



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Zepazo

Software para detección de impactos lunares

Autor

Antonio Cuadros Lapresta

Tutor

Dr. Sergio Alonso Burgos



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Julio de 2021



ugr | Universidad
de **Granada**

Zepazo

Software para detección de impactos lunares

Autor

Antonio Cuadros Lapresta

Tutor

Dr. Sergio Alonso Burgos

Zepazo: Software para detección de impactos lunares

Antonio Cuadros Lapresta

Palabras clave: Impactos, Luna, Meteoroides, Detección, Vídeo.

Resumen

Los meteoroides, objetos sólidos que poseen un diámetro máximo de 50 metros viajan por el sistema solar impactando a multitud de cuerpos celestes como la Tierra y la Luna entre otros desde la formación de los mismos. Los meteoroides de forma continuada impactan contra la atmósfera Terrestre y debido a su pequeño tamaño y altas velocidades, muchos de ellos acaban desintegrándose en su paso por la atmósfera siendo muy pocos los que terminen impactando contra la superficie terrestre. En contraposición a la Tierra, la Luna posee una atmósfera muy diferente a la Tierra formada principalmente por elementos como argón y helio formando una atmósfera muy tenue permitiendo que incluso meteoroides y otros de cuerpos espaciales de pequeño tamaño que al entrar en contacto con la atmósfera terrestre se desintegren, puedan impactar contra la superficie lunar y sean visibles en forma de resplandores lumínicos desde la Tierra haciendo uso de telescopios capturando la zona oscura de la Luna.

En este proyecto se pretende implementar un software que nos permita automatizar el proceso de detección de impactos en la superficie lunar mediante el análisis de vídeos de larga duración de zonas no iluminadas de la superficie lunar obtenidos mediante telescopios en con el objetivo de extraer los impactos lunares ocurridos tratando de minimizar los falsos positivos ocasionados normalmente por fenómenos como la oscilación de la grabación, el titilar de otros cuerpos celestes, ruido en la grabación, paso de satélites artificiales, etc.

Zepazo: Moon impact detection software

Antonio Cuadros Lapresta

Keywords: Impacts, Moon, Meteoroid, Detection, Video.

Abstract

Meteoroids are solid objects that travel through the solar system, impacting multitude celestial bodies such as the Earth and the Moon among others. Meteoroids continuously impact the Earth's atmosphere and due to their small size and high speeds, many of them end up disintegrating as they pass through the atmosphere, with very few ending up impacting the Earth's surface. In contrast to the Earth, the Moon has an atmosphere that is very different from the Earth, consisting mainly of elements such as argon and helium, forming a very tenuous atmosphere, allowing even meteoroids and other small space bodies that in contact with the Earth's atmosphere would disintegrate, can impact against the lunar surface and are visible in the form of luminous glows from the earth making use of telescopes capturing non-illuminated areas of the Moon.

The aim of this project is to implement a software that allows us to automate the process of detecting impacts on the lunar surface by analyzing long-term videos of non-illuminated areas of the lunar surface obtained through telescopes in order to extract lunar impacts occurred trying to minimize the false positives normally caused by phenomena such as the oscillation of the recording, the flickering of other celestial bodies, noise in the recording, passage of artificial satellites and additionally provide tools to facilitate the task of analyzing videos such as an impact validator, a tool that helps us when adjusting parameters, etc..

Yo, **Antonio Cuadros Lapresta**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77146794X, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.



Fdo: Antonio Cuadros Lapresta

Granada a Julio de 2021 .

Dr. **Sergio Alonso Burgos**, Profesor del Departamento Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Zepazo, Software para detección de impactos lunares***, ha sido realizado bajo su supervisión por **Antonio Cuadros Lapresta**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a Julio de 2021.

El tutor:



Sergio Alonso Burgos

Agradecimientos

En primer lugar, gracias a mi tutor Sergio por todo el apoyo recibido así como su dedicación durante toda la realización de este proyecto.

A todos y cada uno de los profesores que han contribuido a mi desarrollo académico durante esta etapa.

Agradecer también a mis padres, abuelos y resto de familiares por su interés, apoyo y confianza tanto en mí como en este proyecto.

A mis compañeros, que me han acompañado durante todo el camino y han supuesto un pilar muy importante en el mismo.

Índice general

1. Introducción	11
1.1. Introducción	11
1.2. Descripción del problema	13
1.2.1. Detección de impactos	14
1.3. Estado del arte	16
1.3.1. Primeros estudios acerca de los impactos en la superficie lunar	16
1.3.2. Proyectos recientes y actuales	17
1.4. Objetivos	19
1.4.1. Diagrama de objetivos	27
1.5. Asignaturas que contribuyen a la realización de este proyecto	28
1.6. Estructura del documento	29
2. Planificación y metodología	31
2.1. Planificación	31
2.1.1. Temporización	31
2.1.2. Presupuesto	43
2.2. Metodología	44
2.2.1. Herramientas	48
3. Análisis y especificación de requisitos	52
3.1. Introducción a la ingeniería de requisitos	52
3.2. Estudio de viabilidad	52
3.3. Obtención de información sobre el dominio del problema y el sistema actual	54
3.3.1. Glosario términos destacados del ámbito del problema	54
3.3.2. Introducción al sistema	55
3.4. Preparación de reuniones de elicitation y negociación	56
3.4.1. Descripción general implicados	56
3.4.2. Descripción perfil implicados	57
3.5. Casos de Uso	58
3.6. Lista estructurada de requisitos	60
3.6.1. Requisitos funcionales	60
3.6.2. Requisitos no funcionales	65
3.6.3. Requisitos de información	66
4. Diseño	67
4.1. Introducción al diseño	67
4.1.1. Prototipo Inicial	67
4.2. Diseño de la aplicación	69

4.2.1. Diseño Zepazo	72
4.2.2. Diseño interfaz selección de parámetros	75
4.2.3. Diseño validador de impactos	79
4.2.4. Diseño cortador de vídeos	80
5. Implementación	81
5.1. Zepazo	81
5.1.1. Clase Impact	83
5.1.2. Clase ImageAnalyzer	84
5.1.3. Clase VideoAnalyzer	99
5.1.4. Clase Dator	108
5.1.5. Clase FSDator	110
5.1.6. Script 'zepazo.py'	111
5.2. ZepazoParams	112
5.2.1. Clase ZepazoParams	113
5.3. ZepazoVerify	126
5.4. ZepazoCrop	128
6. Pruebas	129
6.1. Pruebas unitarias	129
6.1.1. Dockerfile	130
6.1.2. GitHub Action	133
6.1.3. Pruebas	134
6.2. Pruebas de aceptación	148
6.3. Modo debug	153
7. Conclusiones y trabajos futuros	155
A. Manual de usuario	162
A.1. Manual de usuario Zepazo	162
A.1.1. Instalación del conjunto de programas	162
A.1.2. Zepazo	163
A.1.3. ZepazoParams	166
A.1.4. ZepazoVerify	170
A.1.5. ZepazoCrop	171

Índice de figuras

1.1. Fotograma ejemplo de un vídeo a analizar	14
1.2. Fotograma ilustrando un impacto lunar	15
2.1. Diagrama de Gantt	42
2.2. Formulario Reunión	46
2.3. Diagrama metodología	47
3.1. Diagrama de casos de Uso	59
4.1. Diagrama de clases y paquetes prototipo inicial	68
4.2. Diagrama agrupación casos de uso en programas	69
4.3. Diagrama objetivo del diseño	70
4.4. Diagrama Clases Zepazo	73
4.5. Diagrama Paquetes Zepazo	74
4.6. Diagrama Actividad Script zepazo.py	74
4.7. Diseño Interfaz	75
4.8. Diagrama Clases Zepazo Parameter Interface	77
4.9. Diagrama Paquetes Zepazo Parameter Interface	78
4.10. Diagrama Actividad Zepazo Verify	79
4.11. Diagrama Actividad Zepazo Crop	80
5.1. Proceso dilatación píxel	86
5.2. Fotograma detección impactos, falsos positivos en zona iluminada	86
5.3. Mismo fotograma, dilatación aplicada, falsos positivos eliminados	87
5.4. Funcionamiento método moonEnclosingCircle: Ejemplo real. .	91
5.5. Fórmula pertenencia punto a elipse rotada	92
5.6. Funcionamiento método markHits: Ejemplo real.	94
5.7. Funcionamiento método getDifferences.	95
5.8. Eliminación de ruido	98
5.9. Uso de máscaras: Ejemplo práctico	103
5.10. Interfaz programa ZepazoParams	112
5.11. QT 5 Designer, creación Interfaz	114
5.12. Interfaz, jerarquía de elementos visuales	114
5.13. Interfaz, panel superior	118
5.14. Interfaz, panel central	119
5.15. Interfaz, panel inferior	119
5.16. Interfaz, mensaje error	120
5.17. Funcionamiento método adjustEllipse.	121

5.18. Interfaz, elipse calculada automáticamente	122
5.19. Interfaz, máscara añadida	123
5.20. Interfaz, funcionamiento botón mostrar parámetros	124
5.21. Interfaz, mostrando el comando a utilizar	124
6.1. GitHub Commits History	133
A.1. Interfaz programa ZepazoParams sin vídeo seleccionado	166
A.2. Interfaz programa ZepazoParams, vídeo seleccionado	167
A.3. Interfaz programa ZepazoParams, elipse ajustada	168
A.4. Interfaz programa ZepazoParams, máscaras seleccionadas	168
A.5. Interfaz programa ZepazoParams, comando generado	169

Capítulo 1: Introducción

1.1. Introducción

Los meteoroides son objetos sólidos que se encuentran en el espacio interplanetario y que son más grandes que partículas de polvo, pero más pequeños que los asteroides, entre los 100 μm y los 50 m. Los meteoroides apenas son observables en el espacio interplanetario, no obstante, la atmósfera terrestre es uno de los mejores detectores de meteoroides ya que debido a la existencia de la atmósfera estos objetos se desintegran produciendo destellos visibles [1]. La Luna en contraposición a la Tierra posee una atmósfera mucho más tenue formada por elementos como argón, helio, sodio y potasio contando con apenas 100 toneladas de masa siendo catalogada como una SBE (surface-bounded-exosphere) [2] lo que permite que cuerpos que en contacto con la atmósfera terrestre se desintegren y no llegarán a impactar contra su superficie debido a su reducida masa y altas velocidades (entre 20km/s y 70km/s) consigan impactar contra la superficie lunar produciendo destellos visibles desde la tierra gracias a telescopios y generando los cráteres que dotan a la luna de su aspecto tal y como la conocemos hoy en día.

