe disponer de varias imágenes con rotaciones bien conocidas, como las anteriores, y a partir de ahí formular transformaciones que, como bien vemos, hacen distinción entre ángulos negativos y positivos. A su vez, 125º es una rotación base que se aplica a todas las imágenes, dado que por defecto, todas aquellas que tienen ángulos de rotación *yaw* positivos quedarían con el sur prácticamente alineado con la esquina superior izquierda, como bien muestra la Ilustración 73.

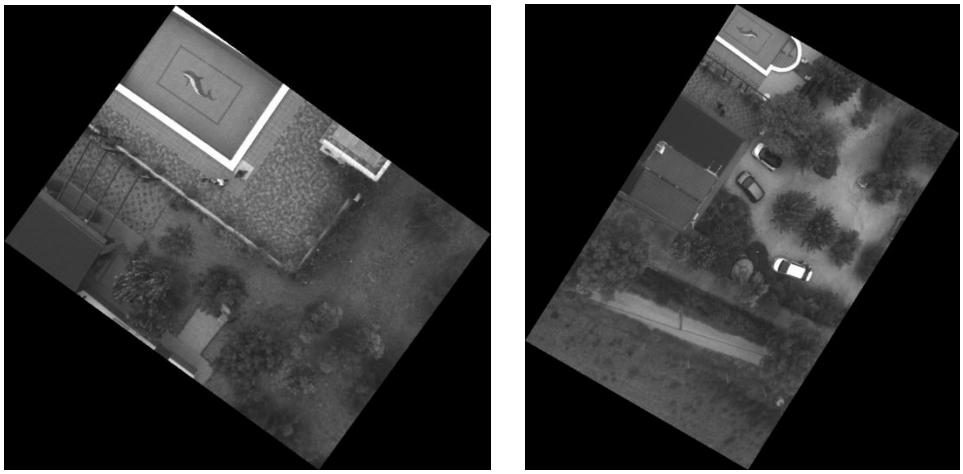


Ilustración 73. Imágenes con ángulos rectificados sin aplicar la corrección de 125º.

Nótese como ambas se orientan hacia dicha esquina, aunque la que tenía el ángulo negativo está justamente volteada. Por lo tanto, si se aplica el algoritmo anterior al completo se obtienen imágenes como las de la Ilustración 74.

Ilustración 74. Rectificación de las imágenes con el algoritmo anteriormente presentado.

- 143.96º pasaría a ser -268.96º y giraría hacia la derecha.
- -147.86º pasaría a ser -542.86º y giraría también hacia la derecha.

Se puede observar claramente como existe cierto fallo, intolerable para la georreferenciación, pero es ciertamente mejor que la situación de partida. En cualquiera de los casos, esta situación se mantiene para el resto de conjuntos de datos que tenemos a nuestra disposición, aunque normalmente se tratan de cultivos con pocas estructuras a su alrededor, y por lo tanto, la comprobación de su corrección es más difícil. Para afirmar que esta transformación funciona en el resto de conjuntos

de datos hay que fijarse en estructuras aisladas que se representan en el terreno, tales como postes eléctricos, con la ayuda de algún mapa como el ilustrado anteriormente (Google, s.f.).

Por último, cabe destacar cómo se rota la imagen, ya que el algoritmo no deja de ser un *mapeo* como el que se efectuó para eliminar la distorsión, con la diferencia de que en este caso ha sido implementado por completo para controlar el nuevo tamaño de la imagen. Por lo tanto, habrá que resolver ciertos problemas de los que nos liberaraba OpenCV:

- El nuevo tamaño de la imagen se obtiene aplicando la transformación M_i, una composición de rotaciones, a las esquinas de la imagen que se debe rectificar. A partir de estos puntos transformados se pueden obtener los correspondientes mínimos y máximos, que nos dan un área suficiente para contener la nueva imagen.

- Se encuentra además el inconveniente de que al calcular la nueva posición de un punto $p_{x,y}$, los valores resultantes son números flotantes, y hay que hacer alguna operación de redondeo. Esto implica también que existirán píxeles de color nulo distribuidos a lo largo de la imagen resultado, describiendo un determinado patrón. La solución que aquí se propone es un filtro gaussiano, de tal forma que los píxeles nulos toman un color promedio de los vecinos. Dado que se está difuminando la imagen, se debe aplicar un núcleo de tamaño bastante pequeño.

- Esto último también tiene su influencia en el tamaño de la imagen, ya que si se redondea un valor flotante es posible que se intente asignar una intensidad de píxel a una posición fuera de la imagen.

Parte de lo que hemos aprendido en este apartado se debe utilizar necesariamente en el siguiente, que se corresponde al almacenamiento de información de cada punto de la imagen, entre la que se encuentra su posición en el sistema de coordenadas UTM. Sin embargo, para esta funcionalidad queda poco más que implementar, salvo aquellas vistas correspondientes a la aplicación gráfica.

Almacenamiento de datos georreferenciados

Este apartado gira en torno a los dos tipos de datos que se van a asociar a cada píxel: intensidad en cada imagen del modelo multicapa y localización en el sistema de coordenadas UTM.

Antes de comenzar a explicar cómo se calculan los dos tipos de datos citados, es necesario hacer una reflexión sobre la imagen que se va a utilizar como entrada de esta funcionalidad, teniendo en cuenta siempre que el modelo multicapa ya está registrado.

- Se parte de la imagen con su rotación rectificada, en cuyo caso habrá que incluir en el archivo sólo píxeles no nulos. Problema: cómo se distingue un píxel de intensidad cero que pertenece a la imagen y uno que pertenece al fondo de la imagen (nulo). Esta consideración sí que tiene la ventaja de tener ya todas las intensidades calculadas si se rectificaran todas las capas (que no es lo que antes se propone), sin embargo, el problema llegaría de nuevo si nos piden un nivel de muestreo inferior al píxel. Habrá que interpolar en cualquier caso, y se presenta además la dificultad de cómo hacer el barrido de la imagen real (descartando fondo) para hacer la distinción de esos puntos que no se corresponden necesariamente a píxeles.

- **Se parte de que todas las capas están registradas, y se le aplica a un punto de la imagen** (el mismo en todas ellas) **una transformación definida por** $\boldsymbol{M_i}$. Su nivel de intensidad en todas estas capas debe ser interpolado (consume tiempo, teniendo en cuenta además que se debe comprobar si existen determinados vecinos: bordes de la imagen), pero en el caso de que nos pidan un nivel de precisión mayor (muestreo más frecuente), podemos hacerlo eficientemente (sólo hay que generar más puntos entre píxeles en dirección horizontal y vertical).

Sea este segundo el enfoque finalmente elegido, hay que definir el método de interpolación de niveles de intensidad, así como la obtención de la coordenada UTM de un píxel:

- Para obtener el nivel de intensidad de un punto que no pertenece necesariamente a un píxel se ha empleado una **interpolación bilineal**, que viene a ser una extensión de la interpolación lineal en un espacio 2D. Por lo

tanto, se deben tener en cuenta los cuatro vecinos más cercanos al punto p_i, tal y como refleja el esquema de la Ilustración 75.

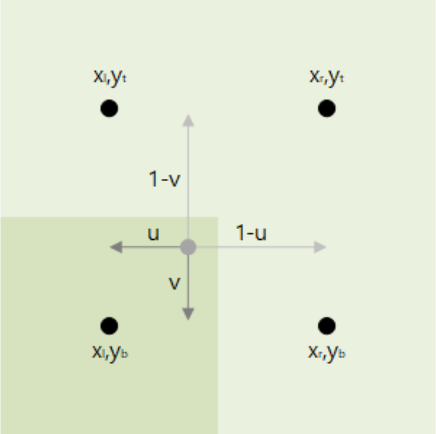


Ilustración 75. Esquema de selección de vecinos y cálculo de distancias en la interpolación bilineal.

Sin establecer ninguna comprobación para $p_{x,y}$ de si sus píxeles vecinos existen o no, el algoritmo de interpolación bilineal puede quedar como se muestra en la Ilustración 76.

Algorithm 5: Bilinear interpolation to calculate the gray value at a certain position of the image

```
1  function GetGrayValue (x, y);
   Input  : position to be sampled
   Output: gray value for that position
2  left ← round(x − 0.5);
3  right ← round(x + 0.5);
4  top ← round(y + 0.5);
5  bottom ← round(y − 0.5);
6  u ← x − left;
7  v ← y − bottom;
8  gray ← 0;
9  gray ← (1 − u) ∗ (1 − v) ∗ image[l][b] + u ∗ (1 − v) ∗ image[r][b];
10 gray ← gray + (1 − u) ∗ v ∗ image[l][t] + u ∗ v ∗ image[r][t];
11 return gray
```

Ilustración 76. Interpolación bilineal sin comprobación de existencia de vecinos.

- Cálculo de la coordenada UTM de un punto $p_{x,y}$. Una vez se rota la imagen multiespectral, se supone que la coordenada UTM calculada a partir del punto $(latitud, longitud)$ de los metadatos se corresponde al punto principal de la imagen, que se mantiene invariable en todo este proceso de adecuación. Por tanto, para un punto cualquiera, $p_{x,y}$, se supone conocida

su nueva posición (i, j), que debe ser el resultado de aplicar la matriz compuesta calculada en apartados previos, $M_{i.}$

$$(i, j, k, l) = M_i * (x, y, 0, 1) \quad (50)$$

Hay que tener en cuenta además que las coordenadas UTM se expresan en metros, y que, de forma estimada, conocemos la altura en la que se ha capturado una imagen, y por lo tanto, la cantidad de metros que son representados por píxel (*Ground Sample Distance* ó GSD). Nótese como la transformación no es tan inmediata como se puede suponer, teniendo en cuenta que:

- Al eliminar la distorsión de una imagen multiespectral, el número de píxeles se mantiene pero el número de metros a considerar se reduce sustancialmente. El área que se representaba en 1280 píxeles, ahora lo hace en $1280 + offset$ píxeles, de tal forma que en nuestra imagen rectificada sólo podemos ver una parte de toda esa área inicial.

  Por lo tanto, habrá que tener en cuenta en qué factor se debe reducir ese GSD. Para ello, es necesario observar que a partir de las esquinas de la imagen se obtiene una representación bastante fiable de la cantidad en que habrá que variar esa distancia en metros (la eliminación de la distorsión estira la imagen a partir de estos puntos). De esa forma, se puede calcular la nueva distancia tanto en X como en Y.

  Nótese además como la rectificación de la posición de un píxel en (x, y) no produce valores negativos sino positivos, ya que lo que indica la función *remap* es el píxel de la imagen distorsionada, (i, j), de donde se toma el valor de intensidad. Esto implica que se obtiene una distancia en X y en Y máxima menor que las dimensiones de la imagen original, 1280x960.

  Al hacer esta operación, lo que ocurre es que los píxeles representan una porción menor de terreno (menor área en los mismos píxeles), y por lo tanto, la GSD o distancia por píxel se reduce. El factor en el que se reduce se puede calcular como sigue:

$$xFactor = \frac{(x_{max} - x_{min})}{width} < 1 \quad (51)$$

$$yFactor = \frac{(y_{max} - y_{min})}{height} < 1 \quad (52)$$

$$GSD_{factor} = \max(xFactor, yFactor) \quad (53)$$

Se debe calcular la posición (i, j) de las cuatro esquinas de la imagen porque la transformación no es simétrica, dado que el punto principal c_p no se halla en el centro exacto de la misma. Asimismo, dado que el factor de escala del GSD debe ser menor, en el denominador debe aparecer la anchura de la imagen, que es necesariamente mayor que las distancias entre cualquier par de puntos generados por la función de eliminación de la distorsión, tanto en X como en Y.

El factor de escala en X e Y puede diferir, y de hecho lo suele hacer, mientras que nuestro GSD considera que se avanza la misma distancia en X e Y por cada píxel. Como factor representativo se escoge el mayor de ellos, aunque la diferencia entre ambos factores es muy pequeña.

- El punto principal de la imagen no sufre variación alguna en su posición al eliminar la distorsión, dado que es el origen de dicha transformación y se mantiene invariante.

Sin embargo, una vez se elimina la distorsión sí que se registran las capas multiespectrales, lo que implicaba un recorte de todas las capas. Por lo tanto, la posición del punto principal ya sí varía, aunque su traslación es bien conocida dado que viene determinada por la esquina inferior izquierda del rectángulo de corte, es decir, su posición (un rectángulo se define mediante una posición y sus dimensiones).

Así, dado un rectángulo R, se puede obtener la nueva localización del punto principal en la imagen resultante como sigue:

$$c_{np} = c_p - Rectangle.posición \quad (54)$$

Nótese como esta última problemática no solo influye aquí sino también en el anterior apartado, donde se rectificaba la rotación de captura de la imagen.

Una vez se plantean todas las correciones necesarias, se debe calcular la coordenada UTM de cada punto de interés de la imagen. Se habla de punto y no píxel ya que una de las posibilidades que tiene el usuario es guardar información con una determinada precisión, que no tiene por qué corresponderse al píxel, además de la opción de guardar datos exclusivamente de los centros de masas una vez hallados los contornos. La operación de transformación de un punto es la siguiente:

$$newPosition = M_i * (x, y, 0, 1) \tag{55}$$

$$UTM_{easting} = utm_x + metersPixel * (newPosition_x - cp_x) \tag{56}$$

$$UTM_{northing} = utm_y + metersPixel * (cp_y - newPosition_y) \tag{57}$$

Ahora sí se formulan las operaciones teniendo en cuenta que el origen de la imagen se encuentra en la esquina superior izquierda desde nuestra perspectiva.

Si situamos las coordenadas que se obtienen en esta operación en un mapa, teniendo en cuenta que la altura de vuelo se estima en 40 metros debido a la altitud sobre el nivel del mar registrada y la elevación del terreno en el punto de captura, se obtiene una escena como la de la Ilustración 77 a partir de la Ilustración 78.

Ilustración 77. Posicionamiento de las esquinas de una imagen multiespectral en el mapa.

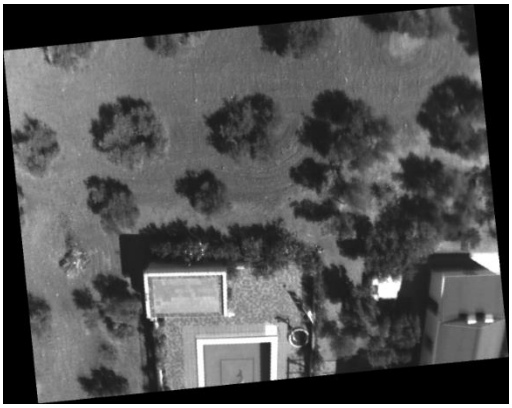


Ilustración 78. Imagen rotada de la que se extraen los datos georreferenciados.

El área de esquinas representadas con puntos rojos indica la salida de todo este proceso, mientras que el área amarilla indica el área de terreno que podemos ver en la imagen. La imagen se encuentra en perspectiva, por lo que este último área no tiene una correspondencia directa con nuestra salida, pero aún así, nos sirve para comprobar que el fallo de este algoritmo es bastante grande para el uso que se le quiere dar.

Se proponen así unos algoritmos que no se pueden comprobar correctos ni justo lo contrario, dado que los datos de entrada sufren de una serie de errores que impiden un resulto preciso. Sin embargo, sí que se han implementado para que la

discrepancia entre la información obtenida y la realidad fuera mínima, teniendo en cuenta las limitaciones de la información disponible. Cabe destacar que todas las utilidades relacionadas con la georreferenciación (todavía nos queda por ver alguna) presentan la misma acumulación de errores, que deriva en resultados como los que acabamos de ver.

Por último, y para terminar esta iteración, se muestra a continuación un listado de metadatos necesarios para implementar todas las funcionalidades aquí desarrolladas, y que aún no han aparecido en este documento (Tabla 12).

Tipo de metadato	Etiqueta de acceso	Descripción
Resolución X del plano focal	Exif.Photo.FocalPlaneXResolution	Valor flotante que indica los puntos por milímetro en el eje X.
Resolución Y del plano focal	Exif.Photo.FocalPlaneYResolution	Valor flotante que indica los puntos por milímetro en el eje Y.
Latitud	Exif.GPSInfo.GPSLatitude	Cadena que indica para cada unidad (grados, minutos y segundos) su valor y su divisor. Ejemplo: *38/1 3/1 2147483647/68387168*
Latitud (Punto cardinal)	Exif.GPSInfo.GPSLatitudeRef	Cadena con el punto cardinal de referencia: norte o sur.
Longitud	Exif.GPSInfo.GPSLongitude	Cadena que indica para cada unidad (grados, minutos y segundos) su valor y su divisor. Ejemplo: *38/1 3/1 2147483647/68387168*
Longitud (Punto cardinal)	Exif.GPSInfo.GPSLongitudeRef	Cadena con el punto cardinal de referencia: este u oeste.
Ángulo yaw	Xmp.Camera.Yaw	Valor flotante.
Ángulo pitch	Xmp.Camera.Pitch	Valor flotante.
Ángulo roll	Xmp.Camera.Roll	Valor flotante.

Tabla 12. Metadatos de una imagen multiespectral utilizados durante la totalidad de este apartado.

2.2.6.2 - Diagramas de secuencia

En este apartado se debe mostrar el flujo de operaciones hasta alcanzar el resultado en las múltiples funcionalidades que aquí se han expuesto: registro de imágenes, eliminación de la rotación de la imagen, configuración de altura estimada, persistencia de información de multiespectrales en fichero, etc, siempre partiendo de que la interacción se produce entre entidades del esquema Modelo-Vista-Controlador (de la Ilustración 79 hasta la Ilustración 83).

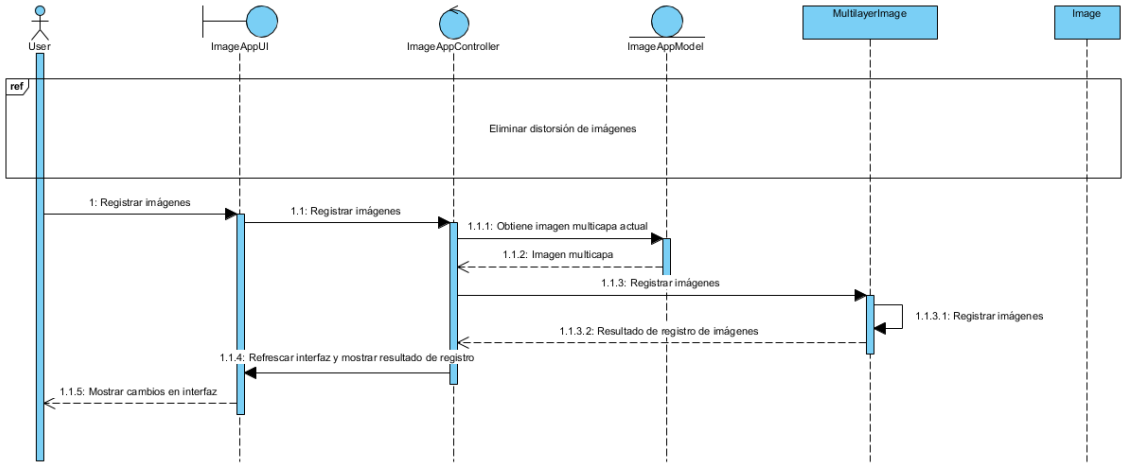


Ilustración 79. Operaciones hasta alcanzar el registro de imágenes.

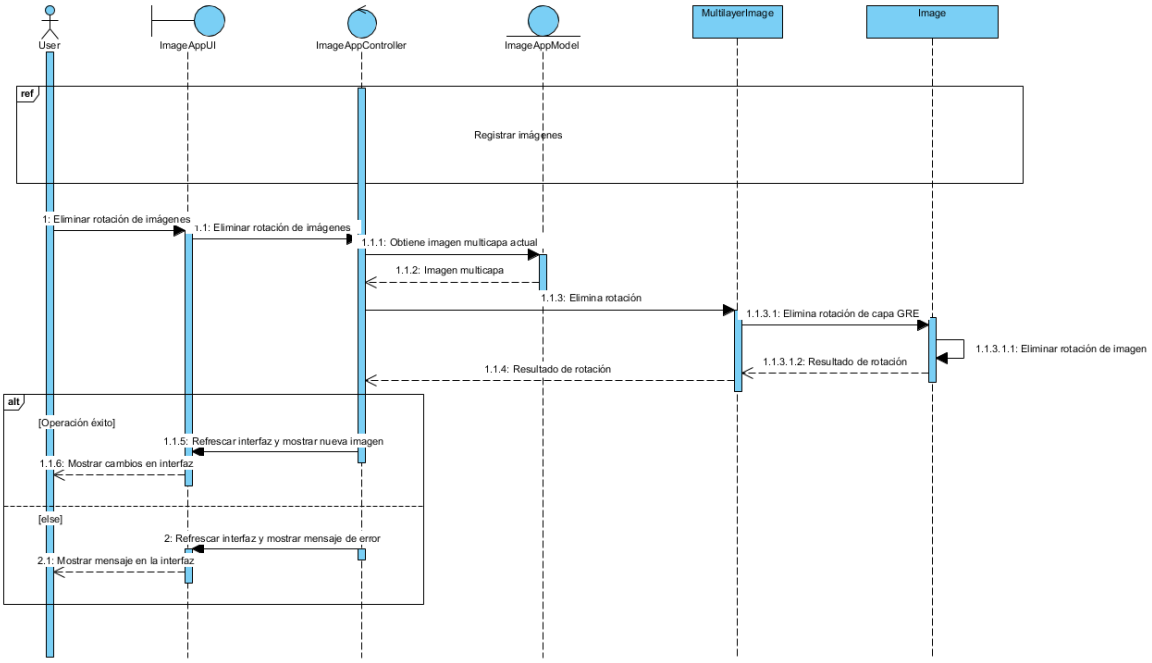


Ilustración 80. Operaciones a realizar hasta eliminar la rotación de una imagen multiespectral.

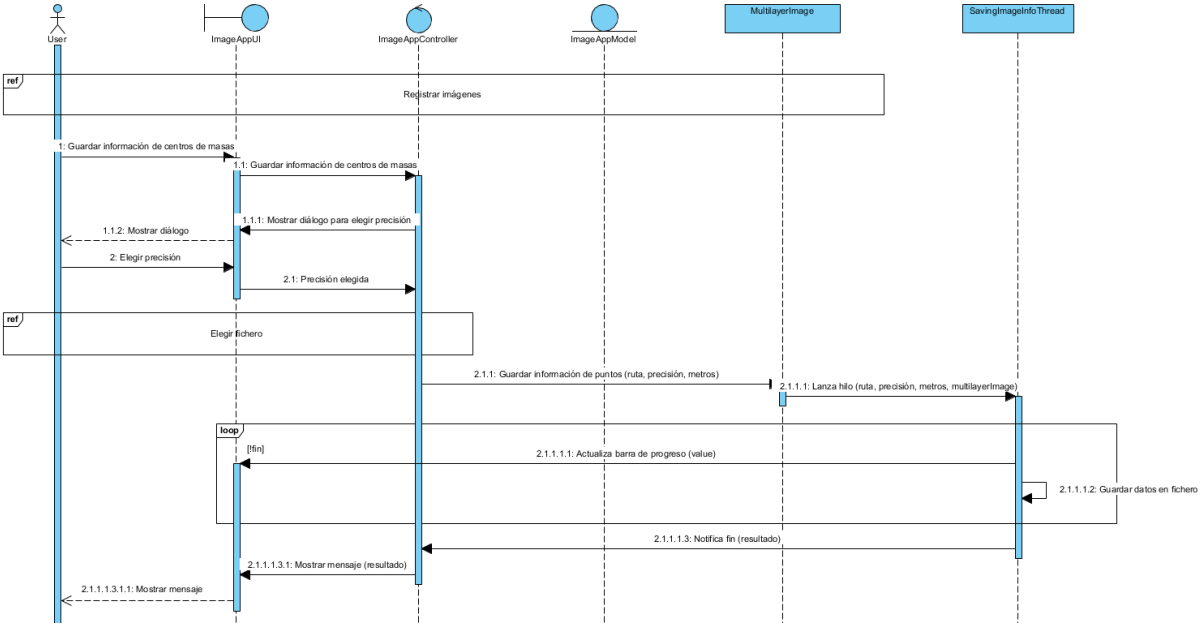


Ilustración 81. Flujo de operaciones hasta obtener un fichero con los datos de la imagen multiespectral georreferenciados.

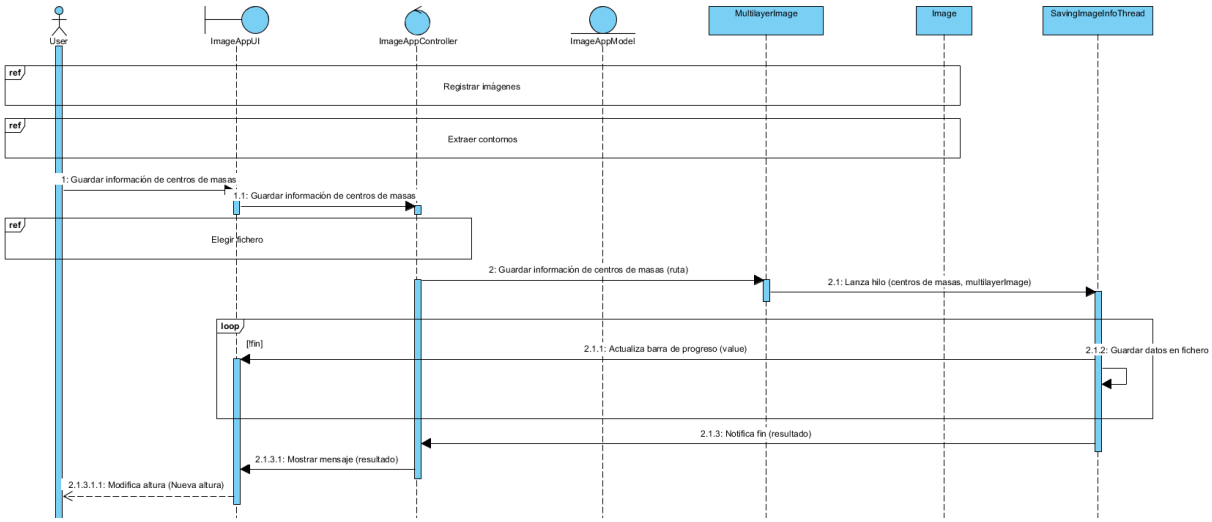


Ilustración 82. Flujo de operaciones hasta obtener un fichero con información exclusivamente de centros de masas de contornos.

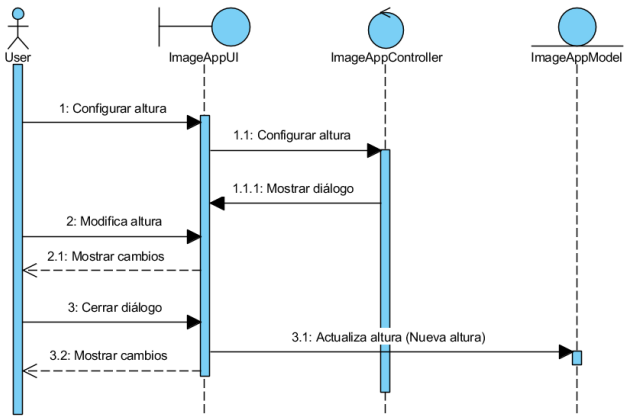
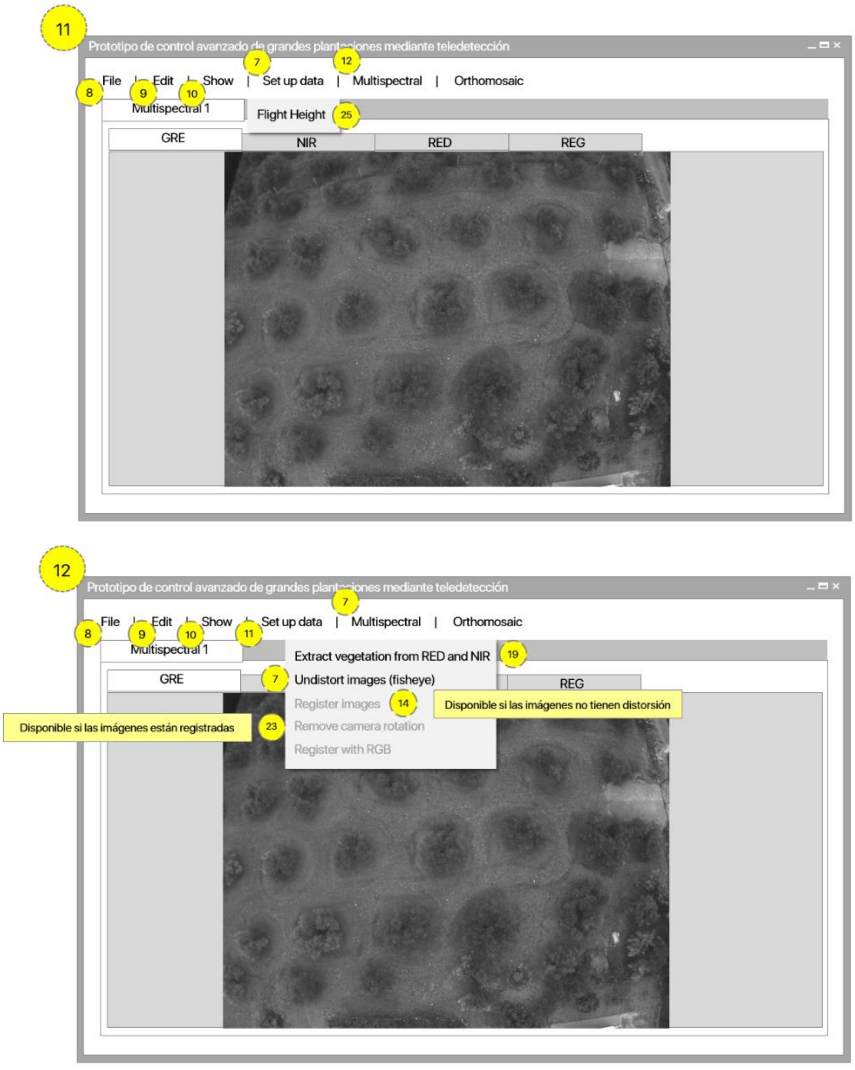


Ilustración 83. Modificación de la altura estimada del dron almacenada en la aplicación.

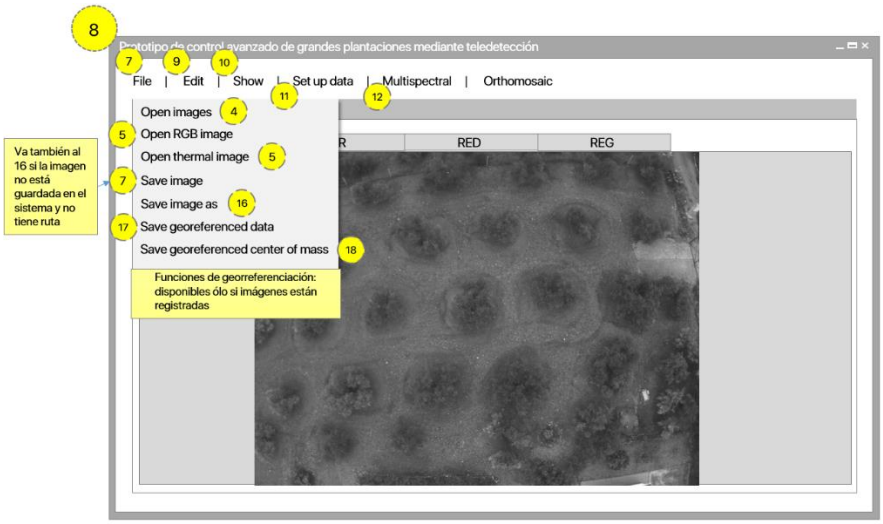
2.2.6.3 - Storyboards

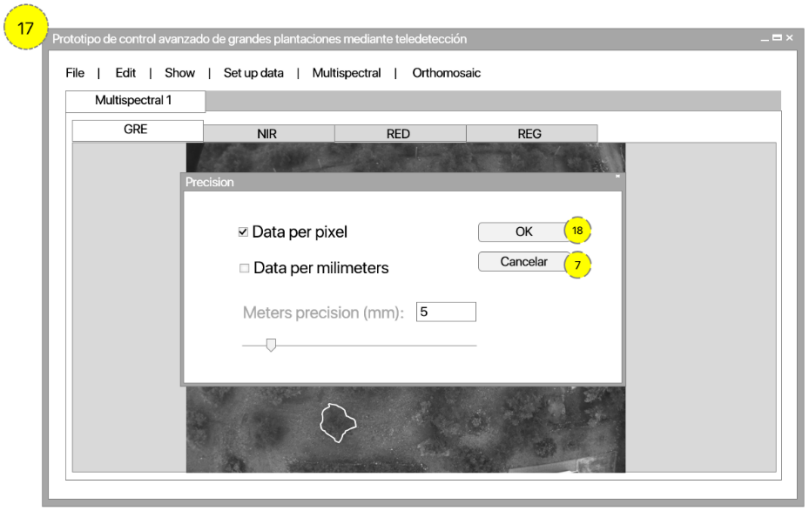
De la misma forma que para los diagramas de secuencia, las vistas involucradas en este largo apartado se representan, de forma esquemática, como se muestra en la Ilustración 84.

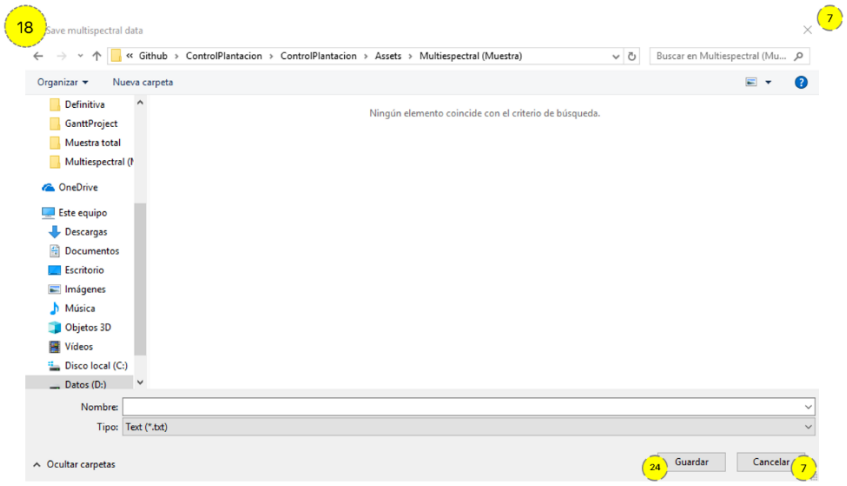


14
Prototipo de control avanzado de grandes plantaciones mediante teledetección
File | Edit | Show | Set up data | Multispectral | Orthomosaic
Multispectral 1
GRE
NIR
RED
REG
Image registration
Image 3 of 4
7 cuando termine proceso

23
Prototipo de control avanzado de grandes plantaciones mediante teledetección
8 File 9 Edit 10 Show 11 Set up data 12 Multispectral | Orthomosaic
Multispectral 1
GRE
NIR
RED
REG
Unrotated image


8
Prototipo de control avanzado de grandes plantaciones mediante teledetección
7 9 10
File | Edit | Show | Set up data | Multispectral | Orthomosaic
11 12
Open images 4
5 Open RGB image
Open thermal image 5
7 Save image
Save image as 16
17 Save georeferenced data
Save georeferenced center of mass 18
Funciones de georreferenciación: disponibles ólo si imágenes están registradas
Va también al 16 si la imagen no está guardada en el sistema y no tiene ruta
RED
REG


17
Prototipo de control avanzado de grandes plantaciones mediante teledetección
File | Edit | Show | Set up data | Multispectral | Orthomosaic
Multispectral 1
GRE
NIR
RED
REG
Precision
Data per pixel
Data per milimeters
OK
18
Cancelar
7
Meters precision (mm):
5


18
Save multispectral data
7
« Github › ControlPlantacion › ControlPlantacion › Assets › Multiespectral (Muestra)
Buscar en Multiespectral (Mu...
Organizar
Nueva carpeta
Definitiva
GanttProject
Muestra total
Multiespectral (I
OneDrive
Este equipo
Descargas
Documentos
Escritorio
Imágenes
Música
Objetos 3D
Vídeos
Disco local (C:)
Datos (D:)
Ningún elemento coincide con el criterio de búsqueda.
Nombre:
Tipo:
Text (*.txt)
Ocultar carpetas
24
Guardar
Cancelar
7

24
Prototipo de control avanzado de grandes plantaciones mediante teledetección
File | Edit | Show | Set up data | Multispectral | Orthomosaic
Multispectral 1
GRE
NIR
RED
REG
Image registration
Saving data in file...
7
cuando termine proceso

Ilustración 84. Vistas de la aplicación necesarias para llevar a cabo todas las operaciones expuestas en este apartado.

2.2.7 - Octava iteración

Como sucediera en alguna iteración previa, el objetivo aquí es reestructurar la aplicación para dar soporte a otras funcionalidades no incluidas hasta ahora, aunque muy posiblemente hayan aparecido a lo largo de las vistas presentadas del *storyboard*. Brevemente, las funcionalidades que se añaden en esta iteración son las siguientes:

- Se permite tener **varios conjuntos de imágenes multiespectrales abiertos al mismo tiempo** (ver Ilustración 85). Si antes teníamos un único gestor de pestañas para controlar hasta 5 imágenes del dispositivo Parrot Sequoia, ahora necesitamos un gestor de mayor entidad que controle al anterior.

  El objetivo de esta transformación es que los ángulos de rotación de las imágenes multiespectrales se realimenten, teniendo en cuenta que una parte de ellas es capaz de alinearse con el norte pero no así otras, que podrían considerarse valores anómalos. En cualquiera de los casos, para llevar a cabo esta operación se necesitaría abrir imágenes en masa para tener en cuenta muchas de ellas, de tal forma que el valor resultante sea representativo de todos los casos. Esto último, sin embargo, tiene difícil implementación en una aplicación gráfica orientada a la interacción con el usuario.

Si entramos en esta temática de ajustar la rotación de las imágenes, se deben tener en cuenta varios aspectos:

- No pueden considerarse los ángulos de rotación originales. Si retrocedemos a la iteración donde esto se trataba se exponía lo siguiente: al aplicar la transformación básica, todas las imágenes se orientaban hacia una misma esquina (y si no es así se podría voltear). Por lo tanto, se podría medir la desviación respecto de esa orientación, algo difícil de implementar porque no todas parten de la misma posición. Ejemplo: una imagen alcanza esa orientación con 90º, y otra con 160º.

- No se pueden realimentar los ángulos o sus desviaciones con una media, si así fuera los valores anómalos influirían en gran medida en esta, haciendo que no tuviéramos un subconjunto de imágenes mal orientadas sino todas ellas. Un método común para resolver esta problemática es RANSAC, que ya ha aparecido al inicio de este documento (J. P. Jhan, 2017), de tal forma que se describe un modelo matemático que nos permite ignorar esos valores anómalos.

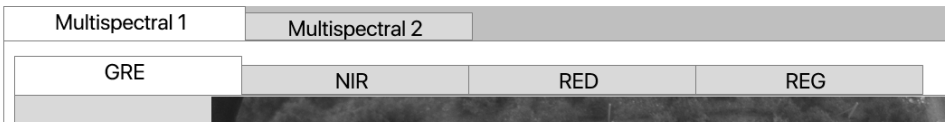


Ilustración 85. Integración de varios conjuntos de multiespectrales en la aplicación.

- Barras de progreso y sus correspondientes diálogos para informar del estado de la operación, bloqueando así la aplicación (ver Ilustración 86). Los procesos de registro y de acceso a fichero para guardar datos son costosos computacionalmente, especialmente este último, por lo que se incluye esta información gráfica que ya ha aparecido en iteraciones previas.

Ilustración 86. Barra de progreso durante el proceso de registro de imágenes.

2.2.8 - Novena iteración

Llegados a esta iteración, somos capaces de hacer un registro de imágenes multiespectrales bastante exacto, y también podemos considerar que somos capaces de georreferenciar sus puntos. Por lo tanto, a partir de esta base se propone añadir una serie de funcionalidades acorde a todo lo que ya hemos conseguido.

2.2.8.1.1 - Estudio de un mapa de disparidad

La elaboración de un mapa de disparidad se enmarca dentro de la visión estereoscópica, y básicamente representa la profundidad de los objetos de la escena. Si conocemos la posición x e y, y hallamos la profundidad z, se podría elaborar una escena 3D.

Se parte en esta sección del algoritmo de registro de imágenes que acabamos de presentar, el cual se presta a la elaboración de un mapa de disparidad como cualquier otra técnica de emparejamiento de imágenes. **No se pretende aquí obtener una representación 3D del terreno, que no es nuestro objetivo, sino estudiar hasta dónde se podría llegar con el algoritmo implementado**. Se dice bien que este no es un objetivo porque una escena 3D debería obtenerse a partir de imágenes de mayor resolución, como las imágenes de color, y no a partir de unas imágenes multiespectrales de dimensiones bastantes limitadas. Por lo tanto, vamos a estudiar qué se puede hallar a partir de la transformación geométrica global obtenida, que nos conducirá inevitablemente a demostrar que el resultado hallado en el registro es simplemente una aproximación.

Se procede por lo tanto a comentar las etapas de la visión estereoscópica (Fuertes García, 2019), que de acuerdo a Barnard y Fischler (Fischler, 1982), serían las siguientes:

- Adquisición de imágenes: en nuestro caso este proceso no depende de nosotros, aunque hay que tener en cuenta ciertos aspectos derivados de la captura de imágenes. Por ejemplo, ¿se han capturado en el mismo instante? Sabemos que sí se han obtenido al mismo tiempo pero existe un desplazamiento, a priori conocido, entre lentes.

  Aunque se dispone de hasta cinco imágenes tomadas en el mismo instante con pequeños desplazamientos, consideraremos sólo dos para simplificar la problemática. Es lo que se conoce como geometría bifocal.
- Modelado de la cámara: como ya sabemos, durante todo este proceso se utilizan dos vistas. La geometría de dos vistas se conoce como geometría epipolar, ya que a cada imagen se le asigna un epipolo. Se parte de la idea de que un punto de la escena 3D, M, puede ser visto en las dos imágenes como m_1 y m_2. También sabemos que una imagen es la proyección del espacio 3D en el plano de la imagen.

  Si partimos tan sólo de una imagen y un punto, m_1, no se puede conocer la profundidad del punto M dado que sólo conocemos la línea <C_1, m_1>, compuesta por múltiples puntos que podrían generar m_1. Necesitamos por lo tanto alguna restricción más.

  Dicha restricción vendrá dada por el centro óptico C_2. A partir de este punto y la recta anteriormente formada, <C_1, m_1>, se puede proyectar esta misma en la segunda imagen. Se obtiene así otra recta diferente en este otro plano, conocida como línea epipolar, y uno de los puntos de esa recta debe ser necesariamente m_2 (ver Ilustración 87).

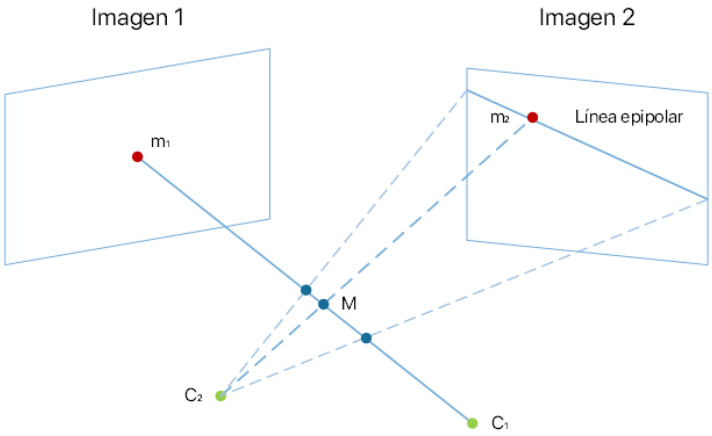


Ilustración 87. Proceso de obtención de la línea epipolar de la imagen 2.

La línea epipolar representa una serie de puntos que podrían ser m_2 en la imagen 2, que sabemos que es la correspondencia de m_1 en la segunda captura. También se podría obtener la línea epipolar de la imagen 1 de forma análoga al proceso que acabamos de ver. La línea epipolar, de forma general, reduce el problema del emparejamiento de puntos haciendo que la búsqueda pase de producirse en un espacio 2D a 1D.

Todos los componentes geométricos que acabamos de ver tienen una relación: son contenidos por un plano epipolar como sugiere la Ilustración 88.

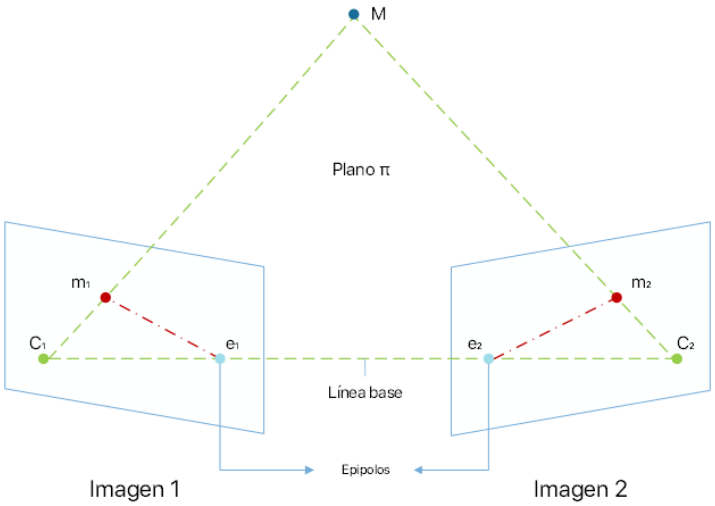


Ilustración 88. Relación geométrica que se establece entre todos los componentes vistos anteriormente para formar un plano.

Nótese como la intersección del plano π y una imagen crea una línea epipolar. Se añade además un nuevo concepto: el epipolo. Si se considerara otro punto en un espacio tridimensional, M', este generaría otro plano totalmente diferente, π'. Incluso así, el epipolo seguiría siendo el mismo.

Una vez comprendida la relación entre un punto 3D, M, y su correspondiente proyección en dos imágenes diferentes, m_1 y m_2, podemos formular una matriz de transformación como la que sigue:

$$\lambda w = KPS'M' \tag{58}$$

- K depende de 5 parámetros: dos factores de escala (x e y), las coordenadas del punto principal (también extraídas en procesos anteriores) y un factor de torcimiento.

- P depende de la distancia focal f.
- S' es una matriz de transformación 3D euclídea que depende de 6 parámetros: traslación y rotación tridimensional en X, Y y Z (2x3).