El estudio y caracterización de los meteoroides a partir de los impactos lunares surge debido a la dificultad encontrada en detectar y predecir la posición de los mismos en su paso por la atmósfera terrestre debido a su pequeño tamaño y altas velocidades. Es por ello que actualmente se estudian los impactos lunares, ya que nos proporcionan una forma más sencilla de detectarlos, ya que se manifiestan como destellos luminosos en la superficie y nos permiten obtener aproximaciones de su tamaño, masa y la velocidad a la que impactaron. Uno de los primeros artículos que encontramos relacionados con el estudio de impactos lunares es 'Remote visual detection of impacts on the lunar surface (975-976), Melosh' [5] donde se expone la idea de que aquellos meteoroides que entrasen en la atmósfera terrestre y se desintegrasen debido a su tamaño, en la luna serían visibles los impactos producidos desde la tierra utilizando un simple telescopio.

De esta forma, la detección y estudio de los impactos de meteoroides en la Luna es fundamental para ampliar el conocimiento sobre las poblaciones de meteoroides cercanas a la Tierra y comprender el sistema solar complementando a técnicas como la observación, el uso de radares y el estudio de tasas de formación de cráteres en la Luna [3].

Adicionalmente, el estudio de estos fenómenos sobre la superficie lunar nos permite conocer la frecuencia de los impactos así como la distribución de los mismos sobre la propia superficie aportando un conocimiento muy importante de cara a las futuras misiones espaciales relacionadas con la Luna como es el caso de la misión Artemis dirigida principalmente por la Nasa cuyo objetivo es volver de nuevo a la Luna para establecer presencia humana de forma prolongada tanto en superficie como en órbita.

En este proyecto se pretende aportar una herramienta que permita obtener a partir de una grabación de zonas no iluminadas de la Luna los posibles impactos ocurridos, de esta forma se pretende crear un software que permita analizar vídeos de zonas no iluminadas de la Luna y que nos permita conocer si han tenido lugar durante la grabación impactos de meteoroides extrayendo los fotogramas donde han tenido lugar así como un área estimada donde han podido tener lugar los impactos. Otro objetivo igual de importante para este proyecto es intentar minimizar los falsos positivos, esto es en nuestro caso indicar que ha tenido lugar un impacto cuando en realidad no ha sido así. Los falsos positivos pueden darse debido a multitud de factores, desde pequeños defectos en la grabación como pueden ser oscilaciones hasta destellos producidos por estrellas que aparecen en la propia grabación.

La realización de este software surge gracias a una propuesta de Dr. Sergio Alonso Burgos, tutor de este proyecto, el cual ha colaborado en diversas ocasiones con miembros del IAA (Instituto de Astrofísica de Andalucía) aportando diversos prototipos previos de un software de detección de impactos lunares facilitando su tarea investigadora. Así en este proyecto se partirá de uno de los prototipos con el objetivo de mejorar la detección de los impactos, disminuir el número de falsos positivos así como aumentar su funcionalidad tal y como se describirá en secciones posteriores.

1.2. Descripción del problema

El problema que se busca resolver consiste en proponer una alternativa a la identificación de impactos lunares de forma manual basada en el análisis de grabaciones de la Luna ya que debido a las grandes cantidades de vídeo que se llegan a obtener y a la dificultad para su detección, ya que la duración del evento luminoso producido por un impacto puede durar décimas de segundo y pasar desapercibido para el ojo humano, suponen problemas para las personas y es necesaria la automatización de dicha tarea. En consecuencia, este proyecto busca aportar una solución software que permita analizar grabaciones de zonas no iluminadas de la Luna de forma automatizada con el objetivo de detectar impactos de meteoroides sobre la superficie lunar.

Adicionalmente, este software pretende dar solución a un problema presente entre el personal del IAA (Instituto de Astrofísica de Andalucía) los cuales trabajan con un software bastante anticuado encargado de analizar vídeos en busca de impactos. Este software con el que trabajan cuenta con problemas como que únicamente les permite trabajar con vídeos en un formato propietario y poco común en la actualidad (utilizado por cámaras antiguas) y que debido a la gran cantidad de tiempo que lleva crear un nuevo software y su falta de tiempo, no les es posible embarcarse en el desarrollo de un programa como el que se presenta en este documento.

De esta forma, este problema se ubica en el contexto del tratamiento, análisis y procesado de archivos multimedia digitales ya que el objetivo principal de este proyecto es la detección de impactos lunares a partir de grabaciones de la zona no iluminada de la luna. De esta forma para resolver el problema planteado será necesario adentrarnos en el ámbito del análisis y procesado de archivos multimedia principalmente vídeo e imagen digital.

Para resolver este problema se plantea la construcción de un software que sea capaz de analizar, tratar y procesar grabaciones de la zona oscura de la luna aportando como resultado información relacionada con los posibles impactos de meteoroides, como por ejemplo si han tenido lugar impactos, cuántos, en qué zona de la Luna aproximadamente y en qué momento de la grabación para su posterior análisis por parte de expertos. En esta parte es necesario y muy importante tener en cuenta que pequeñas oscilaciones, destellos de otros cuerpos y el propio movimiento de la Luna pueden causar la aparición de falsos positivos y será necesario tratar estos casos para evitarlos.

1.2.1. Detección de impactos

La detección de impactos como ya se ha mencionado es la tarea principal de este proyecto, para ello partiremos de vídeos donde se aprecia la superficie lunar. En dichos vídeos encontraremos un conjunto de características comunes que serán importantes tener en cuenta:

- Se muestran zonas no iluminadas de la Luna, aspecto importante para la detección de impactos ya que la energía liberada por los impactos hace que el haz de luz producido sea únicamente visible en zonas no iluminadas y no sea posible su detección en zonas iluminadas
- Normalmente no aparecen zonas iluminadas de la Luna, pero podrían aparecer zonas iluminadas y en caso de aparecer será necesario su tratamiento
- Normalmente en los vídeos se muestra una zona no completa de la cara visible de la Luna, de esta forma la Luna puede quedar recortada por el ángulo de captura de la cámara, no obstante, el programa deberá poder trabajar con vídeos donde se muestre toda la superficie lunar

De esta forma podemos encontrar vídeos donde sus fotogramas sean como el siguiente:

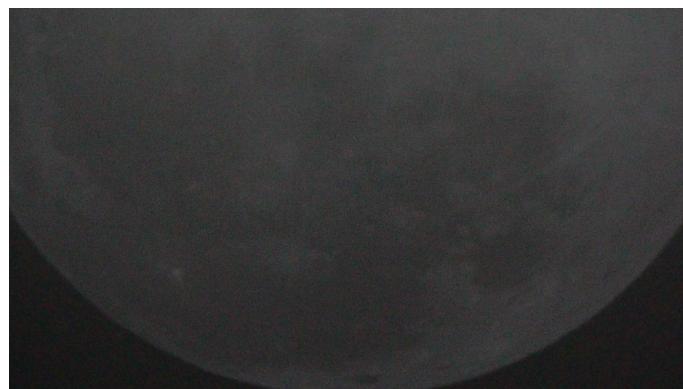


Figura 1.1: Fotograma ejemplo de un vídeo a analizar

Tal y como se ha mencionado anteriormente, encontramos en el fotograma anterior la superficie lunar no iluminada y se puede observar que no nos encontramos ante una grabación donde se muestre la superficie completa, sino que en este caso encontramos en torno a un 50 % de la superficie capturada en cámara, por último, observamos que en este caso no aparece ninguna zona iluminada, en el apartado de implementación se mostrarán ejemplos de vídeos donde aparecen zonas iluminadas y los problemas que pueden suponer así como la solución propuesta.

De esta forma nuestro programa aceptará vídeos con las anteriores características y realizará un proceso de análisis donde se analizarán los diversos fotogramas que forman un determinado vídeo en busca de los destellos producidos por los impactos de meteoroides sobre la superficie lunar. Un ejemplo de como se visualizaría un impacto es el siguiente obtenido del repertorio de impactos analizados por el proyecto NEOLITA [9]:

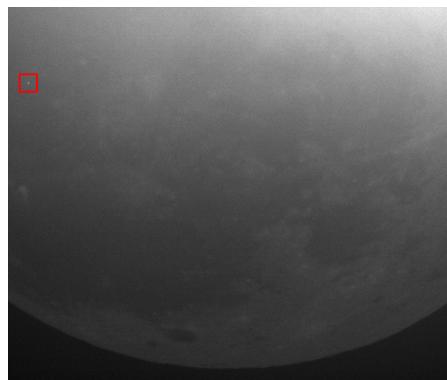


Figura 1.2: Fotograma ilustrando un impacto lunar

1.3. Estado del arte

1.3.1. Primeros estudios acerca de los impactos en la superficie lunar

La detección, caracterización y estudio de impactos sobre la superficie lunar no es una tarea nueva, desde hace más de una década una gran cantidad de investigadores se han volcado en este ámbito. Los primeros estudios relacionados con la detección de impactos en las zonas no iluminadas de la luna surgen en el año 1999 con artículos como por ejemplo el artículo publicado por Dr. J. L. Ortiz 'A search for meteoritic flashes on the Moon' [4] donde se presentan los resultados de una búsqueda de destellos en la superficie lunar relacionados con impactos de meteoroides y presentan el método utilizado para poder encontrar dichos impactos por computador haciendo uso de la resta de fotogramas consecutivos y correcciones a la imagen para intentar evitar los principales problemas encontrados en la detección de impactos como son movimientos en la imagen, estrellas o cambios en las condiciones atmosféricas que puede producir cambios en el fondo de la imagen que daban lugar a falsos positivos. Dr. J. L. Ortiz adicionalmente ha publicado diversos artículos relacionados con la detección y estudio de impactos lunares posteriormente como por ejemplo 'Optical detection of meteoroidal impacts on the Moon' [3] donde se presenta un estudio de los impactos producidos en la zona oscura de la Luna en noviembre de 1999 relacionados con una lluvia de meteoroides ocurrida ese mes y se presenta una técnica para validar los impactos ocurridos mediante la observación de los mismos por diferentes observadores de tal forma que si un impacto ha tenido lugar en un momento temporal concreto y ha sido capturado por más de un observador será validado. La validación de impactos es una tarea fundamental ya que como se ha mencionado, pueden llegar a detectarse una gran cantidad de impactos que sean fruto de falsos positivos originados incluso por las herramientas utilizadas como son la aparición píxeles calientes en las imágenes, es por ello que todos los proyectos relacionados con el estudio de la detección de impactos lunares veremos que usarán un sistema de verificación de impactos basado principalmente en contemplar como posible impacto aquel el cual ha sido capturado por al menos dos observadores simultáneos. Debido a esta cuestión, cobra una gran importancia el contar con sistemas que sean capaces de proporcionar de forma exacta el momento temporal de cada fotograma de la grabación para poder contrastar de forma exitosa las grabaciones obtenidas en busca de impactos en momentos temporales similares pudiendo hacer uso por ejemplo de sistemas GPS que son capaces de proporcionar de forma prácticamente exacta el momento temporal de una determinada grabación.

1.3.2. Proyectos recientes y actuales

NEOLITA

El proyecto NEOLITA cuyas siglas en español son 'Impactos y Transitorios Ópticos en la Luna de Objetos cercanos a la Tierra' es un proyecto de la ESA (Agencia Espacial Europea) que se inició en el año 2017 en los observatorios de Kryoneri y Atenas en Grecia, cuyo objetivo principal es el estudio de la distribución y frecuencia de pequeños objetos llamados NEOs (near-earth objects) mediante la observación y monitorización de los impactos lunares y cuya primera fase ha concluido en enero de este año 2021. En estos años que ha estado activo el proyecto se han conseguido detectar un total de 113 impactos validados y 48 adicionales sin validar sumando un total de 161 posibles impactos capturados en la superficie lunar.

Este proyecto como se ha mencionado se desarrolló simultáneamente a través de dos observatorios, el observatorio de Kryoneri y el de Atenas ambos ubicados en Grecia. Este proyecto ha conseguido automatizar la detección de impactos lunares a través de una arquitectura software/hardware consistente un total de cuatro sistemas principales divididos entre ambos observatorios previamente mencionados, dos de estos sistemas están ubicados en el observatorio de Kryoneri y son encargados de la observación y detección, y los dos restantes se ubican en el observatorio de Atenas y son los encargados del archivado y publicación de resultados.

El observatorio de Kryoneri cuenta con un sistema GPS que tal y como mencionamos anteriormente en estudios previos, es utilizado para poder ubicar temporalmente los impactos con alta precisión. Adicionalmente encontramos el propio telescopio y dos cámaras especiales de fotogramas rápidos que permiten realizar una primera validación los impactos, de tal forma que un impacto pasa esta primera validación si se ha conseguido captar por ambas cámaras en el mismo momento temporal determinado por una antena GPS que proporciona información a un servidor de tiempo GPS. Posteriormente un experto estudiará las imágenes con el objetivo de terminar de validar los impactos y determinar si efectivamente se ha producido o no un impacto sobre la superficie lunar [7].

De esta forma, de los 161 posibles impactos detectados, se han conseguido validar 113 gracias a que fueron detectados por ambas cámaras en el observatorio de Kryoneri y posteriormente revisados por expertos y no se han conseguido validar un total de 48 impactos debido a que o bien no fueron registrados por ambas cámaras o aunque fueron registrados, no fueron validados por expertos.

MIDAS

Por otra parte encontramos el proyecto MIDAS (Sistema de Detección y Análisis de Impactos Lunares), llevado a cabo por Dr. J. M. Madiedo y Dr. J. L. Ortiz del IAA continuando sus trabajos iniciados en 1999 citados previamente relacionados con la detección de impactos sobre la superficie lunar. Este proyecto conocido como MIDAS busca la monitorización sistemática de la zona oscura de la luna a través de diversos telescopios coordinados entre sí con el objetivo de detectar los destellos producidos por los impactos de meteoroides así como poder caracterizarlos a través de parámetros como el tamaño del cráter producido, eficiencia lumínosa producida, ubicación del cráter entre otros.

Adicionalmente continúan trabajando en la construcción y mejora del software conocido como MIDAS que permite la captura y análisis de tanto vídeo en tiempo real como de grabaciones previas con el objetivo de detectar y confirmar impactos lunares. Esta herramienta desarrollada en C++ por Dr. J. M. Madiedo pretende detectar impactos lunares capturados por telescopios de forma automatizada. De esta forma al igual que ocurría en el proyecto NEOLITA, se intenta validar los posibles impactos encontrados haciendo uso de al menos dos sistemas ubicados en los observatorios de Sevilla y La Hita.

Este sistema cuenta con un gran conjunto de opciones como análisis de los vídeos, fotometría para ubicar los impactos, cálculo de ciertos parámetros relacionados con los impactos como el cálculo del tamaño del cráter producido, velocidad estimada del meteorito, su masa entre otros datos, no obstante también cuenta con ciertas limitaciones como la eficiencia del propio programa así como que requiere que la zona iluminada como mucho sea de un 50 o 60% para evitar la aparición excesiva de falsos positivos [8].

Como hemos podido observar, la detección de impactos lunares es una tarea que lleva activa en la comunidad científica desde hace varias décadas con el objetivo de ser capaces de detectar de forma cada vez más precisa, automatizada y con menor esfuerzo. Debido a esto y gracias a las bases sentadas por Dr. J. L. Ortiz con sus primeros estudios, la detección de impactos lunares ha ido cobrando gran importancia entre la comunidad científica gracias además a tanto el proyecto NEOLITA como MIDAS los cuales han conseguido automatizar gran parte del proceso de detección de impactos lunares dejando atrás las largas horas de análisis de los investigadores.

1.4. Objetivos

Este proyecto pretende abordar el desarrollo de un software que permita, de forma automatizada y sistemática, la detección de impactos lunares a partir de vídeos de las zonas no iluminadas de la luna, tomados haciendo uso de telescopios, tratando de minimizar falsos positivos derivados de fenómenos como la oscilación de los dispositivos de captura de vídeo empleados, el destello de otros cuerpos celestes que puedan aparecer en la grabación, [el ruido de los dispositivos de captura](#) y los destellos que provocan los [rayos cósmicos](#).

OBJ 0	Detección de impactos
Descripción	El sistema deberá ser capaz de detectar los impactos de meteoroides sobre la superficie lunar en una determinada grabación.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 1	Carga de vídeos
Descripción	El sistema deberá permitir la carga de vídeos de larga duración.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 1.1	Diferentes formatos vídeo
Descripción	El sistema deberá permitir la carga de vídeos con formatos estándar de vídeo.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 1.1.1	Informe error vídeo
Descripción	El sistema deberá informar de errores relacionados con la carga de vídeos de formatos no permitidos.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 2	Sistema parametrizado
Descripción	El sistema deberá estar parametrizado para poder ajustarse a las diferentes condiciones de las grabaciones.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 2.1	Ajuste al brillo del vídeo
Descripción	El sistema permitirá ajustar un parámetro para ajustarse a las condiciones de brillo del vídeo.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 2.2	Ajuste zonas iluminadas
Descripción	El sistema permitirá ajustar un parámetro para ajustarse a las condiciones del vídeo cuando aparecen zonas iluminadas.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 2.3	Ajuste superficie lunar
Descripción	El sistema permitirá ajustar un parámetro para ajustarse a las condiciones del vídeo y poder detectar qué parte del vídeo es Luna y que parte no.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 3	Falsos positivos
Descripción	El sistema deberá minimizar en la medida de lo posible el número de falsos positivos.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 3.1	Falsos positivos fuera Luna
Descripción	El sistema detectará la posición luna en la escena para descartar impactos producidos en el exterior del área de la luna.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 3.2	Falsos positivos zonas iluminadas
Descripción	El sistema contará con un parámetro para evitar falsos positivos derivados de los cambios de brillo de las zonas iluminadas.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 4	Almacenamiento fotogramas
Descripción	El sistema almacenará los fotogramas donde han sido detectados impactos.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 4.1	Selección carpeta guardado fotogramas
Descripción	El sistema deberá proporcionar la opción de seleccionar la carpeta donde se desean guardar los resultados.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 4.1.1	Ubicación por defecto guardado fotogramas
Descripción	El sistema por defecto, si no se indica ninguna ubicación almacenará los resultados por defecto en la misma carpeta donde se encuentra el vídeo que se está analizando.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 4.2	Almacenamiento fotogramas previos y posteriores
Descripción	El sistema deberá dar la opción al usuario de almacenar varios fotogramas antes y después del impacto.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 4.2.1	Número fotogramas almacenar por defecto
Descripción	El sistema por defecto almacenará únicamente el fotograma del impacto.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 4.3	Almacenamiento fotograma marcado y normal
Descripción	El sistema deberá poder almacenar tanto el fotograma original como otro con una marca donde se indique la ubicación del impacto detectado.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 4.4	Nombre almacenamiento fotogramas
Descripción	El sistema deberá dotar de un nombre significativo a los fotogramas de tal manera que se pueda identificar de que vídeo han sido extraídos, el número de impacto, así como si se ha elegido guardar fotogramas anteriores y posteriores al impacto se deberá indicar que dichos fotogramas son relativos a un impacto determinado.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 5	Previsualización proceso análisis
Descripción	El sistema deberá dar la posibilidad de previsualizar el proceso de análisis del vídeo.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 5.1	Previsualización contorno lunar
Descripción	El sistema deberá mostrar en caso de detectarse el contorno lunar, el trazo que lo recubre.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 5.2	Previsualización impactos
Descripción	El sistema deberá mostrar en el momento que se produzca un impacto un cuadrado marcando el impacto.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 5.3	Terminar previsualización
Descripción	El sistema deberá dar la posibilidad de abortar el proceso de visualización previo.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 6	Configuración a través de archivo
Descripción	El sistema deberá poder cargar parámetros a través de un fichero.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 7	Interfaz ajuste parámetros
Descripción	El sistema deberá contar con una interfaz sencilla para la fase de preprocesado que permita ajustar los parámetros de forma sencilla.
Importancia	Media
Obligatoriedad	Opcional