Se puede simplificar la notación en base a lo que hemos visto de la siguiente manera:

$$\lambda m = AM, \tag{59}$$

donde m será el punto proyectado y A será una matriz de tamaño 3x4.

Si se considera la geometría bifocal anteriormente planteada entonces se establece un sistema de ecuaciones como el que sigue:

$$\begin{cases} \lambda_1 M_1 = AM \\ \lambda_2 M_2 = AM \end{cases} \tag{60}$$

donde M se expresa en el mismo sistema de coordenadas para ambas ecuaciones.

Otro aspecto importante para hallar la correspondencia entre los puntos de dos imágenes diferentes es el sistema formado por las dos cámaras o los instantes entre los que se toman las capturas. El sistema ideal sería en el que se presenta en la Ilustración 89, donde los ejes ópticos de ambas lentes son paralelos, algo que sabemos que no sucede en nuestra situación. Si así fuera, el problema del registro se podría solventar con una simple translación de sistemas de coordenadas.

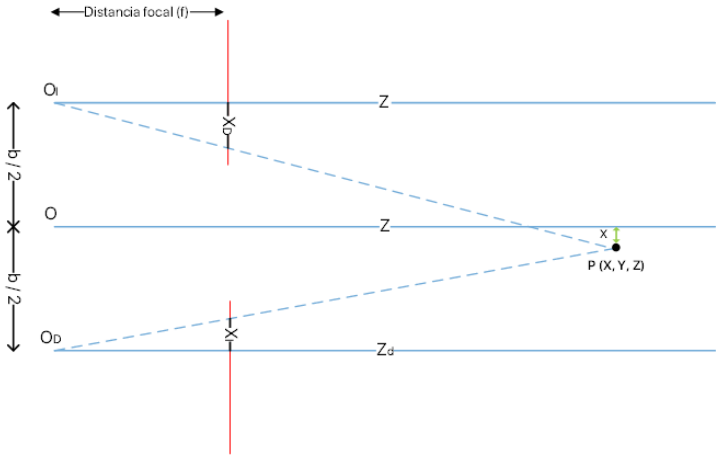


Ilustración 89. Representación de un sistema estereoscópico ideal, donde los ejes ópticos son claramente paralelos.

Si se considerara este esquema ideal, la disparidad, d, se podría obtener a través de la operación $x_I - x_D$. No se hallaría la profundidad real del punto sino una profundidad relativa que serviría para elaborar un mapa en escala de grises.

- Extracción de características: se buscan puntos característicos en las imágenes que se utilizan en el siguiente paso, ya sea por intensidad de píxel o bien buscando formas como cadenas de puntos o puntos aislados.

  Como bien sabemos, nuestro algoritmo de registro, ECC, se basa en un coeficiente de correlación, de tal forma que a lo largo de las iteraciones este prueba transformaciones hasta que alcanza el nivel de precisión especificado. Por lo tanto, este paso de extracción de características desaparece por completo de nuestro proceso.

- Correspondencia entre imágenes: en esta operación se busca el punto p_j correspondiente a p_i, ambos de capturas diferentes, ya sea en un área pequeña (más eficiente) o de forma global en la imagen (más costoso). Tanto este punto como el anterior vienen dados por el algoritmo de *Enhanced Correlation Coefficient*.

  Tomando en consideración el algoritmo ECC, se podría suponer conocida para cada punto p_i su correspondencia p_j, sin embargo, el principal problema de este algoritmo es que no se hace una búsqueda de cada punto individual, sino una búsqueda de todos los puntos de la imagen a la vez. Esto implica que se obtiene una matriz de transformación que minimiza el valor de una función que estima cómo de similares son dos imágenes.

- Obtención de la distancia. Supuesto el sistema estereoscópico ideal, se podría hallar la distancia d a partir de las siguientes ecuaciones:

$$x_I = \frac{f}{z}\left(x + \frac{b}{2}\right) \tag{61}$$

$$x_D = \frac{f}{z}\left(x + \frac{b}{2}\right) \tag{62}$$

$$d = x_I - x_D = f * \frac{b}{z} \tag{63}$$

- Interpolación: sería un último paso opcional suponiendo que sólo se halla la correspondencia para ciertos puntos (muestreos). Para el resto de puntos se interpola la profundidad.

Nuestro método, ECC, se puede enmarcar totalmente como un método de correspondencia e interpolación en una misma operación. Se puede suponer hallada la correspondencia de ciertos puntos, para los que sí se obtiene su emparejamiento real (o bien existe un error por debajo de un umbral), pero el resto de correspondencias tienen como objetivo minimizar una función de correlación, de tal forma que se pueden definir como interpolaciones obtenidas a partir de la aplicación de una matriz de transformación. El problema real es que el número de puntos para los que se halla su verdadera correspondencia es desconocido, y aunque los supiéramos, no se garantiza que estos sean los puntos más representativos para obtener el mapa de disparidad, dado que en ningún momento el algoritmo tiene este objetivo en mente.

Se puede comprobar fácilmente que el algoritmo de registro empleado es una mera interpolación a través de la implementación del algoritmo que se muestra en la Ilustración 90, dedicado a la obtención del mapa de disparidad a partir de M_i.

Algorithm 6: Algorithm that proves the applied registration is just an approximated process by obtaining the disparity map between two different layers of a multispectral image

```
1  function GetDisparityMap (alignedGRE, alignedRED);
   Input  : any pair of multispectral images. The process is simplified
            when considering the reference layer and another one
   Output: grayscale image that represents the disparity map obtained by
            applying the alignment matrices from the registration process
2  if !alignedRED.alignmentMatrices then
3  |  return null;
4  end
5  width ← alignedRED.width;
6  height ← alignedRED.height;
7  depthMap ← Matrix(height, width, Float_1);
8  disparityMap ← Matrix(height, width, Unsigned_3);
9  while col < width do
10 |  while row < col do
11 |  |  points.push(Point(col, row));
12 |  end
13 end
14 while matrixIndex < alignedRED.alignmentMatrices do
15 |  transform(points, points, alignedRED.alignmentMatrix(matrixIndex);
16 end
17 while col < width do
18 |  while row < col do
19 |  |  depthMap(row, col) ← abs(col − points(col ∗ height + row).x);
   |  |  Update(max, min, depthMap(row, col);
20 |  end
21 end
22 max ← max − min;
23 while col < width do
24 |  while row < col do
25 |  |  disparityMap(row, col) ←
   |  |  Vector3(depthMap(row, col) − min) ∗ 255/max;
26 |  end
27 end
28 returnImage(disparityMap);
```

Ilustración 90. Pseudocódigo del algoritmo de generación del mapa de disparidad entre dos imágenes multiespectrales tras el registro.

Si se aplica este pseudocódigo sobre un par de imágenes multiespectrales GRE y RED cualquiera, conocida la matriz de transformación global M_i, se obtiene un gradiente como el de la Ilustración 91.

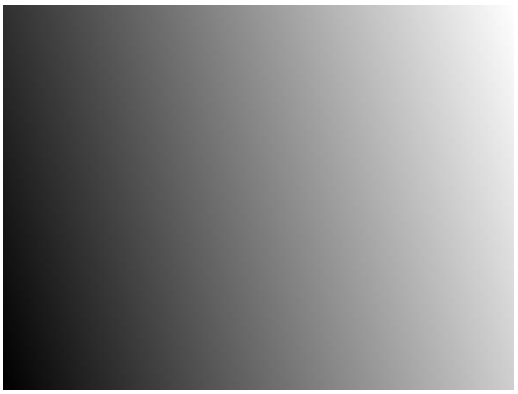


Ilustración 91. Gradiente que representa la disparidad de los puntos al aplicarse una transformación global.

Por lo tanto, se debe descartar la generación de un mapa de disparidad a partir del algoritmo ECC tal y como se ha empleado. Sin embargo, sí que existe una solución basada precisamente en este mismo algoritmo (Evangelidis, 2005), de su mismo autor, que se basa en los siguientes conceptos:

- Una primera fase de registro de imágenes mediante ECC para hallar la transformación geométrica que aproxime la superposición de las imágenes utilizadas.
- Una vez registradas se reduce el espacio de búsqueda en gran medida: se debe definir un área de búsqueda y repetir el proceso de ECC ahora en una subimagen para hallar la localización exacta de cada punto (ver Ilustración 92).

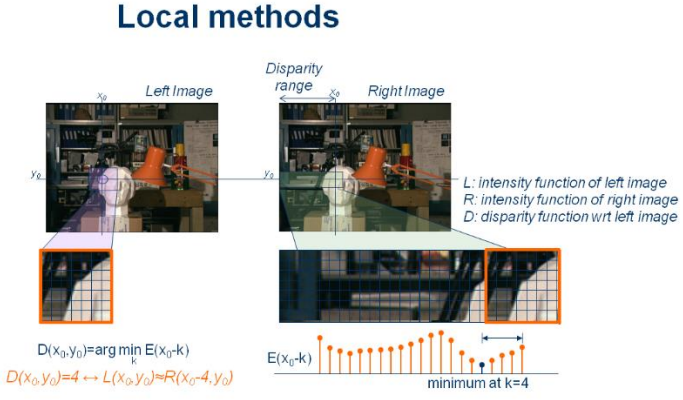


Ilustración 92. Subpixel Stereo Correspondence (Evangelidis, 2005).

También partíamos en esta sección de una georreferenciación, ya fuera de mayor o menor precisión, por lo que otro área de estudio es la generación del ortomosaico.

2.2.8.1.2 - Generación del ortomosaico

Como bien se comentaba al inicio de este documento, se debe hacer una clara distinción entre un ortomosaico y un **cosido o pegado de imágenes** (*image stitching*), dado que en esta segunda situación las imágenes no siempre se encuentran en las condiciones óptimas para que se produzca esa unión, y el resultado no presenta el escenario de la forma más precisa posible. De ahora en adelante, seguiremos hablando de ortomosaico por simplicidad, pero nos referimos claramente al pegado de imágenes o *image stitching*. En cualquiera de los casos, el resultado que se desea obtener sería algo similar a lo que se muestra en la Ilustración 93.

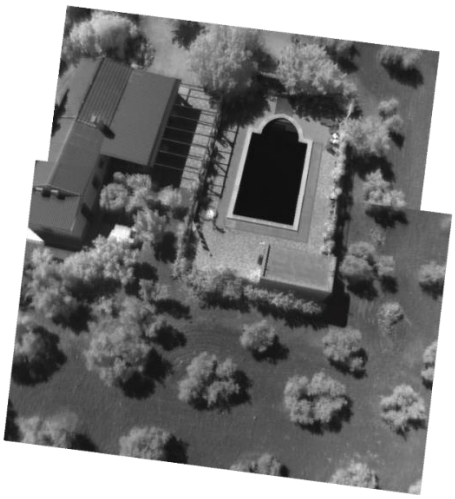


Ilustración 93. Ortomosaico objetivo de este apartado.

Aún con la precisión mostrada en esta imagen, se debería de hablar de *image stitching*, dado que la perspectiva no se puede considerar revertida. Asimismo, antes de proponer el algoritmo para la generación del ortomosaico, se intentan identificar todos aquellos problemas que podrían afectar (y lo harán) a la composición del mismo, teniendo en cuenta que se parte de imágenes multiespectrales rotadas:

- Poca precisión en la georreferenciación, derivada del sistema de posicionamiento global de la cámara multiespectral, pero también por el desconocimiento de la altura de captura sobre el terreno. Igualmente, también interviene la perspectiva de las imágenes, aunque por

simplificación podría entenderse que este último problema no introduce un error tan grande como los problemas anteriores.

- Rotación de imagen imprecisa, dado que los ángulos de los metadatos no se corresponden a la realidad. Luego, incluso con la corrección realizada, existe cierta imprecisión que siempre afecta, incluso en aquellos casos donde el ángulo resultante más se aproxima al valor real (siempre hay un error).

El algoritmo que se propone a continuación parte de que la situación de las imágenes de entrada es ideal, y por lo tanto, no hay errores en la localización geográfica, la rotación, o distorsiones no eliminadas, lo que facilita mucho todo el proceso pero también genera un resultado bastante alejado del objetivo que buscamos. Las bases del algoritmo propuesto son las siguientes:

- Conocidas las coordenadas UTM de las esquinas de la imagen original, se le puede asignar también una **posición a las esquinas de la imagen ya rectificada**: hablamos del resultado rotado donde una parte del mismo tiene color nulo. Dicha asignación de coordenadas se debe producir necesariamente combinando las coordenadas *easting* y *northing* de las esquinas reales de la imagen.
- La integración de nuevas imágenes se debe producir siempre y cuando estas estén contenidas en el ortomosaico actual. Lo que se busca con esto es no introducir imágenes muy distantes que hagan nuestro ortomosaico difícil de visualizar (se presentan las imágenes de forma diminuta).
- La inclusión de una nueva imagen implica la actualización de las coordenadas UTM de las esquinas del ortomosaico.

A continuación, se pretende mostrar el esquema del algoritmo utilizado mediante pseudocódigo, de tal forma que comenzaremos con métodos más abstractos hasta llegar al núcleo de todo este proceso. A grandes rasgos, el algoritmo de generación del ortomosaico se define tal y como se muestra en la Ilustración 94.

Algorithm 7: Adds a new image to the current orthomosaic

```
1  function AddImage (newImage, imageUTM);
   Input  : image to be added to the current orthomosaic and the utm
            coordinates of the corners
   Output: success of addition
2  if image already included then
3  |  return false;
4  end
5  if imageUTM.empty then
6  |  return false;
7  end
8  if !overlaps(currentUTM, imageUTM) then
9  |  return false;
10 end
11 insert(newImage, imageUTM);
12 includedImages.insert(newImage);
13 return true
```

Ilustración 94. Pseudocódigo del método principal para la inserción de una nueva imagen en el ortomosaico.

Se guarda la imagen que se acaba de pasar para evitar que se pueda incluir de nuevo al ortomosaico. Nótese como no tendremos el control de qué fichero se ha incluido ya, sino de qué instancia *Image* ya ha sido utilizada.

Como se ha comentado previamente, se incluye una imagen en el ortomosaico siempre y cuando sus coordenadas estén contenidas en la imagen actual, para evitar incrementar el tamaño del resultado en gran medida. En la Ilustración 95 se muestra cómo se podría implementar esta operación, teniendo en cuenta que la situación se reduce a un solape de rectángulos.

Algorithm 8: Checks if the image rectangle and the orthomosaic overlaps

```
1  function Overlaps (currentUTM, imageUTM);
   Input  : UTM coordinates for the orthomosaic and UTM coordinates of
            the new image
   Output: true if the image and the current orthomosaic overlaps
2  leftX ← max(imageUTM[BR].x, currentUTM[BL].x);
3  rightX ← min(imageUTM[BR].x + imageUTM[BL].x −
     imageUTM[BR].x, currentUTM[BL].x + currentUTM[BR].x −
     currentUTM[BL].x);
4  topY ← max(imageUTM[TR].y, currentUTM[BL].y);
5  bottomY ← min(imageUTM[TR].y + imageUTM[BR].y −
     imageUTM[TR].Y, currentUTM[BL].y + currentUTM[TL].y −
     currentUTM[BL].y);
6  return leftX < rightX  &  topY < bottomY;
```

Ilustración 95. Pseudocódigo para comprobar si una imagen se puede integrar en el ortomosaico actual.

Uno de los pasos más importantes cuando se recibe una nueva imagen es actualizar las coordenadas UTM en las que se encuentra el ortomosaico actualmente, ya que va a definir sus límites de cara a futuras inserciones, tal y como se representa en la Ilustración 96.

Algorithm 9: Updates the UTM position of orthomosaic corners

```
1  function UpdateUTMCorners (imageUTM);
   Input  : vector of UTM coordinates for each corner of the new image
2  min_x ← min(imageUTM[TR].x, imageUTM[BR].x);
3  min_y ← min(imageUTM[TL].y, imageUTM[TR].y);
4  max_x ← max(imageUTM[BL].x, imageUTM[TL].x);
5  max_y ← max(imageUTM[BL].y, imageUTM[BR].y);

6  imageUTM[BL] ← (min_x, min_y);
7  imageUTM[BR] ← (max_x, min_y);
8  imageUTM[TL] ← (min_x, max_y);
9  imageUTM[TR] ← (max_x, max_y);

10 min_x ← min(min_x, currentUTM[BL].x);
11 min_y ← min(max_x, currentUTM[BL].y);
12 max_x ← max(min_x, currentUTM[TR].x);
13 max_y ← max(max_x, currentUTM[TR].y);

14 currentUTM[BL] ← (min_x, min_y);
15 currentUTM[TR] ← (max_x, max_y);
```

Ilustración 96. Pseudocódigo para actualizar coordenadas UTM del ortomosaico.

Cuando se accede a los vectores de coordenadas UTM, se hace a través de una serie de índices en forma de enumeración, que delimitan claramente la esquina a la que se hace referencia: TL (*Top*, *Left*), BR (*Bottom*, *Right*), etc, teniendo en cuenta que todas ellas se nombran desde el punto de vista de la imagen, de la cual sabemos que su origen está en la esquina superior izquierda. Por lo tanto, nos referiremos a nuestra esquina *Top*, Left como *Bottom*, *Right*.

Aunque sabemos que la imagen que nos han pasado pertenece al ortomosaico, también debemos conocer la posición exacta donde esta se debe añadir, que viene definida por la coordenada UTM de la esquina inferior izquierda (ver Ilustración 97).

Algorithm 10: Updates the orthomosaic image by including a new multispectral image

```
1  function UTMPositionToPixel (utm_x, utm_y);
   Input  : utm location
   Output: position in the current orthomosaic image
2  utmWidth ← currentUTM[TR].x − currentUTM[BL].x;
3  utmHeight ← currentUTM[TR].y − currentUTM[BL].y;
4  utmXRel ← utm_x − currentUTM[BL].x;
5  utmYRel ← utm_y − currentUTM[BL].y;
6  width ← image.width;
7  height ← image.height;
8  return (width − 1) ∗ (1 − utmXRel/utmWidth), (height − 1) ∗ (1 −
     utmYRel/utmHeight));
```

Ilustración 97. Pseudocódigo para transformar una coordenada UTM en una posición dentro de la imagen del ortomosaico.

En el pseudocódigo de la Ilustración 98 se hace referencia a las esquinas de la imagen, de nuevo, desde el punto de vista de su origen, mientras que el ortomosaico sí que se define tal y como nosotros podemos observarlo.

Algorithm 11: Updates the orthomosaic image by including a new multispectral image

```
1  function UpdateImage (image, previousUTM, imageUTM);
   Input : the new image, the current orthomosaic UTM coordinates
           (which has been already updated), and the new image UTM
           coordinates
2  xDistance ← currentUTM[TR].x − currentUTM[BL].x;
3  yDistance ← currentUTM[TR].y − currentUTM[BL].y;
4  prevxDistance ← previousUTM[TR].x − previousUTM[BL].x;
5  prevyDistance ← previousUTM[TR].y − previousUTM[BL].y;
6  imagexDistance ← imageUTM[TR].x − imageUTM[BL].x;
7  imageyDistance ← imageUTM[TR].y − imageUTM[BL].y;

8  pixelPositionBackground ←
     UTMPositionToPixel(previousUTM[TR].x, previousUTM[TR].y);
9  pixelPositionForeground ←
     UTMPositionToPixel(imageUTM[TR].x, imageUTM[TR].y);

10 newTam_x ← max(pixelPositionBackground.x +
     width, pixelPositionForeground.x + width + 1);
11 newTam_y ← max(pixelPositionBackground.y +
     height, pixelPositionForeground.y + height + 1);

12 background ← zeros(newTam_y, newTam_x);
13 background_x ← floor(pixelPositionBackground.x);
14 background_y ← floor(pixelPositionBackground.y);
15 orthomosaic.copyTo(background(background_x, background_y, width, height));

16 orthomosaic = background;

17 foreground_x ← floor(pixelPositionForeground.x);
18 foreground_y ← floor(pixelPositionForeground.y);
19 image.copyTo(orthomosaic(foreground_x, foreground_y, image.width, image.height));
```

Ilustración 98. Integración de la nueva imagen en el ortomosaico actual.

Buena parte de todo este bloque de código va dirigido a averiguar la posición de cada imagen en el nuevo ortomosaico, así como las dimensiones de este tras insertar la captura multiespectral. Si se implementa el algoritmo aquí expuesto, se generan ortomosaicos como los de la Ilustración 99.

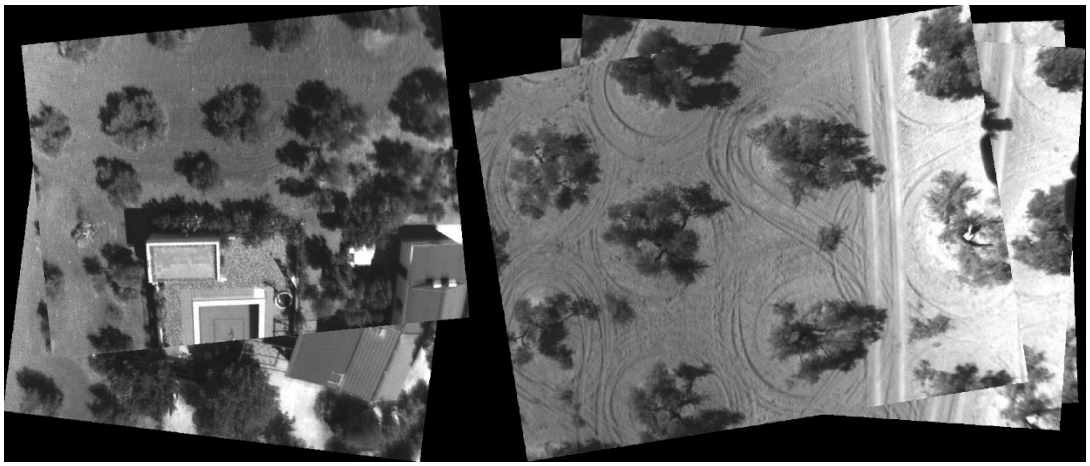


Ilustración 99. Ortomosaicos generados con 2 y 3 imágenes respectivamente.

Como se apreciaba desde procesos anteriores en los ya que intervenía la georreferenciación, la precisión que se obtiene es muy pobre, y son múltiples los problemas, donde el más grave de ellos puede ser la precisión de la localización espacial.

2.2.8.2 - Diagramas de secuencia

Dado que la utilidad de generación del ortomosaico ha sido la única incorporada a la aplicación dentro de todas las funcionalidades expuestas en este apartado, sólo se muestra un diagrama de secuencia asociado a dicha actividad (ver Ilustración 100).

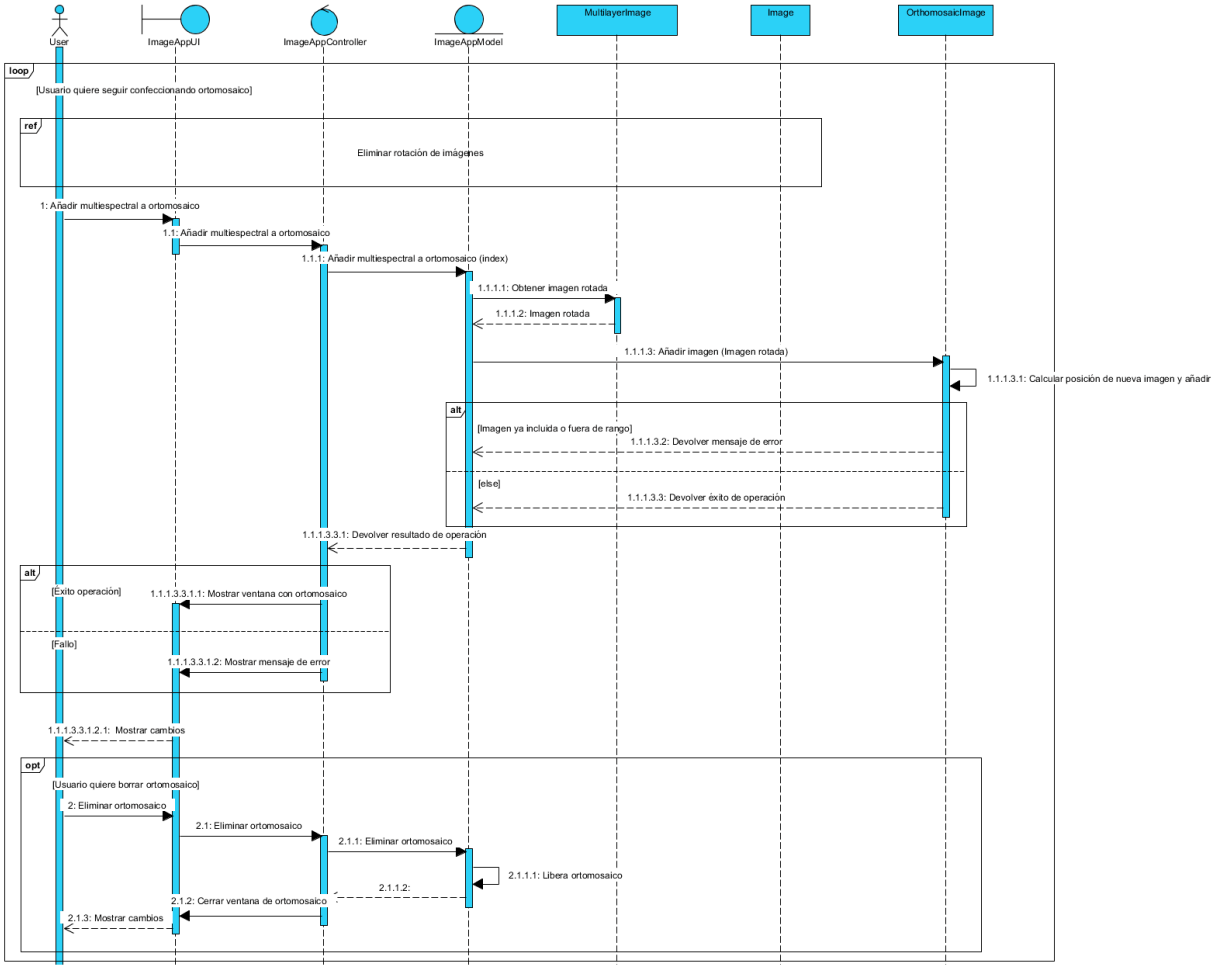


Ilustración 100. Diagrama de secuencia para la acción de componer el ortomosaico.

2.2.8.3 - Storyboards

Como hemos venido haciendo hasta esta iteración, las vistas del *storyboard* implicadas en esta acción de creación del ortomosaico son las que se representan en la Ilustración 101.

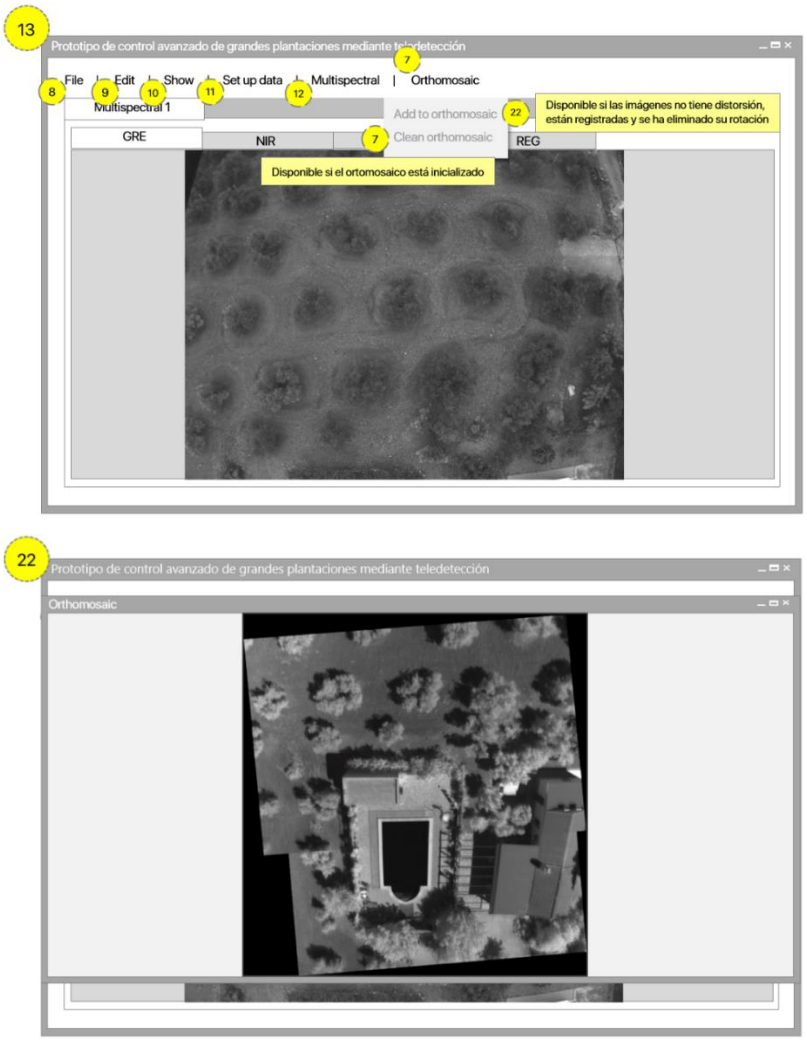


Ilustración 101. Vistas involucradas en la composición del ortomosaico.

2.2.9 - Décima iteración

El objetivo de esta iteración es permitir una visualización alternativa a ciertos resultados obtenidos, como podría ser la extracción de contornos. Se implementa así un **módulo 3D que debe dar soporte a la visualización de los contornos resultantes** en un *canvas*, permitiendo al usuario interactuar con ellos (tipo de renderizado, alturas, etc).

Buena parte de la carga de este módulo recae en el diagrama de clases, por lo que en esta sección nos centramos en explicar decisiones que se hayan tomado sobre el renderizado de la escena. Asimismo, no se entrará en detalles tales como la implementación de modelos de iluminación, que no son el objetivo de este trabajo y que extenderían esta documentación más de lo necesario.

A diferencia de un módulo corriente de esta aplicación, la parte 3D necesita algún controlador que renderice lo que se va a visualizar en el canvas. Además, debe existir un único controlador que se encargue de esta opción, lo que nos lleva inevitablemente a tomarnos la licencia de implementar el controlador y el modelo de este apartado en una misma clase, *Renderer*, dado que nadie más debe consultar los datos necesarios para el *renderizado*.

Todo aquello que se dibuje se hace sobre un canvas de Qt, que también nos permite acceder a eventos de ratón, teclado, etc, que se produzcan sobre él, informando así al controlador.

El resto de componentes importantes de este módulo surgirán conforme tratemos los elementos que se integran en la escena 3D:

- El soporte de dicha escena es un **plano** sobre el que se dibuja la imagen multiespectral (ver Ilustración 102). Habrá que tener en cuenta entonces que se debe crear un material (y su textura) con la imagen para que aporte color a dicho plano, teniendo en cuenta que este último debe tener las proporciones necesarias para representar la imagen correctamente.

  Se puede partir así de un plano cuadrado, donde la altura puede ser la dimensión fija, y la anchura se escale en función de las proporciones de la imagen multiespectral, que siempre son 1280:960 = 4:3. Para futuras transformaciones de los componentes de la escena, se debe tener en cuenta que el plano tiene su centro en el origen del mundo. Asimismo, el plano puede estar compuesto por múltiples subdivisiones, que para esta aplicación no serán necesarias dado que no necesitamos una geometría densa.

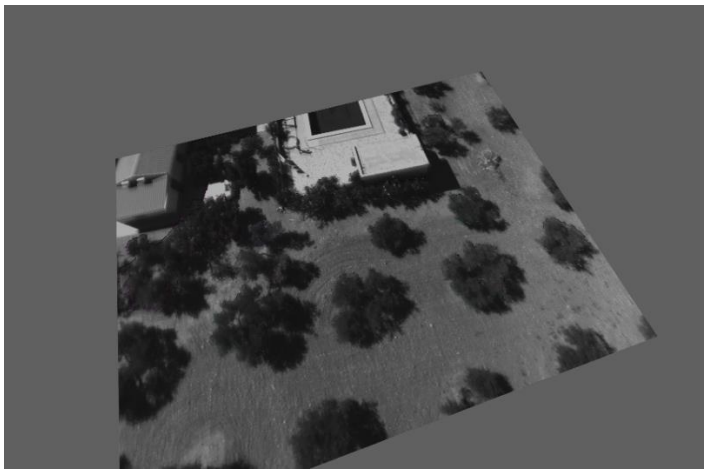


Ilustración 102. Representación de la imagen multiespectral sobre un plano.

- El componente principal de esta escena son los **contornos**. Cada contorno se representa de forma independiente, aunque una posible mejora para aumentar el rendimiento de la aplicación sería combinarlos todos en un mismo VAO y en una misma llamada de dibujo. La causa por la que esto no se ha hecho así es tener el control de cada contorno por separado: diferentes materiales, por ejemplo.

  Un contorno se representa, como ya sabemos, por un polígono, por lo que su representación 3D no necesita más que la duplicación de este mismo en dos alturas: 0 y 1. Se puede dibujar de dos formas: malla de triángulos y *wireframe*. Para la primera de ellas habrá que crear los triángulos correspondientes, sin embargo, para el modo *wireframe* será suficiente con generar una lista de índices de 0 a *size* – 1 (integra los dos polígonos, superior e inferior), separando justamente ambos con el correspondiente índice de reinicio de la primitiva. Con el tipo de primitiva adecuada, *GL_LINE_LOOP*, se dibujan los polígonos y unas líneas verticales que unen puntos equivalentes en la parte superior y en la base del contorno (ver Ilustración 103).

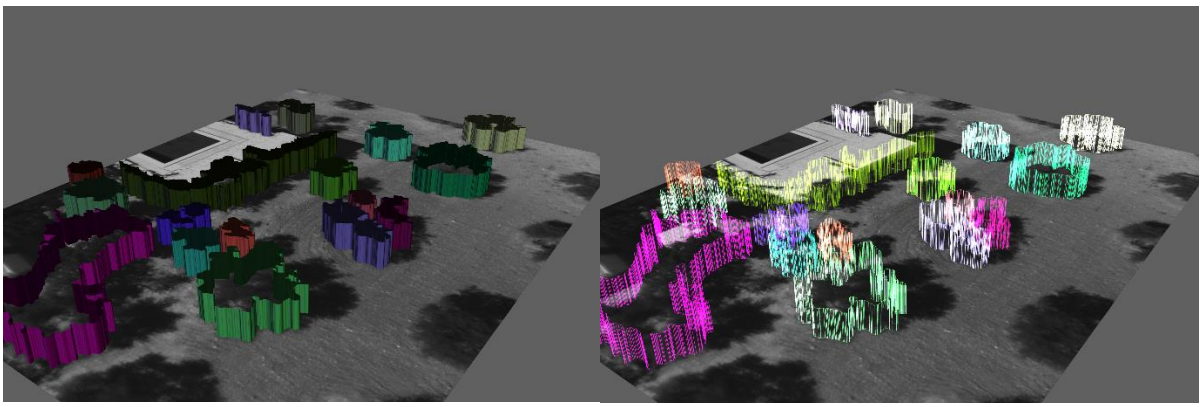


Ilustración 103. Comparativa entre el renderizado en modo malla de triángulos y malla de alambre.

A la representación aquí mostrada se le debe añadir la gestión de la altura de los contornos. Nótese como esta funcionalidad se puede implementar fácilmente, dado que los polígonos tienen altura 0 y 1, por lo que un simple escalado en la Y con un factor equivalente a la de la altura de los polígonos es suficiente.

Sin embargo, la transformación geométrica previa de un contorno consta de múltiples pasos, teniendo en mente todo lo que hemos visto del plano: está centrado en el origen, se parte de un tamaño de plano y se escala la anchura. De esta forma, la posición (500, 100) en una imagen de dimensiones 1280x960, sufriría las siguientes transformaciones:

- Normalización: (500 / 1280, 100 / 960) = (0.39, 0.1)
- Escalado por el tamaño original del plano, que suponemos 50x50: (0.39 x 50, 0.1 x 50) = (19.5, 5)
- Traslación del centro del plano al (0, 0): (19.5 – 50 / 2, 5 – 50 / 2) = (-5.5, -20)
- Escalado para adaptar a la anchura de la imagen: (-5.5 * 1280 / 960, -20) = (-7.3, -20)

Para terminar esta apartado se trata la generación de las tapas de los contornos, cuyo problema se reduce a la triangulación de un polígono (ver Ilustración 105). La principal problemática de este apartado viene dada cuando los polígonos son muy complejos (normalmente asociados a un suavizado muy pequeño y al uso de los contornos originales). La solución implementada es el algoritmo *Ear Clipping* (Eberly, 2015), cuyo pseudocódigo se presenta en la Ilustración 104.

```
Algorithm 12: Decompose a polygon into triangles
function Triangulate (polygon);
  Input  : a polygon
  Output: a list of triangles whose vertices belong to the polygon
if polygon.isClockwise() then
  | polygon.reverseVertices();
end
while polygon.size >= 3 do
  | gotEar ← false;
  | while i < polygon.size & !gotEar do
  |   | if polygon.isVertexConvex(i) then
  |   |   | size ← polygon.size;
  |   |   | a ← polygon[(i − 1 + size) mod size];
  |   |   | b ← polygon[i];
  |   |   | c ← polygon[(i + 1) mod size];
  |   |   | triangle ← Triangle(a, b, c);
  |   |   | validPoint ← true;
  |   |   | while j < size & validPoint do
  |   |   |   | point ← polygon[j];
  |   |   |   | if point ≠ a, b, c & triangle.isInside(point) then
  |   |   |   |   | validPoint ← false;
  |   |   |   | end
  |   |   | end
  |   |   | if validPoint then
  |   |   |   | gotEar ← true;
  |   |   |   | polygon.remove(i);
  |   |   |   | triangles.add(triangle);
  |   |   | end
  |   | end
  | end
  | if !gotEar then
  |   | Invalid polygon; break;
  | end
end
```

Ilustración 104. Pseucódigo para triangular un polígono mediante el algoritmo de Ear Clipping.

En cada iteración se intenta extraer un triángulo formado por V_{i-1}, V_i y V_{i+1}, sea V_i un vértice convexo, y donde el segmento $V_{i-1}V_i$ no intersecta con ningún otro segmento del polígono. Nótese como este algoritmo se propone para polígonos simples, donde sus segmentos, de inicio, no intersectan.

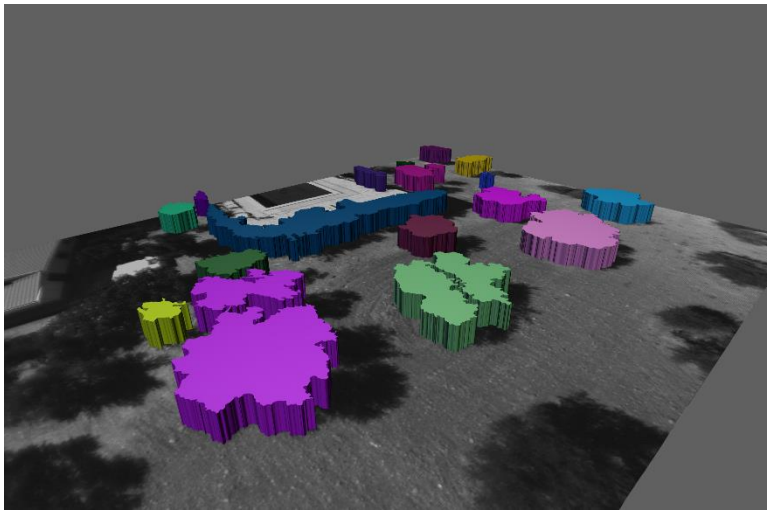


Ilustración 105. Representación de contornos con sus tapas generadas mediante el algoritmo de Ear Clipping.

- El último componente de la escena son aquellas esferas que representan los **centros de masas** de los contornos, y cuya posición debe transformarse de una forma similar a como ya lo hicimos con los contornos, con la diferencia de que ahora el objeto no debe escalarse, sólo modificar su posición (ver Ilustración 106). Por lo tanto, se parte de un modelo cargado desde el inicio, que se dibuja múltiples veces, tantas como contornos haya, y cada vez en un punto distinto. Asimismo, dado que el usuario puede pedir que se renderice la tapa, debe existir un *offset* este modelo y dicho componente.

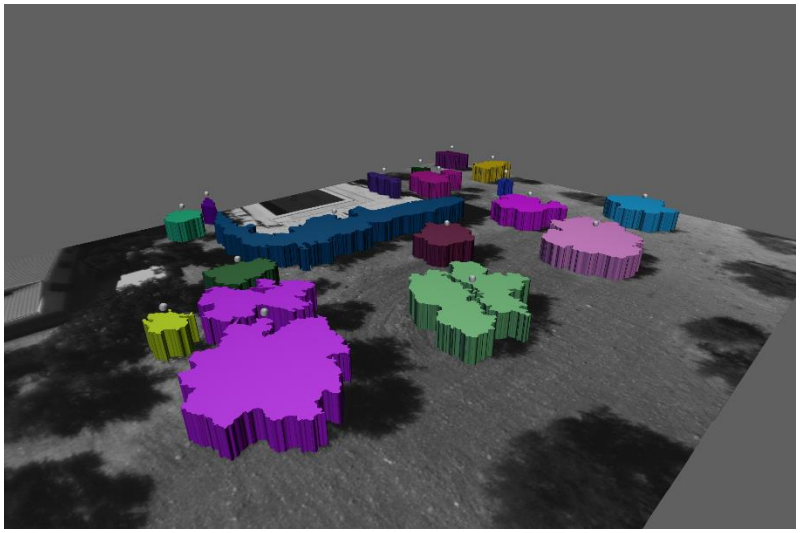


Ilustración 106. Centros de masas de los contornos, representados como esferas.

Otro factor importante en la escena es la **iluminación**, que interviene de forma directa en el coloreado de todos los objetos con topología de malla de triángulos: contornos, plano y centros de masas. Todos los modelos de iluminación aplicados se representan en la Ilustración 107.

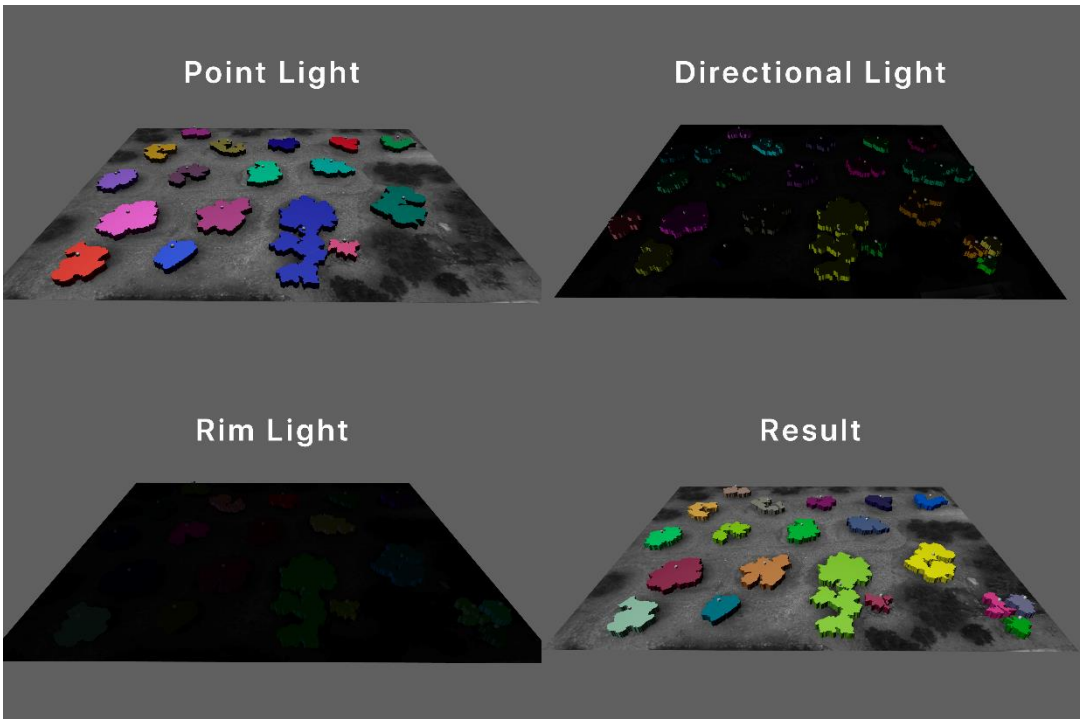


Ilustración 107. Resultado de la escena tras aplicar cada luz y resultado final.

En cuanto a la **cámara en perspectiva** de esta escena, cabe destacar que posee un gran número de controles para permitir libertad de movimiento al usuario, por lo que el manejo de la misma debe mostrarse en el manual de esta aplicación (Manuales de usuario).

Por último, otra funcionalidad disponible para el usuario final de la aplicación es la **captura de la escena al doble de resolución** a través de un FBO, teniendo en cuenta que en ese caso ya no se aplican técnicas de *anti-aliasing*, por lo que hay que utilizar realmente dos *buffers*: en uno se captura la escena y en otro se vuelca el contenido del primero aplicándole una operación de emborronamiento (*blit*), de tal forma que tras esto se obtiene una imagen de la escena sin problemas de *aliasing* (ver Ilustración 108).

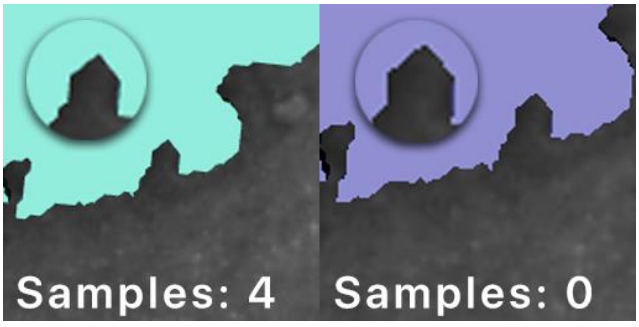


Ilustración 108. Comparativa de aliasing al aplicar un número de samples distinto. A la derecha, la captura por defecto sin buffer secundario.

2.2.9.1 - Diagramas de secuencia

El control de este módulo 3D de la aplicación se lleva a cabo tal y como se muestra en la Ilustración 109.

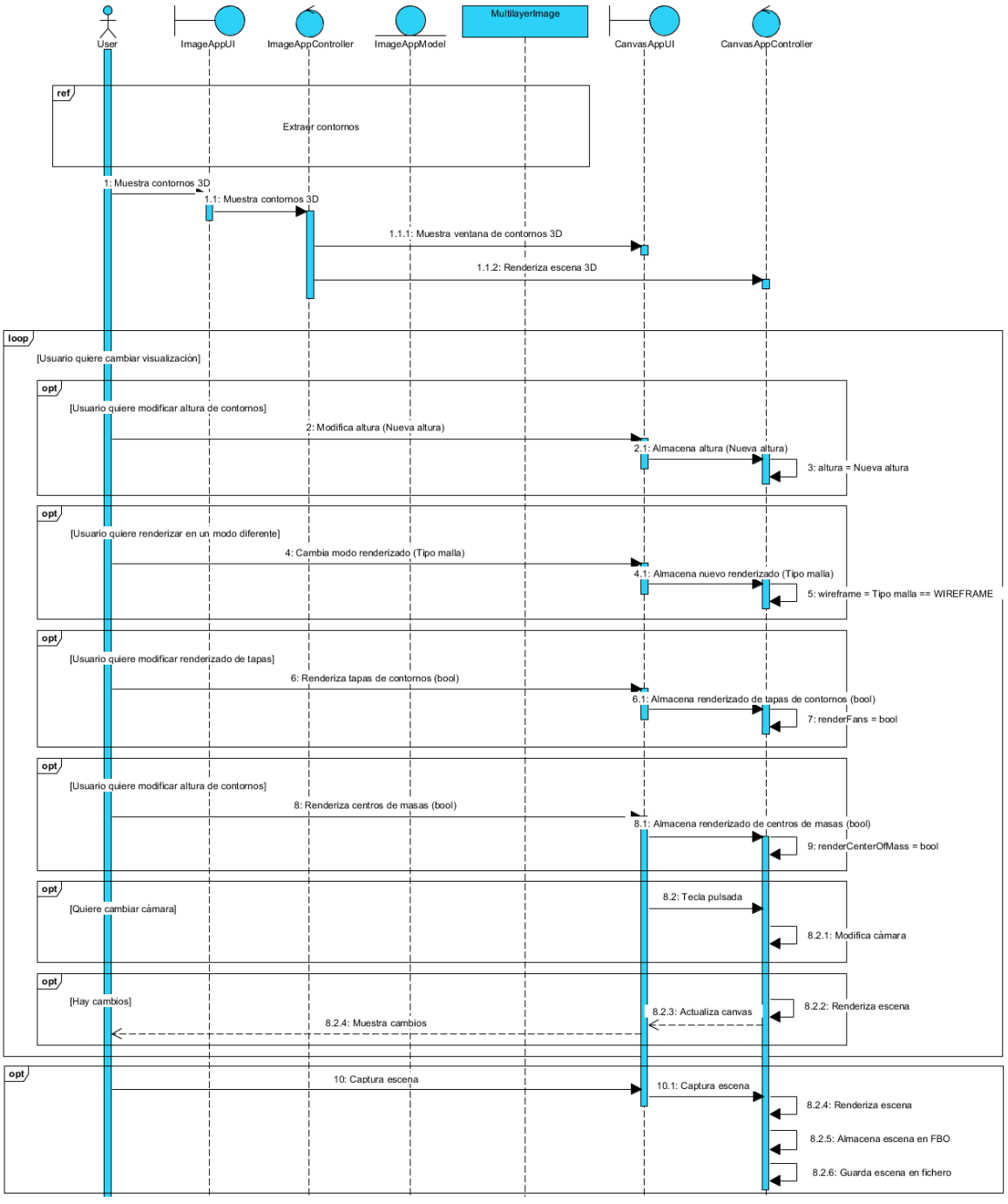


Ilustración 109. Diagrama de secuencia que muestra la interacción que puede llevar a cabo el usuario con el módulo 3D.

2.2.9.2 - Storyboards

Las vistas de esta aplicación involucradas en la visualización 3D de los contornos extraídos en etapas anteriores se muestra en la Ilustración 110.

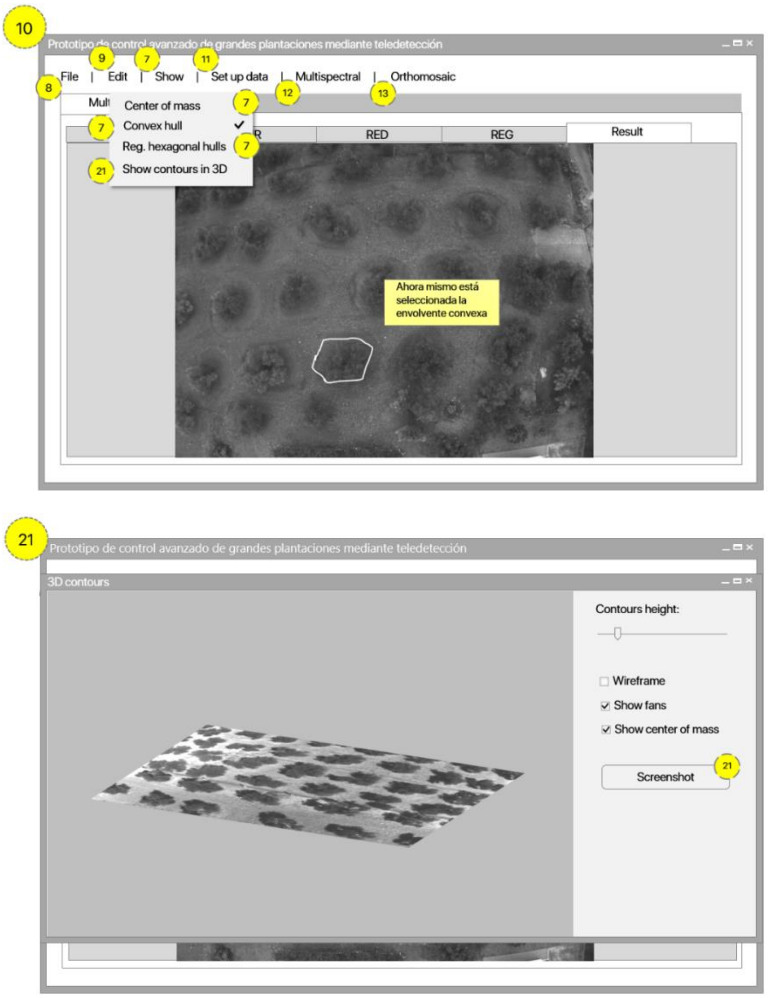


Ilustración 110. Vistas involucradas en la visualización 3D de los contornos extraidos previamente.

2.2.10 - Undécima iteración

El objetivo que se plantea en esta penúltima iteración es **desarrollar un modelo de base de datos que nos permita almacenar información recogida en apartados previos**, como puede ser la posición en un sistema de coordenadas de la Tierra de un centro de masas o un conjunto de puntos correspondientes a árboles extraídos en las primeras iteraciones de este proyecto.

Si bien, no se ha considerado que se deba guardar el color de los puntos que integran un árbol, dado que esa debe ser una característica a explorar en cada vuelo y no a lo largo del tiempo. En cualquiera de los casos, en la base de datos podría guardarse para un instante de tiempo concreto una etiqueta que indique la situación del árbol: enfermo, sano, o cualquier otro estado posible, en lugar de un conjunto de colores con los que habría que operar para averiguar el estado de la vegetación. Por

lo tanto, este enfoque de guardar sólo una etiqueta ahorra memoria y tiempo de cómputo en una base de datos.

Sí que se considera necesario guardar el contorno de un árbol con fines puramente de control de crecimiento, de tal forma que se podría representar de forma gráfica a lo largo del tiempo cómo avanza el polígono que representa al árbol. En cualquiera de los casos, se podría prescindir de este conjunto de puntos y guardar, por ejemplo, un área si así fuese necesario. También, con el fin de ahorrar memoria, puede surgir la pregunta de qué tipo de contorno emplear, en cuyo caso lo más frecuente es que se utilizara la envolvente convexa, que mantiene mínimamente la forma del árbol y ocupa mucho menos espacio.

Por otro lado, en el diagrama de la Ilustración 112 se emplean términos relacionados con una base de datos espacial de Oracle, como puede ser SDO_GEOMETRY, que indica la posición UTM, y que junto a la zona UTM identifica de forma única una localización en dicho sistema de coordenadas.

Dado que se almacena información estructural de un árbol a lo largo del tiempo, debe considerarse precisamente una variable tiempo, no ya sólo en el árbol sino también en aquellas entidades que definen su geometría y son susceptibles de variar en diferentes vuelos: contorno, centro de masas, etc.

Para generar aquella parte del esquema que integra toda la información del terreno en sí (no de los datos extraídos), nos hemos basado en los datos que el propio SIGPAC (Ministerio de Agricultura, Pesca y Alimentación, 2019) aporta sobre una parcela (ver Ilustración 111).

Ilustración 111. Información aportada por SIGPAC sobre una parcela.

En base a todo lo que hemos visto, se obtiene un esquema lógico de tipo Entidad-Relación (ERD) como el de la Ilustración 112.

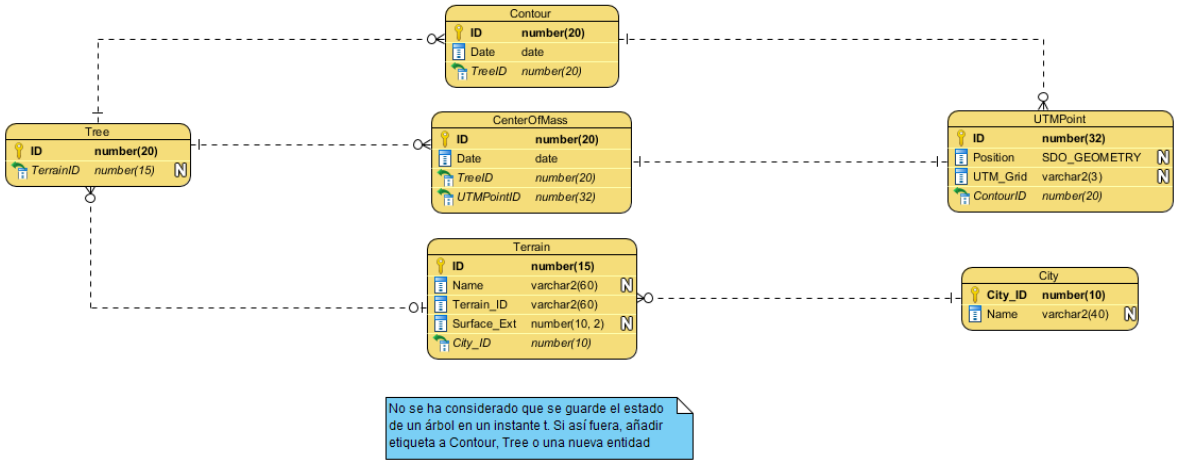


Ilustración 112. Esquema lógico de tipo Entidad-Relación de una base de datos destinada a guardar información estructural de árboles a lo largo del tiempo.

Derivado de este esquema, se genera el correspondiente código SQL que nos permita crear este modelo de base de datos y que se debe adjuntar a la entrega de este proyecto. Dentro de dicho código se debe resaltar la implementación de los identificadores únicos de nuestras entidades (aquellas que tienen como ID números que no se corresponden con ninguna propiedad real) como secuencias, que nos permiten obtener identificadores para nuestros registros de forma incremental:

```
CREATE SEQUENCE terrain_sequence;
CREATE OR REPLACE TRIGGER terrain_id
BEFORE INSERT ON terrain
FOR EACH ROW
```

2.2.11 - Duodécima iteración

Una vez se comprueba la inexactitud de todas aquellas operaciones que necesitan de un modo u otro de la georreferenciación de imágenes multiespectrales, se decide tomar un camino alternativo. Si bien la imagen RGB que capturaba el dispositivo Parrot Sequoia adolecía del mismo problema, la imagen RGB que captura otro dispositivo alternativo posee una precisión mucho mayor (del orden de centímetros).

Debido al instante de tiempo en que se inicia esta iteración, no se procede a implementar la georreferenciación a raíz de dicha imagen RGB sino simplemente su registro con una imagen multiespectral.

2.2.11.1.1 - Registro de imagen multiespectral y RGB

La principal diferencia entre una imagen multiespectral y la nueva imagen RGB es que ya no se capturan en un mismo tiempo instante de tiempo, por lo que el registro debe ser necesariamente más complejo que todo lo que hayamos visto hasta ahora.

Cabe destacar que uno de los metadatos que contiene cualquiera de nuestras imágenes es el instante de captura, por lo que se propone como metodología escoger la imagen RGB más cercana en el tiempo a una imagen multiespectral. Aún así, como ya veremos, esto no garantiza que las imágenes sean tan similares como en apartados previos.

Para comprender mejor este apartado, se muestran a continuación dos imágenes, una multiespectral (GRE), capturada el día 15 de febrero a las 12:43:21 PM, y una imagen RGB tomada el mismo día a las 13:43:22 PM. A raíz de esto, un problema evidente, dado el punto de vista planteado de escoger la imagen más cercana en el tiempo, es precisamente este, la corrección del tiempo que se encuentra en los metadatos. En cualquiera de los casos, las dos imágenes de la Ilustración 113 se reconocen como capturadas en instantes de tiempo muy cercanos.

Ilustración 113. Imagen RGB y multiespectral capturadas en instantes de tiempo muy cercanos.

Si bien somos capaces de asegurar que capturan el mismo área, las diferencias entre ambas son muy grandes como para efectuar un registro tal y como se reciben de los dispositivos de captura. Por supuesto, existen ciertas mejoras que ya hemos venido llevando a cabo y que deberían acercar la imagen multiespectral a la RGB,

como puede ser la eliminación de la distorsión *fisheye*. De esta forma, la comparativa quedaría como se muestra en la Ilustración 114.

Ilustración 114. Comparativa de imagen RGB y multiespectral sin distorsión fisheye.

Cuando hablábamos de la imagen RGB del dispositivo Parrot Sequoia se comentaba que, de acuerdo a sus metadatos, no presentaba distorsión *fisheye*, sin embargo, se podía ver claramente como sí existía cierta distorsión que además, es algo frecuente para distancia focales tan pequeñas. En el caso de esta imagen RGB, su distancia focal es más grande (de 4.9 mm a 8 mm), sin embargo, sigue entrando dentro del intervalo de distancias focales tan pequeñas que podrían presentar esa distorsión. De nuevo, al igual que con la imagen RGB de apartados previos, no se dispone de ningún coeficiente de distorsión ni de otros datos relevantes para revertir esta situación.

Esta distorsión se puede ver claramente en la Ilustración 114, donde podemos fijarnos en el camino justamente a la derecha de ambas imágenes: mientras que la imagen multiespectral, con su distorsión revertida, presenta prácticamente una huella recta, la imagen RGB presenta una huella claramente con curvatura. Para este registro no se decide ir más allá con este problema por las razones que se exponen a continuación:

- En apartados posteriores se decide revertir la distorsión *fisheye* con unos parámetros aproximados, lo que produce claros errores en los extremos de la imagen, que es donde la distorsión también se podía observar de forma clara. A diferencia de apartados posteriores, aquí necesitamos la imagen RGB completa para efectuar el registro, por lo que no podemos aplicar esta solución.

- No necesitamos registrar exactamente las dos capturas para extraer información de color de la imagen RGB, sino para conocer dónde se encuentra el punto principal de la multiespectral así como su orientación dentro de esta segunda.
- La distorsión *fisheye* tiene poca relevancia en la parte central de la imagen, que es la zona donde debiera encontrarse ese punto principal.

Se debe tener en cuenta, eso sí, que el registro de estas imágenes tiene como objetivo una georreferenciación más precisa que todo lo que hayamos hecho antes, por lo que no puede haber una gran diferencia entre ellas.

También se debe tener en cuenta que el algoritmo de registro de imágenes que se emplea es de nuevo ECC, que si se recuerda de apartados anteriores, tenía multiples modelos de transformación, entre los que se encontraba la homografía, la más compleja de todas ellas. Se hablaba entonces de que se podría buscar este tipo de transformación en aquellos casos en los que se desconocen las condiciones de captura, y este puede ser un buen ejemplo. De hecho, lo que intentaremos hacer es revertir ciertamente esa diferencia entre imágenes provocada por la distorsión *fisheye* de la RGB mediante una transformación global tan compleja como la homografía. Nótese como dicha transformación no puede generar el efecto de circunferencia que se produce debido a la distorsión ya citada, sin embargo, sí que podemos utilizarla para deformar la imagen y aproximarla todo lo que sea posible.

Las imágenes a registrar, y por lo tanto la entrada del algoritmo completo, serían las dos primeras imágenes de este apartado, pero la entrada del algoritmo ECC será el par de imágenes de la Ilustración 114, que, como podemos ver, presentan diferencias notables en varios sentidos:

- Existe una clara rotación entre imágenes, derivada del movimiento del dron.
- El ángulo de visión de ambas lentes no es el mismo, también derivado de la distancia focal de cada una de ellas (8 mm para la RGB y 4 mm para la multiespectral, por lo que esta última tiene mayor ángulo).

Todo esto nos conduce, de nuevo, a la aplicación de una transformación global más compleja, pero también a la modificación de los parámetros del algoritmo, que si bien recordamos, eran hasta dos. De momento, teniendo en cuenta lo que hemos visto, un parámetro que claramente se debe modificar es el número de iteraciones,

que se debe incrementar sustancialmente para permitir que el algoritmo pueda probar más transformaciones (aunque todo esto se comprueba en la fase de experimentación).

Si tenemos en cuenta que la imagen RGB tiene mucha resolución (4000x3000), queda claro que hay que reducir su tamaño por dos razones: a) comparar ambas imágenes correctamente, b) ejecutar el algoritmo en un tiempo razonable. Se puede considerar entonces el siguiente enfoque:

- Se puede reducir el tamaño de la imagen RGB al tamaño de la multiespectral para resolver esta última problemática.
- Podemos ir más allá y reducir incluso aún más el tamaño de ambas imágenes. Esto es posible dado que si recordamos el apartado donde se introducía el algoritmo ECC, se comentaba que es capaz de trabajar a nivel de subpíxel. Es decir, se podría trabajar con imágenes muy grandes y buscar un nivel de precisión menor (p. ej. 1^{-5}), o bien trabajar con imágenes más pequeñas y un nivel de precisión mucho más grande (p. ej. 1^{-60}). Si a esto le sumamos el interés anteriormente citado en aumentar el número de iteraciones considerablemente, la segunda opción que acabamos de explicar parece razonable.

Otro factor a tener en cuenta en el registro de imágenes es esa estructura arbórea que se presentaba en el registro multiespectral. Se creaba dicho esquema para evitar emparejar capas de intensidad muy dispares, lo que podría desencadenar un registro erróneo. Tomando en consideración ese problema, se efectúa el registro sólo entre la multiespectral GRE y la imagen RGB, que al reducirse a escala de grises son las más similares, y como prueba de ello, se puede realizar un análisis de correlación de imágenes tal y como se propone en el apartado de experimentación. Así, para una imagen multiespectral cualquiera y su RGB asociada, los coeficientes de correlación entre ambas son los que se muestran en la Tabla 13.

Imagen multiespectral	Coeficiente de correlación: multiespectral - RGB
GRE	0.98066
RED	0.9668
REG	0.9614

NIR	0.9473

Tabla 13. Coeficientes de correlación entre imágenes multiespectrales (Conjunto de datos 1, IMG_6) y su RGB asociada, todas en escala de grises.

Esta decisión que se acaba de justificar también tiene sus beneficios computacionales: si elegimos la imagen GRE no es necesario registrar imágenes multiespectrales para poder compararla con la RGB (ya está registrada porque es la referencia). Tampoco se debe efectuar la rotación de esta capa, ya que es precisamente de operaciones como esa de las que huimos, dada la inexactitud hallada en dichos procesos.

Otro paso a tener en cuenta en todo este algoritmo debe ser necesariamente la eliminación del ruido de las imágenes, especialmente de la imagen RGB. De esta forma, queda un flujo de operaciones como el que se muestra en la Ilustración 115.

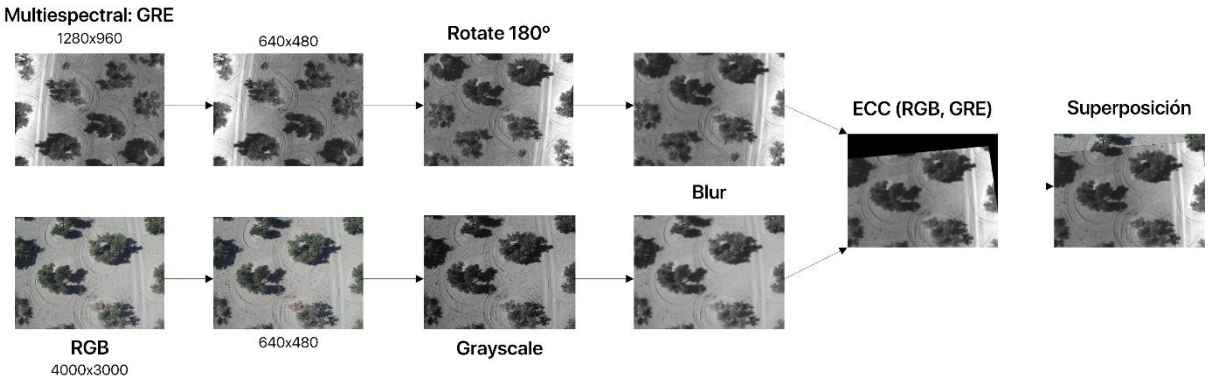


Ilustración 115. Flujo de operaciones a llevar a cabo para registrar una imagen RGB y una multiespectral.

Otra forma de ver este proceso es mediante pseudocódigo (ver Ilustración 116).

Algorithm 13: Registers the multispectral image in a RGB image

```
1  function RegisterRGBMulti (multiGRE, rgb);
   Input  : two images: a multispectral GRE image and a RGB image
   Output: rgb image with an overlapped GRE image which is registered
           at this point
2  rgbResized ← rgb.resize(multiGRE.size/2);
3  multiResized ← multiGRE.resize(multiGRE.size/2);
4  blurredRGB ← rgbResized.smooth(3);
5  blurredGRE ← multiResized.smooth(3);
6  rotatedGRE ← blurredGRE.rotate(180);
7  alignedImage ←
     rotatedGRE.align(blurredRGB, HOMOGRAPHY, 400, 1e − 60);
8  result ← resizedRGB.overlap(alignedImage, color(0, 0, 0));
9  Libera memoria
10 return alignedImage ≠ null
```

Ilustración 116. Pseudocódigo para el registro de una imagen multiespectral en una RGB.

En este bloque de código se han propuesto algunos valores para los parámetros de todas las funciones que se invocan:

- En el caso del emborronamiento se propone un núcleo de tamaño 3 (debe ser impar), teniendo en cuenta que el tamaño de la imagen es ya lo suficientemente pequeño.
- La imagen multiespectral se rota 180º porque este tipo de imágenes, de formato TIFF, suelen venir rotadas respecto a la imagen RGB, que se presupone que es la correctamente representada.
- Para el registro se propone un número de iteraciones alto (400), como ya se justificó antes, y también un gran nivel de precisión (1^{-60}), teniendo en cuenta que las imágenes son más pequeñas. Como comparativa, el algoritmo de registro de multiespectrales funciona (a expensas de lo que suceda en la experimentación) con 30 iteraciones y una precisión de 1^{-10}.

En la Ilustración 117 y Ilustración 118 se muestran algunos resultados de aplicar este algoritmo de registro:

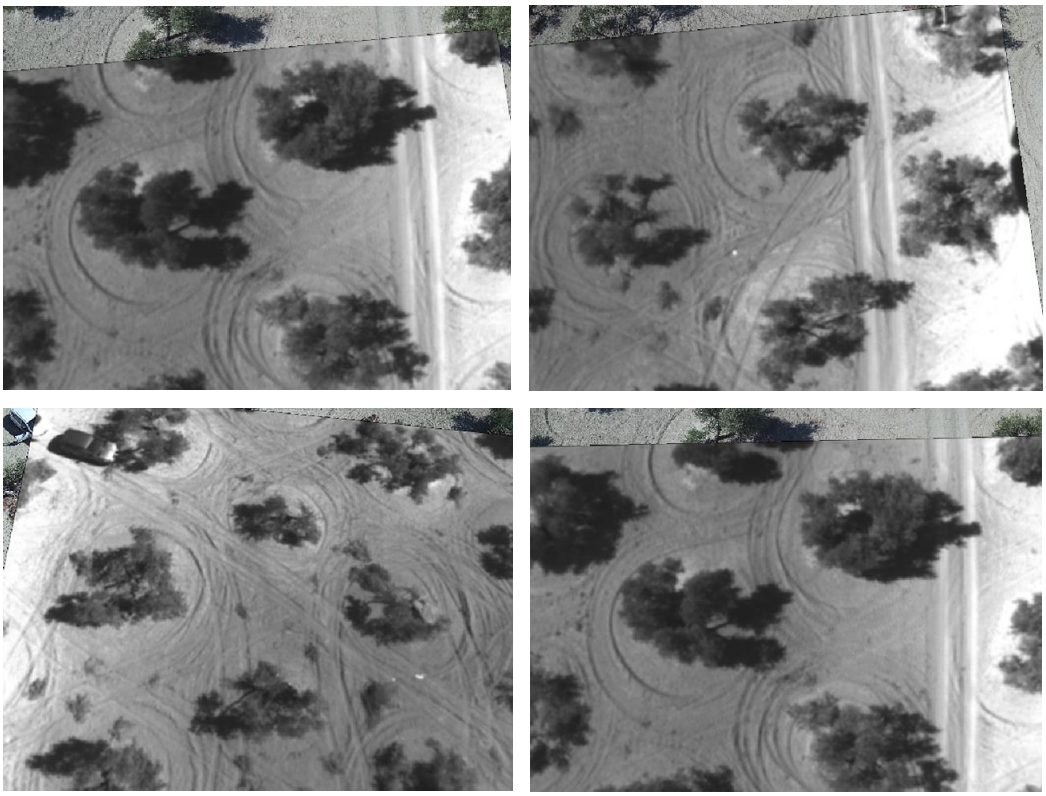


Ilustración 117. Resultados de ejecutar el algoritmo de registro de imágenes RGB-multiespectral.

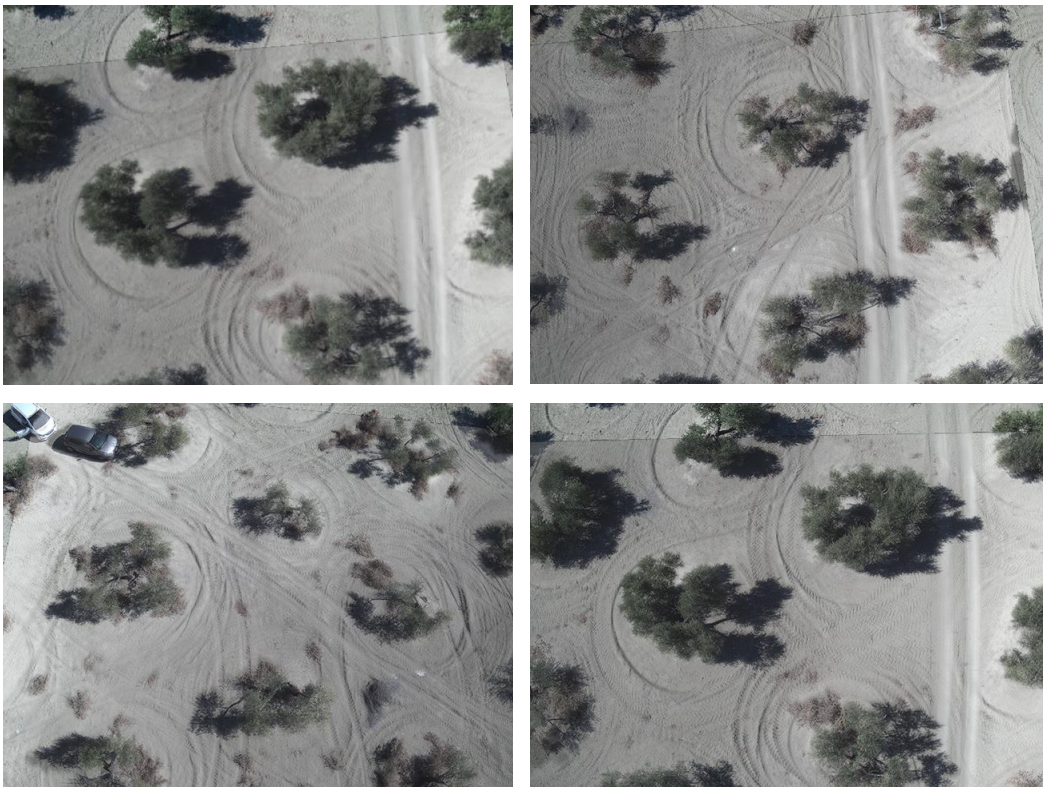


Ilustración 118. Superposición de resultados del registro sobre imágenes RGB originales a través del canal alpha.

Nótese como existe aún cierta distancia entre imágenes derivada mayormente de la distorsión de la imagen RGB. En cualquiera de los casos, el objetivo es claro: identificar dónde está el punto principal de la multiespectral en la RGB (aunque sea con una pequeña distorsión) aplicándole la transformación obtenida, M_i, y rotar la imagen en función de los ángulos que contienen los metadatos de la imagen RGB, que ahora sí deberían ser correctos.

En cualquier caso, antes de proponer la solución que aquí se ha expuesto, se intenta llevar a cabo este registro de una forma completamente diferente, que se trata en el siguiente subapartado.

Alternativa de registro

Para este caso, se parten de los contornos extraídos de la imagen multiespectral, y se intenta encontrar estos mismos en la imagen RGB. Por supuesto, esto implica que hay que hacer una extracción de contornos de la imagen RGB.

Aunque esta primera fase de extracción de contornos parece la más compleja, aún lo es más el emparejamiento de contornos, dado que hay que analizar su similitud.

De hecho, este último paso es el principal inconveniente encontrado, y por ello se dedice proponer una metodología completamente diferente.

Nótese como la extracción de contornos se realizaba en la multiespectral y no en la imagen RGB porque en esta última es mucho más complejo. Sin embargo, en este apartado no se pretende una segmentación precisa sino aproximada, de tal forma que nos basta con conocer dónde se sitúa al menos un contorno de la multiespectral en la imagen RGB.

De forma gráfica, el algoritmo que se propone para extraer los contornos de algunos de los árboles que aparecen en la imagen RGB se muestra en la Ilustración 119.

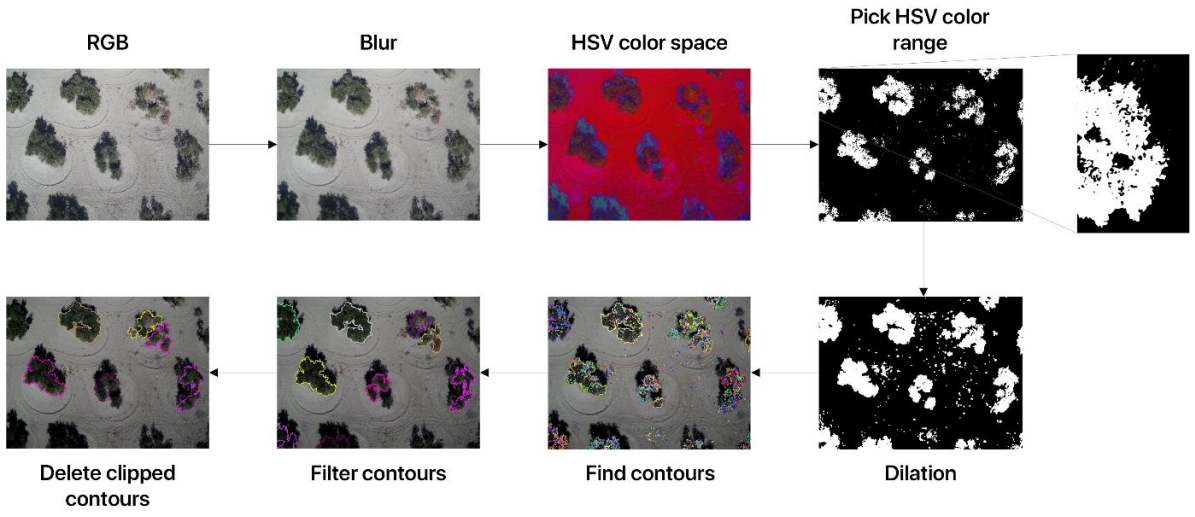


Ilustración 119. Flujo de operaciones para extraer la vegetación de una imagen RGB.

En lugar de trabajar en el espacio de color RGB, donde la segmentación puede llegar a ser prácticamente imposible, dado que se confunde un árbol con su sombra, se trabaja en el espacio de color HSV (Hue, Saturation, Value), donde sí se pueden diferenciar claramente ambos elementos (Feng, 2015).

Algunos parámetros útiles para llevar a cabo este proceso son los siguientes:

- Tamaño del núcleo del filtro gaussiano: se establece el valor 21 dado el tamaño de la imagen original.
- Para la operación de escoger un rango de colores HSV hay que tener en cuenta que ya no hablamos de los 3 canales originales: R, G y B. Se establece así un rango HSV de (5, 20, 0) a (90, 255, 255), donde será el canal Hue el filtro para escoger nuestra vegetación.

- Dado que se extraen polígonos no especialmente bien delimitados a partir de esta selección, se dilatan mediante una operación morfológica (*dilate*) de tamaño de núcleo 2 durante 16 iteraciones.

Sin embargo, se puede comprobar que este método no funciona para todos los casos. Véase la imagen RGB de la Ilustración 120, de otro conjunto de datos diferente, donde el color del terreno varía en gran medida, de tal forma que ahora no se obtendría ningún contorno:

Ilustración 120. Imagen RGB con un color de terreno muy diferente de las imágenes anteriores.

Se refactoriza pues el flujo anterior para que funcione con ambos tipos de imágenes, aunque queda claro que, a falta de experimentación con más conjuntos de datos, podría volver a darse la misma problemática en otros *datasets*.

En este segundo algoritmo propuesto se utiliza otro espacio de color más (ver Ilustración 121), como es CIELAB ó LAB, donde ahora sí se puede diferenciar claramente la vegetación debido a que el valor a* está relacionado con la fluctuación del verde al rojo. Es decir, a* = 0 hará referencia al verde, y justo lo contrario sucede con a* = 100 (ó 255, depende de si está normalizado).

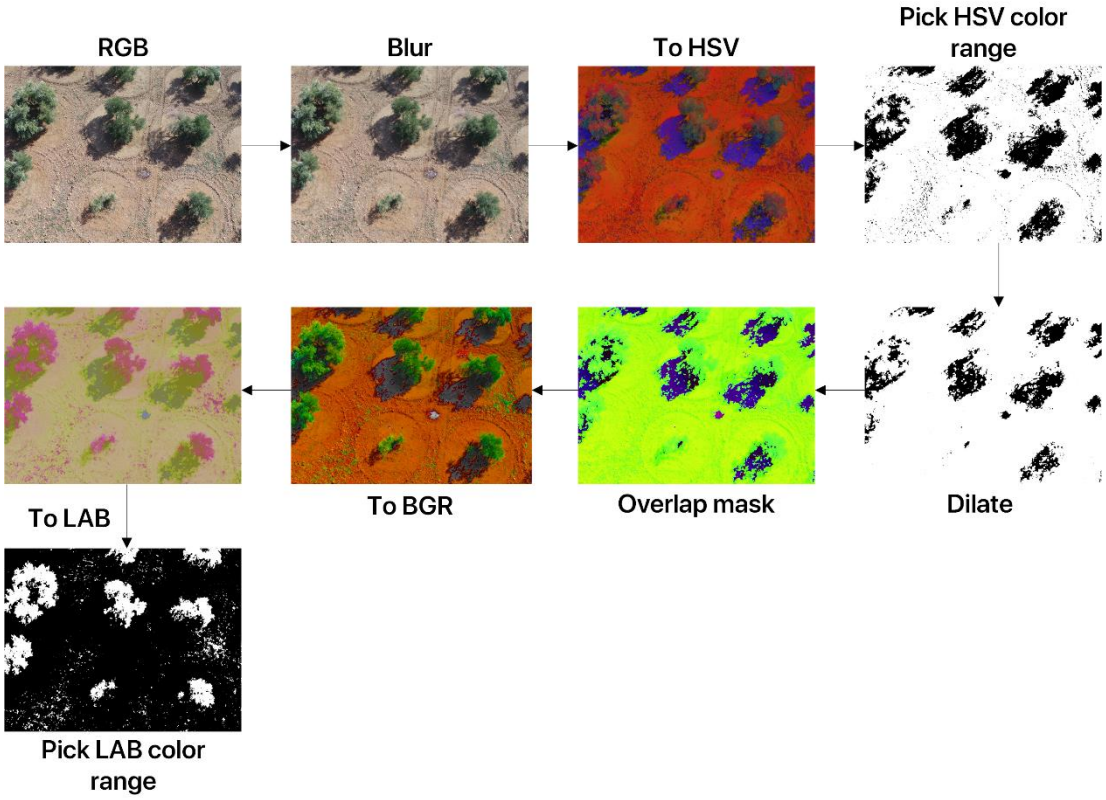


Ilustración 121. Algoritmo mejorado para la extracción de contornos en la imagen RGB.

Algunas anotaciones sobre el algoritmo:

- El principal problema encontrado anteriormente es que no se podía distinguir claramente terreno y árbol, por lo que se obtiene una máscara en el espacio HSV para todo este conjunto. De esta forma, podemos aislar la sombra del resto de elementos representados.
- La máscara obtenida se dilata un poco para reducir la superficie de las sombras (se dilata el área blanca).
- Se solapa la máscara en la imagen HSV, de tal forma que la superficie blanca de la máscara se corresponde con un color totalmente diferente en dicho espacio ((255, 255, 255) en HSV), pero englobamos así a árboles y suelo.
- Se pasa al formato BGR (OpenCV) como un puente hacia el espacio de color LAB. En este espacio ya sí somos capaces de distinguir en todos los casos probados la superficie de árboles respecto de suelo y sombra (ya se hizo distinción de esta última previamente).

Para referencias de posibles implementaciones futuras, se proponen aquí una serie de valores para los parámetros de este algoritmo:

- Tamaño de núcleo de emborronamiento: 21 (de nuevo, derivado del tamaño de la imagen RGB).
- Tamaño del núcleo de dilatación: 2, aplicando 16 iteraciones.
- Rango de colores a seleccionar en HSV: (5, 20, 80), (90, 255, 255), que se corresponde a la vegetación y al suelo.
- Rango de colores a seleccionar en LAB: (0, 0, 0), (255, 125, 255). Recordemos, a* pasa del verde (0) al rojo (255 ó 100 si se encuentra en formato normalizado).

De esta manera, se obtienen resultados como los de la Ilustración 122 para imágenes de varios conjuntos de datos (y diferentes tipos de terrenos).

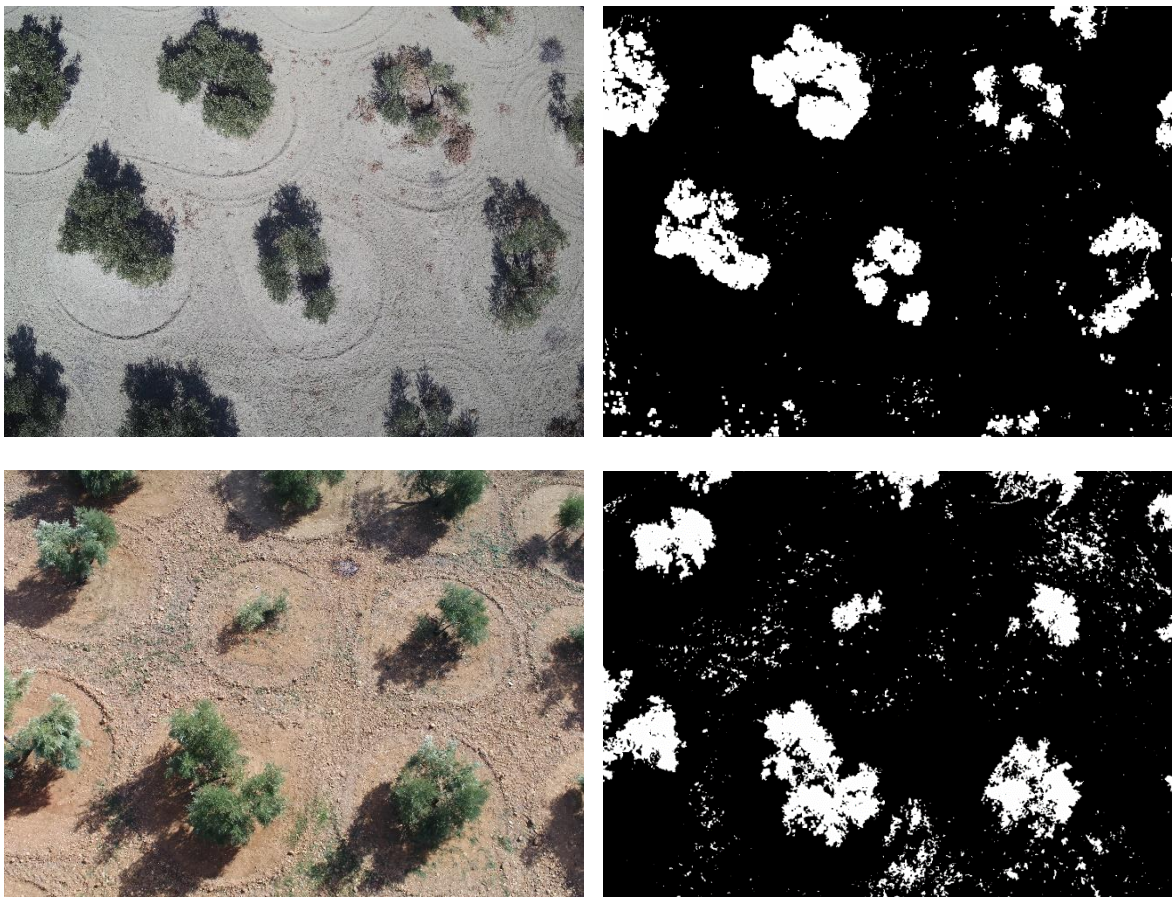


Ilustración 122. Resultados del algoritmo de extracción de árboles para dos tipos de terrenos diferentes.

Llegados a este punto, tenemos nuestros contornos disponibles tanto en la imagen RGB como en la imagen multiespectral, por lo que nos queda aún el paso más difícil, que, de hecho, aunque implementado, no ha llegado a estar operativo en la aplicación dado que no funcionaba para la gran mayoría de pares de imágenes.

El problema que se plantea ahora es el de hallar la similitud entre dos contornos, teniendo en cuenta que estos pueden estar rotados (siempre sucede por el cambio de posición y orientación del dron), deformados por la proyección (lo que integra un escalado), etc. Por lo tanto, es un problema complejo que normalmente se resuelve, o, mejor dicho, se resolvía, con algún tipo de medida de distancias (Shape Similarity, s.f.) (Serge Belongie, 2002) (Yankov, Keogh, Wei, Xi, & Hodges, 2008).

Se dice bien que se resolvía porque dadas las tendencias actuales, un método más sencillo y que podría dar mejores resultados se basa en *deep learning*. En cualquiera de los casos, para seguir la corriente que hasta ahora hemos venido usando se emplean métodos de distancias, y más concretamente, la distancia de Hausdorff, que ya implementa OpenCV debido a que es un método que podría utilizarse para *image matching* (encontrar la posición más apropiada de un objeto en la imagen).

La distancia de Hausdorff unidireccional para dos conjuntos de puntos, A y B, se calcula como sigue (Singhal, 2003):

$$\delta(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\| \tag{64}$$

mientras que la distancia bidireccional será el máximo de $\delta(A,B)\ y\ \delta(B,A)$.

Nótese como la unidireccional puede ser necesaria cuando se delimita claramente el objeto fuente (*source*) y destino, dado que, por ejemplo, la forma origen puede estar ocluida en el destino, luego si la distancia es mayor que un umbral se considera que no aparece. En ese caso no tendría sentido alguno hacer una búsqueda bidireccional.

Se debe tener en cuenta además que este algoritmo no recibe dos polígonos, sino dos conjuntos de puntos, lo que implica que A y B podrían estar formados por varios polígonos. Véase el ejemplo de un polígono (A) y un conjunto de fronteras obtenidas a partir de la aplicación de un filtro de Sobel sobre una imagen (B), en cuyo caso se puede intentar hallar la posición en la que, con mayor probabilidad, puede situarse el polígono A.

En nuestro caso, no se desea saber la posición donde debiera situarse un polígono de A en el conjunto de B, sino saber cómo de similares son dos polígonos. Por lo tanto, la entrada de este algoritmo en nuestra aplicación sí que serán dos polígonos siempre (árboles).

Habría también que discutir criterios tales como el normalizado de las formas para evitar que contornos que pueden aparecer con distinto tamaño se marquen como muy diferentes.

En definitiva, dada la distancia entre dos contornos de cada imagen, se puede escoger la mínima distancia hallada entre un par de ellos. No se adjuntan resultados gráficos a esta proposición porque el resultado, claramente, no es visual (salvo que se representen sólo los dos contornos implicados), pero en cualquiera de los casos, la solución obtenida no es buena, lo que nos lleva en su momento a estudiar la alternativa explicada anteriormente.

2.2.11.1.2 - Registro imagen RGB y térmica

El último apartado de implementación de este trabajo va dirigido al registro de una imagen RGB y una imagen térmica capturadas por el mismo dispositivo, DJI XT2, y en el mismo instante de tiempo, por lo que el problema debe necesariamente simplificarse respecto de la situación anterior.

En cualquiera de los casos, nos encontramos las dificultades que han venido apareciendo durante todo el proyecto:

- Distorsión *fisheye* en la imagen RGB, capturada con una lente de distancia focal 8 mm, a lo que se debe añadir que no se dispone de ningún tipo de información al respecto, en cuanto a coeficientes de distorsión y otros metadatos nos referimos.

  A diferencia del problema anterior, donde teníamos una imagen RGB (presumiblemente con distorsión *fisheye*) y una imagen multiespectral rectificada (donde la diferencia entre ellas quizás no era tan grande), ahora tenemos una imagen RGB y una imagen térmica cuya distancia focal es mucho mayor y por lo tanto no presenta esa distorsión. Esta situación se puede ver claramente en la Ilustración 123.