OBJ 7.1	Exportación parámetros interfaz a fichero
Descripción	La interfaz de selección de parámetros deberá dar la posibilidad de exportar los parámetros a un fichero.
Importancia	Media
Obligatoriedad	Opcional

OBJ 7.2	Importación parámetros a interfaz desde fichero
Descripción	La interfaz de selección de parámetros deberá dar la posibilidad de importar los parámetros desde un fichero.
Importancia	Media
Obligatoriedad	Opcional

OBJ 8	Uso de máscaras
Descripción	El sistema deberá dar la posibilidad de añadir máscaras al vídeo.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 8.1	Máscaras a partir de coordenadas
Descripción	El sistema dará la posibilidad de añadir máscaras a partir de coordenadas.
Importancia	Alta
Obligatoriedad	Obligatorio

OBJ 9	Ubicación impactos
Descripción	El sistema deberá dar la posibilidad de ubicar en la superficie lunar el impacto de forma aproximada.
Importancia	Media
Obligatoriedad	Opcional

OBJ 10	Resumen impactos
Descripción	El sistema deberá mostrar un resumen de impactos al terminar el procesamiento del vídeo.
Importancia	Media
Obligatoriedad	Opcional

OBJ 11	Recorte videos
Descripción	El sistema deberá contar con una herramienta que permita recortar videos
Importancia	Media
Obligatoriedad	Opcional

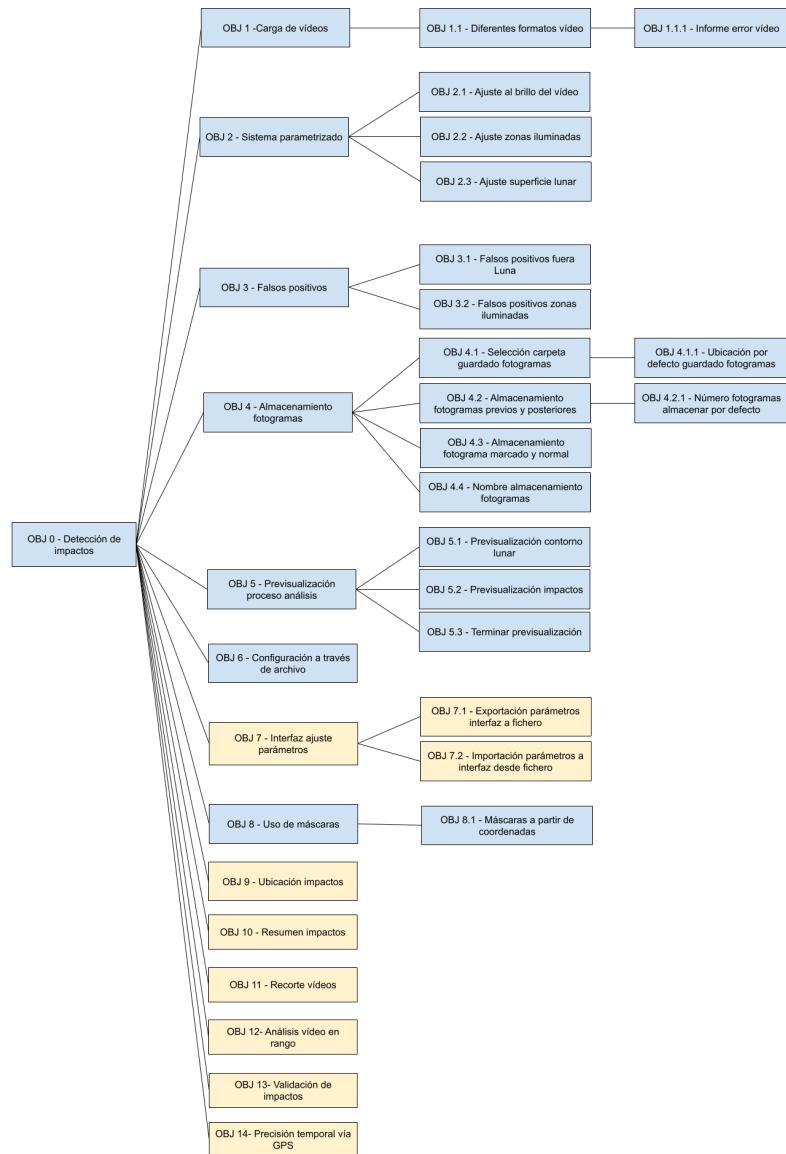
OBJ 12	Análisis video en rango
Descripción	El sistema deberá contar con la posibilidad de analizar videos en un rango determinado de fotogramas.
Importancia	Media
Obligatoriedad	Opcional

OBJ 13	Validación de impactos
Descripción	El sistema deberá contar con la posibilidad de verificar los impactos haciendo uso de dos videos
Importancia	Media
Obligatoriedad	Opcional

OBJ 14	Precisión temporal vía GPS
Descripción	El sistema deberá contar con la posibilidad de tener una mayor precisión temporal a la hora de marcar impactos haciendo uso de un sistema GPS.
Importancia	Baja
Obligatoriedad	Opcional

1.4.1. Diagrama de objetivos

A continuación se muestra de forma esquemática y resumida los objetivos planteados en este apartado mostrando de color azul los objetivos obligatorios y como amarillos los opcionales:



1.5. Asignaturas que contribuyen a la realización de este proyecto

En este apartado se pretende hacer un recorrido a lo largo de la formación recibida a través de las diversas asignaturas cursadas con el objetivo de analizar que asignaturas ayudarán a la realización de este proyecto debido a los contenidos cursados en las mismas. Si bien es cierto, **toda la formación recibida va a ayudar a plantear y realizar este proyecto**, no obstante en este apartado se destacarán aquellas que de forma directa influirán en el desarrollo del mismo.

En primer lugar, encontramos aquellas asignaturas que me han aportado los conocimientos necesarios básicos para iniciarme en la programación y aunque en dichas asignaturas se enseñaran lenguajes de programación como C++, Java o Ruby, me han permitido ser capaz de aprender con facilidad cualquier otro lenguaje. Estas asignaturas son *Fundamentos de la Programación, Metodología de la Programación, y Programación y Diseño Orientado a Objetos*.

En segundo lugar, por el esfuerzo realizado en enseñar el lenguaje de programación Python, aunque las asignaturas no consistiesen en ello, me gustaría destacar asignaturas como *Seguridad y Protección de Sistemas Informáticos y Desarrollo de Aplicaciones para Internet* donde al inicio de las mismas se nos presentó dicho lenguaje y se nos instruyó en el mismo. Destaco la enseñanza de este lenguaje ya que como se verá en secciones posteriores, este será el lenguaje utilizado para la realización de este proyecto.

Asignaturas como Fundamentos de *Ingeniería del Software y Diseño y Desarrollo de Sistemas de Información* influirán de forma directa en el desarrollo de este proyecto ya que en dichas asignaturas se sentaron las bases del proceso de obtención de requisitos, diseño y desarrollo software, aspectos muy importantes para diseñar este proyecto así como para exponer el funcionamiento del software a desarrollar mediante diagramas fácilmente entendibles por otro programador interesado en el proyecto.

Por último una asignatura cuyos contenidos están directamente relacionados con lo que se va a realizar en este proyecto es *Sistemas Multimedia* ya que en dicha asignatura se aprendió a tratar con contenido multimedia tal como imágenes o vídeo, aspecto que será fundamental para este proyecto donde necesitaremos tratar con grabaciones de la luna y para poder analizar dicho vídeo trabajaremos sobre una unidad más sencilla como son los fotogramas, es decir imágenes, donde podré aplicar todo el conocimiento adquirido acerca de imágenes digitales en dicha asignatura y su tratamiento pero enfocado en este caso a detectar los posibles impactos.

1.6. Estructura del documento

El documento que se presenta en este escrito pretende describir el proceso de investigación y desarrollo llevado a cabo para desarrollar la aplicación “Zepazo”. Por ello este documento se estructura en un total de siete capítulos.

El primer capítulo, titulado “**Introducción**” pretende servir como introducción al proyecto incluyendo una primera descripción del ámbito donde se ubica el problema que se desea resolver mediante los apartados “1.1 Introducción” y “1.2 Descripción del problema”. Adicionalmente se expone un pequeño estudio realizado de proyectos relacionados con el que se va a desarrollar en el apartado “1.3 Estado del Arte”. En siguiente lugar se exponen los objetivos del proyectos junto a las asignaturas que notablemente han contribuido a realizar este proyecto en los apartados “1.4 Objetivos” y “1.5 Asignaturas que han contribuido a la realización de este proyecto”.

En el segundo capítulo del documento “**Planificación y metodología**” nos centraremos en indicar la temporización inicial y el presupuesto inicial del proyecto en el apartado “2.1 Planificación” y en un segundo apartado denominado “2.2 Metodología” se explicará la metodología a seguir prevista inicialmente para el proyecto así como las herramientas que se van a utilizar para su desarrollo.