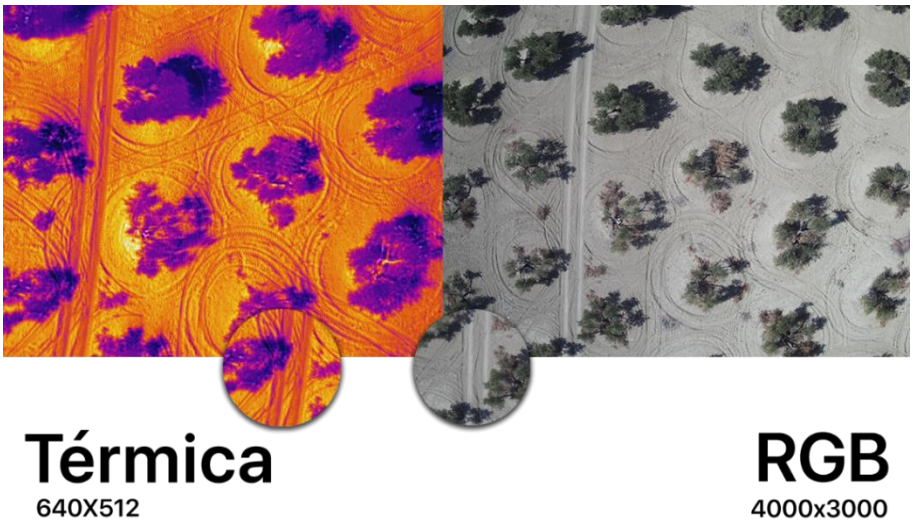


Ilustración 123. Comparativa de imagen térmica y RGB. Detalle de huellas donde se aprecia la distorsión de esta segunda.

- Ambas representan áreas diferentes, y la principal causa es la diferencia entre las distancias focales de las lentes, que originan ángulos de visión muy dispares. Así, la térmica es capaz de capturar un menor área del terreno. De la distribución de las lentes en el dispositivo, que se muestra en la Ilustración 9, se puede extraer que también debe existir cierta traslación horizontal entre ambas imágenes, aunque nosotros partiremos siempre de la premisa de que la térmica se halla en el centro exacto de la imagen RGB.

  Nótese como el ángulo de visión de las lentes depende de la distancia focal, pero también de otros parámetros como la anchura del sensor, algo que ya se explicó en anteriores apartados. Por lo tanto, el área representada en la térmica no se corresponde directamente con el factor $Distancia\ focal_{Térmica}\ /\ Distancia\ focal_{RGB}$.

  Tampoco se conocen esos parámetros de los que depende directamente el ángulo de visión, por lo que el método que a continuación se propone vendrá influenciado por toda esta problemática.

La estrategia a seguir mantiene como base el algoritmo de registro que hemos venido usando: ECC. De esta forma, los pilares fundamentales de este algoritmo son los siguientes:

- Eliminación de la distorsión *fisheye*, estimando un conjunto de parámetros que se pueden aplicar al algoritmo ya bien conocido. Más concretamente,

se aplican los coeficientes de distorsión de las imágenes multiespectrales (son estáticos), y como distancia focal se usa un coeficiente medio que también se corresponde a las multiespectrales, pero multiplicado por dos, que es el factor de escala entre las distancias focales en mm de ambas lentes ($Distancia\ focal_{RGB}\ /\ Distancia\ focal_{Multiespectral}\ =\ 2$).

- Selección de una parte de la RGB en la que sabemos a ciencia cierta que se encuentra la imagen térmica. Se estima así un porcentaje de la primera imagen que determina el área de búsqueda, aunque en realidad el algoritmo seguiría funcionando en el caso de que estimemos un porcentaje demasiado pequeño o demasiado grande. En el primero de los casos no se consideraría toda la información de la térmica (por extender), y en el segundo, al registrarla se dejarían unos bordes negros en el área objetivo, dado que la imagen térmica no pertenece a esa región en su totalidad.

  En realidad, ninguna de estas casuísticas genera ninguna pérdida, simplemente mejoran o empeoran la visualización. Es decir, al final, lo que se obtiene con este algoritmo es una matriz de transformación global M_i para la térmica, que se está estimando de forma bastante precisa en ambos casos. De hecho, para la visualización del resultado queda mucho mejor si escogemos un porcentaje de área menor.

  Para ese caso, el principal problema puede estar en que dicha transformación global no se ajuste correctamente a los extremos de la imagen térmica, debido a que que la intersección de información entre ambas capas no incluye a la imagen térmica por completo, y por lo tanto, no se está calculando M_i teniendo en cuenta dichos extremos. Por lo tanto, si escogemos un porcentaje de área pequeño, el registro se basa sólo en una parte de la térmica.
- Al involucrar tantos parámetros en la falta de alineación entre ambas imágenes, se debe considerar un modelo de transformación global más complejo, como es de nuevo la homografía. Como sucediera en etapas anteriores, para disminuir el tiempo de ejecución del algoritmo se decrementa el tamaño de las imágenes a registrar notablemente, lo que implica que habrá que aumentar la precisión y el número de iteraciones.

El algoritmo a seguir se muestra en la Ilustración 124.

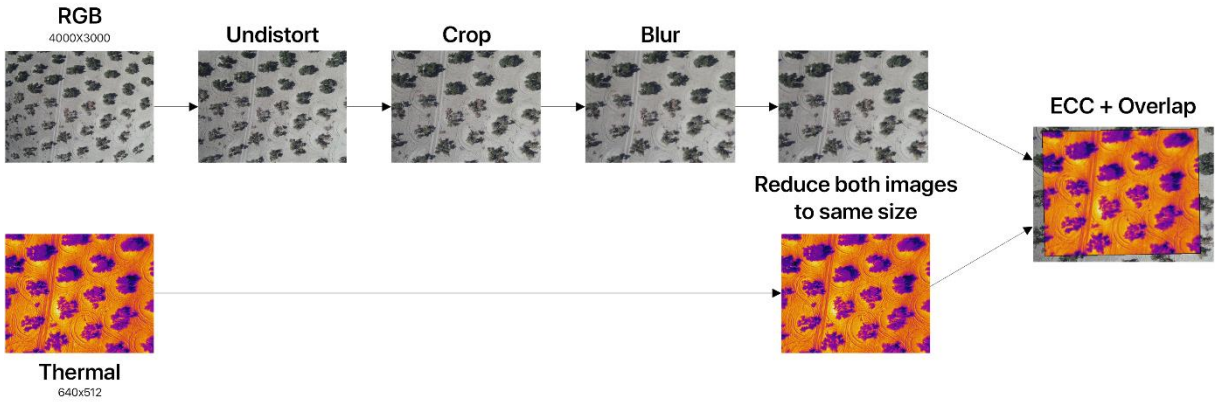


Ilustración 124. Esquema de registro de imagen RGB con una imagen térmica.

Y de forma no tan gráfica, a través de pseudocódigo e indicando los valores de algunos parámetros, se puede ver en la Ilustración 125.

Algorithm 14: Registers the thermal image with an RGB image which captures a larger area

```
1  function RegisterThermalWithRGB (thermal, rgb);
   Input  : two images: thermal and RGB
   Output: rgb image with an overlapped thermal image once it's registered
2  rgbArea ← (rgb.width * WIDTH PERCENTAGE, rgb.height *
     HEIGHT PERCENTAGE);
3  rectangle ← ((rgb.width − rgbArea.width)/2, (rgb.height −
     rgbArea.height)/2, rgbArea.width, rgbArea.height);
4  undistortedRGB ←
     rgb.undistort(FOCAL LENGTH, rgb.size/2, DISTORT COEFF);
5  croppedRGB ← undistortedRGB.crop(rectangle);
6  blurredRGB ← croppedRGB.blur(10);
7  rgbResized ← blurredRGB.resize(thermalSize/2);
8  thermalResized ← thermal.resize(thermalSize/2);
9  alignedImage ←
     thermalResized.alignImage(rgbResized, HOMOGRAPHY, 300, 1e −
     60);
10 resizedAligned ← alignedImage.resize(rgbArea);
11 result ← rgb.copy(resizedAligned, rectangle);
12 Libera memoria;
13 return result;
```

Ilustración 125. Pseudocódigo para obtener el registro de una imagen térmica y una imagen RGB.

Nótese como el área de recorte de la imagen RGB viene definido por el centro de la misma y una anchura y altura que se corresponde a un porcentaje del tamaño de la imagen original. A su vez, la eliminación de la distorsión toma unos parámetros estimados y especificados a través de constantes, mientras que el punto principal de la imagen se considera, por simplificación, que es el centro de la misma. Asimismo,

se reduce el tamaño de la imagen RGB recortada y emborronada a $\frac{tamaño_{térmica}}{2}$, siendo este factor divisor regulable para hacer el algoritmo menos costoso computacionalmente. Imaginemos, por ejemplo, una aplicación en tiempo real donde lo ideal es localizar algún elemento conforme se recorre el terreno, y no se busca una precisión exacta (como sucede normalmente en muchas aplicaciones de la imagen térmica: control de ganado, soporte para control de incendios, etc).

De esta forma, se muestran algunos resultados obtenidos en la Ilustración 126, Ilustración 127 e Ilustración 128, para los cuales se comenta en qué condiciones se han generado.

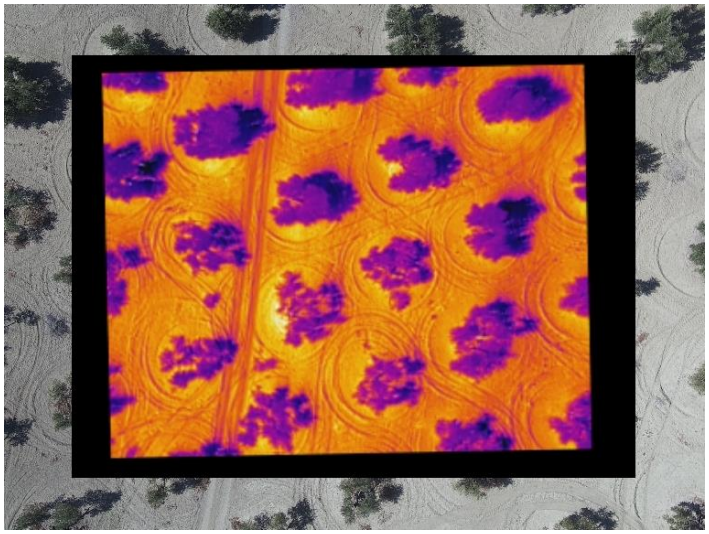


Ilustración 126. Registro utilizando un área de mayor tamaño.

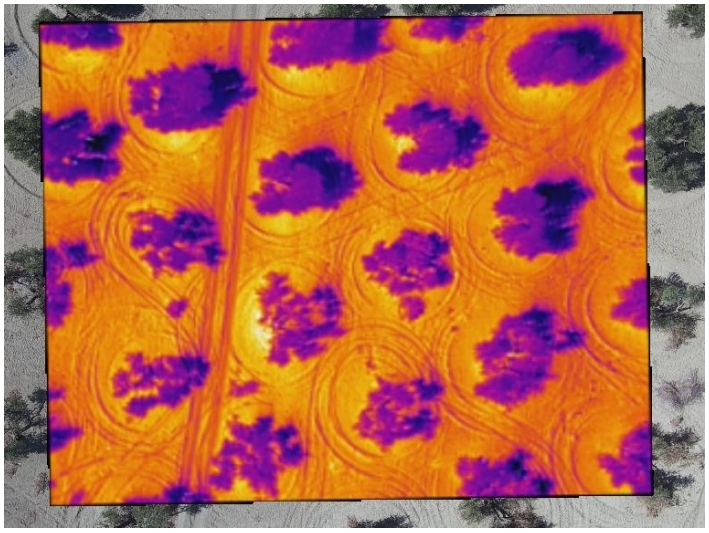


Ilustración 127. Registro de imágenes donde se muestra sólo el área recortada de la imagen RGB y la térmica superpuesta, eliminando el fondo negro.

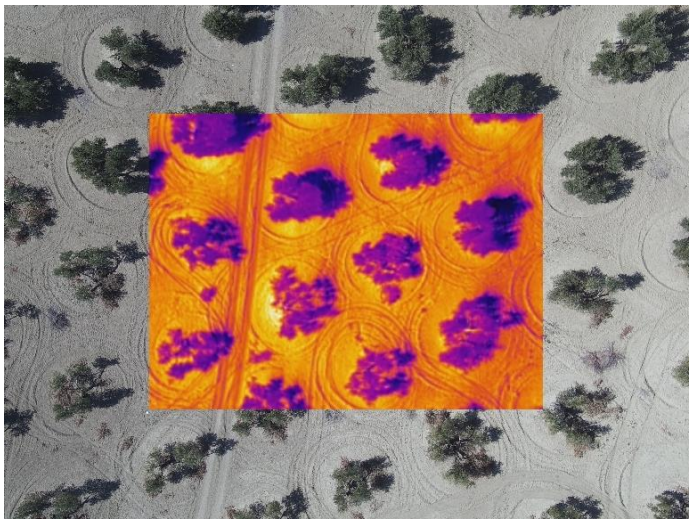


Ilustración 128. Registro utilizando un área más pequeña que en figuras anteriores.

Condiciones de generación de cada resultado:

1. Se ha escogido un área de la RGB que representa el 80% de la misma. Dado que la imagen térmica se corresponde a un área menor, aparecen estos bordes negros, pero en cualquiera de los casos el registro parece correcto (se puede ver en la Ilustración 127).
2. Mismas condiciones que en la figura anterior, sólo que ahora el resultado muestra simplemente el área recortada de la RGB, y encima de esta se solapa la imagen térmica rectificada, eliminando los bordes negros anteriores.
3. A diferencia de resutados previos, ahora se ha escogido un área de la imagen RGB que representa una parte del terreno inferior a lo capturado por la térmica. El registro sigue siendo correcto, aunque parte de la imagen térmica se ha recortado.

Se puede ver, por lo tanto, como el método aquí desarrollado funciona bajo diferentes circunstancias, por lo que tiene características bastantes deseables para el registro de imágenes RGB y térmicas, a lo que se debe sumar que el proceso es automático. Podemos observar además como el ajuste que se realiza sobre la imagen térmica no se corresponde únicamente a una traslación y un escalado, sino también a una rotación (ver Ilustración 126 e Ilustración 127), por lo que si supiéramos el factor de escala entre los ángulos de visión de ambas lentes, seguiríamos sin poder aplicarlo a la hora de seleccionar una parte de la imagen RGB. Esto implica que las imágenes

capturadas por ambas lentes no se obtienen exactamente en el mismo instante tiempo (en una pequeña fracción de tiempo puede haber cambiado la orientación y posición el dron) o bien que la diferencia entre imágenes no se corresponde únicamente a una traslación, como suponíamos a raíz de la distribución física de las lentes, sino también a una rotación debido a la orientación de los ejes ópticos, como ya sucediera en las imágenes multiespectrales.

A diferencia de algoritmos anteriores, donde vagamente podíamos comparar nuestros resultados con los obtenidos por otros investigadores o cualquier otra área de la sociedad, en este caso sí tenemos ejemplos muy cercanos. Las imágenes que a continuación se presentan no se corresponden a DJI XT2 sino al dispositivo ThermalCapture Fusion (ThermalCapture, 2019), el cual dispone de varios niveles de *zoom*, que marcan la precisión del registro de la imagen RGB y térmica (ver Ilustración 129).

Ilustración 129. Ejemplo de registro del dispositivo ThermalFusion Capture.

En esta imagen se pueden ver claros errores de registro en estructuras como coches, muros, etc, que se acentúan y se ven con mayor claridad al desplazar la barra superior, que modifica la opacidad de la imagen térmica.

Por ejemplo, registros realizados por nuestro algoritmo con estructuras que no sean árboles pueden ser los que se muestran en la Ilustración 130.

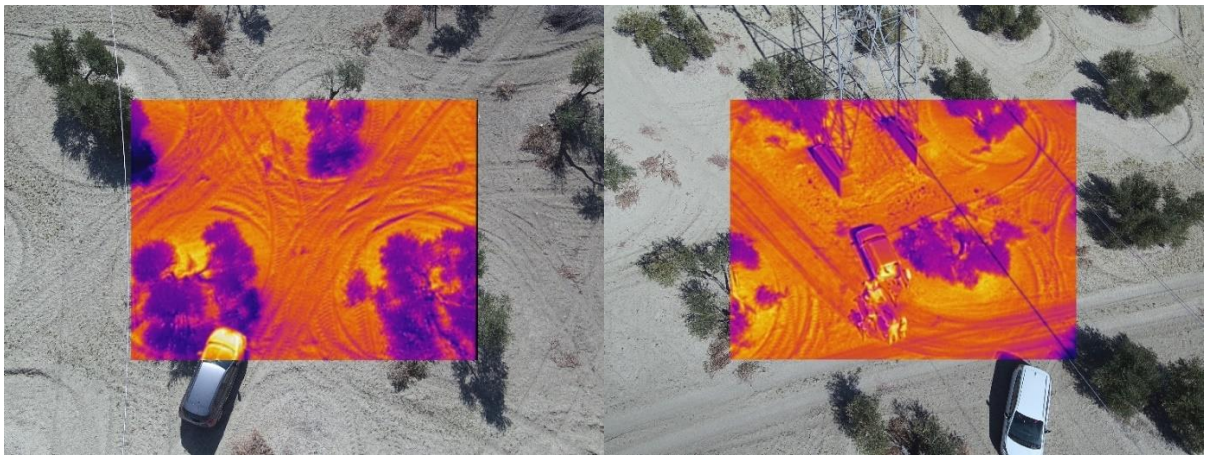


Ilustración 130. Registro de imágenes multiespectrales y térmicas con estructuras alternativas a la vegetación.

También cabe destacar que el dispositivo anteriormente citado dispone de varios niveles de ajuste de precisión, que se definen como *zoom*. En nuestro caso, esta posibilidad podría depender de los parámetros del algoritmo de registro ECC, aumentando o disminuyendo precisión y número de iteraciones.

Como en todos los apartados anteriores, se finaliza este último con aquellos metadatos empleados a lo largo de todos los algoritmos aquí expuestos, y que no se hayan mostrado antes (ver Tabla 14).

Tipo de metadato	Etiqueta de acceso	Descripción
Fecha de captura	Exif.Image.DateTime	Cadena con formato Año:Mes:Día Hora:Minuto:Segundo.

Tabla 14. Nuevos metadatos empleados durante esta iteración.

2.2.11.2 - Diagramas de secuencia

A continuación se muestran aquellos diagramas de secuencia relacionados con las acciones que se han presentado en esta última iteración (Ilustración 131, Ilustración 132, Ilustración 133).

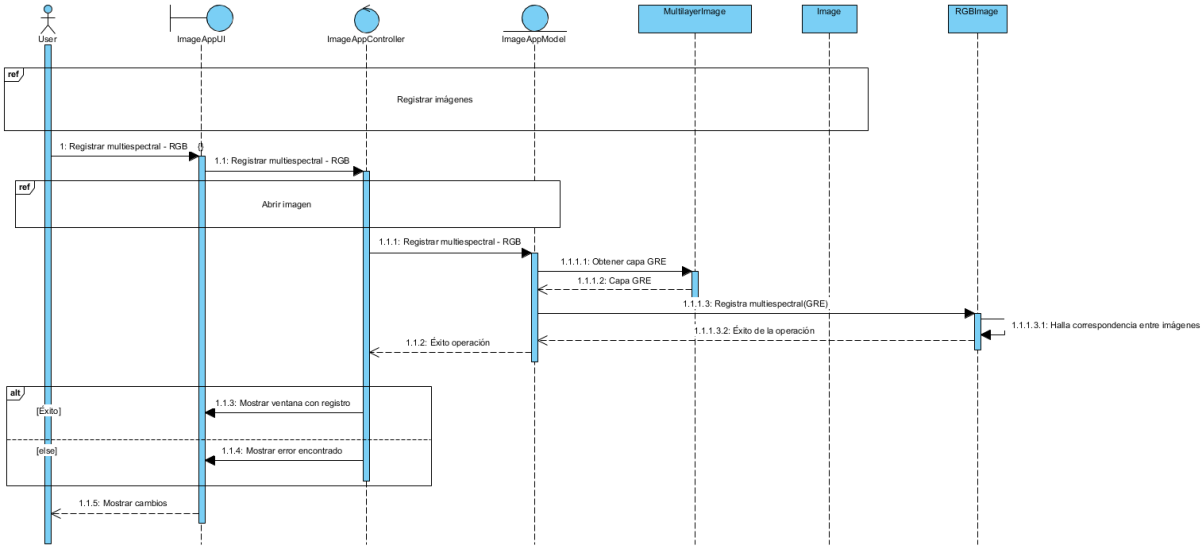


Ilustración 131. Diagrama de secuencia para la acción de registrar imagen multiespectral con RGB.

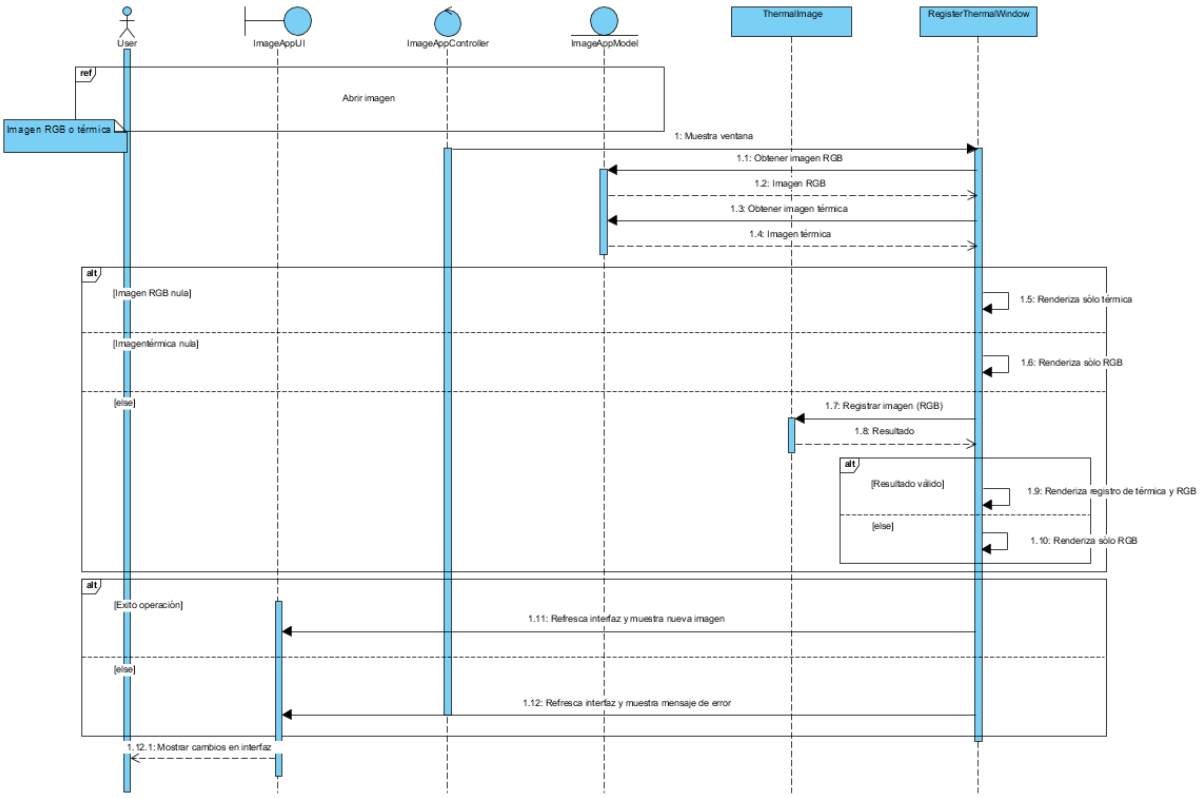


Ilustración 132. Diagrama de secuencia para incorporar a la aplicación una imagen térmica o RGB.

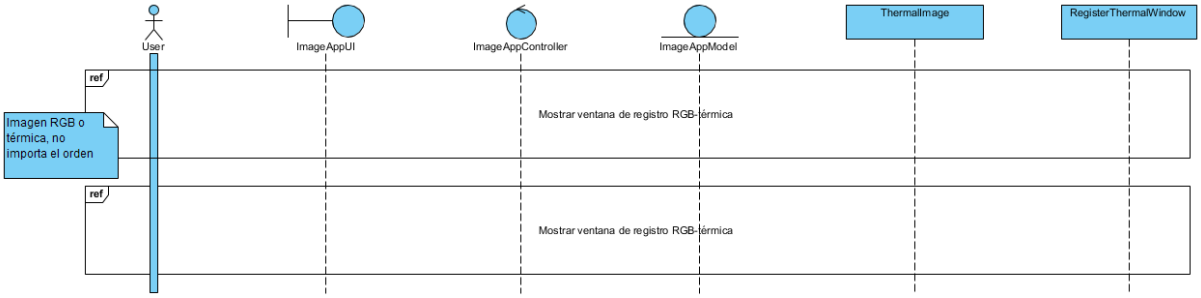


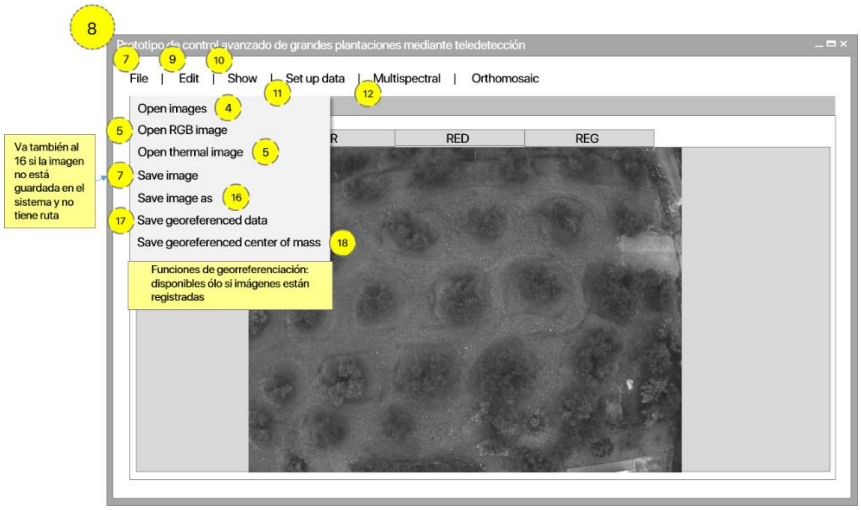
Ilustración 133. Diagrama de secuencia de la incorporación de las imágenes RGB y térmica a la aplicación, desencadenando su registro.

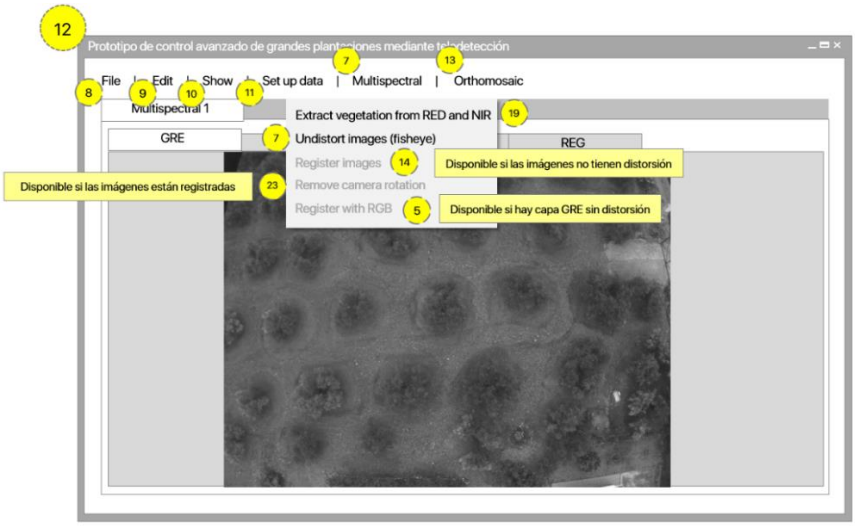
Nótese como para poder registrar una imagen multiespectral con una RGB es necesario un registro previo de la primera, lo que implica que debe existir la capa GRE, que es la que más se asemeja a la imagen RGB en escala de grises, y es que la que se emplea para esta operación.

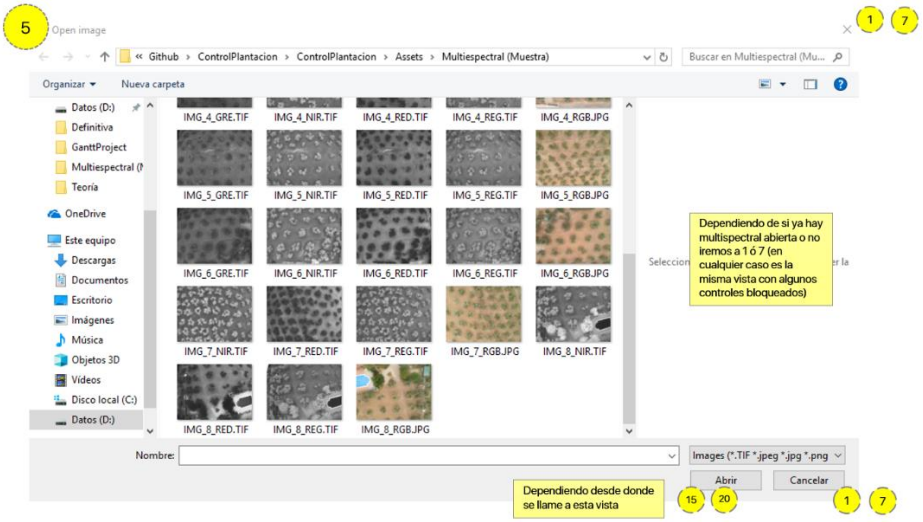
En el caso del registro RGB-térmica, en el momento en el que se abre la segunda de ellas, sin importar el orden, se reconoce que hay dos imágenes y se efectúa el registro.

2.2.11.3 - Storyboards

Las vistas de la aplicación involucradas en la obtención de las imágenes registradas se muestran en la Ilustración 134.




12
Prototipo de control avanzado de grandes plantaciones mediante teledetección
File
Edit
Show
Set up data
Multispectral
Orthomosaic
Multispectral 1
Extract vegetation from RED and NIR
GRE
Undistort images (fisheye)
REG
Register images
Disponible si las imágenes no tienen distorsión
Disponible si las imágenes están registradas
Remove camera rotation
Register with RGB
Disponible si hay capa GRE sin distorsión


5
Open image
Github
ControlPlantacion
Assets
Multiespectral (Muestra)
Buscar en Multiespectral (Mu...
Organizar
Nueva carpeta
Datos (D:)
Definitiva
GanttProject
Teoría
OneDrive
Este equipo
Descargas
Documentos
Escritorio
Imágenes
Música
Objetos 3D
Vídeos
Disco local (C:)
IMG_4_GRE.TIF
IMG_4_NIR.TIF
IMG_4_RED.TIF
IMG_4_REG.TIF
IMG_4_RGB.JPG
IMG_5_GRE.TIF
IMG_5_NIR.TIF
IMG_5_RED.TIF
IMG_5_REG.TIF
IMG_5_RGB.JPG
IMG_6_GRE.TIF
IMG_6_NIR.TIF
IMG_6_RED.TIF
IMG_6_REG.TIF
IMG_6_RGB.JPG
IMG_7_NIR.TIF
IMG_7_RED.TIF
IMG_7_REG.TIF
IMG_7_RGB.JPG
IMG_8_NIR.TIF
IMG_8_RED.TIF
IMG_8_REG.TIF
IMG_8_RGB.JPG
Dependiendo de si ya hay multispectral abierta o no iremos a 1 ó 7 (en cualquier caso es la misma vista con algunos controles bloqueados)
Nombre:
Images (*.TIF *.jpeg *.jpg *.png
Abrir
Cancelar
Dependiendo desde donde se llame a esta vista
15
20
1
7

20
Prototipo de control avanzado de grandes plantaciones mediante teledetección
Register RGB with thermal
Siempre y cuando se puedan registrar porque haya solape entre ambas imágenes.

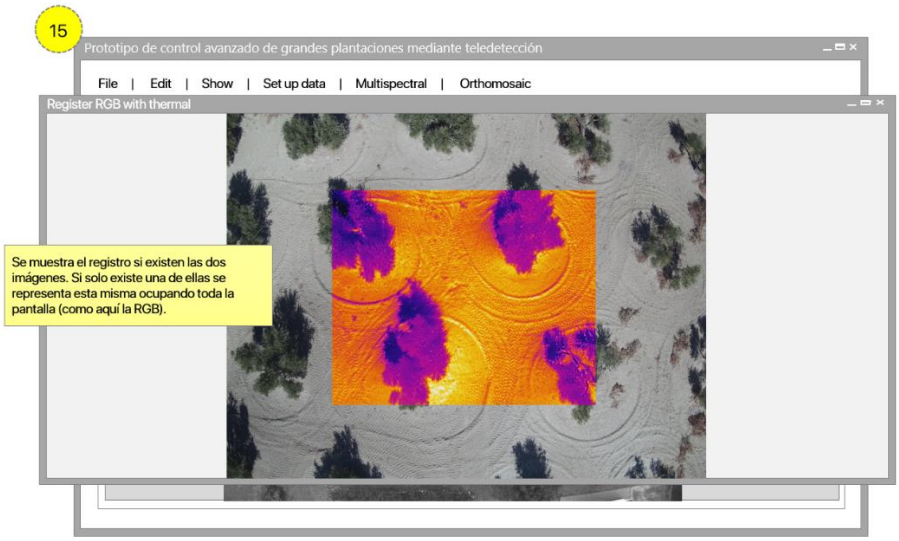


Ilustración 134. Secuencia de vistas necesarias para obtener el registro RGB-térmica y RGB-multiespectral.

2.2.12 - Diagrama de clases

Una vez concluido el proceso de desarrollo de este trabajo, se pretende concentrar en este apartado el análisis de la estructura del software creado. El hecho de integrar esta sección al final de todo este proceso iterativo viene influenciado por el instante del proyecto que representa este diagrama, dado que incluye al mismo en su totalidad. No parece correcto por lo tanto mostrar al lector de este documento parte de software aún no desarrollada en cierta iteración. No se ha hecho esto mismo para el *storyboard* porque se trata de una representación esquemática de la interfaz de usuario, la cual tenía como objetivo ayudarnos a comprender la comunicación con el usuario implementada en cada iteración.

Se divide este apartado en tantos módulos como haya en la aplicación, que a su vez ya aparecieron en la iteración donde se adaptaba el proyecto al esquema Modelo-Vista-Controlador (ver Sexta iteración). Dado que el diagrama de clases de un único módulo ya tiene un tamaño considerable, se intentará hacer un análisis de ciertos subconjuntos de clases, haciendo especial hincapié en el núcleo de este trabajo.

Como punto final a este apartado, se hará una breve mención a aquellos patrones, además del MVC, empleados al desarrollar el software.

2.2.12.1 - Aplicación genérica

Se deben enmarcar aquí aquellas clases que se comunican con varios módulos o realizan un control más genérico, así como otras tantas que se adaptan a cualquiera de estos módulos: vistas genéricas, espacios de nombres con operaciones geométricas, enumeraciones genéricas, etc.

El esquema que representa al módulo completo es el de la Ilustración 135.

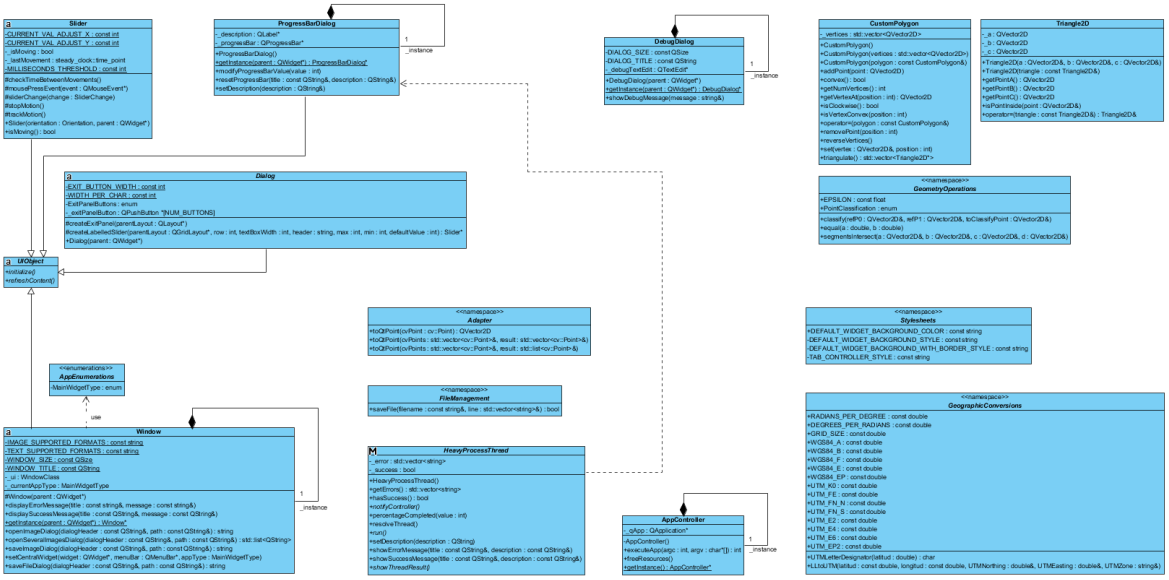


Ilustración 135. Diagrama de clases para el módulo de la aplicación al completo.

Este esquema que se acaba de presentar se divide en varios subconjuntos para facilitar su comprensión (Ilustración 136, Ilustración 137, Ilustración 138 e Ilustración 139).

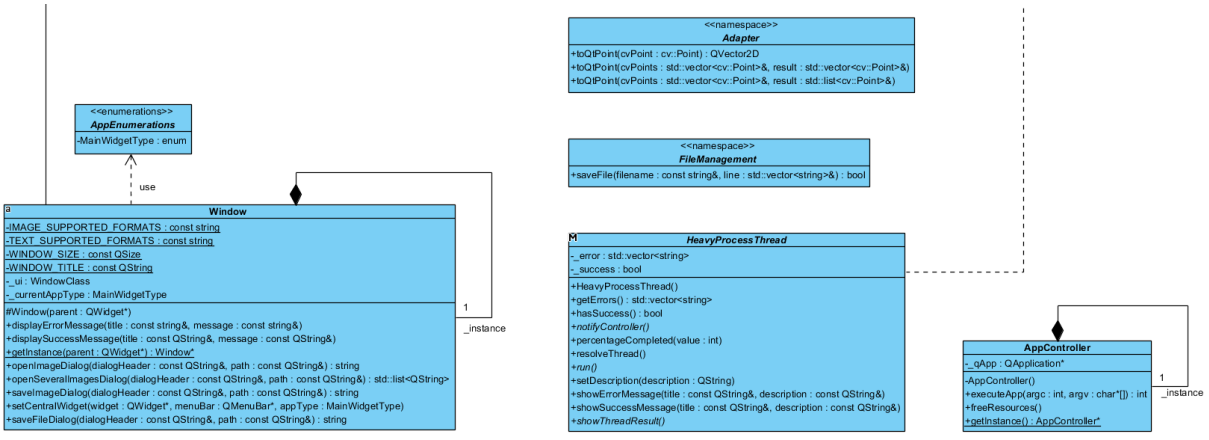


Ilustración 136. Diagrama de clases parcial del módulo de aplicación.

- **AppController**: es el controlador de la aplicación, que como se explicaba previamente, tiene una funcionalidad bastante reducida. Por ejemplo, se encarga de ejecutar la aplicación o liberar recursos de la misma, pero nada más allá de esta función. Su inclusión favorece la adaptación del proyecto a una aplicación con un mayor número de módulos independientes. Asimismo, implementa un patrón que veremos a lo largo de muchos de nuestros diagramas de clases, Singleton. Dado que hay un apartado exclusivo dedicado a esta importante parte del software, no se desarrolla más aquí.
- **Window**: representa la ventana principal de la aplicación, y su función es controlar sus propiedades (tamaño, título, etc), así como mostrar diálogos válidos para todos los módulos, tales como aquellos que permiten el guardado o la apertura de una imagen o cualquier otro fichero de texto, o aquellos mensajes de éxito / error que se representan al finalizar una operación.
- **AppEnumerations**: es simplemente un contenedor de enumeraciones cuyo principal propósito consiste en dar nombre a cada módulo, de tal forma que la ventana o el controlador puedan guardar el identificador del módulo que se encuentra activo en este momento.
- **HeavyProcessThread**: clase base que da soporte a cualquier tipo de proceso pesado, como bien podría ser el registro de imágenes o el almacenamiento de datos en fichero. Se da soporte a algo así porque procesos de tal magnitud deberían informar al usuario del estado del mismo a través de la interfaz. Sin embargo, no basta sólo con actualizar la interfaz desde un proceso tan pesado (no funciona), sino que la actualización, por ejemplo, de una barra de progreso, debe realizarse a través del mecanismo de señales de Qt en un hilo aparte. Por lo tanto, esta clase da soporte a actualizaciones del progreso de un proceso, notificaciones de resultados, etc.
- **FileManagement**: se especializa en el almacenamiento de información en este caso, pero el nombre de este espacio se abre a incluir más operaciones relacionadas con ficheros en un futuro.

- **Adapter**: dado que en este trabajo se incluyen múltiples librerías, y algunas de ellas emplean entidades equivalentes pero asociadas a su propio proyecto, como es el caso del vector 2D (OpenCV: cv::Vec2, Qt: QVector2D), debe existir alguna herramienta que realice la traducción. Relacionado con el patrón Adaptador.

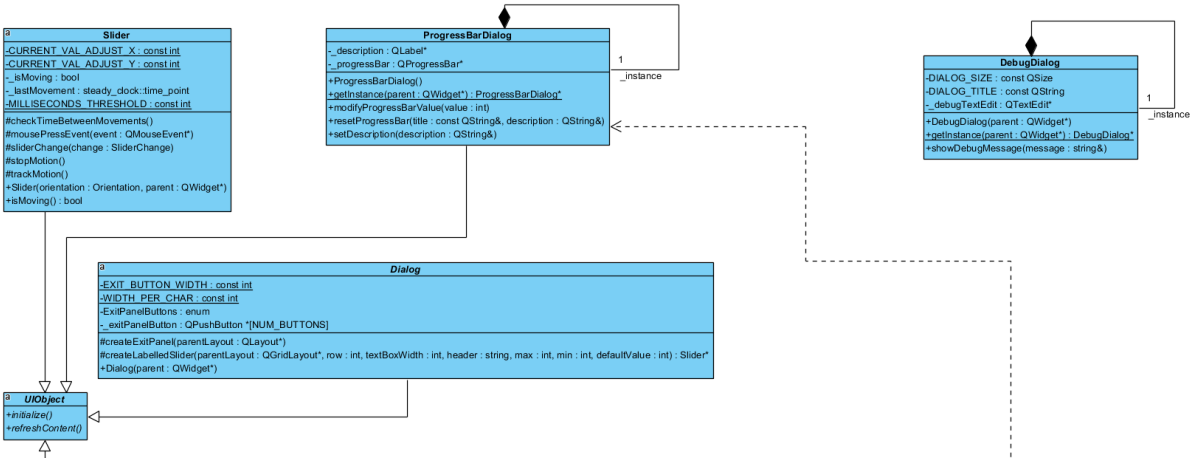


Ilustración 137. Diagrama de clases de entidades relacionadas con la interfaz del módulo de aplicación.

- **UIObject**: interfaz vital para este trabajo, ya que todos los elementos que se correspondan con la vista del proyecto deben implementar esta clase, obligándoles así a inicializar sus componentes y a desarrollar algún mecanismo para refrescar su contenido (si fuera sensible a cambios).
- **Dialog**: se trata de otra clase abstracta que implementa una funcionalidad básica para cualquier diálogo de esta aplicación (modificación de la altura de vuelo estimada, selección de parámetros de extracción de contornos, etc). Encapsula la creación de componentes tan generalizados como dos controles entrelazados, *slider* y *text edit*, los cuales se realimentan al modificarse.
- **ProgressBarDialog**: implementa justamente la clase anterior, de tal forma que da soporte al hilo que controla la ejecución de procesos pesados, por lo que debe permitir modificar progreso, descripción o título (al comenzar una nueva tarea).
- **DebugDialog**: diálogo con un propósito puramente de desarrollo, ya que se encarga de mostrar mensajes generados en el código. Por ejemplo, si un

shader no se puede compilar aparece en esta ventana. De cara al usuario final, este diálogo debería desaparecer. Su única función es añadir un nuevo mensaje a la consola.

- **Slider**: control extendido respecto del *slider* que nos proporciona Qt, dado que este no contiene ciertas funcionalidades deseables en un objeto de este tipo. Algunas características de este elemento de la interfaz son las siguientes:
  - Añade un mensaje sobre el propio *slider* mientras se está deslizando, para visualizar el valor actual.
  - Incluye eventos de cambio de valor incluso aunque el usuario no libere el control. De esta forma, nos permite actualizar la interfaz tras un período de tiempo, aunque el usuario no haya decidido el valor final (se mantiene pulsado el *tick* del *slider*).

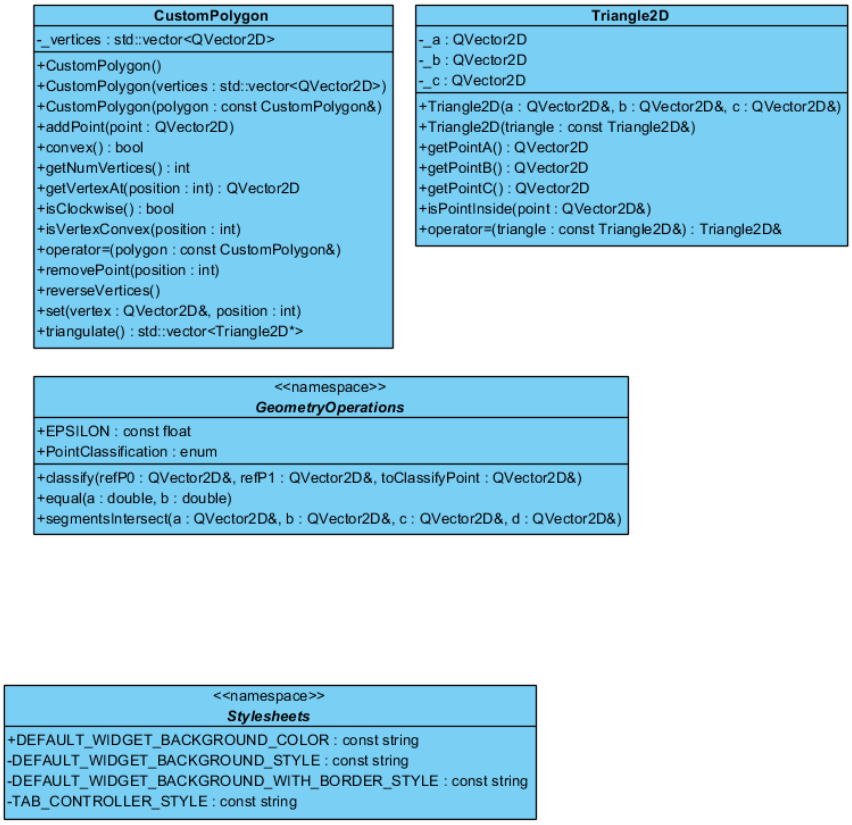


Ilustración 138. Diagrama de clases de espacios de nombres y clases geométricas del módulo de aplicación.