El tercero capítulo, denominado “**Análisis y especificación de requisitos**” está destinada a conocer más de cerca el problema que se va a resolver realizando un primer estudio de viabilidad en el apartado “3.2 Estudio de viabilidad”, posteriormente se obtiene más conocimiento del problema a resolver mediante un pequeño glosario de términos y una introducción al sistema en el apartado “3.3 Obtención de información sobre el dominio del problema y el sistema actual”. A continuación encontramos un apartado llamado “3.4 Preparación de reuniones de elicitation y negociación” donde se muestra una descripción de los implicados del sistema y un apartado llamado “3.5 Casos de Uso” donde se presentan los distintos casos de uso del sistema que dará lugar a lo comentado en el apartado posterior “3.6 Lista estructurada de requisitos” donde se exponen los diversos requisitos del sistema a desarrollar, tanto funcionales, no funcionales como de información.

El cuarto capítulo, “**Diseño**” pretende ilustrar el proceso de diseño de este software que genera como resultado un total de cuatro programas dedicados cada uno de ellos a un tema concreto como pueden ser la detección de impactos, ajuste asistido de parámetros para el análisis, validación de impactos y recorte de vídeos.

En el quinto capítulo titulado “**Implementación**” se mostrarán los detalles de la implementación de cada una de las clases con sus respectivos métodos desarrollados con el propósito de cumplir cada uno de los objetivos y requisitos planteados.

El sexto capítulo denominado “**Pruebas**” pretende ilustrar el conjunto de pruebas realizadas sobre cada uno de los programas desarrollados. De esta forma, se detallarán tanto aquellas pruebas automatizadas realizadas, las pruebas unitarias describiendo cada una de ellas y su proceso de automatización como pruebas manuales realizadas, las pruebas de aceptación donde se pondrán de manifiesto las reuniones realizadas con el cliente y el feedback obtenido. Por último se mostrará el modo Debug de la aplicación principal, Zepazo, que nos ha permitido comprobar en todo momento el correcto funcionamiento del programa.

Como último capítulo encontramos “**Conclusiones y trabajos futuros**” donde se reflexionará acerca de aspectos como la planificación inicial, objetivos conseguidos, aprendizaje, imprevistos y reflexiones relacionadas con el proyecto. Adicionalmente encontraremos una reflexión donde se analizarán las posibilidades futuras de este software.

Adicionalmente encontramos un apéndice “**Manual de usuario**” que es el manual de usuario que encontramos en nuestro repositorio de GitHub que permitirá aprender a utilizar los programas presentados en este documentos de forma sencilla.

Capítulo 2: Planificación y metodología

En esta sección se muestra la *planificación inicial del proyecto*, la cual puede diferir con la conseguida, no obstante se detallará este aspecto en la sección de conclusiones. Adicionalmente se establecerá un presupuesto inicial así como la metodología a seguir para conseguir desarrollar el software propuesto.

2.1. Planificación

2.1.1. Temporización

Descripción

22/02/2021 - 28/02/2021

Durante esta primera semana se tratarán de decidir las herramientas a utilizar, como son el lenguaje de programación, biblioteca de tratamiento de archivos multimedia, herramienta de tests, herramienta de integración continua, herramienta para generar documentación, gestor de tareas, gestor de dependencias así como herramienta para crear y gestionar contenedores.

Horas a dedicar: 5h (Una horas cada día a lo largo de 5 días)

Objetivos que se cumplen: Se contribuye al avance de todos los objetivos mediante la selección de las herramientas básicas que nos permitirán desarrollar el proyecto.

01/03/2021 - 07/03/2021

Durante esta semana se planteará la estructura del proyecto, definiendo inicialmente las clases necesarias así como su estructura general. De esta forma se definirá la estructura general de carpetas del proyecto, clases correspondientes, se inicializará el gestor de dependencias y sus archivos de configuración, se configurará adicionalmente un generador de documentación del código que nos permita seguir un estándar a la hora de documentar nuestro código fuente.

Horas a dedicar: 5h (Una hora cada día a lo largo de 5 días)

Objetivos que se cumplen: Se contribuye al avance de todos los objetivos definiendo la estructura general del proyecto.

08/03/2021 - 14/03/2021

En esta semana se tratará de empezar a dotar de funcionalidad a nuestro programa, para ello nuestro programa aceptará mediante alguna librería de parámetros. Además se tendrá un primer contacto con la biblioteca de manipulación de vídeo e imagen seleccionada y de esta forma, empezaremos esta semana por aceptar como parámetro un vídeo, y si la ruta es correcta el vídeo se cargará y reproducirá. Se cuidará que la ruta del vídeo es correcta y en caso contrario se informará del error correspondiente al usuario. Adicionalmente se plantea la inclusión del cálculo y almacenamiento de ciertos parámetros relacionados con el vídeo como puede ser su duración, fotogramas por segundo, número total de fotogramas, entre otros que pueden ser útiles más adelante.

Horas a dedicar: 10h (Dos horas cada día a lo largo de 5 días)

Objetivos que se cumplen: Se cumplen los objetivos OBJ1, OBJ1.1, OBJ1.1.1 relacionados con la carga de vídeos y se sientan las bases para desarrollar el objetivo OBJ2 relacionado con tener un sistema parametrizado.

15/03/2021 - 21/03/2021

Durante esta semana se implementará una primera versión de detección de impactos basándonos en la diferencia de fotogramas consecutivos. Se intentará manejar la detección de impactos inicial marcándolos de alguna forma en la imagen y se comenzarán a introducir los tests unitarios para poder hacer pruebas sobre nuestro código. Adicionalmente se creará un contenedor Docker donde poder ejecutar dichos tests. Por último se utilizará algún sistema de integración continua para poder ejecutar los tests cada vez que realizamos un cambio.

Horas a dedicar: 10h (Dos horas cada día a lo largo de 5 días)

Objetivos que se cumplen: Se ha conseguido avanzar en la consecución del objetivo OBJ0, detección de impactos que es el objetivo principal del proyecto.

22/03/2021 - 28/03/2021

Durante esta semana se tratará de impedir que aparezcan posibles falsos positivos en el exterior de la luna, para ello se intentará obtener el contorno de la luna mediante una aproximación circular o elíptica de tal forma que si un impacto se produce en fuera de la zona detectada, es decir, fuera de la elipse o circunferencia, deberá ser descartado ya que es un falso positivo. Además se deberá trabajar en hacer más flexible la detección del contorno de la Luna intentándolo parametrizar su detección para adaptarse a las diferentes circunstancias de los vídeos y en consecuencia obtener mejores resultados. Adicionalmente se mejorará la detección de impactos mediante el cálculo de diferencias de fotogramas parametrizando dicho proceso para adaptarse a las diferentes condiciones de los vídeos. Por último, se añadirá una primera versión de un modo debug que mediante algún parámetro nos permita ver el proceso de análisis del vídeo mostrando en dicho vídeo información de depuración como la elipse o circunferencia en cada momento que recubre la Luna, los impactos interiores y los impactos exteriores descartados de otro color.

Horas a dedicar: 15h (Tres horas cada día a lo largo de 5 días)

Objetivos que se cumplen: Se completará el objetivo OBJ2.1 al añadir un parámetro ajustable en función del brillo al calcular las diferencias y el objetivo OBJ2.3 al añadir un parámetro para ajustar la detección del contorno lunar avanzando el Objetivo OBJ2. Adicionalmente se cumplirá el objetivo 3.1 avanzando el objetivo OBJ3. Avanzará el objetivo OBJ5, cumpliéndose el objetivo OBJ5.1, OBJ5.2.

29/03/2021 - 04/04/2021

Durante esta semana se trabajará en intentar implementar algún mecanismo como la dilatación de fotogramas para evitar que en las zonas iluminadas de la Luna que puedan aparecer en las grabaciones puedan producirse detecciones de impactos dando lugar a falsos positivos. Adicionalmente se trabajará en el guardado de fotogramas. En dicho método podremos tanto guardar un único fotograma como varios fotogramas previos y posteriores según indique el usuario. También se deberá dar la posibilidad al usuario de elegir la carpeta donde guardar dichos fotogramas. Adicionalmente se deberá implementar algún mecanismo que calcule de forma aproximada los valores de los que depende la detección del contorno lunar basándose en los valores del propio vídeo para que el usuario pueda partir de una aproximación y a partir de dicha aproximación pueda ajustarlo con un menor esfuerzo.

Horas a dedicar: 15h (Tres horas cada día a lo largo de 5 días)

Objetivos que se cumplen: Se completarán los objetivos OBJ3.2 ya que se implementará la dilatación de fotogramas para evitar falsos positivos derivados de la aparición de zonas iluminadas, completando el objetivo OBJ3. Adicionalmente avanzará el objetivo OBJ2 cumpliendo el objetivo OBJ2.2. Por último se pretende mejorar el objetivo OBJ3.1 añadiendo una forma automática de calcular los valores de los que depende el cálculo del contorno lunar. Por último se completarán los objetivos relacionados el guardado de fotogramas OBJ4 como son OBJ4.1, OBJ4.1.1 relacionados con la carpeta donde guardar dichos fotogramas, OBJ4.2 y OBJ4.2.1 relacionados con el número de fotogramas a guardar antes y después de un impacto, OBJ4.3 y OBJ4.4.

05/04/2021 - 11/04/2021

Durante esta semana se trabajará en la incorporación de máscaras a nuestros vídeos para tapar ciertas zonas que interesen tapar como por ejemplo marcadores de tiempo que pueden ser detectados como falsos positivos. Se incluirá un argumento para poder añadir las máscaras a través de coordenadas, ya que como las máscaras en principio serán rectangulares se deberá indicar para cada máscara el punto superior izquierdo y el inferior derecho. Adicionalmente se trabajará en una forma más intuitiva de añadir máscaras a través de otro parámetro que muestre el primer frame y a base de clicks nos permita añadir dichas máscaras previsualizando su resultado. Adicionalmente se trabajará en mejoras sobre el código previo comprobando que se han realizado todos los tratamientos de errores relacionados con los parámetros que se obtienen a partir del usuario e informando al usuario en caso de que se produzca algún error. Por último se tratará de mejorar el modo debug donde se muestra la previsualización del análisis para que durante dicha previsualización no se guarden fotogramas y se pueda finalizar dicha previsualización mediante algún botón.

Horas a dedicar: 10h (Dos horas cada día a lo largo de 5 días)

Objetivos que se cumplen: Se completará el objetivo OBJ2.4 finalizando OBJ2 al añadir parámetros para añadir máscaras así como el objetivo OBJ8.1 y OBJ8 relacionados con el uso de coordenadas para establecer máscaras y el uso de máscaras en general respectivamente. Se completa en el objetivo OBJ5.3 añadiendo la posibilidad de abortar el proceso de previsualización completando en consecuencia el objetivo OBJ5.

12/04/2021 - 18/04/2021

Durante esta semana se trabajará en unos de los opcionales como es el uso de una interfaz que nos ayude a ajustar los distintos parámetros que acepte nuestro programa. Para ello esta semana se dedicará a la elección de una biblioteca para nuestro lenguaje que nos permita realizar una interfaz sencilla y la realización de dicha interfaz para que puedan ajustarse los argumentos presentes hasta el momento como son: argumento en función del brillo de la imagen a la hora de calcular las diferencias, argumento que nos permite ajustar el círculo o elipse que recubre la Luna, argumento para la posible dilatación de las imágenes, argumento para las máscaras de tal manera que podamos añadirlas y los argumentos para seleccionar la carpeta donde guardar los impactos así como el número de fotogramas a guardar. Adicionalmente se añadirán opciones básicas para poder seleccionar un vídeo o guardar/abrir los argumentos como JSON así como poder previsualizar el proceso de análisis en modo debug para ver si los parámetros han sido correctamente ajustados. En esta semana no se busca aún tener un comportamiento funcional.

Horas a dedicar: 10h (Dos horas cada día a lo largo de 5 días)

Objetivos que se cumplen: Se avanzará en los objetivosopcionales OBJ7, OBJ7.1 y OBJ7.2 ya que se habrá seleccionado la biblioteca para implementar una interfaz en nuestro lenguaje así como se habrá implementado la interfaz para seleccionar y parámetros aunque únicamente la parte visual ya que la parte funcional se completará la semana siguiente finalizando dichos objetivos.

19/04/2021 - 25/04/2021

Durante esta semana se trabajará en incorporar algunas funcionalidades a la interfaz como es la selección de un vídeo, mostrando su primer fotograma en la interfaz para poder previsualizar sobre dicho fotograma los cambios en los parámetros que se realicen. Se deberá implementar el comportamiento de los botones de carga de vídeo como se ha mencionado, poder exportar parámetros a JSON, cargar desde JSON y previ-
visualización de los parámetros cargados, ajuste de todos los ‘parámetros’ mencionados como son el parámetro que se ajusta en función del brillo de la imagen, el que nos permite ajustar la elipse o círculo, el que nos permite dilatar las imágenes para evitar falsos positivos así como el uso de máscaras y la selección de carpeta y número de fotogramas previos y posteriores a guardar.

Horas a dedicar: 15h (Tres horas cada día a lo largo de 5 días)

Objetivos que se cumplen: Se completarán los objetivos opcionales OBJ7, OBJ7.1 y OBJ7.2 ya que esta semana se implementará el comportamiento funcional de la aplicación complementando la implementación visual realizada la semana anterior. De esta forma se completan los objetivos relacionados con la implementación de una interfaz para facilitar el ajuste de parámetros para un determinado vídeo.

26/04/2021 - 02/05/2021

Durante esta semana se trabajará en incorporar a nuestro proyecto la opción de cargar los parámetros necesarios para procesar el vídeo a través de un fichero JSON en vez de hacer uso de una orden con numerosos argumentos. Adicionalmente se trabajará en el objetivo opcional relacionado con obtener un resumen de los impactos obtenidos en el vídeo al terminar el análisis del mismo. Para ello se deberá trabajar en métodos que nos permitan obtener el momento temporal del vídeo cuando se solicite para poder indicar al detectar un impacto en qué hora, minuto y segundo de la grabación se ha producido así como podremos indicar el número de fotograma en el que se ha detectado y el nombre con el que se ha guardado el impacto. Adicionalmente se trabajará en mejorar el método de descarte de impactos fuera de la superficie Lunar de tal forma que se maneje la situación en la que por las características del vídeo no se pueda obtener una circunferencia o elipse, se deberán tomar como posibles impactos todos aquellos que se produzcan ya que si usamos la filosofía de descartar aquellos impactos que se producen fuera de la superficie lunar y no se ha podido detectar la superficie correctamente todos quedarán descartados.

Horas a dedicar: 10h (Dos horas cada día a lo largo de 5 días)

Objetivos que se cumplen: Se completarán los objetivos OBJ6 y OBJ6.1 relacionados con la carga de parámetros en nuestro programa haciendo uso de ficheros JSON para facilitar la tarea de configuración. Adicionalmente se completarán los objetivos OBJ10 y OBJ10.1 relacionados con mostrar un resumen de impactos al finalizar el análisis.

03/05/2021 - 09/05/2021

Durante esta semana se deberá trabajar en realizar las últimas mejoras sobre la interfaz de tal manera que nos permita de la forma más sencilla posible ajustar los parámetros de nuestro programa a un vídeo concreto que se va a utilizar para su posterior análisis haciendo uso de los parámetros ajustados en dicha interfaz. Durante esta semana, una vez la interfaz está prácticamente terminada se plantea la inclusión de posibles mejoras que puedan aparecer como posibles en dicho momento para mejorar la experiencia del usuario. Adicionalmente durante esta semana se plantea la inclusión de un parámetro en nuestro programa que nos permita analizar el vídeo desde un determinado fotograma hasta otro de tal forma que únicamente se analice un fragmento del vídeo.

Horas a dedicar: 5h (Una hora cada día a lo largo de 5 días)

Objetivos que se cumplen: Se mejorará la interfaz relacionada con los objetivos OBJ7, OBJ7.1 y OBJ7.2. Adicionalmente se completará el objetivo OBJ12 que nos permite analizar el vídeo en un rango definido de fotogramas.

10/05/2021 - 16/05/2021

En esta semana se va a trabajar en uno de los opcionales que consiste en poseer una herramienta que nos permita recortar un determinado vídeo para poder analizar únicamente dicha parte. Esta herramienta será útil para los usuarios ya que en muchas ocasiones por ejemplo durante las primeras fases de la grabación se producen muchos cambios en parámetros del vídeo hasta que se ajustan a los valores deseados, esta primera parte debería ser eliminada. Adicionalmente esta herramienta puede ser útil de cara a cuando se obtiene el resumen de impactos poder recortar el vídeo en torno a un impacto.

Horas a dedicar: 5h (Una hora cada día a lo largo de 5 días)

Objetivos que se cumplen: Se completará el objetivo OBJ11.

17/05/2021 - 23/05/2021

Durante esta semana se proponen realizar tests de cobertura sobre nuestro código con el objetivo de tener nuestro código probado en la mayor medida para evitar posibles problemas que no hayan ocurrido hasta el momento y en consecuencia evitar cualquier tipo de excepción no controlada. De esta forma se terminarán de añadir los tests necesarios para obtener el mayor porcentaje posible de cobertura.

Horas a dedicar: 10h (Dos horas cada día a lo largo de 5 días)

Objetivos que se cumplen: En esta semana se ayudará a conseguir el objetivo central, OBJ0 que nos permite detectar los impactos en un determinado vídeo.

24/05/2021 - 20/06/2021

Durante este último mes se plantea conseguir el objetivo opcional OBJ9 donde se intentan ubicar los impactos ocurridos en un mapa lunar, el OBJ13 y adicionalmente se trabajarán en mejoras o aspectos que hayan quedado pendientes al llegar a este punto.

Horas a dedicar: 15h.

De esta forma, podemos observar el siguiente **diagrama de Gantt** donde adicionalmente se incluye la fase de análisis y diseño previa y la fase de documentación del proyecto:

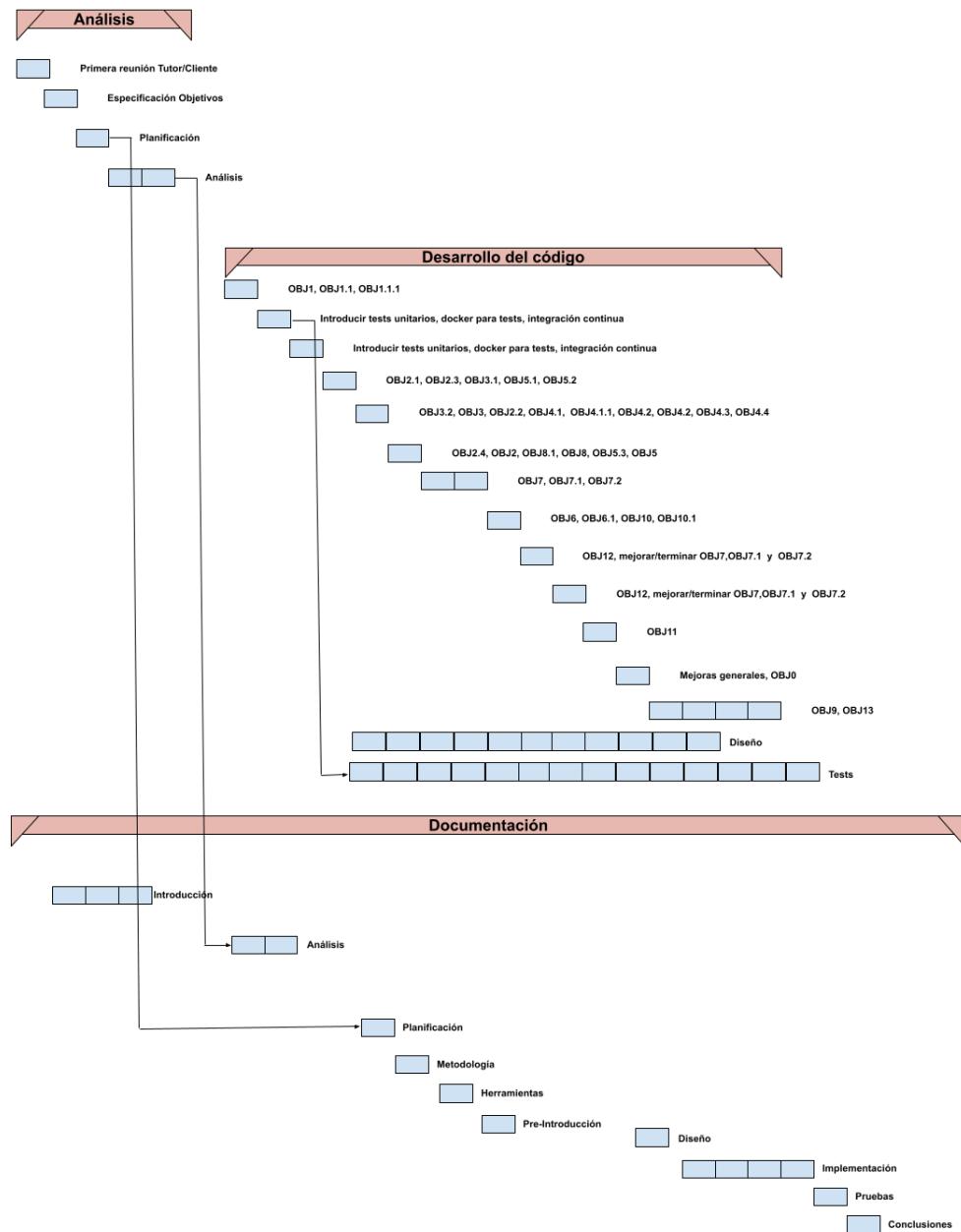


Figura 2.1: Diagrama de Gantt

2.1.2. Presupuesto

En primer lugar vamos a calcular el *costo de mi trabajo por hora*. Como hemos visto en el apartado anterior donde se ha expuesto la planificación temporal de la implementación de este proyecto, se ha dividido su codificación en un total de 16 semanas a lo largo de las cuales se dedicarán según la planificación inicial propuesta un total de 140 horas. Según lo investigado a través de plataformas empresariales como por ejemplo Linkedin, y fijándome en los puestos Junior, ya que estos son los puestos más básicos en el desarrollo software en el ámbito empresarial, encontramos sueldos medios/bajos de 15.000€ al año a media jornada, es decir, unos 1250€ al mes, lo que aproximadamente arroja un total de 15.625€ la hora a media jornada. No obstante como este es mi primer proyecto y tengo poca experiencia considero que unos 13€ la hora es un precio justo. De esta forma teniendo en cuenta las 140 aproximadas de trabajo a realizar obtenemos un total de 1820€ por el tiempo dedicado a su codificación.

En segundo lugar, vamos a tener en cuenta los *gastos de operación*. En cuanto al gasto energético realizado, como hemos visto, se dedicará a la codificación del proyecto un total de 140 horas, si nuestro ordenador consume un total de 400W por hora tenemos un total de 56000W (56Kw) consumidos. Teniendo en cuenta que el precio del vatio en España se encuentra en torno a 0,12663 €/kWh según la Tarifa General 2.0A lo que proporciona un total de gasto eléctrico aproximado de 7€.

Si nos fijamos en el coste de internet, actualmente cuento con una tarifa de internet que ofrece unos 600Mb de fibra simétrica y móvil por un total de 50€ al mes. De esta forma asumimos que la mitad de esta factura es por trabajo y la mitad para ocio. Como el proyecto se desarrolla entre los meses de Febrero a Julio, tenemos un gasto de internet de 6 meses por 25€ el mes que da un total de 150€.

De esta forma los gastos considerados y el precio final se reflejan en la siguiente tabla:

Tipo Gasto	Precio
Costo trabajo	1820€ (13€/hora)
Costo electricidad	7€
Costo internet	150€
Total	1977€

Cuadro 2.2: Presupuesto

2.2. Metodología

Para el desarrollo del software de este proyecto se ha decidido utilizar una **metodología de desarrollo software similar a la metodología llamada TDD** siglas de Test-driven development o en español Desarrollo guiado por pruebas al cual he sido iniciado en la asignatura Infraestructura Virtual.

En este tipo de metodología cobra gran **importancia la existencia de tests**, generalmente las pruebas conocidas como pruebas unitarias destinadas a comprobar el correcto comportamiento de una determinada unidad de código como puede ser una función o una parte de la misma. Esto nos permitirá asegurar que nuestras distintas unidades de código funcionen correctamente teniendo el comportamiento esperado.

Idealmente en esta metodología la primera tarea a realizar es la escritura de un test, con el posterior resultado esperado de fallo ya que el código no ha sido implementado y posteriormente se escribe una primera aproximación de la unidad de código requerida, se ejecutarán los tests, si los tests pasan, es decir, se completan con éxito se puede pasar a desarrollar la siguiente unidad de código (recordemos primero el test de dicha unidad), y si falla debemos realizar un cambio en la unidad de código desarrollada y así de forma continuada hasta que el test o tests se completen.

No obstante como ya se ha comentado se va a seguir una metodología similar a TDD, en el sentido de otorgar de importancia a la pruebas pero sin garantizar el orden de prueba y posterior codificación, no obstante todo el proceso estará guiado por pruebas y toda unidad de código deberá ser evaluado por una prueba para verificar que dicha unidad posee el comportamiento deseado. Por esto se menciona que se va a utilizar una metodología similar a la metodología de desarrollo TDD y no TDD puramente. Adicionalmente se tendrá **constante realimentación con el cliente** realizando ciertas entregas parciales de **prototipos** que permitan una mejor comunicación con el cliente y de esta forma poder realizar los cambios necesarios durante el proceso de desarrollo en vez de esperar al final. Debido a esto podremos incorporar nuevos requisitos o refinar alguno de los requisitos definidos inicialmente.

De esta forma conseguimos el siguiente ciclo:

- 1. Elección de un requisito: Se selecciona un determinado requisito.
- 2. Desarrollar la unidad de código y prueba: Escribir o mejorar la unidad de código a implementar y escribir o mejorar la prueba.
- 4. Ejecutar la prueba: Se ejecuta la prueba asociada al código, si falla se vuelve al paso 2, si no falla, paso 5
- 5. Actualizar requisitos: Se debe marcar como completado el requisito en el que se ha trabajado.
- 6. Comprobar si existen cambios significativos: Si existen cambios significativos desde la última vez que se le mostró un prototipo al cliente vamos al paso 7, en otro caso, paso 5.
- 7. Preparación y muestra de un prototipo: Se muestra al cliente los nuevos cambios implementados desde la última vez que se le mostró un prototipo, adicionalmente se prepara un pequeño informe de la reunión para poder rastrear los cambios a introducir en el programa. El cliente puede proponer cambios o mejoras, por ello volvemos al paso 5.

Para conseguir desarrollar esta metodología como se comentará en el siguiente apartado donde se discutirán las herramientas utilizadas, se ha hecho uso de **Github**, plataforma donde he organizado las tareas a realizar del proyecto en un conjunto de **Milestones**, es decir, conjuntos de tareas a realizar, donde cada una de estas tareas será un **Issue**. Adicionalmente para conseguir ejecutar los tests de una forma más automática y sencilla se utilizará una **Github action** que construirá un contenedor donde se instalarán todas las dependencias necesarias para ejecutar los tests y se ejecutarán de forma sistemática con cada cambio subido al repositorio de tal forma que con cada cambio realizado todos los tests se ejecutarán para poder comprobar si nuestro código cumple con la funcionalidad deseada o en contraposición fallan y somos capaces de detectar fallos en el mismo.

Tal y como se ha mencionado en el paso 7, tras cada reunión se generará un informe que resumirá los puntos principales tratados en dicha reunión, para ello se ha preparado un pequeño formulario que se llenará durante cada reunión y es el siguiente:

FORMULARIO REUNIÓN

Fecha:	
Prototipo presentado:	Nuevas funciones: Funciones modificadas: Funciones eliminadas:
Feedback del cliente:	Añadir: Modificar: Eliminar:

Figura 2.2: Formulario Reunión

Podemos ver el ciclo anterior de forma gráfica:

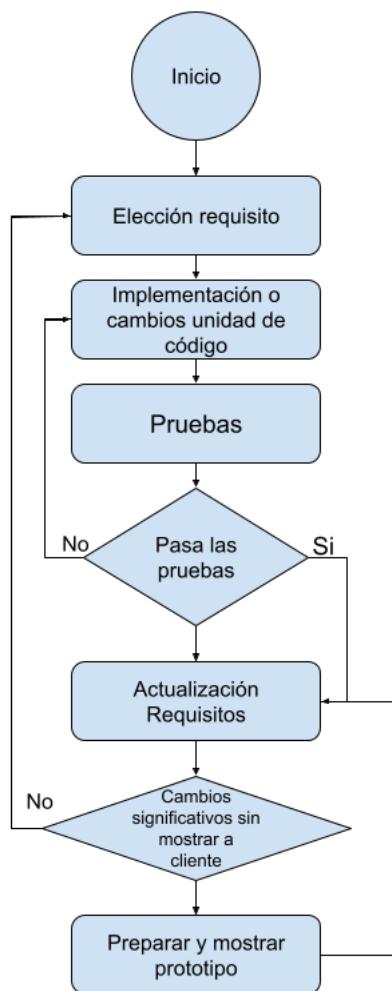


Figura 2.3: Diagrama metodología

2.2.1. Herramientas

Repository

Con el objetivo de agrupar todo el contenido de nuestro proyecto en un repositorio se ha decidido utilizar la plataforma **GitHub** que nos permitirá gestionar las distintas versiones de nuestro proyecto, pudiendo controlar de forma sencilla todos los cambios realizados sobre el mismo. Adicionalmente se ha decidido utilizar GitHub ya que contiene un conjunto de herramientas muy interesantes como son las GitHub Actions que nos permitirán por ejemplo ejecutar las pruebas unitarias con cada cambio subido a la plataforma, realizando lo que se conoce como CI, siglas de Integración Continua, con el objetivo de detectar los fallos que hayan ocurrido lo antes posible debido a modificaciones de nuestro código.