- **Clases geométricas**: CustomPolygon y Triangle2D. Representan entidades que sirven de apoyo para la triangulación del polígono que forma el contorno de un árbol. Tienen sólo las operaciones necesarias para llevar a cabo dicha operación, dado que es el único punto desde el que se requieren estas herramientas. Buena parte de sus métodos se refieren al acceso a los puntos que componen la forma así como a la sobrecarga de operadores, dado que son objetos que con frecuencia se copian o procesan de manera similar.
- **GeometryOperations**: de nuevo, es un espacio que alberga operaciones geométricas básicas, como la obtención de la posición relativa de un punto C respecto de un segmento, o un operador igual algo menos estricto (utiliza ε).
- **Stylesheets**. Espacio de nombres que almacena el estilo de la interfaz: color de fondo, bordes, etc.

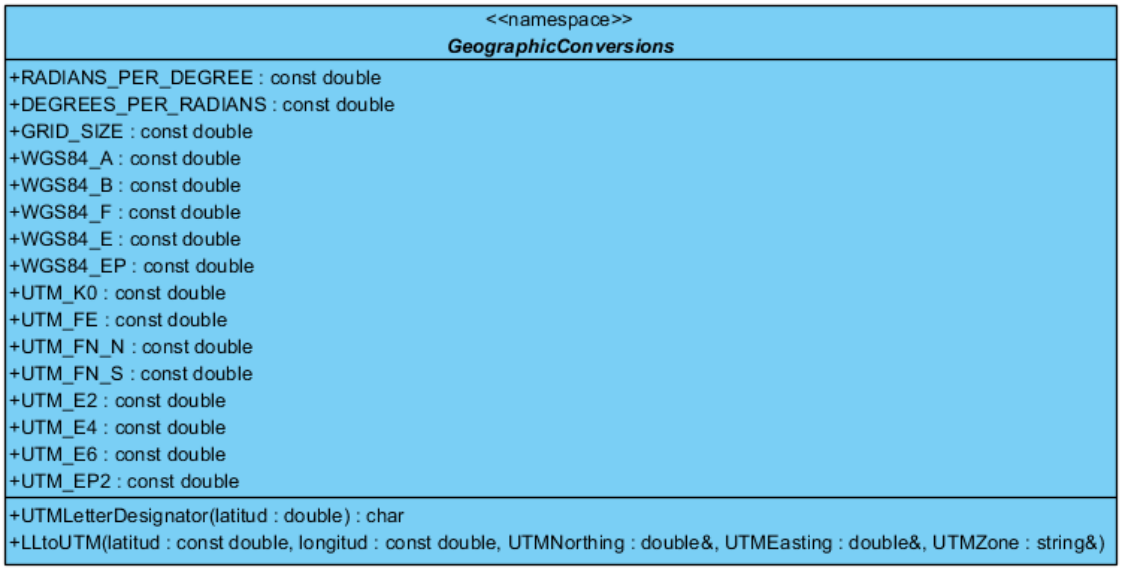


Ilustración 139. Esquema de la clase encargada de la traducción de coordenadas de sistemas de referencia diferentes.

- La última clase de este módulo se encarga de la traducción de coordenadas del tipo ($Latitud, Longitud$) a UTM, que es nuestra única operación necesaria en el ámbito de la georreferenciación. Se disponen en forma de constante todas aquellas variables de entrada que se definían cuando

describíamos el flujo de operaciones necesario para realizar la traducción $(Latitud, Longitud)$ a UTM.

2.2.12.2 - Módulo de procesamiento de imágenes

Una vez descritas las bases de la aplicación, se procede a definir todos aquellos elementos necesarios en el módulo principal de este trabajo y, por lo tanto, es el de mayor extensión y carga. De forma general, las clases se relacionan como sigue:

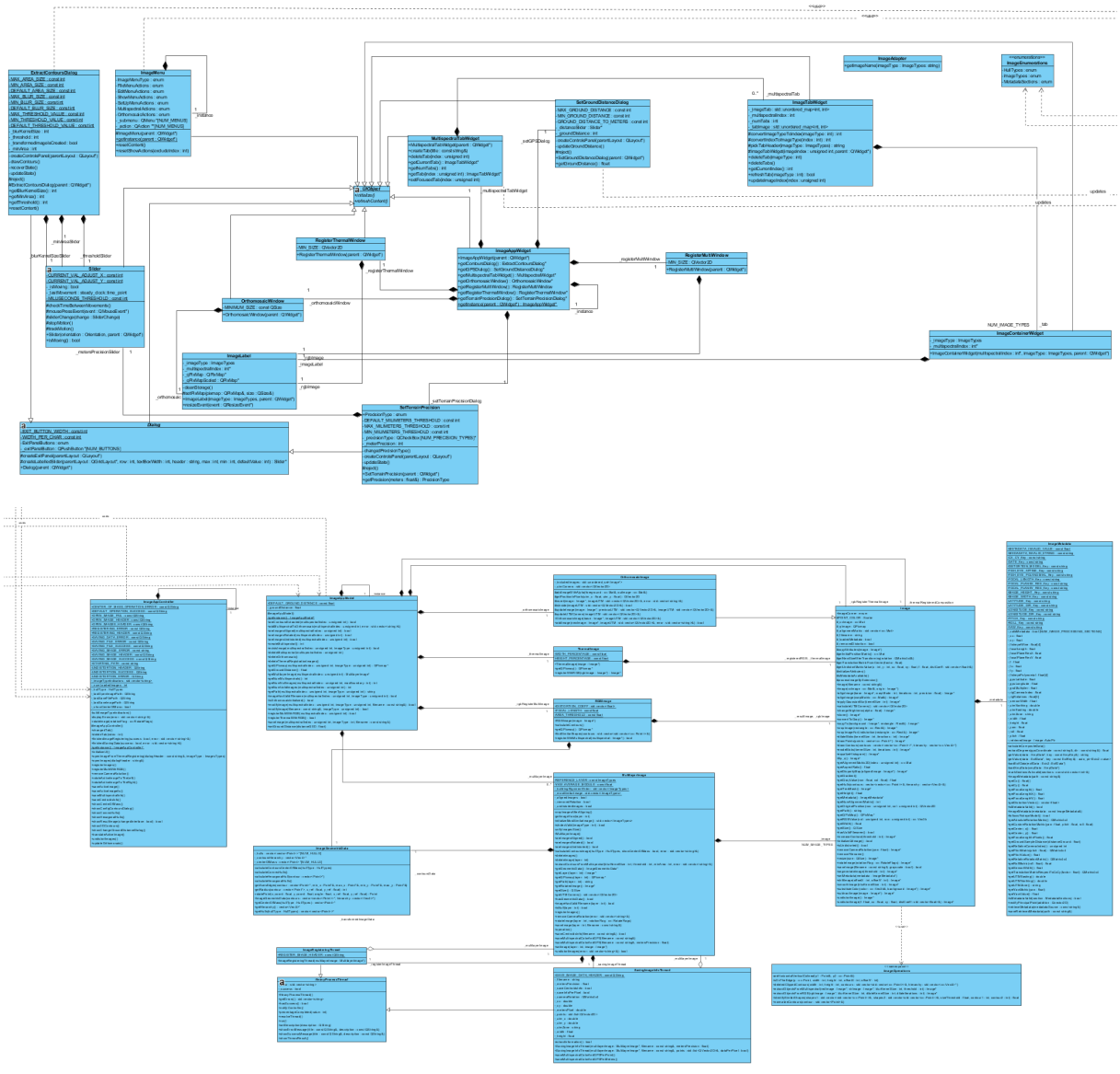


Ilustración 140. Diagrama de clases del módulo de procesamiento de imágenes al completo.

El objetivo de esta ilustración no es el de analizar cada entidad por separado, como es evidente, sino presentar cómo se comunican y relacionan las diferentes clases implementadas en este módulo. Para orientar al lector de este documento

correctamente, se volverán a introducir clases ya tratadas en diagramas de clases de secciones anteriores, dado que las entidades específicas de este módulo también utilizan, por supuesto, aquellos elementos necesarios del esquema genérico. Se dividen los diagramas de clases anteriores en subconjuntos, que van desde la Ilustración 141 hasta la Ilustración 149.

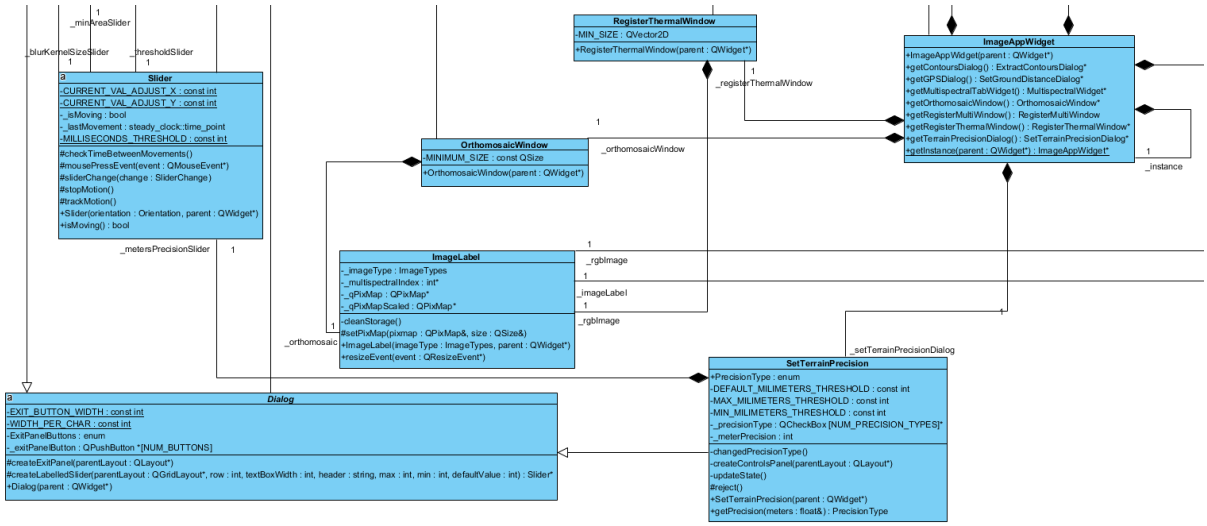


Ilustración 141. Diagrama de clases parcial de la vista (UI) del módulo de procesamiento de imágenes (I).

Dentro de la parte de la interfaz de este módulo se forma una jerarquía, no tanto a nivel de implementación sino de estructura visual, que se puede comprender a partir de la Ilustración 142.

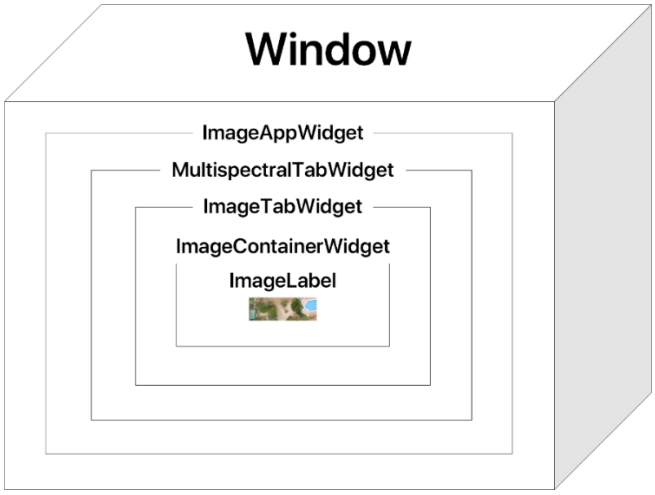


Ilustración 142. Jerarquía de componentes de la interfaz del módulo de procesamiento de imágenes.

En cualquiera de los casos, las clases presentadas en la Ilustración 141 son las siguientes:

- **ImageAppWidget**: se puede definir como el componente de mayor entidad dentro de la interfaz de este módulo. Crea todos aquellos diálogos relacionados con este apartado y permite el acceso a los mismos. Por debajo de él se integran todos los componentes que desembocan en la visualización de las imágenes multiespectrales.
- **ImageLabel**: justo al contrario que la última entidad, un *label* se utiliza para mostrar una imagen, y por lo tanto, es el último punto de la jerarquía. Se comunica con el modelo para pedirle una imagen concreta, y por ello, guarda qué tipo de imagen debe mostrar. En el caso de que fuera una multiespectral, debe almacenar también qué capa representa. Una parte muy importante de este elemento es su adaptación al entorno donde se muestra: debe mantener la relación de aspecto, pero integrándose dentro de las dimensiones de la ventana que alberga la instancia.
- **OrthomosaicWindow**: ventana secundaria que se limita a mostrar el ortomosaico, que no deja de ser una imagen.
- **RegisterThermalWindow**: misma descripción que para la clase anterior, con la diferencia de que ahora debe discriminar varios casos (existe sólo RGB, existe sólo térmica, o ambas, y en esta última situación la imagen resultado es el registro, siempre y cuando la operación termine con éxito). En cualquiera de los casos, la función de esta clase se sigue reduciendo a contener un *ImageLabel*.
- **SetTerrainPrecision**: diálogo para modificar la precisión con la que se guardan los datos georreferenciados. Debe inicializar al menos tantos *checkboxes* como tipos de precisión se permitan (información por píxel o por saltos de una determinada cantidad de metros), y un *slider* para la segunda opción citada (escoger cada cuántos metros se quiere información). Forma parte de un paso intermedio hacia la persistencia de los datos extraídos. Nótese como debe permitir el acceso a la precisión o método de almacenamiento elegido por el usuario, dado que el controlador sabe que el diálogo se ha cerrado, pero no qué opción ha sido elegida.

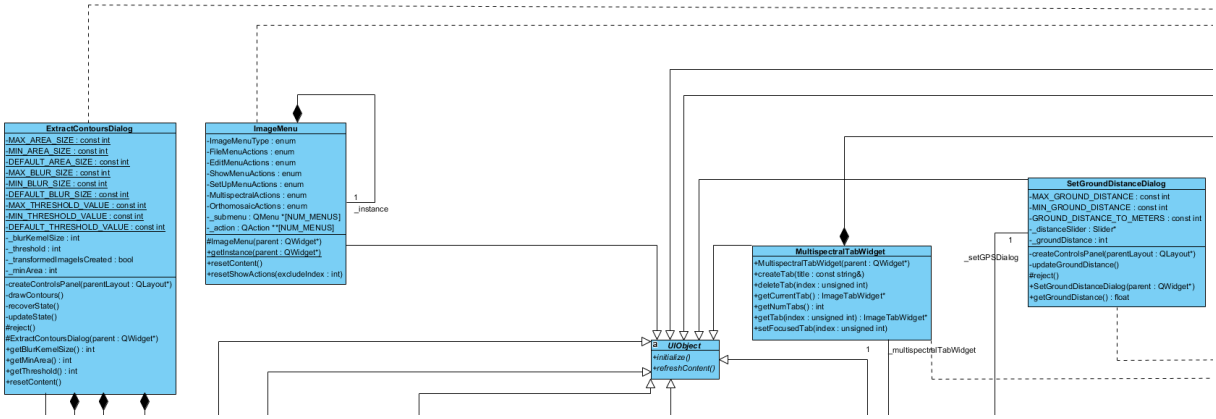


Ilustración 143. Diagrama de clases parcial de la vista (UI) del módulo de procesamiento de imágenes (II).

- **ExtractContoursDialog**: diálogo que contiene todos aquellos controles que permiten modificar los parámetros de extracción de contornos (tamaño del núcleo del filtro gaussiano, umbral de área y umbral de intensidad de gris). Debe definir claramente los límites en los que se mueven todos estos controles a través de constantes. Asimismo, el resultado de la modificación de estos parámetros, y de la extracción de contornos en general, se presenta en una ventana resultado. Nótese además como debe almacenar estados anteriores debido a que el diálogo contiene controles de Aceptar / Cancelar, por lo que este segundo puede implicar deshacer cambios, o incluso cerrar la pestaña de resultado, creada para mostrar los cambios en tiempo real.

- **ImageMenu**: menú superior que en ningún caso se ha considerado como un elemento genérico de la aplicación, debido a que sólo integra operaciones relacionadas con este módulo. Organiza todos los controles por submenú, y sus fases más importantes son la inicialización, donde relaciona cada uno de estos controles con un determinado *slot*, así como la fase de refresco de dichos controles, para habilitarlos o deshabilitarlos en función del estado de la aplicación. De acuerdo al modelo MVC, es el controlador quien recibe los eventos del usuario, y por lo tanto dichos *slots* pertenecen a la clase *ImageAppController*.

- **SetGroundDistanceDialog**: diálogo para modificar la altura estimada de vuelo. Dado que permite cancelar la operación, debe mantener una copia de su último estado (una variable *int*). Si bien garantiza acceso al valor de

la altura, esta misma clase se encarga de actualizar el modelo, por lo que no debiera ser necesario.

- **MultispectralTabWidget**: gestor de pestañas que a su vez contiene otro gestor de menor nivel en la jerarquía, pero con una funcionalidad mayor que la que este posee. Algo muy interesante sobre esta clase viene dado también por otra que ya hemos visto, *ImageLabel*, que guardaba el índice de la multiespectral por la que debía preguntar al modelo para obtener la imagen a representar. Si se borra una pestaña (multiespectral), debe actualizarse el índice al que estos *labels* deben hacer referencia, por lo que se produce una comunicación como la siguiente: *MultispectralTabWidget* > *ImageTabWidget* (tantos como imágenes multiespectrales) > *ImageContainerWidget* (tantos como imágenes multiespectrales y resultados: modifican el índice de la multiespectral, y dado que *ImageLabel* contiene un puntero a este valor, también se ve reflejado dicho cambio en este tipo de instancias).

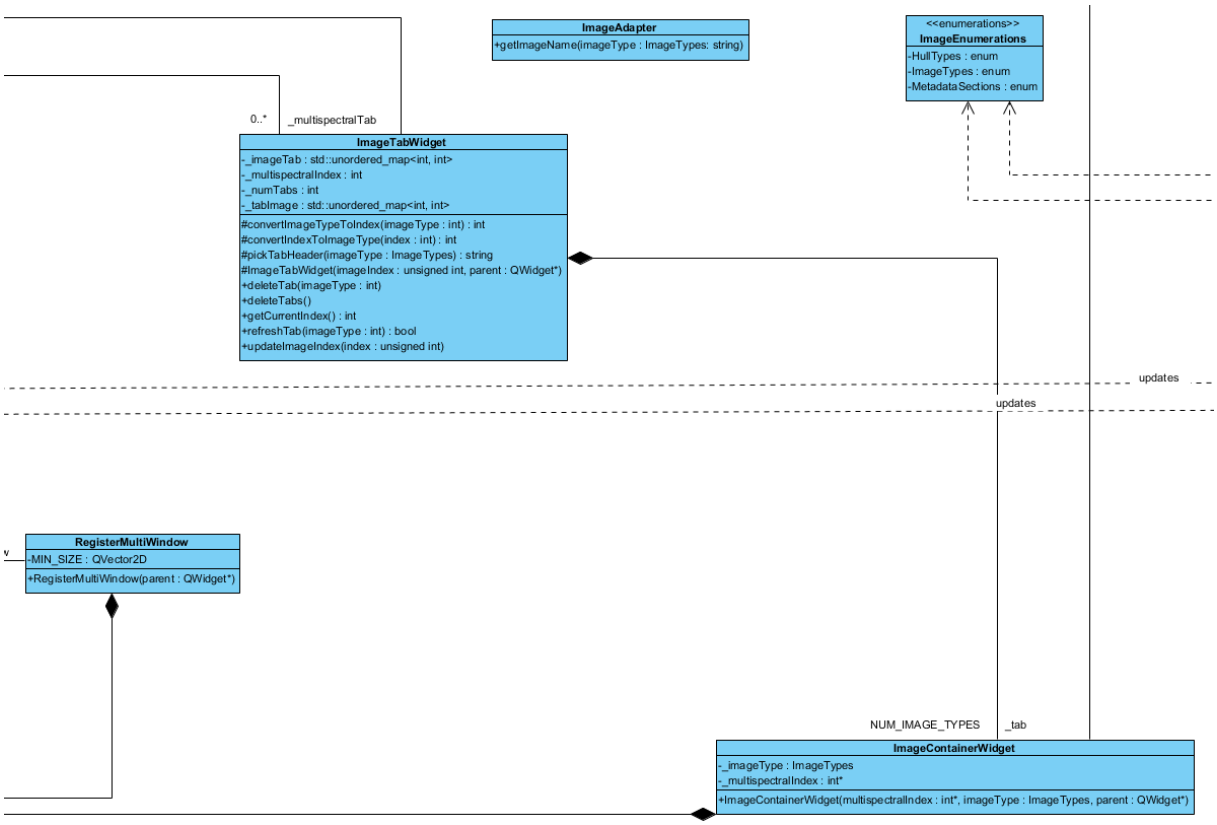


Ilustración 144. Diagrama de clases parcial de la vista (UI) del módulo de procesamiento de imágenes (III).

- **ImageTabWidget**: otro elemento más de la jerarquía anteriormente presentada. Gestiona directamente las pestañas de las imágenes multiespectrales y resultados de operaciones varias. La complejidad de esta clase radica en el hecho de que no siempre deben existir todos los tipos de imágenes, por lo que el identificador de una de ellas (se especifican en la clase *ImageEnumerations*) no debe corresponderse con el índice de la pestaña en la aplicación. Dado que se nos pide crear cada pestaña de forma individual, se puede mantener algún conjunto de estructuras como las que aquí se muestran:
    - **imageTab**: para cada índice de imagen se guarda su pestaña correspondiente en forma de tabla *hash*.
    - **tabImage**: para cada pestaña se guarda el índice de la imagen contenida.
- **ImageAdapter**: patrón adaptador exclusivo de este módulo, aunque su funcionalidad no es muy extensa dado que se limita a la conversión de un índice de imagen a una cadena representativa.
- **ImageEnumerations**: enumeraciones exclusivas de este módulo tales como índices de tipos de imágenes, identificadores de conjuntos de metadatos e identificadores de tipos de contornos. A pesar de lo que pueda parecer, es una clase vital para el desarrollo del proyecto, garantizando un acceso mucho más claro a todos los datos del modelo.
- **ImageContainerWidget**: se limita a almacenar un *label* con una imagen. Mientras que el *label* representa el espacio justo que necesita la imagen, guardando su relación de aspecto, este *widget* se expande y crea un fondo cuyo color viene determinado por *Stylesheets*. Contiene el índice de la multiespectral que se debe representar tal que *ImageLabel* contiene un puntero hacia esta variable.
- **RegisterMultiWindow**: ventana secundaria que representa el resultado de registrar una imagen RGB y una multiespectral.

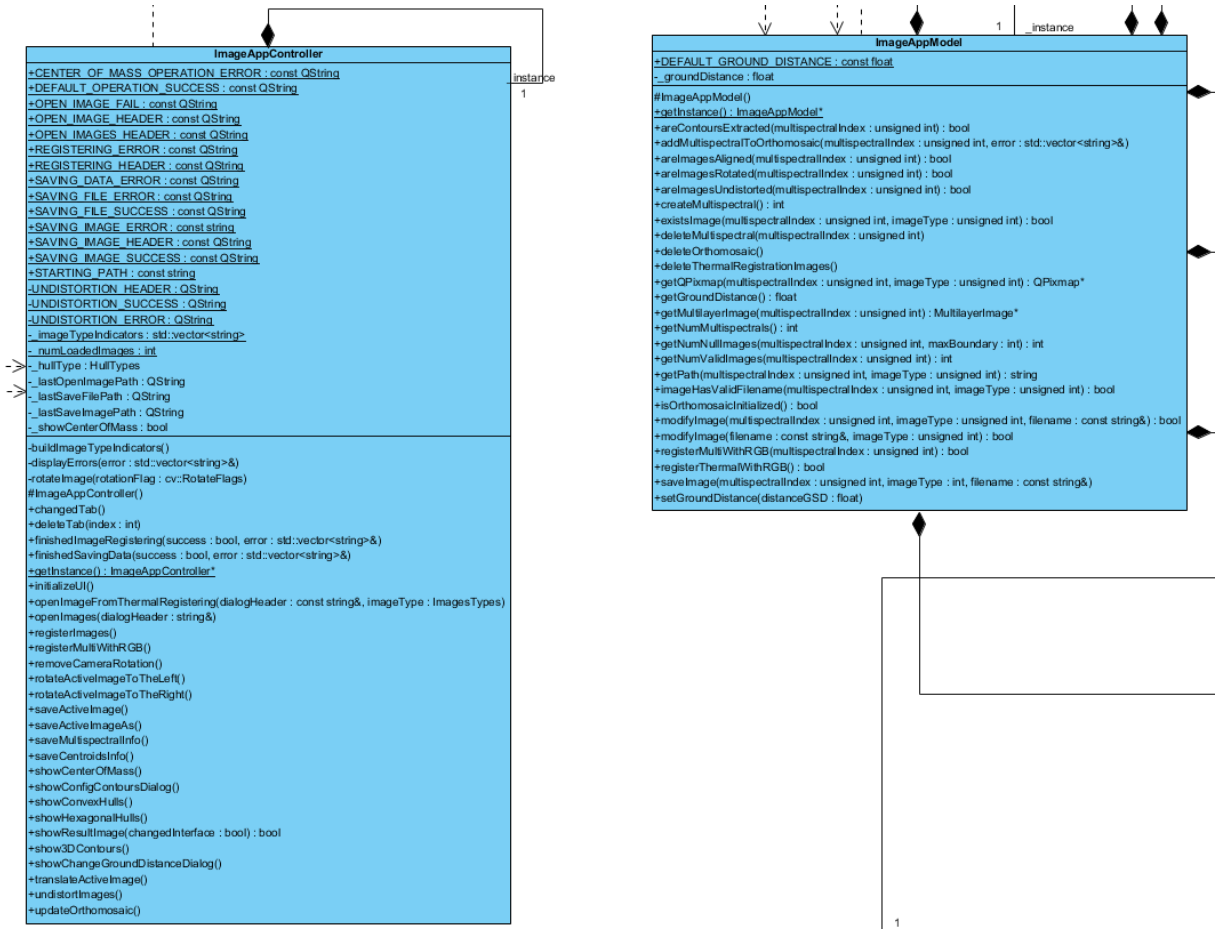


Ilustración 145. Diagrama de clases del modelo y controlador de este módulo.

Se incluyen en el esquema de la Ilustración 145 dos de las principales clases de este módulo, como son el controlador y el modelo del patrón MVC. Algunos detalles de estas entidades son los siguientes:

- **ImageAppController**: si bien se trata de una de las clases más extensas de este apartado, se puede reducir su descripción a comentar que todos estos métodos son *slots* de controles repartidos por toda la aplicación, aunque eso sí, tiene una gran carga de implementación siempre compartida con la vista y el modelo. Por supuesto, ninguno de los algoritmos propuestos se definen aquí, sino que esta clase es un intermediario entre prácticamente todos los componentes. Veáse también como almacena un gran número de constantes que servirán de títulos y descripciones para ventanas de éxito o error tras finalizar una operación desencadenada por la acción del usuario.
- **ImageAppModel**. Supone el acceso a los principales datos de este módulo: imágenes multiespectrales, RGB, térmica, etc. Tampoco se deben incluir

aquí los algoritmos expuestos, sino que esta clase actúa como un nivel de abstracción de cara al acceso a los datos de esta aplicación. Contiene muchos métodos para preguntar sobre ciertos detalles tales como como el número de multiespectrales activas, dado que el menú necesita esta información al actualizarse y no requiere las imágenes en sí.

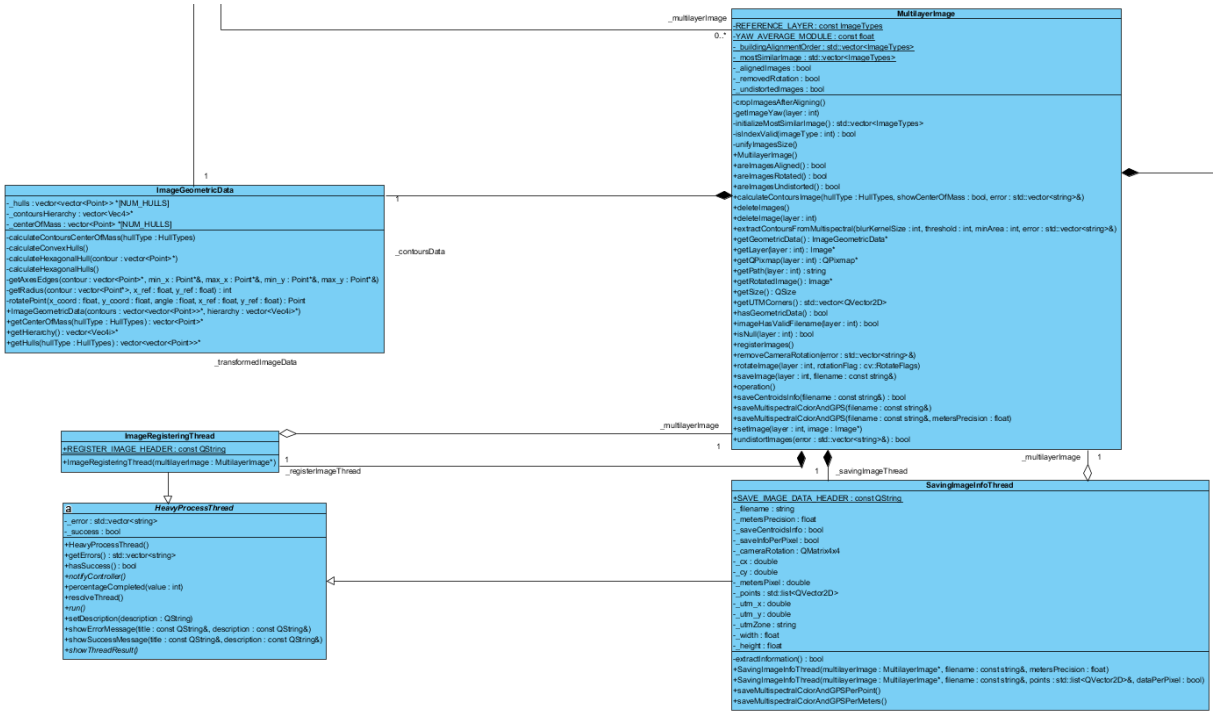


Ilustración 146. Diagrama de clases de utilidades del módulo de procesamiento de imagen (I).

- **ImageRegisteringThread**: se incluye esta clase para ejecutar el proceso de registro de imágenes multiespectrales, que se presupone costoso computacionalmente, en un hilo que nos permite actualizar la interfaz.
- **SavingImageInfoThread**. Tiene el mismo objetivo que la clase anterior, aislar un proceso pesado tal y como es el almacenamiento de datos en un fichero del sistema. Esta clase está algo más elaborada porque existen múltiples formas de guardar datos georreferrenciados: por píxel, cada cierto número de metros, etc, y no todas tienen la misma complejidad, o se pueden optimizar de cierto modo no válido para otras opciones.
- **ImageGeometricData**: almacena información asociada a los contornos de una imagen. Es por ello que debe ser un modelo multicapa el que almacene una instancia así, dado que se obtiene al combinar dos de sus capas.

Almacena tantos tipos de contornos como se definan en *ImageEnumerations*, así como los centros de masas para cada uno, teniendo en cuenta que estos varían de un tipo a otro. Nótese como no es necesario calcular un tipo de contorno hasta que se pida, y esto es lo que se comprueba precisamente en *getHulls*.

- **MultilayerImage**: modelo multicapa que almacena diversas imágenes multiespectrales y resultados de operaciones entre ellas. Veáse en el esquema como se almacenan tantas imágenes como tipos haya en *ImageEnumerations,* dado que ya se han incluido en esa enumeración los posibles resultados: contornos extraídos, imagen rotada, etc.

  Si bien esta clase aún no se encarga de las operaciones de más bajo nivel, sí que integra aquellas operaciones que involucran más de una captura, como puede ser el registro de imágenes. Asimismo, sirve como un nivel de abstracción para obtener información de imágenes, además de solicitar operaciones tales como el almacenamiento de información en el sistema de ficheros.

Ilustración 147. Esquema de la clase que representa a una imagen.

La clase *Image* es prácticamente el núcleo de esta aplicación, teniendo en cuenta que aquí se deben incluir todas las operaciones de bajo nivel de la imagen, desde una modificación del tamaño hasta una eliminación de la distorsión. Dado que muchas de esas operaciones necesitan de información de los metadatos, se almacena una instancia de la clase *ImageMetadata* que se presenta a continuación.

No se pueden repasar todos los métodos aquí definidos, pero en cualquiera de los casos se considera que el nombre de cada uno de ellos ya es lo suficientemente autoexplicativo. Sí cabe destacar que el resultado de muchas de estas operaciones es una nueva imagen, es decir, no se modifica la imagen original. Aunque en ciertas

ocasiones se desea modificar una determinada captura, normalmente lo que se propone en este trabajo son algoritmos cuya entrada viene dada por imágenes que se deben seguir representado en la aplicación, y en cada paso se genera una imagen derivada que muestra el terreno de una forma diferente.

También cabe señalar que, como bien sabemos, en este proyecto se integran las librerías Qt y OpenCV, ambas con capacidad de procesar imágenes, por lo que cada una dispone de su propio esquema para abstraer dichas entidades. De forma básica, una imagen tiene al menos su representación en OpenCV (útil para operaciones de algoritmos), y si esta fuera una imagen que se debe mostrar en la aplicación, se puede pedir que se inicialice su representación en Qt, QPixmap.

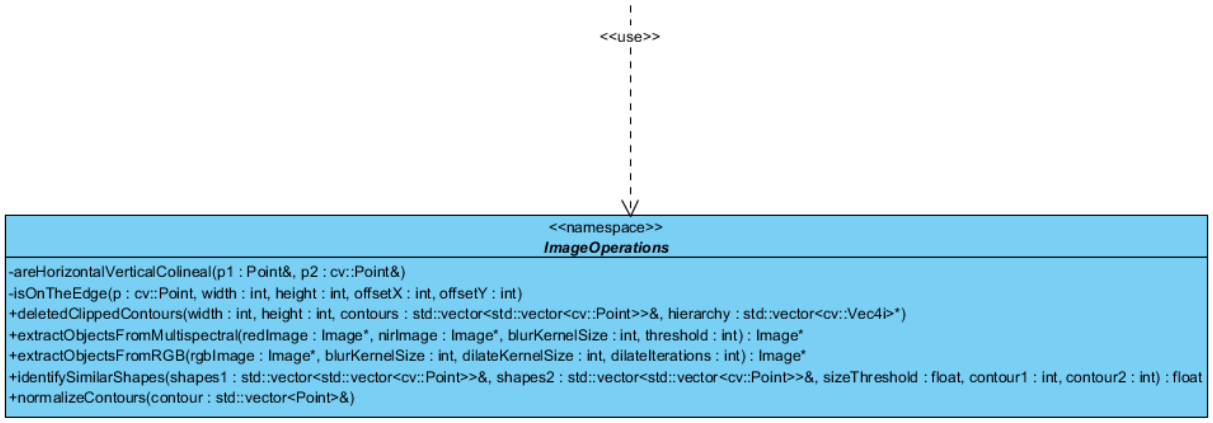


Ilustración 148. Espacio de nombres que define operaciones con imágenes.

Como soporte a la clase anterior, se crea un espacio dirigido únicamente a operaciones que involucran varias imágenes o procesos complejos, como puede ser la extracción de contornos de una imagen RGB. Mientras que una imagen se considera que debe definir operaciones atómicas, este espacio define operaciones de mayor entidad haciendo uso de esas herramientas que le proporciona una imagen.

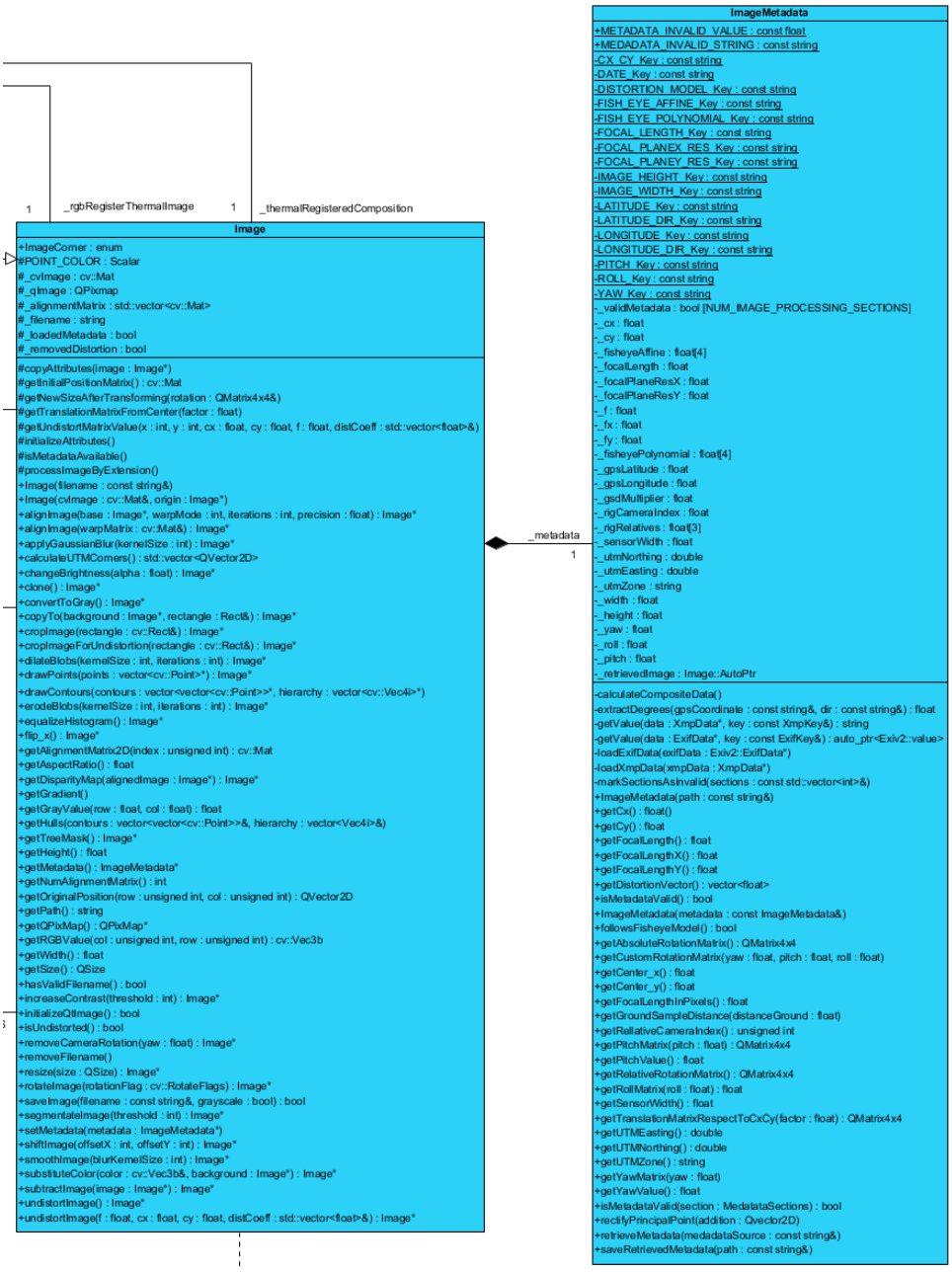


Ilustración 149. Diagrama de clases para la imagen y sus metadatos asociados.

Esta última clase aquí presentada, *ImageMetadata*, no tiene complejidad alguna a nivel de implementación, pero es una clase ardua de definir debido a que se debe acceder a cada metadato por su etiqueta de forma manual, y si bien parte del proceso se puede hacer genérico, cada una de las etiquetas presenta su propio formato y se guarda en una variable diferente, ya sea una cadena, un vector o un número real.

Una parte de las fórmulas expuestas en este documento, como el cálculo del GSD o las matrices de rotación, se aplican aquí dado que muchos de los datos de

entrada necesarios se extraen de los metadatos de la imagen. Es decir, esta entidad guarda datos directamente extraídos de la imagen pero también derivados.

Por último, cabe destacar que todas las etiquetas de acceso que se han ido exponiendo a lo largo de este documento, independientes de la librería, se definen como constantes en la clase.

2.2.12.3 - Módulo de visualización 3D

El último módulo de esta aplicación va dirigido a la visualización 3D de resultados tales como los contornos extraídos de las imágenes multiespectrales. Si bien su funcionalidad en este trabajo es bastante reducida, el esquema que nos permite representar una escena 3D es relativamente complejo. Asimismo, este módulo podría extenderse mucho más con la explicación de los diferentes modelos de iluminación, texturas, etc, pero dado que no es el objetivo de este proyecto, nos limitaremos a comentar qué hace cada entidad a un nivel de abstracción alto.

De forma general, el diagrama de este módulo quedaría como se muestra en la Ilustración 150.

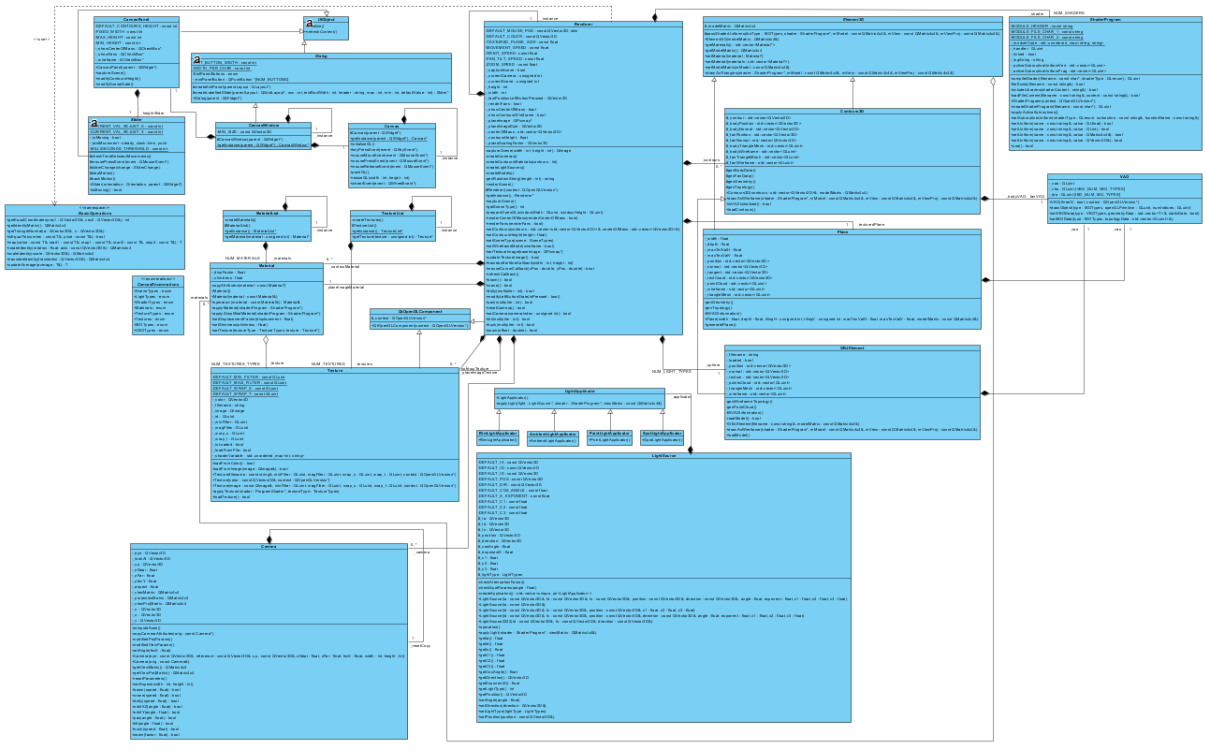


Ilustración 150. Diagrama de clases del módulo de visualización 3D.

Podemos dividir este esquema a su vez en varios subconjuntos (desde la Ilustración 151 hasta la Ilustración 154).

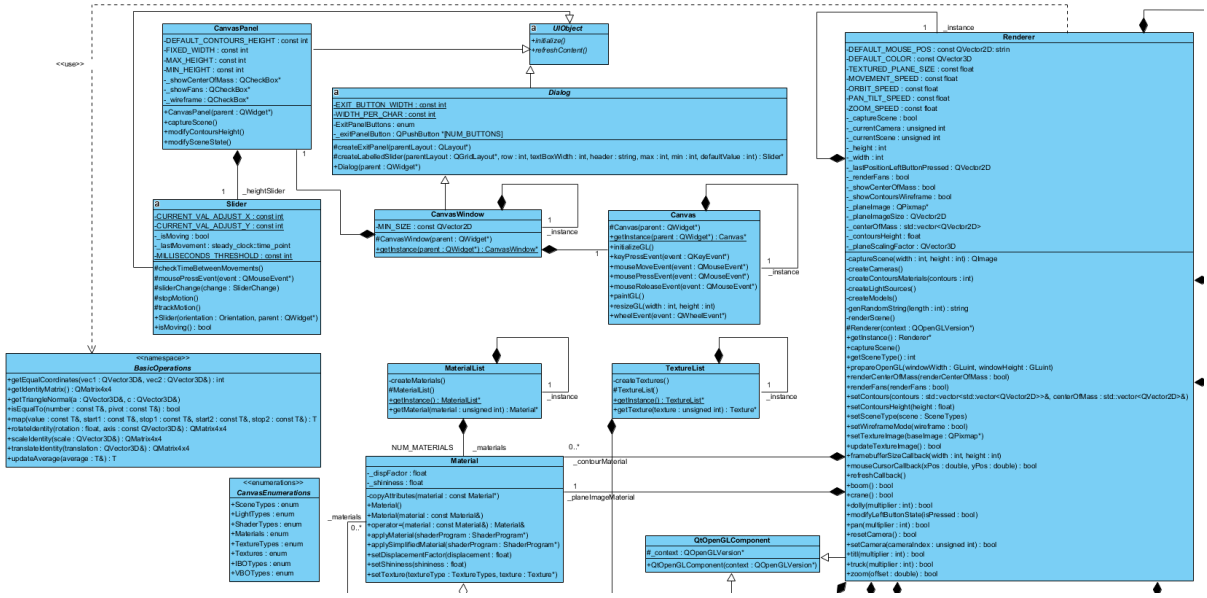


Ilustración 151. Diagrama de clases parcial del módulo de visualización 3D (I).

- **Renderer**: clase principal de este módulo, que hace las veces de controlador y modelo, simplificándose así el esquema MVC, dado que ninguna otra clase debe acceder a dichos elementos (sólo debe haber una entidad encargada de dibujar). Por lo tanto, debe contener todos aquellos objetos necesarios para el renderizado de la escena: cámaras, luces, contornos en formato 3D, plano, material de la imagen, modelos para los centros de masas, etc.

  Dado que esta entidad también responde como controlador, debe recibir los eventos de usuario de la aplicación, que generalmente se basan en eventos de teclado y ratón para controlar la cámara, o bien se corresponden a controles de paneles que permiten modificar la propia escena.

- **CanvasWindow**: dado que este módulo parte de una funcionalidad reducida, los resultados se presentan en una ventana secundaria. Esta entidad sólo sirve para dar un entorno a aquellos elementos de la interfaz realmente importantes.

- **Canvas**: representa el *widget* donde se puede visualizar la escena y donde el usuario puede interactuar con la misma a través de eventos de ratón y teclado. Por lo tanto, la funcionalidad de esta clase se reduce a recibir dichos eventos y comunicarlos a la entidad correspondiente (*Renderer*). En este

caso, los eventos no se redirigen directamente al *Renderer*, sino que el procesamiento de los mismos se hace aquí, y en función de esto se llama a un método u otro del controlador. El objetivo que se persigue con esto último es desacoplar al *Renderer*, teniendo en cuenta que los eventos son específicos de Qt, mientras que el controlador sólo conoce órdenes de OpenGL. Entre todos los eventos que se reciben debe estar necesariamente el de redibujado de la escena, que se genera cuando deben producirse cambios en la misma.

- **CanvasPanel**: panel anexo al *canvas* que permite controlar los parámetros de la escena a través de múltiples controles tales como un *slider*. De nuevo, los eventos que aquí se produzcan se deben comunicar al controlador (ahora sí de forma directa).
- **BasicOperations**: espacio que alberga operaciones básicas para este módulo tales como la obtención de matrices identidad de Qt con ciertas transformaciones geométricas ya aplicadas (escalado, rotación, traslación), normal de un triángulo, *mapeo* de un intervalo a otro, etc.
- **CanvasEnumerations**: conjunto de enumeraciones necesarias para este módulo, de tal forma que se delimitan claramente los tipos de *IBO* / *VBO* que se pueden crear (a qué topología y qué tipo de geometría se asocian), tipos de luces en la aplicación, identificadores de *shaders*, etc.
- **QtOpenGLComponent**: clase base que representa a cualquier entidad de este módulo que necesite un contexto de OpenGL para operar, como bien podría ser el controlador, un *VAO* o un *ShaderProgram* (utilizan operadores de OpenGL: definir un estado, una textura, etc).
- **MaterialList**: almacén de materiales por defecto, de tal forma que dados los índices de materiales definidos en *CanvasEnumerations* se podría acceder a la referencia de cualquiera de ellos.
- **Material**: define la apariencia de cualquier elemento representable en la escena. A pesar de que su papel es importante, un material no hará más que aunar texturas de diferentes tipos para componer la apariencia final del objeto. Por lo tanto, buena parte de la carga de implementación recae sobre

la textura y no aquí, donde lo único que se hace es aplicar una textura u otra en función de las órdenes recibidas.

- **TextureList**: ídem que *MaterialList*, con la que diferencia de que se accede a las texturas aquí almacenadas para asignarlas a un material.

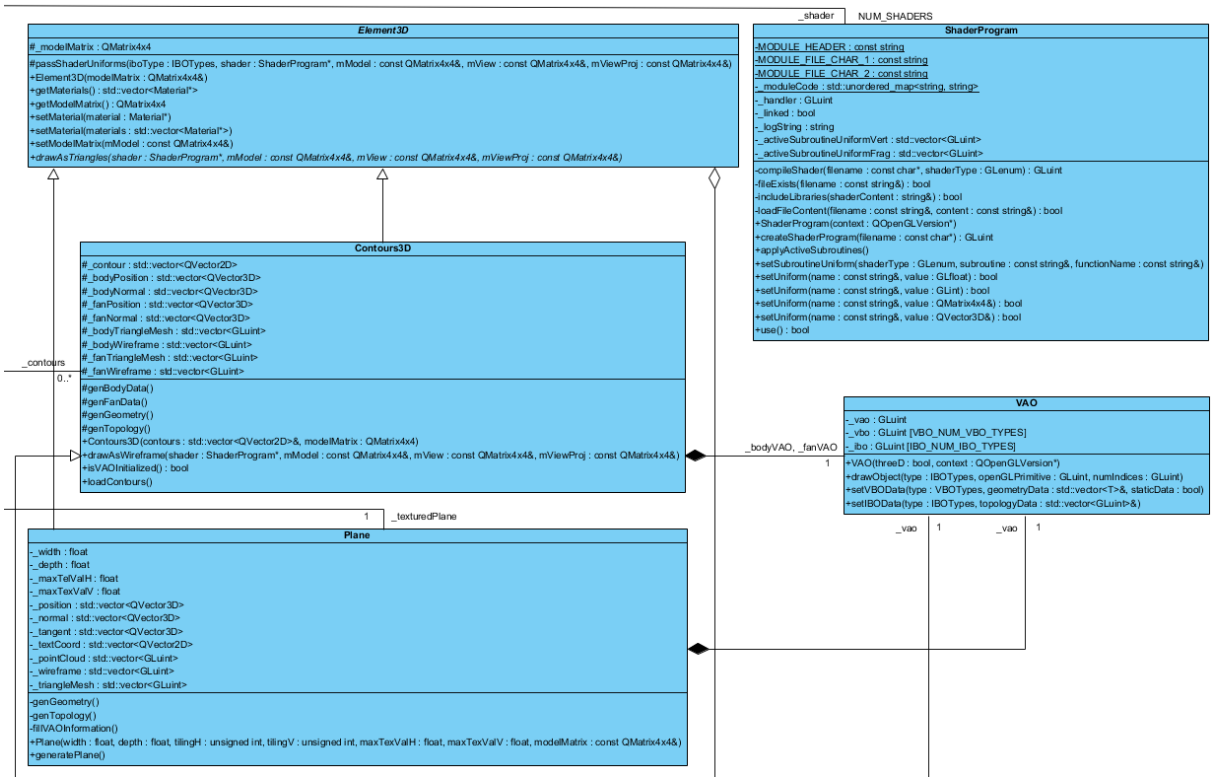


Ilustración 152. Diagrama de clases parcial del módulo de visualización 3D (II).

- **Element3D**: clase abstracta que incluye todas aquellas operaciones básicas para cualquier objeto que podamos dibujar en nuestra escena, además de definir una serie de métodos que se deben implementar en aquellas clases que hereden de esta misma. También almacena un conjunto de materiales que los objetos podrán aplicar al dibujarse. Se define un conjunto y no sólo uno para aquellos objetos que puedan contener más de un componente.
- **Contours3D**: representa un contorno con altura, tanto en modo malla de alambre como de triángulos. Se encarga de triangular los polígonos para generar toda la geometría y topología a través del método *loadContours*.

- **Plane**: representa un plano centrado en el origen con tantas divisiones como sean necesarias. Nosotros lo empleamos para representar la imagen donde se sitúan los contornos, por lo que con cuatro puntos sería suficiente.
- **ShaderProgram**: su labor es cargar y compilar un *shader* de GLSL, que define el modelo de iluminación y posicionamiento de nuestra escena tras aplicarle las transformaciones geométricas oportunas. Si bien esta clase incluye la posibilidad de incluir un *Geometry Shader*, nosotros sólo utilizaremos *Vertex* y *Fragment Shader*.
- **VAO**: define la geometría (posiciones, normales y coordenadas de textura) y topología (malla de alambre y malla de triángulos) de un componente en la GPU a través de VBOs e IBOs, respectivamente.

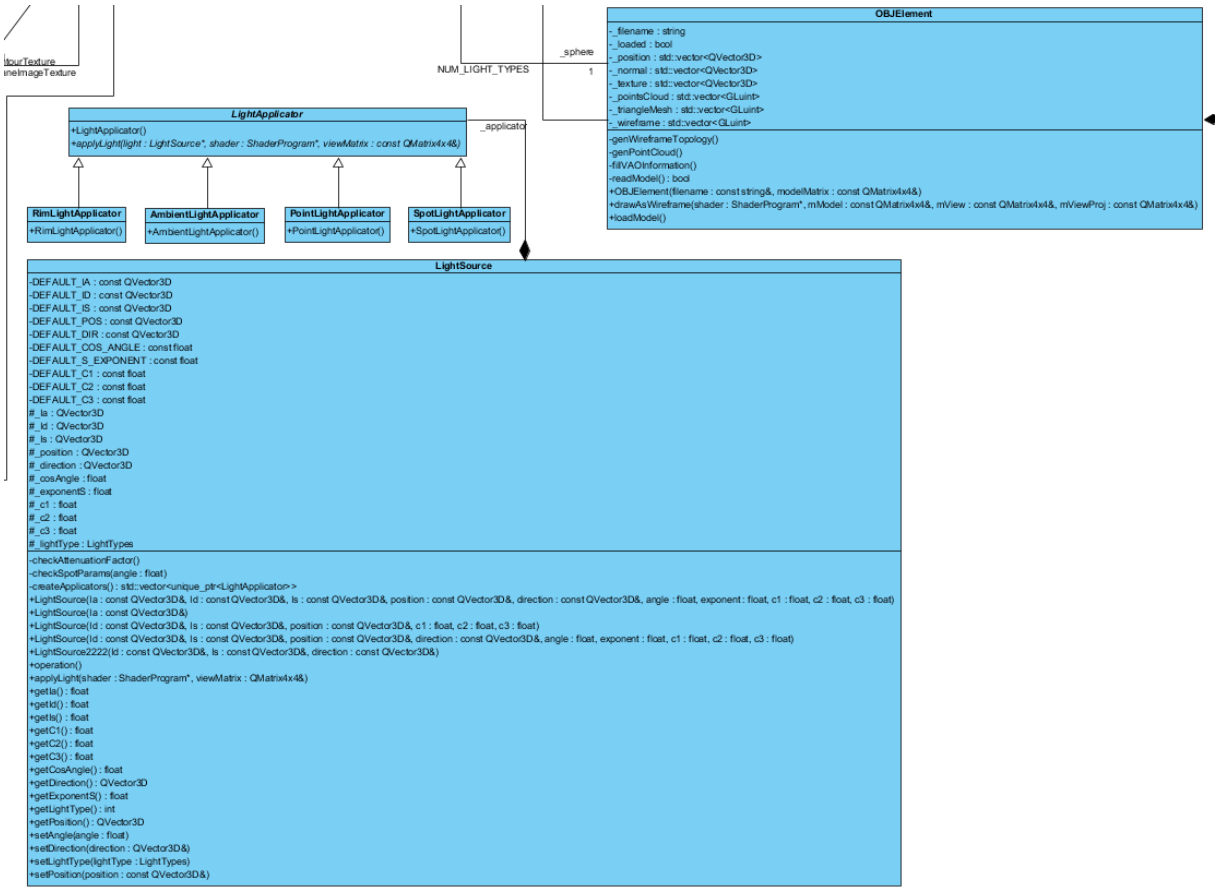


Ilustración 153. Diagrama de clases parcial del módulo de visualización 3D (III).

- **OBJElement**: representa un objeto leído desde un fichero de extensión .obj, de tal forma que buena parte de la implementación se basa en la lectura del archivo. Como cualquiera de los objetos anteriores, define su geometría y

topología cuando así se indique. Tanto el plano como esta entidad se representan comúnmente como una malla de triángulos, por lo que es el contorno el único que defina su malla de alambre.

- **LightSource**: representa un modelo de luz genérico, incluidas las propiedades que cualquier tipo de luz podría contener. Su especialización se producen a través de un aplicador, adaptándose así al patrón Estrategia que se describirá en profundidad al finalizar la explicación de todos los diagramas de clases. Se definen así múltiples constructores que podrían adaptarse a cada modelo, o bien uno genérico (aunque posteriormente pueda modificarse cada atributo). Los aplicadores no son específicos de una luz, sino que simplemente definen un comportamiento, y por lo tanto, pueden compartirse entre instancias (se especifican como objetos estáticos). Como sucediera con un material, una luz debe también aplicarse a la escena para que un determinado aplicador defina los parámetros necesarios de acuerdo a un modelo de iluminación específico.
- **LightApplicator**: interfaz que define el método de aplicación para cualquier modelo de iluminación. Se mencionan a continuación cuáles son los modelos implementados, sin entrar en más detalle de cómo funcionan: *PointLightApplicator* (luz puntual), *DirectionalLightApplicator* (luz direccional), *SpotLightApplicator* (luz similar a la de un foco) y *RimLightApplicator* (parte de la base de la *SpotLightApplicator* para resaltar las figuras, de tal forma que no se tiene en cuenta el color del modelo sino que se aplica siempre un color estático, por ejemplo, similar al azul).

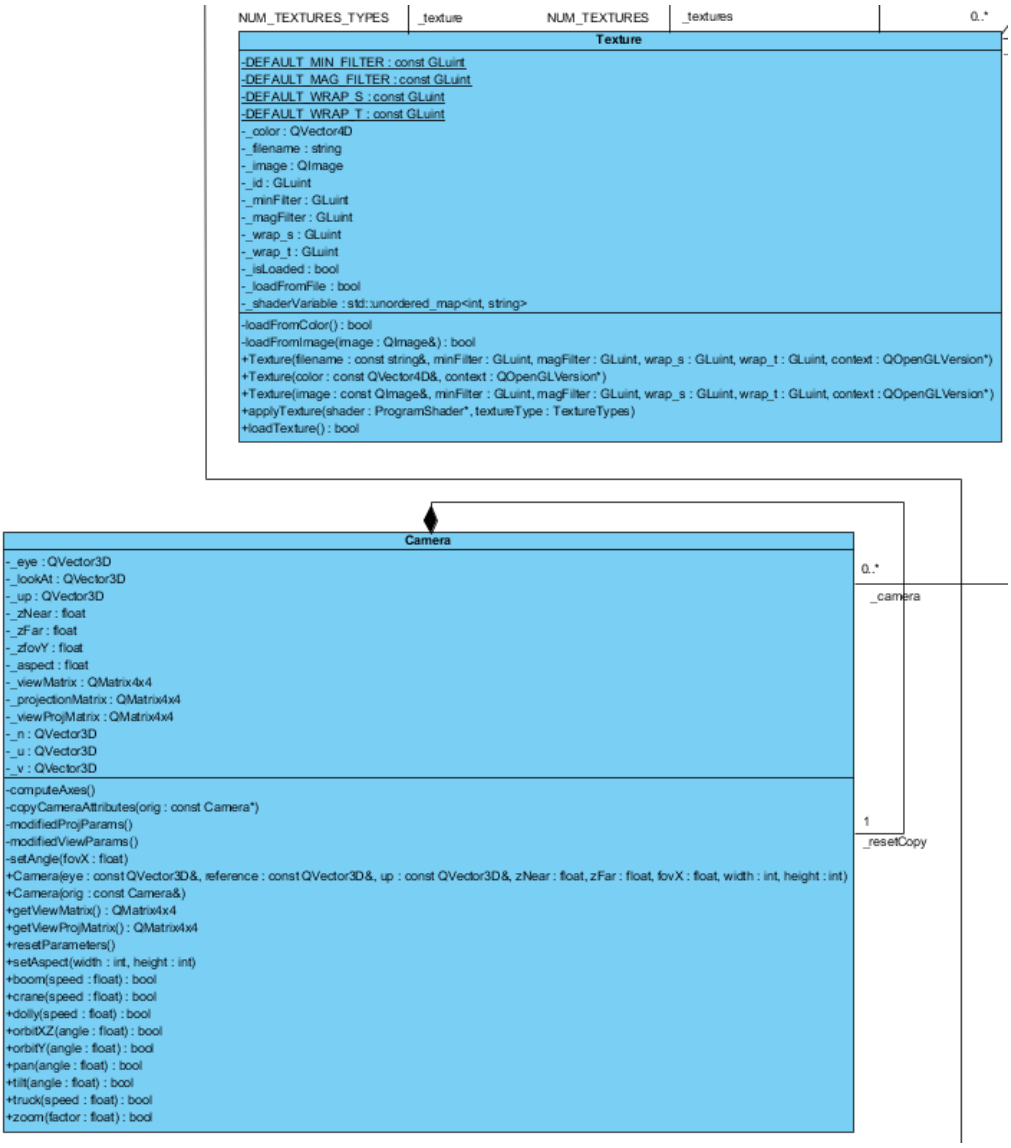


Ilustración 154. Diagrama de clases parcial del módulo de visualización 3D (IV).

- **Texture**: define una distribución de color a lo largo de un espacio 2D en el intervalo [0, 1], donde los patrones a seguir fuera de estos límites se pueden regular. Carga una textura desde archivo, o la genera por sí mismo si fuera un único color, de tal forma que crea un *array* de intensidades que se corresponden a los canales de un color RGB. Se debe definir como un *QtOpenGLComponent* debido a que debe registrar dicha textura en el contexto, obteniendo así un identificador, que será el que se aplique cuando se pida.
- **Camera**: cámara en perspectiva que define todos los movimientos que el usuario puede realizar sobre la escena 3D para modificar el punto de vista

de la misma. Si bien la implementación de dichos movimientos se realiza aquí, los parámetros de velocidad que hacen más cómoda la visualización se definen en el controlador, garantizando una mayor flexibilidad. Debe permitir el acceso a las matrices que modelan la cámara para calcular más tarde la posición final de los objetos en la escena.

Una vez comentadas todas las clases que se han expuesto en estos diagramas, podemos mostrar qué patrones se han considerado a la de hora de desarrollar el software resultante de este trabajo.

2.2.13 - Patrones de diseño

Una parte muy importante de la construcción de software es la utilización de patrones de diseño, que se pueden definir como soluciones software reutilizables que se proponen para resolver problemas comunes y cuyo principal objetivo debe ser aumentar la calidad del sistema. Para comprender los patrones de diseño aquí empleados, se dividirá este apartado en tantas secciones como soluciones se hayan utilizado:

2.2.13.1 - Singleton

La consigna de este patrón es bien clara: garantizar que sólo existe una instancia de un tipo de objeto en un determinado momento, permitiendo asimismo el acceso a global a dicha instancia. Su implementación se puede ver en la Ilustración 155.

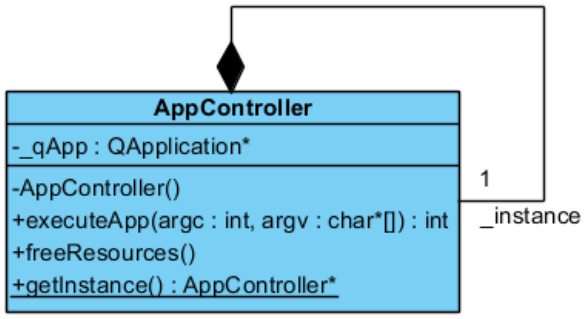


Ilustración 155. Esquema de una clase que aplica el patrón Singleton.

Se garantiza que sólo existe una instancia ocultando el constructor, de tal forma que la propia entidad contiene un atributo estático, al que podemos referirnos como

instance, que por defecto será nulo. Un método estático, *getInstance*, será la vía de acceso a la instancia única, de tal forma que si no hubiera una (*instance* con valor nulo) podría crearse, siempre de forma controlada por la propia clase, y si ya existiera simplemente se devolvería la propia instancia.

Algunos de los beneficios que se obtienen al usar este patrón, además de los dos comentados al inicio de esta sección, son los siguientes (Eric Freeman, 2009); (Vlissides, 1994):

- Control de la instancia: se puede establecer un estricto control sobre dicha instancia en términos de cómo y cuándo.
- Representa una mejora respecto de una variable global asociada a un espacio de nombres, de tal forma que ahora tenemos una instancia asociada a una clase, accesible a través de una forma a determinar.
- Es fácil generar una herencia de una clase que implemente el patrón Singleton, permitiendo extender la funcionalidad de la entidad padre.
- Inicialización *lazy*: la instancia no se crea hasta que se pide por primera vez, evitando inicializar algo que es posible que no se use.
- Se presta a modificar el patrón para extender el número de instancias a n, sea n un número diferente de uno, que es la cantidad que hemos venido definiendo.

2.2.13.2 - Modelo-Vista-Controlador

Dado que una iteración del desarrollo de este trabajo consistía en la aplicación de este patrón, nos remitimos a dicho apartado para ver en profundidad en qué consiste este esquema y cómo se ha adaptado a nuestra aplicación. Sólo se hará un breve resumen aquí para mantener la estructura de este anexo al desarrollo.

El patrón MVC es un patrón para la creación de interfaces de usuario que distingue claramente tres tipos de objetos (Vlissides, 1994):

- **Modelo**: representa a los objetos de la aplicación y hará las veces de base de datos, permitiendo a cualquier instancia el acceso a la información disponible siempre que posea los suficientes privilegios. Por simplicidad, se pueden integrar también aquí aquellos objetos que se almacenan en la base

de datos y que sirven de soporte para estos mismos (que incluye buena parte de la aplicación).

- **Controlador**: responde a los eventos del usuario y consulta y actualiza el modelo. De la misma forma que recibe estos eventos, el controlador puede comunicarse con la interfaz. No es más que un intermediario entre la vista y el modelo.
- **Vista**: parte de la aplicación con la que interactúa el usuario. Debe mostrar información acorde al estado de la aplicación, que se refleja directamente en el modelo.

La comunicación entre estas entidades se puede adaptar a cada aplicación, y de hecho, podemos encontrar varios esquemas. Nuestro modelo de comunicación queda propuesto en la Sexta iteración.

Tres claros ejemplos de Modelo, Vista y Controlador respectivamente son los que se representan en la Ilustración 156.

Ilustración 156. Ejemplos de entidades asociadas a los objetos del esquema MVC.

2.2.13.3 - Adapter

El patrón Adapter permite que clases cuyas interfaces son incompatibles puedan trabajar juntas. De esta forma, se transforma una interfaz en otra. Se puede extrapolar esta situación fácilmente a un adaptador de corriente. En cualquiera de los casos, este patrón traduce una entrada inviable para cierta parte de la aplicación en un objeto que ya sí se puede procesar.

La mejor manera de exponer esta problemática es explicar lo que en esta aplicación sucede:

1. Disponemos de múltiples librerías, y algunas de ellas están ligadas al procesamiento de información espacial (imagen, 3D, etc). Sin embargo, cada una tiene su propia representación, por ejemplo, de un punto 2D.
2. La visualización 3D y toda la vista asociada está muy ligada a Qt, por lo que utilizan las clases que este *framework* les proporciona. Sabemos también que este módulo tenía como principal funcionalidad mostrar los contornos extraídos desde otra perspectiva.
3. Los contornos se extraen desde OpenCV, que devuelve un vector de sus propios puntos 2D, no procesables, por ejemplo, por un VAO de nuestra aplicación.

Veáse como el patrón Adapter encaja perfectamente en esta problemática, de tal forma que no habrá que modificar nuestro VAO, sino implementar una traducción de un tipo de dato a otro (ver Ilustración 157).

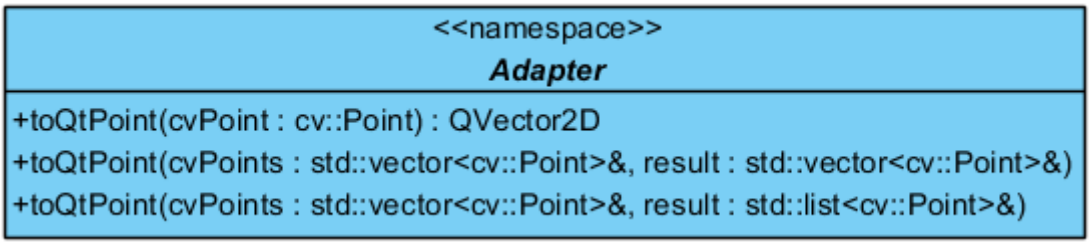


Ilustración 157. Ejemplo de espacio de nombres que hace las veces de traductor.

2.2.13.4 - Strategy

Una forma de extender la funcionalidad de una clase es mediante la herencia, como ya bien sabemos. Sin embargo, esto presenta varias desventajas:

- La solución es poco flexible: se herede o se sobreescriba una solución, esta no puede variar en tiempo de ejecución.
- Todas las instancias de una clase deben mantener el mismo comportamiento. Si se desea que puedan tener diferentes algoritmos, se deben implementar subclases, una por comportamiento.

Este patrón propone el uso de una familia de algoritmos, de tal forma que un contexto podrá modificar su comportamiento seleccionando un algoritmo diferente,

que se representa en nuestro software como una subclase de una base que engloba a la familia completa. Cada uno de esos algoritmos se conoce como *strategy* (Vlissides, 1994).

Esto nos será de utilidad para definir nuestros modelos de iluminación: teníamos hasta cuatro tipos, por lo que definir cuatro subclases parece un trabajo bastante arduo. Se puede simplificar el problema definiendo una única clase, *LightSource*, que hará referencia a una familia de algoritmos que en nuestro caso definen cómo se aplica la luz en función del tipo. De esta forma, pasar de una luz puntual a una luz direccional se reduce a cambiar un puntero. En nuestra aplicación, la principal ventaja de este patrón se puede observar en que disminuye la complejidad de la implementación, pero podemos imaginarnos una aplicación como Unity3D, donde se representa una luz y el usuario puede cambiar su tipo. Con herencia tendríamos que hacer algún proceso de traducción, mientras que con este patrón el cambio sólo produce una modificación del valor de un puntero.

Se puede ver un ejemplo de implementación de este patrón en el esquema de la Ilustración 158.

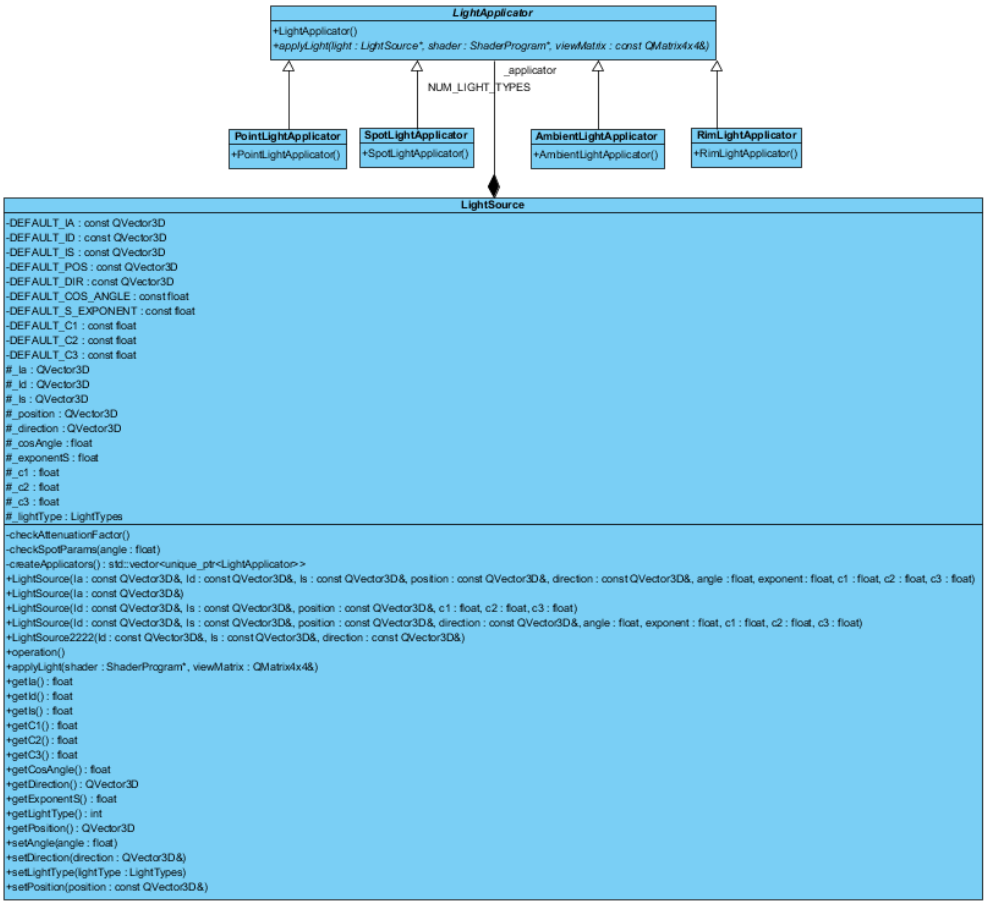


Ilustración 158. Ejemplo de implementación del patrón Strategy.

En esta situación, un cambio de tipo de luz implica no ya un cambio de puntero sino de un entero que se corresponde a un índice de una enumeración. Lo que se ha hecho aquí es una pequeña mejora: suponiendo que tenemos un conjunto de luces considerables, no se crea una estrategia para cada luz, sino que se crea un *pool* de estas, de tal forma que varias luces podrían apuntar a la misma estrategia, teniendo en cuenta que los aplicadores no almacenan información de una fuente concreta, sino que conocen la fuente (y sus datos) en el momento de aplicación (se pasa como parámetro).

2.2.13.5 - Observer

Se menciona este patrón aquí porque aparece en la aplicación, aunque su implementación, de forma general, no nos corresponde a nosotros sino a Qt mediante su variante de *signals / slots* (nosotros somos meros usuarios). En nuestro caso nos limitamos a especificar qué señales de un control (botón, canvas, etc) son recibidas por otras instancias, de las que además se debe especificar el método destinatario, conocido como *slot* (se debe declarar como tal).

De forma general, el patrón observador define una dependencia de uno a muchos, de tal forma que cuando un objeto cambia su estado, todas las dependencias son notificadas (Vlissides, 1994). En nuestro caso, por ejemplo, cuando un botón se pulsa se avisa al controlador correspondiente.

El mecanismo específico de *signals / slots* presenta una serie de ventajas (Hong, 2018):

- Se puede utilizar para notificar eventos de Entrada / Salida asíncronos.
- Es apropiado para notificar eventos de *timeout*.
- Toda la infraestructura (registro, invocación de métodos, desregistro, etc) lo aporta Qt mediante su *Meta Object Compiler* (MOC).

3 - EXPERIMENTACIÓN

En este apartado se desarrollan las pruebas oportunas para mostrar las configuraciones que mejor resultado obtienen en cada uno de los algoritmos que se han planteado a lo largo de este documento.

Si bien nuestra aplicación no se presta a comprobar los resultados como correctos o no, verdadero o falso, sí que se pueden realizar operaciones entre imágenes para determinar cómo de similares son estas. Incluso así, habrá ciertas funcionalidades que difícilmente se pueden evaluar de tal manera, como podría ser la extracción de contornos, dado que se carece del conocimiento de que está bien o mal, de tal forma que incluso para el ojo humano es una tarea ardua.

Se procede por lo tanto a experimentar con aquellos métodos con los que se haya probado que se obtienen resultados cercanos a lo deseado, dado que no tendría sentido analizar algoritmos que no suponen una vía de trabajo futuro y que se han implementado con fines puramente de investigación.

3.1 - Conceptos previos

Algunos conceptos que se utilizarán durante esta fase de experiementación son los siguientes:

3.1.1 - Coeficiente de correlación

Un coeficiente de correlación es un índice que mide el grado de relación entre distintas variables relacionadas linealmente. Nótese como se hace hincapié en una relación lineal, lo que implica que entre dos variables podría no existir una relación del tipo lineal pero sí de cualquier otro tipo (a estudiar mediante un algoritmo alternativo). El valor del coeficiente se encuentra en el intervalo [-1, 1], donde el signo será el sentido de la relación.

- Si $r = 1$, existe una correlación positiva perfecta. El índice indica una dependencia total entre las dos variables, denominada relación directa; cuando una de ellas aumenta, la otra también lo hace en proporción constante.
- Si $0 < r < 1$, existe una correlación positiva.
- Si $r = 0$, no existe relación lineal. Pero esto no necesariamente implica que las variables son independientes: pueden existir todavía relaciones no lineales entre las dos variables.
- Si $-1 < r < 0$, existe una correlación negativa.

- Si $r = -1$, existe una correlación negativa perfecta. El índice indica una dependencia total entre las dos variables llamada relación inversa: cuando una de ellas aumenta, la otra disminuye en proporción constante.

Unos de los coeficientes de correlación más conocidos es el coeficiente de Pearson, que mide el grado de covarianza entre dos variables.

3.2 - Extracción de contornos

Como se comentaba con anterioridad, la dificultad de experimentar con esta funcionalidad radica en reconocer qué es correcto y qué no. No se dispone de las soluciones a la segmentación de la imagen, por lo que se desconoce cómo de bueno es un resultado.

También hay que tener en cuenta que el proceso no se ha presentado como automático, sino que depende de ciertos factores de los cuales el que más dependencia puede tener de un operador humano es el umbral de intensidad.

Por otro lado, también hay que tener en cuenta **qué se considera como correcto a la hora de segmentar la vegetación**:

- Se identifica cada uno de los árboles como un individuo aislado del resto. Se debe tener en cuenta que el algoritmo no tiene acceso al tronco de un árbol ni a sus ramificaciones, por lo que un olivo podría estar compuesto por varias áreas de vegetación, que el algoritmo distingue como árboles diferentes. Asimismo, el ángulo de captura de un área del terreno puede hacer que árboles distintos a pie de campo se perciban como conectados a ojos del proceso aquí definido (ver Ilustración 159). Por lo tanto, estos dos últimos casos no se consideran como errores sino más bien limitaciones del algoritmo con las habrá que lidiar. Algunos puntos a tener en cuenta son los siguientes:
    - En el caso concreto del olivo, se puede percibir como si fuera más de un árbol porque podría tener múltiples pies y copas. Por lo tanto, puede ser interesante capturar cada área de vegetación de ese olivo por separado.
    - Aquellos árboles que se perciben conectados debido al ángulo de visión, incluso para el ojo humano, normalmente se encuentran en

los extremos de la imagen, en cuyo caso podría ser aceptable que no se tuvieran en cuenta.

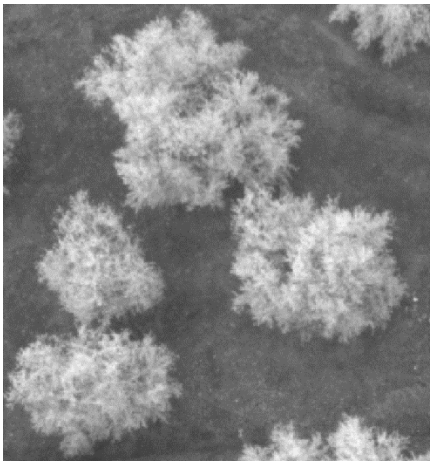


Ilustración 159. Árboles representados en la imagen como si fueran una única masa de vegetación.

- Al hilo de qué es correcto, aquellos árboles seccionados tampoco deben aparecer en la imagen dado que no se dispone de información del contorno al completo. No obstante, esta funcionalidad no es una opción y siempre se debe dar. Será un error sólo en aquellos casos en los que se perciba, incorrectamente, árboles conectados, donde uno de ellos se encuentra justamente en los extremos de la imagen, derivando en su eliminación del resultado final.
- Se deben desechar aquellas áreas de vegetación no útiles para este trabajo, por lo que los contornos extraídos se deben corresponder a árboles presentes en la escena, y no a cualquier otro tipo de vegetación representada.

Dado que el tiempo de cómputo no es un problema en este algoritmo, nos limitaremos a extraer la configuración óptima para un conjunto de imágenes. Hay que tener en cuenta que la metodología que en este apartado se propone tiene poco rigor científico, dado que se apoya exclusivamente en el ojo humano y en aquello que se ha considerado que era correcto en cada captura.

Por lo tanto, se procede a realizar las pruebas pertinentes sobre un conjunto de imágenes de hasta tres *datasets* distintos, donde las imágenes se han seleccionado al azar. Se exploran 20 pares de imágenes RED y NIR, donde de cada conjunto se

experimenta con un número acorde a su tamaño. Se elimina la distorsión y se registran las imágenes multiespectrales originales para partir de condiciones óptimas. En cualquiera de los casos, estas dos capas están muy poco desplazadas, teniendo en cuenta que la extracción de contornos se desarrolló antes de la adecuación de las imágenes multiespectrales.

Por último, lo que se pretende extraer de esta experimentación es que no existen unos parámetros constantes que sirvan para todas las imágenes multiespectrales, sino que se deben adaptar a cada caso. Se procede por lo tanto a hallar la configuración óptima para cada par de imágenes multiespectrales (ver Tabla 15).

Imagen multiespectral	Tamaño de filtro gaussiano	Umbral de área	Umbral de intensidad
Primer conjunto de datos (1-8). ID: 1			
IMG_8	3	1933	40
IMG_5	15	1896	58
Segundo conjunto de datos (35-58). ID: 2			
IMG_41	5	5390	63
IMG_35	6	4015	62
IMG_55	13	595	37
IMG_56	6	3347	37
IMG_50	10	1227	72
Tercer conjunto de datos (0-150). ID: 3			
IMG_64	18	892	231
IMG_79	9	3941	11
IMG_137	7	2045	9
IMG_51	8	3160	2
IMG_66	15	2007	7
IMG_30	16	669	32
IMG_102	5	743	38

IMG_46	21	1524	15
IMG_69	12	2045	3
IMG_19	14	2268	9
IMG_36	16	2045	1
IMG_135	13	2193	15
IMG_90	14	3271	27
Media	11.3	2255.3	358.45

Tabla 15. Configuraciones óptimas halladas para la extracción de contornos en imágenes multiespectrales.

Nótese además como hay variables que producen resultados deseables no sólo para los valores aquí propuestos, sino para un rango, que por complejidad no se ha incluido en esta experimentación. Dada la gran diversidad de valores aquí expuestos, queda claro que no existe una configuración única para todas las imágenes multiespectrales, por lo que tanto en las conclusiones como en el trabajo futuro se debe tratar en profundidad este tema.

3.3 - Registro de imágenes multiespectrales

A diferencia del algoritmo anterior, ahora sí disponemos del conocimiento de qué es correcto y qué no, aunque no de forma completa. Dicho conocimiento se basa en una imagen, la multiespectral de referencia, GRE, que es con la que deben alinearse el resto de capas. Se dice además que no se dispone del conocimiento completo porque, incluso exactamente alineadas, existe cierta disparidad entre imágenes a consecuencia de la diferencia de valores de intensidad.

Para estudiar el grado de similitud entre dos imágenes podemos utilizar un coeficiente de correlación, que como se comentó en el apartado de Conceptos previos, será un valor en $[-1,1]$ ($[0,1]$ en OpenCV: se comprueba con una imagen blanca y otra negra) que indica la relación entre dos variables cualesquiera, que en este apartado serán dos imágenes en escala de grises, $f(x,y)$ y $f'(x,y)$.

Para calcular dicho coeficiente nos ayudaremos de OpenCV, que ya implementa esta herramienta dado que es de interés en el ámbito de *image matching*. Nótese como una imagen de entrada, *source*, se puede desplazar sobre una imagen

plantilla, *template*, de tal forma que se obtiene la posición donde *source* mejor encaja dentro de *template* a partir del coeficiente de correlación para cada iteración realizada.

Si bien, nosotros no necesitamos un proceso iterativo para conseguir la mejor posición, sino un simple análisis de correlación para dos imágenes de entrada que no se van a modificar.

Por otro lado, existen múltiples variables a valorar en este proceso para obtener una configuración óptima:

- **Nivel de precisión**: indica cuál es el ajuste deseado entre dos imágenes. Normalmente son valores muy cercanos al cero, pero cuanto más pequeño sea más difícil es que converja a ese nivel (indica un mayor nivel de precisión).
- **Número de iteraciones**. Tiene una relación directa con el nivel de precisión, dado que mayor precisión implica que se necesita más tiempo, y por lo tanto, más iteraciones. En cualquiera de los casos, aunque sabemos que ambos parámetros deben crecer en el mismo sentido, se debe encontrar una relación adecuada.
- **Tamaño de las imágenes**: conforme el tamaño de las imágenes decrece, el proceso es menos costoso, pero también hay que valorar los dos atributos anteriores para lograr un buen resultado.

Se debe considerar además que la correlación no es el único parámetro que mide cómo de bueno es un resultado, sino también el tiempo de cómputo. Una parte de la interfaz (diálogo de barra de progreso) y otras utilidades, como hilos para ejecutar procesos pesados, se orientan en esta dirección. Si bien se dispone de herramientas para informar al usuario del progreso de una operación, esta no debe ocupar un tiempo excesivo. Se puede garantizar así que se escoge una configuración óptima mediante alguna función que relacione la correlación y el tiempo de cómputo resultante.

A diferencia de antes, donde disponíamos de muchas imágenes donde se podía probar el proceso de extracción de contornos, ahora será algo más difícil seleccionar imágenes aleatorias, dado que en pocos casos se dispone de las cuatro multiespectrales. Se establece así un conjunto de diez imágenes para cada una de las de pruebas que se efectuan a continuación.

Para no añadir más tiempo de ejecución al proceso, el análisis de correlación se hace después del registro, de tal forma que se deben guardar los resultados en

fichero, y será mediante el módulo de experimentación con el que se consigan los coeficientes. Asimismo, el tiempo de ejecución que se muestra en cada prueba no debe incluir estas operaciones de guardado, que se presuponen que son costosas e influirían negativamente en los resultados, sin que esta sea una operación necesaria para el algoritmo.

Dado que el coficiente de correlación entre las imágenes no alineadas (sin distorsión) no parte ni mucho menos de 0, sino que ya tiene una base alrededor de 0.9, lo que se hace en la Tabla 16 es mostrar los valores que devuelve el algoritmo para el estado inicial de las capturas. Por supuesto, aquí no nos interesa el tiempo de cómputo.

Imagen multiespectral	Corr. GRE-RED	Corr. GRE-NIR	Corr. GRE-REG
Primer conjunto de datos (1-8). ID: 1			
IMG_1	0.8954	0.8777	0.9306
IMG_4	0.9777	0.9381	0.9618
IMG_6	0.9756	0.9520	0.9727
Segundo conjunto de datos (35-58). ID: 2			
IMG_48	0.9265	0.8918	0.9316
IMG_58	0.9146	0.8878	0.9242
IMG_53	0.9777	0.9381	0.9618
IMG_37	0.9286	0.9100	0.9371
Tercer conjunto de datos (0-150). ID: 3			
IMG_35	0.9578	0.9453	0.9739
IMG_107	0.9602	0.9537	0.9686
IMG_136	0.9755	0.9612	0.9692
IMG_78	0.9707	0.9504	0.9672
IMG_142	0.9728	0.9375	0.9656
IMG_144	0.9731	0.9316	0.9622
IMG_9	0.9669	0.9545	0.9714

IMG_141	0.9732	0.9551	0.9744
Media	0.9564	0.9323	0.9581

Tabla 16. Correlación entre imágenes originales sin distorsión (no alineadas).

La primera de las pruebas que realicemos (Tabla 17) parte de la configuración que se consideró al desarrollar este problema en el proceso iterativo: 30 iteraciones en las que el objetivo es alcanzar el nivel de precisión 1^{-10}, de tal forma que se intenta hallar el coeficiente de correlación de cada capa con la multiespectral GRE, además del tiempo de cómputo general (para el registro de las cuatro imágenes). La distorsión de las imágenes se elimina para seguir el proceso completo, aunque no se incluye dentro del tiempo calculado.

Tamaño inicial. Número de iteraciones máximo: 30. Nivel de precisión: 1^{-10}				
Imagen multiespectral	**Corr. GRE-RED**	**Corr. GRE-NIR**	**Corr. GRE-REG**	**Tiempo de cómputo (ms)**
Primer conjunto de datos (1-8). ID: 1				
IMG_1	0.9545	0.9186	0.9403	3627
IMG_4	0.9888	0.9473	0.9681	3576
IMG_6	0.9861	0.9591	0.9772	3622
Segundo conjunto de datos (35-58). ID: 2				
IMG_48	0.9654	0.9107	0.9368	3580
IMG_58	0.9546	0.9088	0.9321	3607
IMG_53	0.9888	0.9473	0.9681	3573
IMG_37	0.9628	0.9265	0.9461	3569
Tercer conjunto de datos (0-150). ID: 3				
IMG_35	0.9973	0.9654	0.9785	3543
IMG_107	0.9973	0.9621	0.9776	3558
IMG_136	0.9973	0.9677	0.9767	3577
IMG_78	0.9957	0.9640	0.9731	3575

IMG_142	0.9962	0.9632	0.9749	3567
IMG_144	0.9964	0.9580	0.9715	3586
IMG_9	0.9967	0.9713	0.9810	3585
IMG_141	0.9966	0.9746	0.9791	3518
Media	0.9849	0.9496	0.9654	3577.53
Media conjunta			0.9666	3577.53
Coeficiente de correlación / Tiempo * 1000				0.2701

Tabla 17. Resultados obtenidos en el registro de imágenes multiespectrales para un nivel de precisión 1^{-10}, utilizando 30 iteraciones como máximo.

Es necesario comentar algunos aspectos para comprender los resultados que aquí se muestran:

1. Las imágenes más afines en intensidad respecto a GRE son RED y REG, por lo que es normal que presenten coeficientes más altos (ver Tabla 13).
2. Los dos primeros conjuntos de datos suelen incluir en la mayoría de sus imágenes estructuras creadas por el ser humano, que sí varían con mayor frecuencia su reflectancia entre bandas, provocando coeficientes más bajos. El último de los conjuntos, salvo alguna imagen aislada, no contiene estructuras de tal tipo.
3. Dentro de este tiempo de cómputo no sólo se registra sino también se modifica tamaño, se recortan imágenes, etc. En definitiva, se ejecuta el algoritmo completo que en su momento se propuso.
4. Los tiempos aquí representados se han obtenido con el ordenador que se presenta en el presupuesto de este trabajo.

Se puede observar como el tiempo de cómputo es relativamente elevado, y lo será aún más si se lleva a un dispositivo de capacidades reducidas, por ejemplo, para obtener información durante el vuelo.

Se comprueba además, incrementando el número de iteraciones y nivel de precisión, que hemos alcanzado un máximo a priori local, dado que configuraciones mas estrictas consiguen el mismo nivel de correlación. Nótese como aquí sólo se muestran 4 decimales del coeficiente, por lo que es posible que con una determinada

configuración pudiera mejorarse dicho valor en niveles muy pequeños, lo que no parece que sea rentable teniendo en cuenta que el tiempo de cómputo se dispara.

Se procede a comprobar qué sucede cuando se utilizan niveles de precisión más bajos (valor mayor), lo que implica que se necesitan menos iteraciones para converger (Tabla 18).

Tamaño inicial. Número de iteraciones máximo: 15. Nivel de precisión: 1^{-2}				
Imagen multiespectral	**Corr. GRE-RED**	**Corr. GRE-NIR**	**Corr. GRE-REG**	**Tiempo de cómputo (ms)**
Primer conjunto de datos (1-8). ID: 1				
IMG_1	0.9545	0.9181	0.9395	1128
IMG_4	0.9814	0.9473	0.9681	950
IMG_6	0.9817	0.9591	0.9767	1079
Segundo conjunto de datos (35-58). ID: 2				
IMG_48	0.9654	0.9107	0.9368	1282
IMG_58	0.9547	0.9088	0.9321	1224
IMG_53	0.9814	0.9473	0.9681	955
IMG_37	0.9463	0.9265	0.9460	1100
Tercer conjunto de datos (0-150). ID: 3				
IMG_35	0.9973	0.9645	0.9766	1090
IMG_107	0.9973	0.9621	0.9776	1356
IMG_136	0.9781	0.9623	0.9760	581
IMG_78	0.9956	0.9615	0.9696	1353
IMG_142	0.9772	0.9632	0.9748	989
IMG_144	0.9781	0.9580	0.9715	994
IMG_9	0.9967	0.9713	0.9810	1471
IMG_141	0.9967	0.9713	0.9810	1110
Media	0.9788	0.9488	0.965	1110.8
Media conjunta			0.9642	1110.8

Coeficiente de correlación / Tiempo * 1000	0.8602

Tabla 18. Resultados del registro de imágenes al reducir nivel de precisión.

Si bien la correlación entre imágenes decrece, el tiempo de cómputo disminuye también en gran medida. Parece por lo tanto necesario encontrar alguna solución intermedia que se acerque al máximo local de resultados previos (que es o está muy cerca del global) y que obtenga un tiempo de cómputo más cercano a esta situación. Se propone a continuación considerar el nivel de precisión de la primera de las pruebas y el número de iteraciones de esta última (Tabla 19).

Tamaño inicial. Número de iteraciones máximo: 15. Nivel de precisión: 1^{-10}				
Imagen multiespectral	**Corr. GRE-RED**	**Corr. GRE-NIR**	**Corr. GRE-REG**	**Tiempo de cómputo (ms)**
Primer conjunto de datos (1-8). ID: 1				
IMG_1	0.9545	0.9186	0.9403	1933
IMG_4	0.9866	0.9474	0.9682	1900
IMG_6	0.9838	0.9592	0.9772	1927
Segundo conjunto de datos (35-58). ID: 2				
IMG_48	0.9654	0.9107	0.9367	1884
IMG_58	0.9546	0.9088	0.9321	1894
IMG_53	0.9866	0.9474	0.9682	1921
IMG_37	0.9628	0.9265	0.9461	1895
Tercer conjunto de datos (0-150). ID: 3				
IMG_35	0.9973	0.9654	0.9785	1905
IMG_107	0.9973	0.9621	0.9776	1920
IMG_136	0.9966	0.9645	0.9766	1900
IMG_78	0.9957	0.9640	0.9731	1903
IMG_142	0.9941	0.9632	0.9749	1909
IMG_144	0.9909	0.9580	0.9715	1898

IMG_9	0.9967	0.9713	0.9810	1911
IMG_141	0.9966	0.9746	0.9791	1873
Media	0.9839	0.9493	0.9654	1904.86
Media conjunta			0.9662	1904.86
Coeficiente de correlación / Tiempo * 1000				0.5072

Tabla 19. Resultados obtenidos tras ejecutar el registro de imágenes para unos parámetros intermedios entre las dos pruebas anteriores.

De nuevo, los principales problemas se encuentran en aquellas imágenes donde se representan estructuras como edificios. Por otro lado, se alcanzan cotas de correlación muy cercanas a las de la primera prueba, que se consideraba prácticamente como un máximo local, reduciendo a la vez el tiempo de ejecución, por lo que se puede haber alcanzado una configuración de compromiso entre las dos anteriores. En cualquiera de los casos, dada una aplicación de este algoritmo habrá que decidir qué es lo más conveniente. Por supuesto, si no hay ningún problema en obtener los resultados en un tiempo mayor, la primera configuración deberá ser necesariamente la mejor. Tampoco se podría descartar la segunda configuración para dispositivos más reducidos.

Una última prueba que podemos hacer es la reducción de tamaño de la imagen, que manteniendo estos parámetros, debe mejorar en gran medida el tiempo de ejecución (Tabla 20). La desventaja de esta técnica es que habrá que aumentar el nivel de precisión (menor valor) y el número de iteraciones, aunque el hecho de la que imagen sea más pequeña implica que el cálculo de correlación que define el algoritmo de registro, ECC, se puede realizar en un tiempo menor.

Tamaño inicial / 3. Número de iteraciones máximo: 150. Nivel de precisión: 1^{-60}				
Imagen multiespectral	**Corr. GRE-RED**	**Corr. GRE-NIR**	**Corr. GRE-REG**	**Tiempo de cómputo (ms)**
Primer conjunto de datos (1-8). ID: 1				
IMG_1	0.9540	0.9181	0.9394	1797
IMG_4	0.9883	0.9473	0.9665	1780
IMG_6	0.9849	0.9610	0.9767	1818

Segundo conjunto de datos (35-58). ID: 2				
IMG_48	0.9651	0.9113	0.9368	1746
IMG_58	0.9651	0.9113	0.9368	1774
IMG_53	0.9883	0.9473	0.9665	1760
IMG_37	0.9618	0.9266	0.9455	1774
Tercer conjunto de datos (0-150). ID: 3				
IMG_35	0.9972	0.9659	0.9778	1805
IMG_107	0.9970	0.9618	0.9762	1788
IMG_136	0.9969	0.9671	0.9752	1761
IMG_78	0.9948	0.9633	0.9714	1798
IMG_142	0.9961	0.9632	0.9740	1773
IMG_144	0.9963	0.9580	0.9702	1774
IMG_9	0.9964	0.9713	0.9806	1789
IMG_141	0.9965	0.9750	0.9786	1789
Media	0.9852	0.9499	0.9648	1781.73
Media conjunta			0.9666	1781.73
Coeficiente de correlación / Tiempo * 1000				0.542

Tabla 20. Resultados obtenidos al realizar el registro de imágenes multiespectrales reduciendo previamente el tamaño de las mismas.

Si bien los resultados obtenidos son mejores que los de cualquier otra prueba anterior, la sensación desde la propia aplicación, al visualizar las imágenes, es que los resultados distan mucho de los obtenidos con configuraciones previas. Esto se debe a que al reducir el tamaño de la imagen, la información que antes se encontraba en varios píxeles ahora se debe concentrar en un único píxel. Si a esto le añadimos que el algoritmo de correlación utilizado para esta experimentación trabaja a nivel de píxel, se obtienen estos resultados, que muestran como la última configuración es la mejor, pero desde el punto de vista del usuario, en el que también nos incluimos nosotros como desarrolladores, la solución es aparentemente peor que en pruebas previas.

En cualquiera de los casos, siempre hay que algo que se debe tener en cuenta, especialmente para experimentos posteriores: aunque se trabaje en el registro con imágenes de menor tamaño, se puede extraer la matriz de transformación obtenida y aplicársela a la imagen original. Por lo tanto, no debe haber ningún rechazo a trabajar con imágenes con algo menos de detalle, dado que el algoritmo de registro trabaja a nivel de subpíxel.

De forma gráfica, los coeficientes de correlación y los tiempos de ejecución medios obtenidos para cada configuración se representan tanto en la Ilustración 160 como en la Ilustración 161.

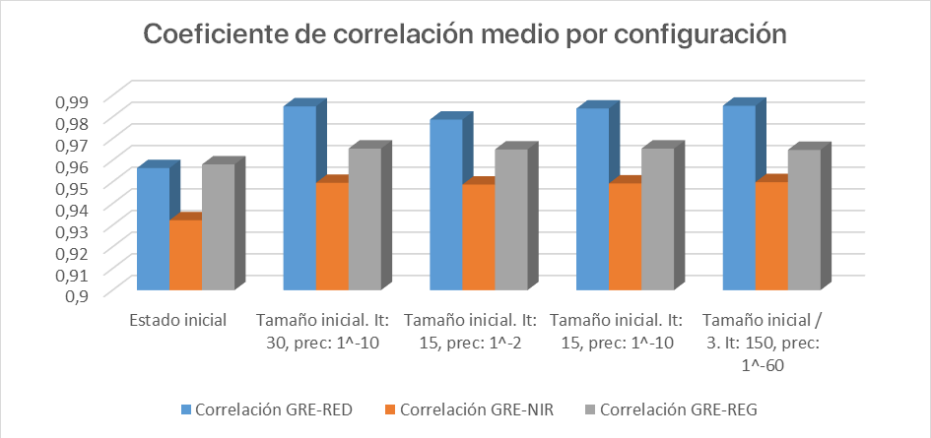


Ilustración 160. Representación gráfica de los coeficientes de correlación medios obtenidos para cada configuración.

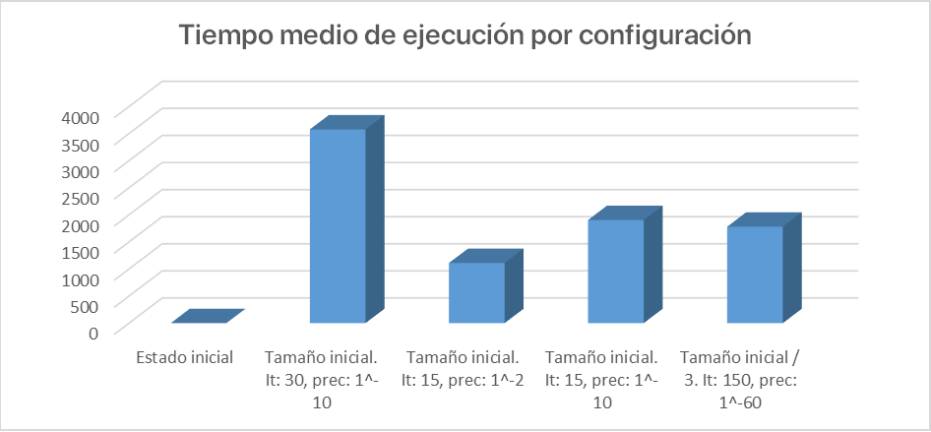


Ilustración 161. Representación gráfica del tiempo de ejecución medio por configuración.

Por lo tanto, se han presentando aquí un conjunto de configuraciones y ninguna de ellas ha resultado ser la óptima para cualquiera de los casos posibles, aunque la primera podría ser claramente la mejor si su tiempo de cómputo, alrededor de tres segundos y medio, no se considera excesivo. Debe escogerse alguna de estas

configuraciones o una cercana según el uso que se desee dar: ejecución en dispositivos de capacidad reducida, análisis de imágenes en laboratorio después del vuelo, etc.

Tampoco se debe descartar por completo la última opción presentada, porque de hecho aparece en apartados posteriores. El principal problema que nosotros tenemos aquí es que la imagen multiespectral tiene muchos detalles que se pierden al reducir su tamaño, generando un registro lejos de lo esperado.

3.4 - Registro de imágenes RGB y multiespectrales

A diferencia de otras situaciones que se irán viendo en este apartado, en esta operación de registro se desconoce el lugar que ocupa la imagen multiespectral dentro de la RGB, dado que su captura es totalmente independiente: no se obtienen del mismo dispositivo ni en los mismos instantes de tiempo. Se fijaba así como pivote la imagen multiespectral y se intentaba encontrar la imagen RGB más cercana en el tiempo.

El principal problema que tendremos a la hora de comprobar todo esto es que no es fácil hallar dos imágenes, cada una de un tipo, que representen un mismo área; el hecho de que ambas se capturen en instantes diferentes hace que se obtengan zonas de terreno muy dispares, donde la intersección, si la hubiera, sería una pequeña parte de la imagen (veáse la Ilustración 162). Nótese que estamos tratando con instantes de captura muy distantes, incluso con la selección de imágenes de ambos tipos más cercanas en el tiempo.

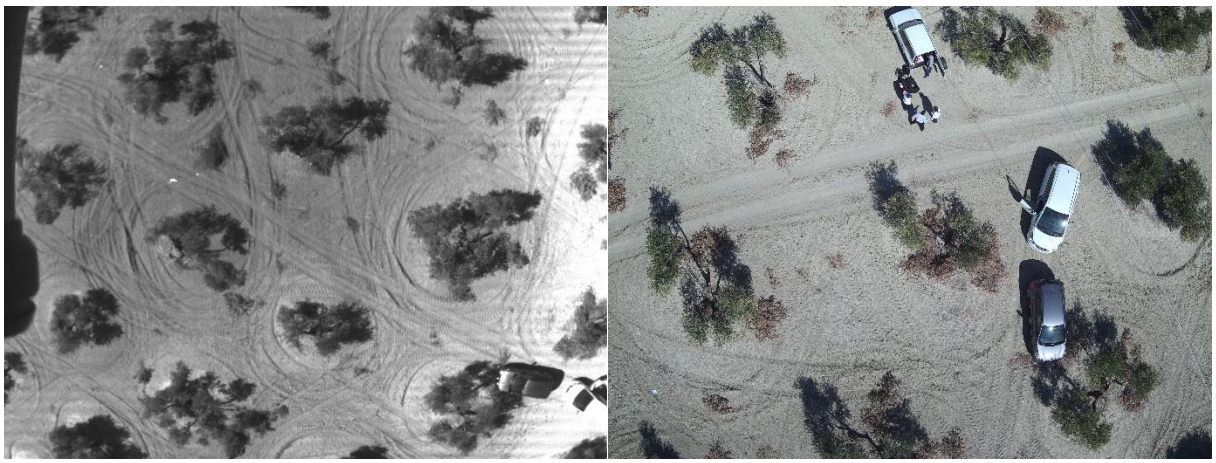


Ilustración 162. Imagen RGB más cercana en el tiempo a la multiespectral de la izquierda.

Por lo tanto, el número de imágenes que se propone para este ejercicio de experimentación será escaso, y la entrada del algoritmo que calcula el coeficiente de correlación serán dos imágenes, la RGB en su estado inicial y la imagen RGB con la multiespectral superpuesta (ya alineada). Esto nos lleva a pensar que el coeficiente tendrá siempre una base correspondiente a esa parte de la imagen RGB que se mantiene invariable.

De nuevo, toda la experimentación que aquí se desarrolle gira en torno a la configuración del registro de imágenes. Se parte de unos valores iniciales (Tabla 21), que a priori generan un tiempo de ejecución relativamente elevado, para ir disminuyéndolo.

Tamaño: multiespectral / 2. Número de iteraciones máximo: 400. Nivel de precisión: 1^{-60}			
Imagen multiespectral	**Imagen RGB**	**Coeficiente de correlación**	**Tiempo de cómputo (ms)**
Primer conjunto de datos. ID: 3			
IMG_146	IMG_916	0.9683	8090
IMG_148	IMG_918	0.9664	7882
IMG_142	IMG_904	0.9711	7506
IMG_137	IMG_892	0.9761	7221
IMG_144	IMG_906	0.9690	7266
Media		0.9701	7593
Coeficiente de correlación / Tiempo * 1000			0.1277

Tabla 21. Resultados del registro de imágenes RGB y multiespectrales con la configuración inicial.

Derivado de una configuración con tanta precisión e iteraciones, se obtiene un tiempo de cómputo elevado. Lo que se hace en pruebas sucesivas es intentar reducir ese tiempo, intentando mantener el coeficiente de correlación, que se presupone, debe ser un máximo global para el tamaño de imagen propuesto.

La Tabla 22 se corresponde a una prueba donde se reduce notablemente el número de iteraciones y el nivel de precisión.

Tamaño: multiespectral / 2. Número de iteraciones máximo: 150. Nivel de precisión: 1^{-30}			
Imagen multiespectral	**Imagen RGB**	**Coeficiente de correlación**	**Tiempo de cómputo (ms)**
Primer conjunto de datos. ID: 3			
IMG_146	IMG_916	0.9683	2810
IMG_148	IMG_918	0.9664	2839
IMG_142	IMG_904	0.9711	2757
IMG_137	IMG_892	0.9761	2718
IMG_144	IMG_906	0.9690	2724
Media		0.9701	2769.6
Coeficiente de correlación / Tiempo * 1000			0.3502

Tabla 22. Resultado del registro de imágenes multiespectrales y RGB al reducir iteraciones y nivel de precisión.

Se obtiene así una configuración cuyo coeficiente de correlación es exactamente es el mismo, reduciendo el tiempo de ejecución. Por lo tanto, se consideran estos últimos valores como los óptimos, empleando un tamaño de imagen equivalente al de la multiespectral entre dos. Muy probablemente existen configuraciones con aún menor carga, pero con estos valores se obtiene un tiempo razonable utilizando un número de iteraciones que se supone que serán suficientes para registros de imágenes que pudieran ser más complejos.

La última prueba que en este apartado se realiza va dedicada al tamaño de la imagen (Tabla 23). Se comprueba así qué sucede si se mantiene el tamaño de la multiespectral, lo que implica que es necesario reducir número de iteraciones y precisión por dos razones:

1. Evitar tiempos de cómputo demasiado elevados.
2. Dado que las imágenes tienen más resolución, no es necesario especificar tanto nivel de precisión, que tiene su equivalencia directa con el término de subpíxel (hay más detalles en la imagen que con la configuración anterior).

Tamaño: multiespectral. Número de iteraciones máximo: 100. Nivel de precisión: 1^{-20}			
Imagen multiespectral	**Imagen RGB**	**Coeficiente de correlación**	**Tiempo de cómputo (ms)**
Primer conjunto de datos (ID: 3)			
IMG_146	IMG_916	0.9569	7405
IMG_148	IMG_918	0.9408	7022
IMG_142	IMG_904	0.9453	7058
IMG_137	IMG_892	0.9747	7448
IMG_144	IMG_906	0.9680	7087
Media		0.9571	7204
Coeficiente de correlación / Tiempo * 1000			0.1328

Tabla 23. Resultados del registro de imágenes multiespectrales y RGB al aumentar el tamaño de imagen.

Además de las diferencias obvias en cuanto a tiempo, el registro como tal es erróneo en varios de los casos aquí presentados, por lo que la configuración que acabamos de plantear se debe descartar por completo. Para mantener el tamaño original de la imagen habría que aumentar el número de iteraciones y el nivel de precisión hasta tal punto que el tiempo de ejecución sería excesivo.

Por último, y como ya hemos hecho en el apartado anterior, se muestran a modo de resumen las gráficas que representan los resultados obtenidos para todas las configuraciones que aquí hemos probado (ver Ilustración 163 e Ilustración 164).

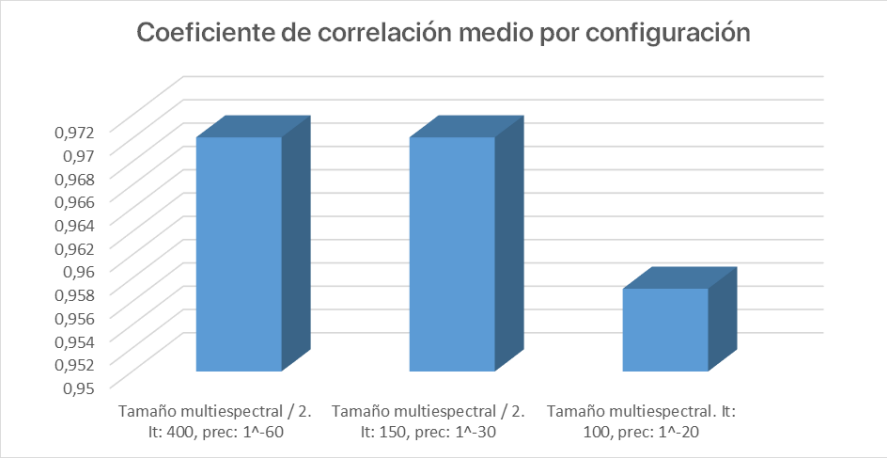


Ilustración 163. Representación gráfica del coeficiente de correlación medio para cada configuración propuesta.

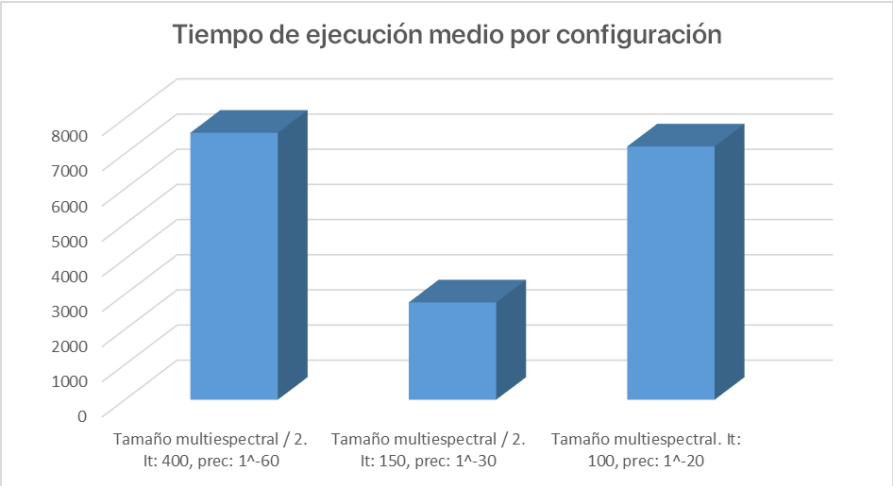


Ilustración 164. Representación gráfica del tiempo de ejecución medio para cada configuración propuesta.

3.5 - Registro de imágenes RGB y térmicas

Como bien sabemos, el proceso de registro de imágenes RGB y térmicas consiste en el recorte de parte de la imagen RGB, que es el área donde se debe registrar la imagen térmica. Se proponían además múltiples soluciones para esta operación: escoger un área lo suficientemente grande (deja bordes negros en la térmica alineada), escoger un área menor que incluye a la imagen térmica siempre, etc.

Dada la técnica de experimentación que aquí se propone, basada en un coeficiente de correlación, la primera de las operaciones de recorte mencionadas posiblemente influya negativamente en el valor del mismo, sin embargo, es la que nos garantiza que se registra la imagen completa y no sólo una parte de la misma. Por lo tanto, aunque se esperan coeficientes menores que en apartados previos, se propone utilizar el caso en el que el área recortada de la imagen RGB representa un área mayor que la imagen térmica.

A continuación se propone una situación de registro, donde el algoritmo toma como entrada imágenes como las de la Ilustración 165, a partir de las cuales se obtiene el resultado de la Ilustración 167, derivado de la combinación del recorte de la RGB y la térmica alineada después del registro (Ilustración 166).

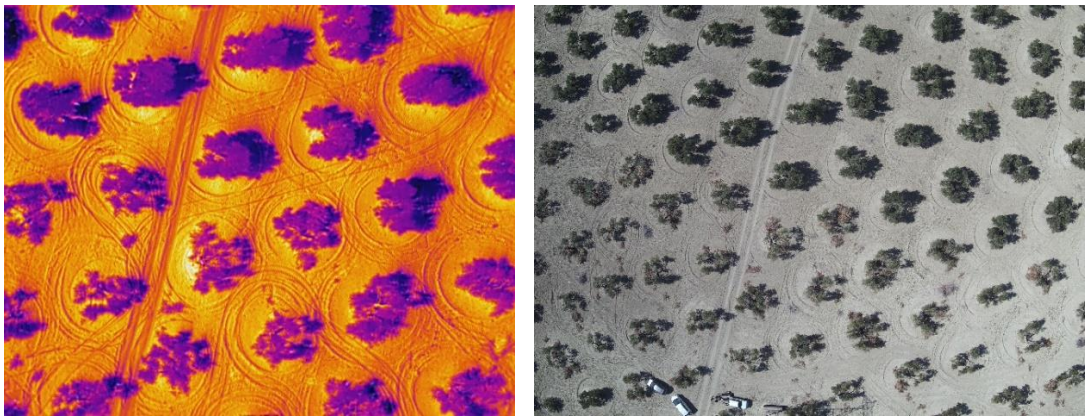


Ilustración 165. Imágenes propuestas como entrada del algoritmo de registro de imágenes RGB y térmicas.

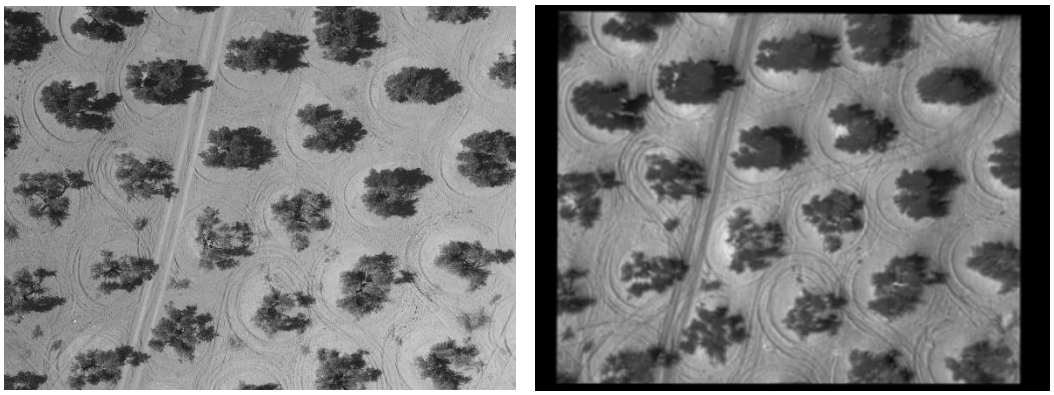


Ilustración 166. Imagen térmica alineada (derecha) con el recorte de la imagen RGB (izquierda).

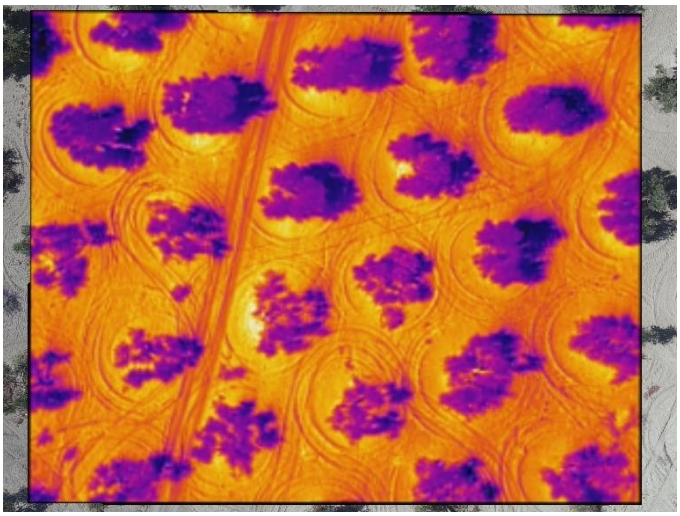


Ilustración 167. Resultado de combinar las dos imágenes anteriores.

Por lo tanto, lo que hay que comparar mediante nuestro coeficiente de correlación es la primera imagen de la Ilustración 166 y la Ilustración 167. De nuevo,

se proponen una serie de parámetros para el registro de imágenes de tal forma que nos ayuden a decidir cuál sería la mejor configuración.

A diferencia de las imágenes multiespectrales, donde la reducción de las dimensiones de la imagen conducía a un registro incorrecto a nivel visual, la ventaja que tiene la térmica es que no representa las figuras como tal de la RGB, sino que suele contener un área del mismo color, derivado de la temperatura, alrededor de los objetos. Esto permite que ahora sí podamos perder detalles en la RGB y seguir haciendo un registro correcto, aunque esto no quita que una resolución mayor conduzca a un registro más exacto a cambio de un tiempo de ejecución mayor.

Para esta prueba se dispone de dos conjuntos de datos, con imágenes térmicas y RGB, aunque el segundo de ellos contiene imágenes del primer tipo en escala de grises. Es decir, el algoritmo de registro se podrá efectuar pero nada más, no sirve para extraer información en un futuro. Se propone así para cada prueba 12 imágenes, 6 de cada conjunto.

En esta ocasión se parte de una configuración bastante costosa, donde el número de iteraciones es 300 y el nivel de precisión es 1^{-60}. Asimismo, la imágenes se reducen al tamaño de la térmica dividido entre dos (ver Tabla 24).

Tamaño: térmica / 2. Número de iteraciones máximo: 300. Nivel de precisión: 1^{-60}		
Par RGB-térmica	**Correlación entre RGB y térmica**	**Tiempo de cómputo (ms)**
Primer conjunto de datos. ID: 3		
IMG_394	0.9756	2779
IMG_626	0.9728	2834
IMG_726	0.9728	2951
IMG_922	0.9674	2791
IMG_554	0.9786	2790
IMG_928	0.9756	2761
Segundo conjunto de datos. ID: 4		
IMG_467	0.9416	2731
IMG_643	0.9511	2789

IMG_405	0.9266	2716
IMG_607	0.9511	2794
IMG_963	0.9444	2881
IMG_475	0.9396	2946
Media	0.9581	2813.58
Coeficiente de correlación / Tiempo * 1000		0.3405

Tabla 24. Resultados del registro de imágenes térmicas y RGB con la configuración inicial.

Nótese como el coeficiente de correlación debe ser necesariamente menor que en apartados anteriores, dado que aquí permanece aún parte de ese fondo de color nulo en la imagen térmica. En cualquiera de los casos, no podemos analizar cómo de buenos son los coeficientes obtenidos porque no tenemos ninguna otra referencia, dado que no se pueden comparar las imágenes iniciales (no se corresponden al mismo área, no tiene sentido extraer la correlación).

Como bien sabemos, al reducir el tamaño de la imagen podría obtenerse una alineación algo peor (aunque como bien se ha explicado antes, aquí es una operación completamente necesaria, que además produce buenos resultados). Por lo tanto, en la siguiente prueba trataremos de realizar el registro con el tamaño original de la térmica, pero habrá que reducir nivel de precisión y número de iteraciones para evitar un tiempo de ejecución excesivamente grande.

Tamaño térmica. Número de iteraciones máximo: 100. Nivel de precisión: 1^{-20}		
Par RGB-térmica	**Correlación entre RGB y térmica**	**Tiempo de cómputo (ms)**
Primer conjunto de datos. ID: 3		
IMG_394	0.9771	3204
IMG_626	0.9745	3320
IMG_726	0.9741	3211
IMG_922	0.9690	3200
IMG_554	0.9803	3267
IMG_928	0.9770	3341

Segundo conjunto de datos. ID: 4		
IMG_467	0.9385	3134
IMG_643	0.9494	3137
IMG_405	0.9265	3187
IMG_607	0.9502	3236
IMG_963	0.9425	3177
IMG_475	0.9358	3257
Media	0.9579	3222.58
Coeficiente de correlación / Tiempo * 1000		0.2972

Tabla 25. Resultados de registro de imágenes RGB y térmicas, conservando el tamaño de estas últimas.

El efecto que se puede ver en estos resultados es el mismo que se exponía en el registro de imágenes multiespectrales: a menor tamaño de imagen, el coeficiente de correlación aumenta (por poco que sea) al perderse detalles y homogeneizarse la imagen. En cualquiera de los casos, al contrario de lo que pasara en la experimentación citada previamente, no se encuentran diferencias notables en el registro de estas imágenes con distintos tamaños (a nivel visual), por lo se que puede optar perfectamente por la configuración de la Tabla 24, teniendo en cuenta que el tiempo de cómputo es aún menor que este aquí encontrado.

Para comprobar que la primera configuración es la mejor obtenida hasta este momento, se propone bajar el número de iteraciones y el nivel de precisión, de tal forma que así se puede descartar que el registro converja antes y estemos usando una opción demasiado compleja, gastando así iteraciones en una mejora ya imposible (a nivel de coeficiente de correlación).

Tamaño: térmica / 2. Número de iteraciones máximo: 60. Nivel de precisión: 1^{-20}		
Par RGB-térmica	**Correlación entre RGB y térmica**	**Tiempo de cómputo (ms)**
Primer conjunto de datos. ID: 3		
IMG_394	0.9756	1419

IMG_626	0.9728	1389
IMG_726	0.9728	1411
IMG_922	0.9674	1396
IMG_554	0.9786	1402
IMG_928	0.9756	1401
Segundo conjunto de datos. ID: 4		
IMG_467	0.9417	1381
IMG_643	0.9511	1404
IMG_405	0.9266	1400
IMG_607	0.9511	1382
IMG_963	0.9444	1403
IMG_475	0.9396	1390
Media	0.9581	1398.16
Coeficiente de correlación / Tiempo * 1000		0.6852

Tabla 26. Resultados de registro de imágenes RGB y térmicas, repitiendo la primera prueba con menos precisión e iteraciones.

Se puede observar como se obtiene la misma correlación de la primera prueba y un menor tiempo de ejecución. Se ha hallado pues una nueva configuración que mejora los resultados de la primera, utilizando la reducción de tamaño de las imágenes RGB y térmica a la mitad de esta última. Si bien pueden existir configuraciones aún mejores, esta ya es lo suficientemente buena en cuanto a tiempo de cómputo, aunque si fuera necesario decrementar aún más dicho tiempo, podrían probarse configuraciones con menos precisión y menos iteraciones, pero siempre es recomendable dejar un margen, que se apoya en estas iteraciones sobrantes para aquellos casos más complejos.

Otros tantos parámetros a tener en cuenta en este algoritmo son los relacionados con la eliminación de la distorsión de la imagen RGB, dado que estos no vienen dados por los metadatos, y por lo tanto, habrá que especificarlos como constantes en la aplicación. Sin embargo, los valores especificados tienen una base

razonable dado que proceden de las imágenes multiespectrales, a excepción de la distancia focal, que se debe adaptar a la de esta imagen, y lo mismo sucede para el punto principal de la misma, que por simplificación se sitúa justamente en la mitad.

Por último, se representan todos los datos obtenidos de forma gráfica tanto en la Ilustración 168 como en la Ilustración 169.

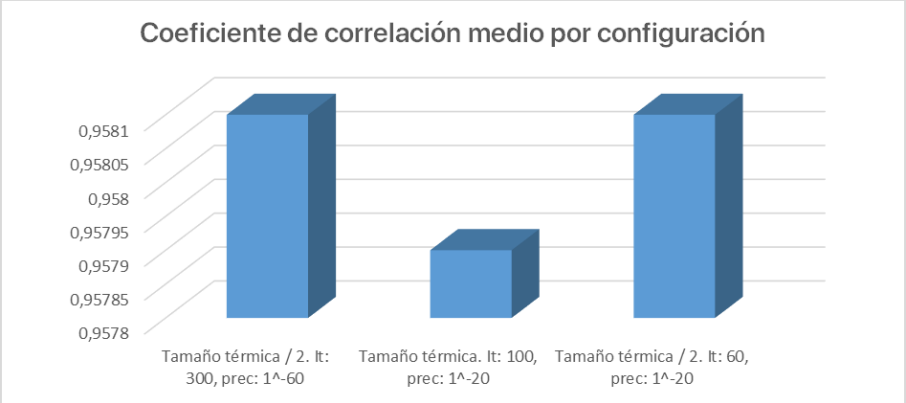


Ilustración 168. Representación gráfica del coeficiente de correlación medio para cada configuración propuesta.

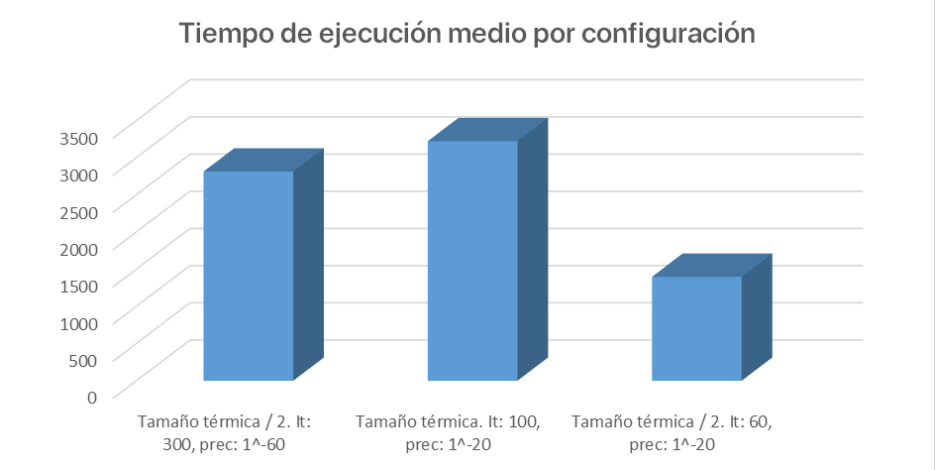


Ilustración 169. Representación gráfica del tiempo de ejecución medio para cada configuración propuesta.

3.6 - Especificación de prototipos realizados

El software desarrollado para las experimentaciones consiste en un único *script* que recibe dos o más imágenes almacenadas en el sistema para hallar la correlación entre una captura de referencia y el resto. En este caso se ha utilizado el lenguaje de programación Python en el entorno de desarrollo PyCharm. No es necesario profundizar mucho más en este *script,* dado que apenas contiene las funciones de

lectura y el cálculo del coeficiente de correlación, que ya nos los da OpenCV. En cualquiera de los casos, se adjunta un proyecto de PyCharm a la entrega de este trabajo con la experimentación llevada a cabo.

Para realizar la medición de tiempos sí que es necesario incorporar un nuevo fichero en el proyecto, que se encarga de controlar un reloj cuando se le pida (iniciarlo y obtener duración, que se corresponde al comienzo y fin del algoritmo). Se propone así el esquema de la Ilustración 170.

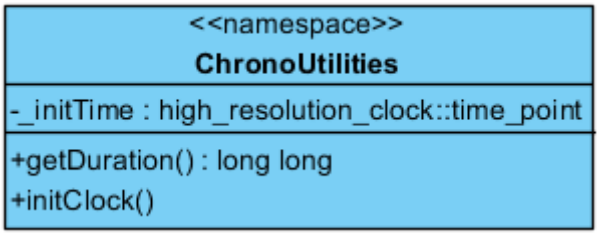


Ilustración 170. Diagrama de clases del espacio de nombres que nos permite medir el tiempo de ejecución.

Si bien la experimentación es una parte importante de cualquier proyecto de investigación, en nuestro caso, por el ámbito que trata, y dado que no tenemos una solución cierta para cada caso, era particularmente complejo buscar una forma de validar y probar los resultados obtenidos. Es por ello que el software de experimentación desarrollado se limita a estos dos ficheros aquí presentados.

4 - RESULTADOS Y DISCUSIÓN

Partíamos en este trabajo comentando los problemas que presentaban los sensores a utilizar y qué era lo que intentaríamos obtener. Una vez se ha desarrollado cada iteración, se ha ido viendo como los problemas que se encontraban era incluso más numerosos. El objetivo ha sido siempre aunar todos los datos disponibles para obtener información de interés, aunque cada unos de esos datos por sí mismos suponían de por si una fuente de información muy valiosa (véase la obtención del estado de salud de un árbol a través de una imagen multiespectral).

Sin embargo, si se consigue aunar y dividir toda esa información en pequeños conjuntos, uno por cada árbol, se puede incorporar la dimensión tiempo a todo este esquema con mayor fuerza. Es por ello que nuestro primer objetivo era **extraer los contornos de la vegetación**, y más concrementamente, cada contorno debía

pertenecer a un árbol representado como un individuo. **El algoritmo que se propone es capaz de extraer toda la vegetación de la imagen de forma muy precisa**, dado que se basa en la información multiespectral. Ahora bien, **todas las operaciones que se proponen tienen como objetivo diferenciar esos individuos, algo que llegado a este punto, parece difícil de obtener de forma automática**.

Son varias las razones que nos llevan a pensar esto último:

- El ángulo de visión puede representar árboles diferentes como si fueran un único árbol.
- No todas las plantaciones presentan una distancia tan grande entre sus árboles; ya aquí se daba esa situación y había dificultades para obtener los contornos de cada árbol.
- Justo al contrario que en el primer caso, puede haber árboles que presenten más de una copa. A través de la información multiespectral eramos capaces de distinguir la vegetación, lo que no incluye ramas y tronco del árbol, que nos permitirían ilustrar toda esa vegetación de forma conexa.

Llegados a este punto debemos plantearnos si de verdad se necesita diferenciar cada árbol para hacer un seguimiento de la plantación. Esto que aquí se expone se puede extrapolar al centro de masas que obteníamos: había una clara distinción entre el centroide de una superficie y el de un conjunto de puntos, pero se optaba por ejecutar el algoritmo más eficiente, dado que no necesitábamos un punto con una representación semántica concreta, sólo necesitábamos un punto que nos identificara ese árbol. Sucede lo mismo para la segmentación de la vegetación, quizás no se puede extraer cada árbol, pero sí una silueta que contenga varios de ellos, de tal forma que se podría almacenar información de ese conjunto e igualmente sería válido para algunos fines de control de la plantación.

Sí es cierto que con este punto de vista se está dejando ir la oportunidad de hacer un seguimiento de un árbol a lo largo del tiempo, por ejemplo, para controlar su crecimiento, y se está enfocando hacia una base de datos que guarda contornos de cualquier masa de vegetación en un instante de tiempo concreto, cuya utilidad también es discutible.

En definitiva, **el algoritmo de extracción de contornos propuesto es capaz de extraer la masa de vegetación de forma precisa, pero necesita de un operador**

que controle el proceso y modifique los parámetros si lo que queremos es distinguir cada árbol de forma aislada.

Justo al contrario que para la extracción de contornos, **los algoritmos descritos para la adecuación de las imágenes multiespectrales sí que pueden ser de utilidad en cualquier proyecto futuro**. **La eliminación de la distorsión *fisheye* queda completamente resuelta, y lo mismo sucede para el registro de imágenes**. Para este último proceso, que se supone complejo, la configuración que mejor aproximaba el resultado deseado lo hacía en poco más de 3.5 segundos de media y obtenía un coeficiente de correlación entre imágenes multiespectrales cercano a 0.96, teniendo en cuenta que nunca se puede alcanzar la perfección por la diferencia de intensidad entre ellas.

Además, no sólo se han obtenido resultados interesantes para un futuro en el registro de imágenes multiespectrales, sino también de imágenes RGB con estas últimas y con las imágenes térmicas. Esto nos permitiría unir toda la información que tenemos a nuestro alcance, pudiendo obtener múltiples índices de interés para la vegetación: información térmica, multiespectral, etc.

Más concretamente, **en el caso del registro de la imagen térmica y la RGB los resultados son muy buenos**, alcanzándose una correlación media entre ambos tipos de imágenes de 0.9581, para lo que se empleaba poco más de un segundo (1398 ms también de media). Hay que tener en cuenta además que se alcanza este resultado desconociendo parámetros que podrían ser de interés para la operación, y que están relacionados con la eliminación de la distorsión *fisheye* de la RGB. Por lo tanto, habría que discutir si es necesario una estimación de parámetros que garantice que el registro se produce en condiciones óptimas. A juzgar por los resultados no parece que esto sea necesario, pero siempre puede haber un margen de mejora.

En el caso del registro de la imagen RGB y multiespectral, se vuelven a encontrar problemas ya mencionados, como la distorsión *fisheye* de esa imagen RGB, a lo que se debe sumar que los sensores empleados para ambos tipos de imágenes no hacen sus capturas en el mismo instante de tiempo. Se propone un enfoque basado en la búsqueda de imágenes RGB cercanas temporalmente para cada imagen multiespectral. Por lo tanto, los problemas que se encuentran se pueden resumir en los siguientes puntos:

- Se parte de entrada de una imagen multiespectral sin distorsión y una imagen RGB con distorsión. Ahora bien, comentábamos antes que se podía

eliminar dicha distorsión con unos parámetros aproximados, ¿por qué no aplicarlos ahora?

A diferencia de la térmica, que ocupa una pequeña parte de la imagen RGB, una multiespectral puede ocupar buena parte de esta última. También sabemos que la distorsión tiene especial repercursión en los extremos, por lo que no se puede aplicar en este caso la corrección definida para la imagen RGB en el registro de esta última con una térmica.

Esto plantea un problema importante, dado que al haber tanta diferencia entre imágenes hay que registrarlas con una transformación tan compleja como la homografía. En cualquiera de los casos, se ha probado en la experimentación como se alcanza una correlación bastante elevada en poco tiempo.

- **La búsqueda de la imagen RGB más cercana en el tiempo no soluciona el problema de que ambas imágenes se solapen**. Puede, y suele haber, diferencias entre ellas de varios segundos, tiempo suficiente para que en la siguiente captura se represente un área totalmente diferente (traslación + rotación del dron).

Si bien se plantea esta solución como una posibilidad de registro entre ambas imágenes, lo cierto es que se ha podido probar con pocos datos, entre otras cosas porque las condiciones de captura no facilitan este proceso. Por lo tanto, **mientras que dichas condiciones se mantengan igual, el registro de una imagen multiespectral y una RGB no parece viable, no por los resultados obtenidos del algoritmo propuesto sino porque las dos imágenes que aquí se tratan capturan áreas de terreno completamente diferentes**.

Además de todos estos algoritmos de registro, también se proponen operaciones con las imágenes multiespectrales que nos conducen hasta la **georreferenciación**. **En este caso sí que se debe descartar desde el comienzo de este párrafo la viabilidad de todo este proceso**, dado que las dificultades encontradas son diversas y difícilmente se pueden solucionar considerando exclusivamente las imágenes multiespectrales:

- Posicionamiento global impreciso, con errores de 5-6 metros.
- Metadatos de ángulos de rotación incorrectos.

- Altura de vuelo desconocida (sólo se almacena la altura sobre el nivel del mar).

Si se acumulan todos estos errores, se obtiene una georreferenciación lejos de lo esperado, y no sólo se deriva de este apartado una simple operación de cálculo de coordenadas de un punto de la imagen, sino también otros procesos completos como el pegado o cosido de imágenes (que hemos generalizado como ortomosaico, aunque no lo es). Por lo tanto, en esta área se han propuesto múltiples algoritmos relacionados con la georreferenciación que obtienen resultados imprecisos a consecuencia de todos los errores citados de las imágenes multiespectrales. En cualquiera de los casos, la investigación de este último apartado ha servido para descartar esta vía como una posibilidad de trabajo futuro.

5 - CONCLUSIONES Y TRABAJOS FUTUROS

Para terminar este trabajo, en base a todos los resultados que se han ido exponiendo durante el mismo, se pueden obtener una serie de conclusiones que nos conducen también a proponer mejoras y futuras líneas de trabajo.

En primer lugar, **este trabajo ha servido para explorar capacidades de las imágenes multiespectrales así como ciertas operaciones hasta ahora poco tratadas en el ámbito de la agricultura de precisión**. En todo este proceso de investigación se ha demostrado que las imágenes multiespectrales no son aptas para operaciones que involucren el posicionamiento sobre la superficie terrestre, al menos por sí solas. Sin embargo, las imágenes como tal sí que contienen información de interés, y por esta línea de procesamiento de la imagen multiespectral se han obtenido resultados más interesantes: desde la extracción de vegetación del área capturada hasta el registro con una imagen RGB, teniendo en cuenta que para alcanzar todo esto hay que adecuar primero las imágenes, y este es un proceso que se ha resuelto por completo, involucrando tanto la eliminación de la distorsión de ojo de pez como el registro de imágenes multiespectrales.

Igualmente, cabe destacar que hay ciertas operaciones, como el registro de la imagen RGB con las capturas multiespectrales o térmicas, que suponen procesos poco explorados y en los que se han obtenido buenos resultados.

En definitiva, el objetivo de este trabajo ha sido explorar vías de trabajo que involucran información útil para la agricultura de la precisión, de tal forma que se han expuesto claramente los problemas, se han propuesto soluciones, y se ha analizado la viabilidad de estas en función de los resultados obtenidos. Unas vías de trabajo se descartan completamente, otras aún presentan problemas y otras tantas se han resuelto prácticamente por completo.

Incluso para este último grupo se pueden proponer **mejoras**, y algunas en las que podemos pensar son las siguientes:

- En este trabajo queda definido un algoritmo de registro de imágenes multiespectrales para imágenes capturadas desde un dron a una altura prudente. Sin embargo, durante el proceso de despegue, o en el caso de que se utilice el sensor desde el terreno, la diferencia entre imágenes será mayor, y muy posiblemente no se alcance la convergencia si empleamos la configuración definida en la experimentación. Por lo tanto, se debe proponer una mejora que no aumente el tiempo de ejecución en gran medida, y una posible solución pasa por considerar menores dimensiones de imágenes, al menos para alcanzar una aproximación, aunque luego podría ejecutarse el registro con los valores inicialmente propuestos para obtener ya una solución con más precisión. En definitiva, una posible mejora sería considerar una pirámide donde cada nivel está formado por la imagen con dimensiones diferentes, tal que una primera aproximación se podría obtener con la dimensión más pequeña. Hecho esto, la diferencia entre imágenes ya no será tan grande; ahora podría registrarse el nivel inferior (mayor tamaño), al que se le ha aplicado la matriz de transformación que se acaba de hallar, y así hasta alcanzar un nivel o un determinado umbral (no debe ser una correlación por los principios expuestos en la experimentación).
- En el área del registro de imágenes térmicas y RGB no se puede dar la situación expuesta en la mejora anterior, pero sí que sabemos que se han empleado unos parámetros de eliminación de la distorsión que aproximan el estado deseado en la parte central de la imagen. Sin embargo, para mejorar este algoritmo podrían estimarse los parámetros de distorsión de la imagen de tal forma que se maximizara la correlación entre capturas.

- De nuevo, para el registro de imágenes multiespectrales y RGB se propone la misma mejora que antes, con la diferencia de que aquí sí es evidente la diferencia entre unos parámetros de distorsión correctos y cualquier otros. Otra mejora que se puede añadir a este apartado pasa por el proceso de captura y no por el de implementación, por lo que se propone realizar capturas de imágenes multiespectrales y RGB en el mismo instante, o bien minimizar esa diferencia de tiempo hasta valores de pocos milisegundos.
- Se ha hablado largo y tendido en el apartado de resultados de la extracción de contornos, y una mejora, siempre y cuando se decida seguir por el mismo camino, podría ser incorporar algo más de inteligencia al proceso, debido a que la imagen por sí sola no nos da detalles para distinguir cada árbol en todas las situaciones. Otra posible mejora sería descartar un porcentaje del terreno situados en los extremos, garantizando así que el ángulo de visión de la cámara no nos hace ver vegetación separada a pie de campo como si formara parte de un mismo conjunto.

No se proponen mejoras para los procesos descritos de georreferenciación porque se incluyen precisamente en ese apartado de vías de trabajo descartadas, aunque esto no implica que no sea un tema de estudio en **trabajos futuros** tales como los siguientes:

- Si bien se han propuesto muchas operaciones en este trabajo, algunas de ellas han quedado inconexas para los objetivos finales que se buscan. Una vía de trabajo futura que toma como base este proyecto podría ser la integración de toda la información en un modelo multicapa compuesto por imágenes multiespectrales, RGB y térmica. Obviamente, todo esto conduce a la extracción de índices e información de interés para el agricultor a partir de todas estas capas.
- Aprovechando el trabajo anterior, y registradas las imágenes RGB y multiespectrales, una posible vía de estudio sería esa georreferenciación que nosotros descartamos al utilizar sólo estas últimas imágenes, teniendo en cuenta que la información contenida en la imagen RGB tiene mayor precisión para este tipo de operaciones.

- Un trabajo futuro podría proponer una visualización 3D de la información multiespectral si fuéramos capaces de enlazar imágenes RGB y multiespectrales, basándonos para esto mismo en el módulo de representación tridimensional que se ha incluido en este proyecto. A partir de las primeras se podría construir una nube de puntos, y con la información multiespectral se podría asignar información a esa nube, no ya a nivel de color de la nube, sino simplemente etiquetas de información en ciertas zonas del espacio en base a los índices extraídos. Nótese como esto mismo se podría extender a realidad aumentada / virtual para ayudar al agricultor a conocer el estado de su plantación.
- Una vez se decida si el camino a seguir sigue siendo la extracción individual de árboles, debe establecerse una base de datos que almacene la información a lo largo del tiempo para que se conozcan antecedentes de ciertos estados del terreno, o bien seguir el crecimiento de cada árbol.

6 - APÉNDICES

Se utiliza este último apartado para incluir documentación adicional que pueda servir de ayuda para poner en marcha y comprender el proyecto realizado.

6.1 - Instalación y configuración del sistema

Antes de comenzar este apartado se debe hacer una clara distinción entre el ejecutable que se entrega con este proyecto y la preparación del sistema para continuar con el desarrollo del mismo.

Para el primero de los casos el único requisito es que la tarjeta gráfica soporte OpenGL 4.5, mientras que para la segunda opción el proceso será más complejo, ya que este proyecto incluye ciertas librerías que habrá que referenciar. En cualquiera de los casos, para este segundo caso tampoco se parte de cero, teniendo en cuenta que se entrega un proyecto con, al menos, el código fuente y aquellas librerías que ocupan poco espacio. Para seguir toda la preparación se muestra un proceso paso a paso hasta conseguir que nuestra aplicación pueda ejecutarse en el entorno de desarrollo Visual Studio.

1. Instalación del entorno de desarrollo Microsoft Visual Studio. Para este proyecto se ha utilizado Microsoft Visual Studio 2017 Community. En la instalación hay que tener en cuenta una serie de detalles:

    - En primer lugar, los componentes de Windows elegidos son los que aparecen en la Ilustración 171.

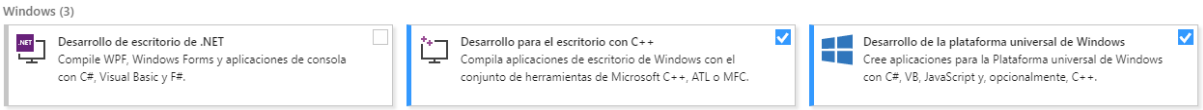


Ilustración 171. Componentes de Windows elegidos en la instalación de Microsoft Visual Studio.

    - También es necesario el SDK de Windows apropiado (con el que se ha desarrollado el proyecto). En concreto, este sería Windows 10 (10.0.17763.0), aunque en el peor de los casos podríamos descargarnos el conjunto de SDK disponibles.

2. Instalación del *framework* de creación de interfaces gráficas Qt. Si bien la versión empleada en este proyecto es la 5.11, se ha comprobado su funcionamiento con la última versión en el momento en el que se escribe este documento (5.12.3), y funciona correctamente. Parece por lo tanto existir compatibilidad bidireccional, siempre y cuando la versión introduzca cambios menores. Durante la instalación será suficiente con escoger el módulo para Windows (5.12.3 > MSVC 2017 64 bit), así como otros tantos que puedan venir seleccionados por defecto.

3. Integrar Qt en Microsoft Visual Studio. Una de las ventajas que tiene este entorno de desarrollo es que es capaz de integrarse con Qt a través de un módulo que se puede descargar, lo que nos permitirá crear un proyecto con todas las referencias a Qt ya integradas. Utilizamos el menú Herramientas > Extensiones y actualizaciones y buscamos el módulo *Qt Visual Studio Tools* (ver Ilustración 172). Aparece así un nuevo control en el menú, *Qt VS tools*, que tiene otro control, *Qt Options*, en el submenú asociado. Lo normal es que la ruta de Qt no venga desde un comienzo, por lo que hay que buscar la carpeta adecuada, que en nuestro caso es *C:/Qt/5.x.y/msvc2017_64* (ver Ilustración 173).

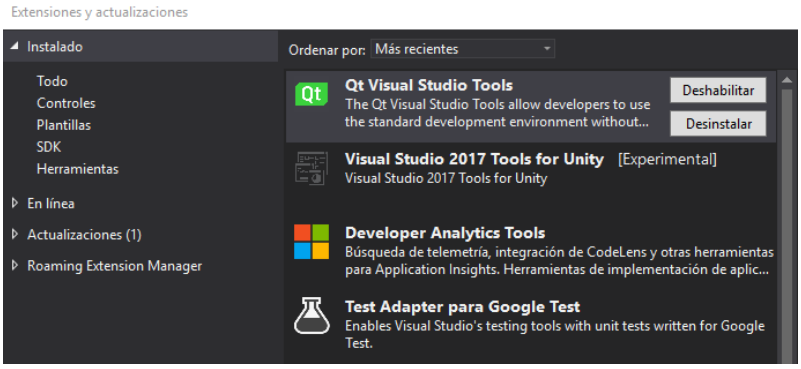


Ilustración 172. Extensiones de Visual Studio, incluyendo la correspondiente a Qt.

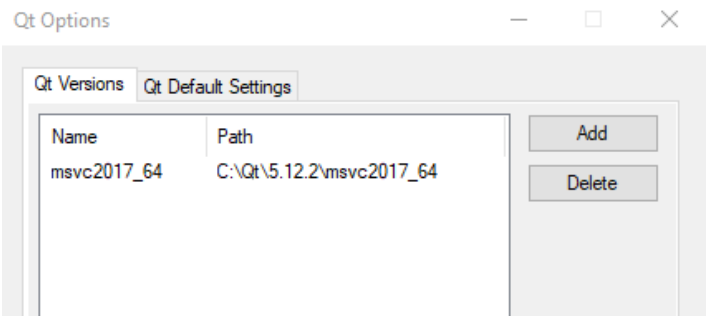


Ilustración 173. Especificación de la versión de Qt y ruta a utilizar.

De esta forma, ya deberíamos estar en condiciones de crear un proyecto de Qt, aunque nosotros no partiremos de cero sino de un proyecto que se entrega con esta documentación.

4. Aunque se ha incluido en el proyecto la librería Exiv2 dado que ocupa poco espacio (alrededor de 11 MB), existen otras de mayor peso que no deben incluirse en el mismo, y este el caso, por ejemplo, de OpenCV. Para evitar extender esta documentación más de lo necesario, se acude al ejecutable de OpenCV en lugar de al proyecto *source*, disponible tanto en Github como en su página, que se puede compilar. Al instalar OpenCV se nos genera una carpeta como *C:/opencv*, de tal forma que la librería estaría lista para su uso. Un problema que presenta OpenCV es que incluye en sus archivos *.dll* la versión del proyecto (que se actualiza frecuentemente), por lo que si en el instante de lectura e instalación de este proyecto no es posible acceder a la versión 4.0, es necesario acudir al siguiente apartado para modificar los parámetros.

5. Este apartado sólo es necesario para comprender cómo se han enlazado las librerías y qué cambios habría que realizar para modificar las mismas, como podría ser el caso de OpenCV en la problemática planteada en la sección anterior. No obstante, el proyecto entregado ya tiene estos parámetros guardados, con lo cual si no es necesario un cambio, este apartado puede omitirse.

   Antes de entrar en el proceso, dado que es mayormente repetitivo, se mostrarán aquellos apartados que habrá que modificar y sus valores asociados. Para ello es necesario entrar en la configuración del proyecto y

escoger el modo *Release* (es con el que hemos trabajado debido a Exiv2, cuya versión sólo ha podido utilizarse en dicho modo):

- C/C++ > General > Directorios de inclusión adicionales. Se debe especificar la carpeta *include* de la librería a introducir.
- Vinculador > Directorios de bibliotecas adicionales. Debe incluirse aquí la carpeta *lib* de la librería a introducir.
- Vinculador > Entrada > Dependencias adicionales. Deben incluirse aquí los archivos *.lib*, tantos como sean necesarios (no tienen por qué ser todos los que aparezcan en la carpeta de la librería).

Se detallan a continuación los archivos y carpetas que deben especificarse para cada apartado anterior:

- OpenCV:
  - *include: C:\opencv\build\include*
  - *lib: C:\opencv\build\x64\vc15\lib*
  - *Archivos .lib: opencv_world400d.lib* (aquí entra en juego la versión).
- Exiv2:
  - *include: Libraries\Exiv2\include* (Ruta relativa)
  - *lib: Libraries\Exiv2\lib* (Ruta relativa)
  - *Archivos exiv2.lib, xmp.lib*

6. Para poder ejecutar la aplicación es necesario añadir los archivos *.dll* a la carpeta correspondiente, que en este caso, tal y como se ha expuesto en la configuración de Visual Studio, será la carpeta *x64/Release* en la raíz del proyecto. Los archivos *.dll* de las librerías referenciadas son *opencv_world400.dll* (si añade una d al final es para el modo *Debug*), y *exiv2.dll*. Para el archivo de OpenCV hay que explorar su carpeta de instalación, y más concretamente la carpeta *opencv/build/x64/vc15/bin*. El archivo de Exiv2 ya viene incluido en la librería que se almacena dentro del proyecto (*ControlPlantacion/ControlPlantacion/Libraries/Exiv2*).

6.2 - Manuales de usuario

Dado que el producto final de este proyecto incluye un conjunto de utilidades accesibles a través de la interfaz, aquí se presentan todas las herramientas al alcance del usuario y una guía de uso. Comentaremos cada una de las opciones posibles basándonos en el orden en el que se presentan los submenús en la interfaz, tal y cómo se muestra en los *storyboards* representados durante el apartado de desarrollo iterativo:

1. Utilidades de archivos (***File***): herramientas para habilitar el almacenamiento y la apertura de archivos en la aplicación.

    - ***Open images***: opción destinada a abrir varias imágenes multiespectrales. Nótese como en el caso más extremo se deben abrir hasta cinco imágenes (REG, GRE, RED, NIR y RGB). Además de la restricción expuesta de que sean 5 capas diferentes, se filtra la selección para comprobar si el nombre del fichero se puede asociar a alguna de las imágenes multiespectrales citadas. De esta forma, para asociar una imagen a la capa RED debe contener las palabras clave 'RED' o 'red' en el nombre del fichero, tal y como ocurre cuando se extraen del propio dispositivo. Incluso si se modificara el nombre externo, las imágenes seguirían almacenando su nombre original en los metadatos.

      Al igual que para los siguientes controles de apertura de archivos, se posibilita la selección de diversos formatos de imágenes (TIFF, PNG, JPG, etc).
    - ***Open RGB image***: nos permite abrir una sola captura RGB únicamente con fines de registro de imágenes. Más concretamente, en este caso nos sirve para el registro de la imagen RGB con la térmica. Por lo tanto, si se abre una imagen con esta opción debe aparecer una ventana secundaria que la muestre: en el caso de que ya hubiera una imagen térmica se efectúa el registro, y en caso contrario se muestra sólo la imagen RGB.
    - ***Open thermal image***: utilidad de apertura de imágenes térmicas (bien podría ser una imagen en escala de grises, como sucede en algún

dataset adjunto). De nuevo, sucede lo mismo que antes: si ya existe una imagen RGB se hace el registro, y de lo contrario se muestra sólo la térmica.

- ***Save image***: guarda la imagen que actualmente se muestra en la aplicación en el sistema de ficheros. Se trata de una utilidad de guardado rápido, sin embargo, si el fichero no procedía del sistema sino que ha sido creado dentro de la aplicación, debe especificarse la ruta y equivaldría a la siguiente opción.
- ***Save image as***: herramienta de guardado donde siempre se pregunta por la ruta donde se desea que se almacene la imagen, independientemente de si el fichero ha sido creado en la aplicación o no.
- ***Save GPS data***: nos permite guardar la información georreferenciada de cada punto de un modelo multicapa (conjunto de multiespectrales) en un fichero, donde también se almacena para cada uno los colores asociados a todas las imágenes en ese punto. Cabe destacar que hay dos niveles de precisión a la hora de guardar los datos: información por cada píxel o cada x milímetros, suponiendo que sabemos a cuántos metros corresponde un solo píxel (calculado a partir de la altura estimada) (ver Ilustración 174). En este segundo caso hay que tener en cuenta que no se conoce el color del punto sino que se aproxima mediante una interpolación bilineal.

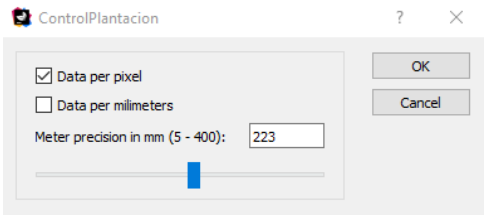


Ilustración 174. Diálogo para seleccionar el nivel de precisión de la información.

Una vez se elige el nivel de precisión, aparece otro diálogo para seleccionar el fichero donde se debe guardar toda la información (como un .txt) en formato *Comma Separated Values* (CSV).

- ***Save centroids info***: ídem que para el apartado anterior, pero ahora sólo se guarda la información de los centros de masas de los contornos extraídos, si los hubiera. Hay que seleccionar de nuevo la ruta destino donde se guarda toda la información.

2. Edición de la imagen (***Edit***): utilidades para modificar el estado inicial de las imágenes multiespectrales, derivado de ciertos defectos encontrados en imágenes de algunos *datasets* (por ejemplo, imágenes rotadas). Sólo se puede aplicar a las imágenes de entrada y no a los resultados obtenidos.
    - ***Rotate left***: rotación de la imagen hacia la izquierda. Sentido horario.
    - ***Rotate right***: rotación de la imagen hacia la derecha. Sentido antihorario.
3. Flexibilidad de visualización de contornos extraídos (***Show***): permite escoger un modo de visualización de los contornos, así como la representación de características adicionales tales como los centros de masas. Por defecto, este submenú está deshabilitado mientras no existe una imagen con sus contornos calculados.
    - ***Center of mass***: habilita o deshabilita el renderizado de los centros de masas de cada contorno hallado al combinar las imágenes multiespectrales.
    - ***Convex hulls***: si se utiliza este control se muestran los contornos con una silueta reducida que se corresponde a la envolvente convexa del polígono original. El objetivo de esta visualización es obtener un contorno aproximado con menos puntos que el perfil inicial, que puede ser muy abrupto.
    - ***Hexagonal hulls***: en lugar de reducir un polígono a su envolvente convexa, se engloba en un hexágono regular. Otro modo de visualización más que reduce la geometría del contorno. A tener en cuenta al cambiar tipos de contornos: son *checkboxes*, si se vuelve a presionar una opción marcada se deshabilita y se muestran los contornos originales.

- ***Show contours in 3D***: propone una alternativa a la visualización de los contornos, pasando del 2D al 3D para consultar los resultados extraídos desde otra perspectiva y poder así manipularlos. Se muestra una ventana secundaria que contiene un *canvas* que se encarga de renderizar los contornos, así como un panel para manipular las formas geométricas derivadas y ocultar o habilitar una forma de renderizado o sus atributos asociados. Véase el panel extraído de la solución final que se muestra en la Ilustración 175.

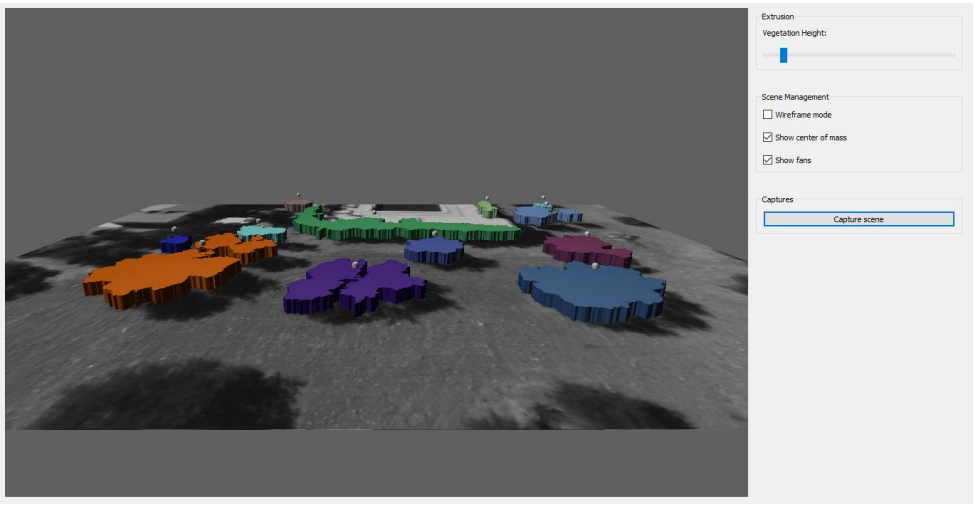


Ilustración 175. Visualización en 3D de los contornos extraídos.

Dentro del panel hay ciertos controles que modifican el estado de la escena:

- Altura de los contornos: se corresponde a una extrusión de la forma geométrica.
- Visualización en modo malla de alambre o malla de triángulos.
- Renderizado de las tapas de los polígonos o no (huecos).
- Renderizado de los centros de masas como esferas sobre el contorno.
- Captura de la escena al doble de resolución a través de un FBO, de tal forma que guarda una imagen con un nombre aleatorio en la carpeta del proyecto.

4. Configuración de datos estáticos de la escena (***Set up data***): nos permite cambiar valores que se aplican a todas las imágenes multiespectrales, aunque para esta aplicación sólo existe un control disponible en este aspecto.

    - ***Flight height***: modifica la distancia estimada al suelo desde el dispositivo de captura, de tal forma que se puede conocer cuántos metros cubre un píxel de la imagen (ver Ilustración 176). Se abrirá un diálogo para elegir la distancia al suelo entre un rango, teniendo en cuenta que no debe ser demasiado bajo, en cuyo caso una imagen debería descartarse porque probablemente pertenece al despegue del dispositivo de vuelo.

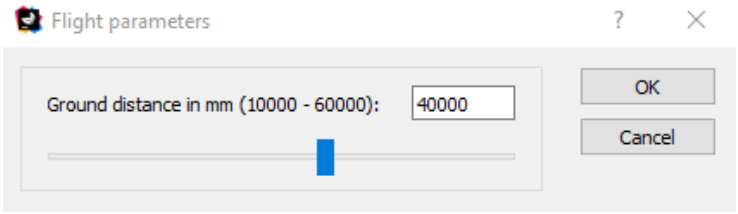


Ilustración 176. Selección de distancia del dispositivo de captura al suelo, en milímetros.

5. Operaciones con imágenes multiespectrales (***Multispectral***): en este submenú se encuentran las principales operaciones que se proponen para solventar los problemas citados al comienzo de este documento.

    - ***Extract vegetation from RED and NIR***: extracción de vegetación a partir de las dos capas, RED y NIR, de tal forma que al mezclarlas de alguna manera se obtiene una nueva pestaña, *Result*, que muestra la imagen RED con los contornos obtenidos dibujados sobre ella. Para alcanzar este estado, y dado que el proceso no es completamente automático, se deben especificar una serie de parámetros, para lo cual se muestra un diálogo con una serie de controles que nos permiten visualizar el resultado en tiempo real con el fin de escoger los mejores valores (ver Ilustración 177).

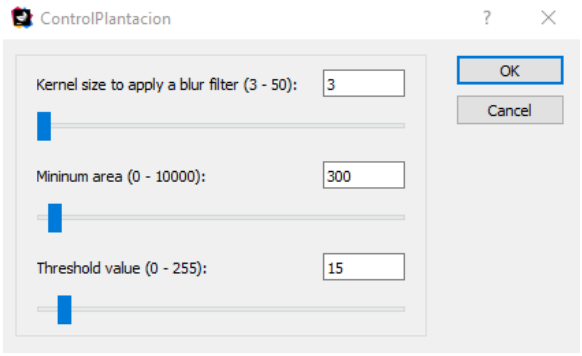


Ilustración 177. Diálogo con los parámetros que podemos modificar para controlar el proceso de extracción de contornos.

El primer parámetro emborrona la imagen, resultando en contornos más simples, pero también implica que la imagen de entrada del algoritmo pierde detalles importantes (por ejemplo, si se difumina la imagen se pueden percibir dos árboles como si estuvieran conexos cuando realmente no es así). El segundo parámetro indica el área mínima que debe tener una forma geométrica para aparecer en el resultado, con el fin de evitar así que se represente vegetación de menor tamaño y que no es de interés para el proyecto. Por último, el umbral de intensidad indica a partir de qué valor de gris se tiene en cuenta un determinado punto de la imagen. Cuanto menor sea el valor, más puntos serán considerados y más impreciso será el resultado.

- ***Undistort images***: rectifica la distorsión de ojo de pez de las imágenes multiespectrales, de tal forma que esta será la base para el resto de operaciones. Nótese como la imagen RGB, en caso de que esta estuviera presente, no sigue un modelo distorsión de ojo de pez, y por lo tanto, debe aparecer un mensaje de error indicando dicha situación.
- ***Register images***: alinea todas las imágenes multiespectrales presentes siempre y cuando exista la capa GRE, ya que esta es la capa maestra respecto de la cual se registran el resto. Incluso la ausencia de otras capas podría originar un registro parcialmente incorrecto, dado que hay dependencias entre ellas (ver Tabla 27). En el caso de la RGB, las diferencias entre esta imagen y una

multiespectral son bastante evidentes, por lo que el registro finaliza pero con un resultado algo peor que para el resto de imágenes. No obstante, es una capa que no presenta utilidad alguna en la aplicación, teniendo en cuenta que la georreferenciación es igualmente imprecisa y se prefiere relacionar una imagen multiespectral con otra RGB más adecuada.

Imagen multiespectral	Dependencias
GRE	-
RED	GRE
REG	GRE
NIR	REG

Tabla 27. Imágenes de las que depende el registro de cada multiespectral.

- ***Remove camera rotation***: el objetivo último de este apartado es posicionar la imagen de tal forma que el norte de la imagen coincida con el norte de la Tierra. En principio, esto no será posible dado que las multiespectrales presentan datos incorrectos, pero el objetivo debería ser este. No sólo elimina este tipo de rotación para todas las imágenes, sino también rotaciones derivadas del vuelo, como la inclinación del propio dron. Sólo está disponible si se han registrado las imágenes.
- ***Register with RGB***: nos permite registrar una imagen multiespectral con una imagen RGB. Para tal fin podemos escoger un fichero del sistema a través de un diálogo, de tal forma que si la multiespectral y la RGB se han podido registrar aparece una ventana secundaria con las dos imágenes solapadas. No es necesaria la rotación previa, teniendo en cuenta que la imagen RGB y la multiespectral se deberían haber capturado en instantes de tiempo cercanos y por lo tanto, la orientación del dron no debería haber variado en gran medida. Será la capa GRE la que se intente registrar con la RGB.

6. Generación del ortomosaico (***Orthomosaic***): nos permite añadir imágenes al ortomosaico en generación o bien eliminarlo.

    - ***Add to orthomosaic***: añade una imagen representativa del conjunto multiespectral (como la RED), con su rotación ya revertida, al ortomosaico en construcción. Una imagen multiespectral no se puede añadir más de una vez al ortomosaico. Conforme se vayan introduciendo imágenes se compone el ortomosaico y se muestra el resultado en cada paso en una ventana secundaria.
    - ***Clean orthomosaic***: elimina el ortomosaico en creación por completo y cierra la ventana oportuna.

6.2.1 - Controles de teclado y ratón

Además de las opciones gráficas, existen otros tantos atajos que nos conducen a la misma operación (ver Tabla 28). Por otro lado, dentro de la sección de esta aplicación destinada a la representación 3D sí que habrá ciertas operaciones que deben hacerse necesariamente con teclado o con ratón sobre el *canvas* (ver Tabla 29).

Controles del apartado de tratamiento de imágenes	
Operación	**Combinaciones**
Abrir / Guardar imágenes y archivos	
Abrir imágenes multiespectrales	Ctrl + O
Abrir imagen RGB	Ctrl + B
Abrir imagen térmica	Ctrl + T
Guardar imagen (modo rápido)	Ctrl + S
Guardar imagen en sistema de ficheros	Ctrl + Shift + S
Guardar datos multiespectrales georreferenciados	Ctrl + P
Guardar datos multiespectrales georreferenciados sólo de centros de masas	Ctrl + Shift + P
Edición de imágenes	
Rotar imagen hacia la derecha	Ctrl + R
Rotar imagen hacia la izquierda	Ctrl + L
Operaciones con contornos extraídos	

Habilitar / deshabilitar dibujado de centros de masas	Ctrl + M
Habilitar / deshabilitar dibujado de envolventes convexas	Ctrl + H
Habilitar / deshabilitar dibujado de celdas hexagonales	Ctrl + Shift + H
Mostrar contornos en 3D	Ctrl + 3
Configuración de datos estáticos	
Mostrar diálogo de configuración de distancia al suelo	Ctrl + G
Procesamiento de imágenes multiespectrales	
Extraer contornos de imágenes multiespectrales	Ctrl + C
Eliminar distorsión de imágenes multiespectrales	Ctrl + U
Registro de imágenes multiespectrales	Ctrl + Shift + R
Eliminar rotación de imágenes	Ctrl + Shift + C
Registrar imagen multiespectral con RGB	Ctrl + Shift + G
Gestión de ortomosaico	
Añadir multiespectral a ortomosaico	Ctrl + A
Eliminar ortomosaico	Ctrl + Shift + O

Tabla 28. Combinaciones de teclado para acceder a las operaciones de tratamiento de imágenes.

Controles de escena 3D	
Operación	**Combinaciones**
Control de cámara	
Arcball (orbit horizontal y vertical)	Presionar botón izquierdo del ratón y moverlo por la pantalla
Rotación alrededor del eje Y de la cámara (Pan)	V o Ctrl + V para rotar en dirección inversa
Rotación alrededor del eje Y de la cámara (Tilt)	B o Ctrl + B para rotar en dirección inversa
Traslación en el eje Z de la cámara (Dolly)	Z o Ctrl + Z para trasladar en la dirección inversa
Traslación positiva en el eje X de la cámara (Truck)	→ (Flecha derecha)
Traslación negativa en el eje X de la cámara (Truck)	← (Flecha izquierda)
Traslación positiva en el eje Y de la cámara (Boom)	↑ (Flecha superior)
Traslación negativa en el eje Y de la cámara (Crane)	↓ (Flecha inferior)

Zoom in	Scroll up (Rueda)
Zoom out	Scroll down (Rueda)
Reset cámara	R
Captura de escena	
Renderizar escena al doble de resolución y guardar captura en fichero	C

Tabla 29. Controles de la cámara y la escena 3D.

6.3 - Publicaciones Científicas

Derivado de este trabajo se ha publicado un artículo, *Multispectral Registration, Undistortion and Tree Detection for Precision Agriculture*, que versa sobre el proceso íntegro que es necesario llevar a cabo sobre las imágenes multiespectrales para extraer la vegetación que en ellas se representa. Dicho proceso incluye las fases de adecuación de estas capturas: eliminación de la distorsión *fisheye* y registro de imágenes.

Este artículo debe formar parte de las actas del Congreso Español de Informática Gráfica 2019[8], celebrado en San Sebastián. El documento se adjunta a la entrega de este trabajo junto a este mismo fichero.

6.4 - Guía original del Trabajo Fin de Título

La guía original de este TFG fue publicada en la web de la EPSJ[9] en el segundo cuatrimestre del curso 2018/2019.

[8] https://ceig2019.vicomtech.org/programa/
[9] http://eps.ujaen.es/TFG/mostrarTFG.php?id=2474

7 - BIBLIOGRAFÍA

Adamchuk, R. G. (2010). Precision Agriculture and Food Security. *Science, 327*, 828-831.

Agencia Tributaria. (2017). *Tabla de coeficientes de amortización lineal.* (Agencia Tributaria) Recuperado el 31 de Mayo de 2019, de https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2017/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml

Apple Developer. (6 de Abril de 2018). *Model-View-Controller*. Obtenido de Apple Developer: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

Athos Agapiou, D. G. (2012). Evaluation of Broadband and Narrowband Vegetation Indices for the Identification of Archaeological Crop Marks. *Remote Sensing, vol. 4, issue 12*, 3182-3919.

Berg-Jürgens, N. L. (2015). Accuracy of Orthomosaic Generated by Different Methods in Example of UAV Platform MUST Q. *IOP Conference Series: Materials Science and Engineering*, 012041.

Capri, A. A. (2018). Chapter One - Modern Agriculture in Europe and the Role of Pesticides. En *Sustainable Use of Chemicals in Agriculture* (págs. 1-22). Elsevier.

CHRobotics. (s.f.). *Understanding Euler Angles*. Obtenido de CHRobotics: http://www.chrobotics.com/library/understanding-euler-angles

Ciarán Hughes, M. G. (2008). Review of Geometric Distortion Compensation in Fish-Eye Cameras. *Review of Geometric Distortion Compensation in Fish-Eye Cameras.* Galway: Irish Signals and Systems Conference.

cprime. (2018). *What is Agile | What is Scrum?* Obtenido de cprime: https://www.cprime.com/resources/what-is-agile-what-is-scrum/

Dhane, P. a. (2012). A Generic Non-linear Method for Fisheye Correction. *International Journal of Computer Applications*, 58-65.

Eberly, D. (16 de Agosto de 2015). *Triangulation by Ear Clipping.* Obtenido de Geometric Tools: https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf

Eric Freeman, E. F. (2009). *Head First Design Patterns.* O'Reilly.

Erickson, J. (2008). *Convex hulls.* Obtenido de University of Illinois: http://jeffe.cs.illinois.edu/teaching/compgeom/notes/01-convexhull.pdf

Erickson, J. (2008). *Convex Hulls.* Obtenido de University of Illinois: http://jeffe.cs.illinois.edu/teaching/compgeom/notes/01-convexhull.pdf

Espejo, V. M. (2008). *Registro de Imágenes mediante Transformaciones Lineales por Trozos.* Málaga: Universidad de Málaga.

Esquivel, S. W. (2007). Calibration of a Multi-camera Rig from Non-overlapping Views. *Esquivel, S., Woelk, F., & Koch, R. (2007). Calibration of a Multi-camera Rig from Non-overlapping Views. DAGM-Symposium.* Germany. doi:10.1007/978-3-540-74936-3_9

Esther Salamí, C. B. (2014). UAV Flight Experiments Applied to the Remote Sensing of Vegetated Areas. *Remote Sens*.

Evangelidis, E. P. (2005). An enhanced correlation-based method for stereo correspondence with subpixel accuracy. *Tenth IEEE International Conference on Computer Vision (ICCV'05)* (págs. 907-912). Beijing: IEEE.

Exiv2. (26 de Abril de 2019). *Metadata reference tables*. Obtenido de Exiv2: https://www.exiv2.org/tags.html

Feng, W. Y. (2015). Greenness identification based on HSV decision tree. *Information Processing in Agriculture*, 149-160.

Fischler, S. T. (1982). Computational Stereo. *ACM Computing Surveys*, 553-572.

Friedl, J. (s.f.). *Jeffrey's Image Metadata Viewer*. Obtenido de Exif Regex: http://exif.regex.info/exif.cgi

Friedl, M. (2018). 6.06 - Remote Sensing of Croplands. En *Comprehensive Remote Sensing* (págs. 78 - 95). Oxford: Elsevier.

Fuertes García, J. M. (27 de Febrero de 2019). *Tema 3: Estereoscopía y Generación del Par Estéreo.* Obtenido de Docencia Virtual: https://dv.ujaen.es/goto_docencia_file_967059_download.html

Georgios D. Evangelidis, E. Z. (2008). Parametric Image Alignment Using Enhanced Correlation Coefficient Maximization. *IEEE Transactions on Pattern Analysys and Machine Intelligence*, 1858–1865.

Github. (2019). *Choose an open source license*. Obtenido de Choose a license: https://choosealicense.com/

Google. (s.f.). *Google Maps*. Obtenido de Google Maps: https://www.google.com/maps

Grupo Acre. (1 de Junio de 2019). *Grupo Acre: Home*. Obtenido de Grupo Acre: https://grupoacre.es/

Hemerly, E. M. (2014). Automatic Georeferencing of Images Acquired by UAV's. *Elder M. Hemerly*, 347–352.

Hong, K. (2018). *DESIGN PATTERNS - OBSERVER PATTERN*. Obtenido de BotoToBogo: https://www.bogotobogo.com/DesignPatterns/observer.php

Huete, P. S. (1990). *Hyperspectral indices and image classifications for agriculture and vegetation.* CRC Press.

Humboldt State University. (2018). *Spectral reflectance*. Obtenido de Humboldt State Geospatial Online:

http://gsp.humboldt.edu/OLM/Courses/GSP_216_Online/lesson2-1/reflectance.html

J. Michael Fitzpatrick, D. L. (2000). Image registration. En D. L. J. Michael Fitzpatrick, *Medical Imaging* (págs. 449-505). Washington: SPIE.

J. P. Jhan, J. Y. (2017). Investigation of parallax issues for multi-lens multispectral camera band co-registration. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Ssciences - ISPRS Archives 42*, 157-163.

Joint Research Centre (JRC) of the European Commission. (2014). Precision agriculture: an opportunity for EU farmers - Potential Support with the cap 2014-2020. Bruselas: Departamento de Políticas de la Unión Europea.

Juliane Bendig, K. Y. (2015). Combining UAV-based plant height from crop surface models, visible, and near infrared vegetation indices for biomass monitoring in barley. *International Journal of Applied Earth Observation and Geoinformation*, Pages 79-87.

Magnetic North vs Geographic (True) North Pole. (Febrero de 2017). Obtenido de GISGEOGRAPHY: https://gisgeography.com/magnetic-north-vs-geographic-true-pole/

Mallick, S. (1 de Julio de 2015). *Image Alignment (ECC) in OpenCV (C++ / Python).* Obtenido de Learn OpenCV: https://www.learnopencv.com/image-alignment-ecc-in-opencv-c-python/

Mesas-Carrascosa, F. a.-S.-B.-F.-G.-G. (2014). Generating UAV accurate ortho-mosaicked images using a six-band multispectral camera arrangement. *Second International Conference on Robotics and Associated High-Technologies and Equipment for Agriculture and Forestry.* Madrid.

Mesas-Carrascosa, F. a.-S.-B.-F.-G.-G. (2017). Accurate ortho-mosaicked six-band multispectral UAV images as affected by mission planning for precision agriculture proposes. *International Journal of Remote Sensing*, 2161-2176.

MicaSense. (2018). *Active Image Alignment*. Obtenido de Micasense: https://micasense.github.io/imageprocessing/Alignment.html

Ministerio de Agricultura, Pesca y Alimentación. (2019). *SIGPAC*. Obtenido de SIGPAC: http://sigpac.mapa.es/fega/visor/#

Ministerio de Empleo y Seguridad Social. (6 de Marzo de 2018). *XVII Convenio colectivo estatal de empresas de consultoría, y estudios de mercado y de la opinión pública.* Obtenido de Boletín Oficial del Estado: https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf

Montana State University. (s.f.). *Convert Geographic Units*. Obtenido de Montana State University: http://www.rcn.montana.edu/resources/converter.aspx

Nayak, S. (10 de Abril de 2017). *Alpha Blending using OpenCV (C++ / Python)*. Obtenido de Learn OpenCV: https://www.learnopencv.com/alpha-blending-using-opencv-cpp-python/

Nebiker, S. a. (2016). LIGHT-WEIGHT MULTISPECTRAL UAV SENSORS AND THEIR CAPABILITIES FOR PREDICTING GRAIN YIELD AND DETECTING PLANT DISEASES. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 963-970.

Open Source Initiative. (6 de Junio de 2019). *Home page*. Obtenido de Open Source Initiative: https://opensource.org/

OpenCV. (15 de Diciembre de 2015). *Remapping*. Obtenido de OpenCV Documentation: https://docs.opencv.org/3.1.0/d1/da0/tutorial_remap.html

OpenCV. (2019). *About OpenCV*. Obtenido de OpenCV: https://opencv.org/about/

OpenGL. (2019). *OpenGL Overview*. Obtenido de OpenGL: https://www.opengl.org/about/

Ordnance Survey. (2018). *A Guide to Coordinate Systems in Great Britain.* Southampton: Ordnance Survey.

Palmero, C. a. (2016). Multi-modal RGB-Depth-Thermal Human Body Segmentation. *International Journal of Computer Vision*, 217-239.

Panotools. (3 de Mayo de 2017). *Lens correction*. Obtenido de Panotools: https://wiki.panotools.org/Lens_correction_model#Lens_distortion_a.2C_b_.26_c_parameters

Panotools. (4 de Enero de 2019). *Fisheye projection*. Obtenido de Panotools: https://wiki.panotools.org/Fisheye_Projection

Parrot . (25 de Abril de 2017). *Application Note: How to register images from Sequoia singled-band cameras.* Obtenido de Parrot for developers: https://forum.developer.parrot.com/uploads/default/original/2X/7/7b4ae9060b9482975495333c118f22a799fa39db.pdf

Parrot Sequoia. (2019). *Parrot Sequoia: Datos del producto*. Obtenido de Parrot Sequoia: https://www.parrot.com/soluciones-business/profesional/parrot-sequoia

Pix4D. (2018). *How are defined yaw, pitch, roll?* Obtenido de Pix4D: https://support.pix4d.com/hc/en-us/articles/205675256-How-are-defined-yaw-pitch-roll

Pix4D. (2018). *How are the Internal and External Camera Parameters defined?* Obtenido de Pix4D: https://support.pix4d.com/hc/en-us/articles/202559089-How-are-the-Internal-and-External-Camera-Parameters-defined

Pix4D. (2018). *https://www.exiv2.org/tags.html*. Obtenido de Pix4D: https://www.exiv2.org/tags.html

Pix4D. (2018). *Specifications of xmp.camera tags*. Obtenido de Pix4D : https://support.pix4d.com/hc/en-us/articles/360016450032-Specifications-of-xmpcamera-tags

Pix4D. (2019). *Photo stitching vs orthomosaic generation*. Obtenido de Pix4D: https://support.pix4d.com/hc/en-us/articles/202558869-Photo-stitching-vs-orthomosaic-generation

PyOpenGL 3.x. (s.f.). *PyOpenGL 3.x: The Python OpenGL Binding*. Obtenido de PyOpenGL 3.x: http://pyopengl.sourceforge.net/

Qt. (2019). *Qt: Home page*. Obtenido de Qt: https://www.qt.io/

Scrum Methodology. (2017). *Scrum Reference Card.* Obtenido de Scrum Methodology: https://www.collab.net/sites/default/files/uploads/CollabNet_scrumreferencecard.pdf

Sebastian Candiago, F. R. (2015). Evaluating multispectral images and vegetation indices for precision farming applications from uav images. *Remote sensing*, 4026–4047.

Serge Belongie, J. M. (2002). Shape Matching and Object Recognition Using Shape Contexts. *IEE Transactions on pattern analysis and machine intelligence*, 509-522.

Shape Similarity. (s.f.). Obtenido de Umeå University: http://www8.cs.umu.se/kurser/TDBAfl/VT06/algorithms/BOOK/BOOK5/NODE196.HTM

Shen X., X. L. (2014). Multi-modal and multispectral. *European Conference on Computer Vision, ECCV* (págs. 309–324). Zurich: Springer International Publishing.

Shore, S. N. (2008). *Forces in physics: a historical perspective.* London: Greenwood .

Singhal, N. (22 de Abril de 2003). *Algorithms for Structure and Motion in Biology.* Obtenido de Stanford University: http://web.stanford.edu/class/cs273/scribing/2004/class8/scribe8.pdf

Sklansky, J. (1982). Finding the convex hull of a simple polygon. *Pattern Recognition Letters*, 79 - 83.

Snyder, J. P. (1935). *Map projections: A working manual.* Washington: U.S. Government Printing Office.

Snyder, J. P. (1986). *Map Projections Used by the U.S. Geological Survey.* Department of the interior.

Stephens, R. (29 de Julio de 2014). *Centroid of polygon in C#*. Obtenido de C# Helper: http://csharphelper.com/blog/2014/07/find-the-centroid-of-a-polygon-in-c/

Structural Analysis and Shape Descriptors. (Agosto de 2014). Obtenido de OpenCV documentation: https://docs.opencv.org/3.0-alpha/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

Süsstrunk, D. F. (2011). Multispectral interest points for RGB-NIR image registration. *18th IEEE International Conference on Image Processing* (págs. 181-184). Bruselas: IEEE.

ThermalCapture. (2019). *Home page*. Obtenido de ThermalCapture: https://thermalcapture.com/

Tscharntke, T. a. (2012). Combining biodiversity conservation with agricultural intensification. En *Land Use Intensification Effects on Biodiversity and Ways to Mitigate Them* (págs. 7-15). CSIRO Publishing.

University of Tel Aviv. (s.f.). *OpenCV 2.1 Cheat Sheet (C++).* Obtenido de University of Tel Aviv: https://www.cs.tau.ac.il/~hezy/workshop/cheat%20sheet.pdf

Usery, M. L. (2010). *Proyecciones cartográficas y sistemas de referencia.* Obtenido de International Cartographic Assoaciation: https://icaci.org/files/documents/wom/09_IMY_WoM_es.pdf

Vlissides, E. G. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional.

Wei, J. a.-F.-M.-L. (2011). Fisheye Video Correction. *IEEE transactions on visualization and computer graphics*.

Xiaoyong Shen, L. X. (2014). Multi-modal and Multi-spectral Registration for Natural Images. *European Conference on Computer Vision 2014* (págs. 309-324). Zurich: Springer International Publishing .

Yankov, D., Keogh, E., Wei, L., Xi, X., & Hodges, W. (2008). Fast Best-Match Shape Searching in Rotation-Invariant Metric Spaces. *IEEE Transactions on Multimedia*, 230-239.

Yasir R., E. M. (2018). Data-driven multispectral image registration. *15th Conference on Computer and Robot Vision, CRV* (págs. 230–237). Toronto: IEEE.