Adicionalmente en esta plataforma podremos organizar nuestro proyecto y el avance del mismo haciendo uso de lo que se conocen como Issues y Milestones.

Los Issues nos permitirán definir tareas, mejoras o correcciones a realizar sobre nuestro código, de esta forma podremos relacionar un determinado avance de código con cada Issue definido.

Los Milestones por otro lado definirán agrupaciones de tareas relacionadas, es decir Issues relacionados, de tal manera que el Milestone será completado al completar todos los Issues que contiene.

El repositorio utilizado puede encontrarse en el [siguiente enlace](#).

Lenguaje de Programación

La elección del lenguaje de programación ha estado muy vinculada con la elección de una biblioteca que nos permita tratar imágenes y vídeo añadiendo funciones útiles para el desarrollo de este proyecto. Es por ello que se ha elegido como lenguaje de programación **Python** en su versión 3. Adicionalmente para la elección de este lenguaje influyeron otros aspectos como ser un lenguaje orientado a objetos, poseer una sintaxis muy intuitiva y sencilla, ser libre, de código abierto, es un lenguaje muy utilizado por la comunidad científica (también por el personal del IAA) y es multiplataforma. El único problema que se podría encontrar derivado del uso de este lenguaje es que al ser un lenguaje interpretado en vez de compilado podría pensarse que sus ejecuciones pudiesen ser más lentas que las de un lenguaje compilado, pero gracias a los prototipos previos que cuenta de mi tutor, sabemos que con Python somos capaces de obtener buenos resultados en tiempos razonables.

Adicionalmente se había planteado el uso de Java debido a que en asignaturas como Sistemas Multimedia donde aprendí sobre tratamiento de imágenes y vídeo se hizo uso de este lenguaje y en consecuencia ya sabría que

bibliotecas utilizar así como hubiese facilitado la creación de interfaces gracias al entorno Netbeans. No obstante se rechazó finalmente el uso de este lenguaje y se optó por Python debido a la gran variedad de bibliotecas gráficas que encontramos para python así como la simple y sencilla sintaxis de Python, el tipado dinámico de Python puede ser útil adicionalmente para el desarrollo de este proyecto así como influyó mi curiosidad por aprender a utilizar otras herramientas nuevas que permitan mi desarrollo académico.

Biblioteca gráfica

Como ya se ha mencionado anteriormente, en Python encontramos una gran cantidad de bibliotecas para trabajar sobre imágenes y vídeo, entre las que encontramos bibliotecas como SciPy que suministra funciones de filtrado lineal y no lineal así como interpolaciones sobre matrices de píxeles entre otras funciones, pero dichas funciones no aportarían para realizar las tareas necesarias. Por otra parte encontramos otra biblioteca conocida como scikit-image que contiene una colección de algoritmos para trabajar sobre imágenes y vídeos muy completos.

No obstante se ha decidido utilizar la biblioteca **OpenCV** siglas de Open Computer Vision es decir Visión Artificial Abierta que es una biblioteca libre y que en un inicio fue desarrollada por Intel. Adicionalmente es multiplataforma siendo compatible en Linux, Windows, Mac OS y Android. Además cuenta con una gran cantidad de documentación tanto oficial como por parte de la comunidad lo que hace que la curva de aprendizaje sea sencilla. Por último se ha elegido esta biblioteca debido a que como se verá en el apartado de implementación nos permitirá tratar imágenes y vídeo de forma muy sencilla y nos permite utilizar un conjunto de algoritmos sobre imágenes definidos en funciones que serán muy útiles de cara a la realización de muchas de las tareas propuestas en los objetivos del proyecto.

Gestor de dependencias

Como gestor de dependencias se ha utilizado **Poetry** que nos permitirá gestionar todas las dependencias de nuestro proyecto permitiendo de forma sencilla encapsularlas definiendo cada una de las dependencias pudiendo indicar adicionalmente su versión, así como actualizarlas, instalar todas las dependencias lo cual será útil de cara a utilizar contenedores y eliminar dependencias entre otras muchas operaciones. Posee una sintaxis muy sencilla y una amplia documentación además de ser casi un estándar en Python debido a la gran cantidad de funciones que aporta para manejar dependencias. De esta forma, Poetry se ayuda del fichero `pyproject.toml` donde con una sintaxis muy sencilla y legible quedan reflejadas todas las dependencias así como sus versiones.

Adicionalmente se podrá extender su funcionamiento con Taskipy como se verá a continuación para poder ejecutar tareas de forma muy sencilla como por ejemplo las pruebas unitarias.

Gestor de Tareas

Se ha elegido **Taskipy** como gestor de tareas. Dicha elección ha sido motivada por la elección previa de Poetry, esto es debido a que Taskipy funciona conjuntamente a Poetry ya que hace uso de los mismos ficheros que Poetry para funcionar y definir sus tareas lo que evita de tener que generar sus propios ficheros y a su vez mantiene una sintaxis sencilla y clara.

Como ya se ha mencionado, hace uso de uno de los ficheros de Poetry, más concretamente el fichero `pyproject.toml` añadiendo simplemente una nueva sección para las tareas a realizar y dichas tareas se ejecutan de forma muy intuitiva haciendo uso de la orden `'poetry run task nombre-tarea'` y además nos permite realizar otras acciones como listar el conjunto de tareas posibles a ejecutar, agrupar tareas, establecer dependencias de ejecución entre otras posibilidades.

Herramienta de tests

Como herramienta para escribir y ejecutar las pruebas unitarias se ha decidido utilizar **Pytest** que es un framework que nos permite escribir pruebas unitarias de forma muy sencilla haciendo uso de aserciones. Proporciona un gran conjunto de utilidades como aserciones, plugins externos para aumentar su funcionamiento así como facilidades para ejecutar todas las pruebas unitarias o alguna en específico.

Adicionalmente en Python encontramos otras herramientas para escribir y ejecutar pruebas unitarias como por ejemplo PyUnit basado también en aserciones pero nos obliga a escribir con una sintaxis algo más rígida. Encontramos también la herramienta Behave pero fue descartado su uso debido a que los informes generados son menos significativos que los generados a Pytest aunque la sintaxis y uso es muy parecido.

Generador de documentación

Con el objetivo de contar con un código documentado de acuerdo a una determinada sintaxis uniforme para todo el código se ha decidido optar por la utilización de un generador de documentación que nos permitirá generar documentación acerca de nuestro código.

Debido a la gran cantidad de documentación existente, su sintaxis muy clara y legible así como la gran cantidad de formatos de salida como HTML

y Latex entre otros, su estructura jerárquica y la extensión de sus funcionalidades haciendo uso de la inclusión de extensiones se ha decidido utilizar **Sphinx**.

CI y contenedores

Como herramienta de integración continua para ejecutar las pruebas unitarias se ha decidido utilizar **GitHub Actions** ya que actualmente es de los pocos sistemas que no utilizan créditos para su funcionamiento y aunque quizás no es de los más rápidos ejecutando tareas, es bastante competente y es gratuito. Es por el sistema de créditos que han sido descartados sistemas como TravisCI o CircleCI utilizados en asignaturas como Infraestructura Virtual.

Adicionalmente con el objetivo de tener un entorno donde poder ejecutar las pruebas unitarias teniendo agrupadas todas las dependencias para evitar errores se ha decidido hacer uso de contenedores, en este caso se ha decidido utilizar **Docker**. De esta forma se utilizarán contenedores para ejecutar tanto los tests como para poder ejecutar la aplicación de forma que todas las dependencias estén contenidas y de esta forma sea portable a cualquier otro sistema y de esta forma no existan problemas de compatibilidad.

Biblioteca generar interfaces

De cara a uno de los objetivos opcionales donde se pide crear una interfaz que facilite a los usuarios elegir los valores de los parámetros para analizar un determinado vídeo es necesario elegir una biblioteca que nos permita crear interfaces gráficas en Python.

Para este lenguaje he encontrado destacadas dos bibliotecas, PyQt y Tkinter. Tkinter finalmente fue descartada debido a que la interfaz que se genera visualmente es más antigua, simple y no permite realizar cosas complejas, en cambio PyQt permite una mayor flexibilidad a la hora de programar, es muy parecido a la generación de interfaces vista en Java en asignaturas como Sistemas Multimedia, posee un aspecto visual básico más moderno y totalmente personalizable, herramientas gráficas similares a las que proporciona Netbeans en Java para generar las interfaces de forma sencilla como es la herramienta **QT 5 Designer** entre otras muchas facilidades. Debido a todo esto se ha decidido utilizar **PyQt** en su versión 5.

Capítulo 3: Análisis y especificación de requisitos

3.1. Introducción a la ingeniería de requisitos

La ingeniería de requisitos nos permite obtener de forma clara y sin ambigüedades las tareas que se deben realizar, las necesidades de los clientes y la solución a la que se debe llegar. En este capítulo se presenta el proceso de análisis y obtención de requisitos siguiendo las recomendaciones del artículo 'Metodología para la elicitation de Requisitos de Sistemas Software' [6] así como lo visto en la asignatura Fundamentos de Ingeniería del Software, donde se recomiendan realizar los siguientes pasos: Obtener información del dominio del problema, realizar reuniones de elicitation, obtención de diagramas de caso de uso, obtención de requisitos funcionales, no funcionales y de información.

3.2. Estudio de viabilidad

En este primer apartado nos preguntamos si es viable el proyecto que se desea realizar con el objetivo de responder de forma breve y sencilla a una pregunta fundamental y que debería ser la primera en plantearnos al comenzar el proceso de análisis, ¿Es viable y/o conveniente realizar el desarrollo del sistema propuesto?

Como hemos visto, el problema consiste en desarrollar un software que nos permita dada una determinada grabación obtener de forma automática, en caso de que se produzcan, los impactos que han tenido lugar durante la grabación con el objetivo de eliminar problemas en la detección de impactos derivados de las limitaciones humanas, que como dijimos eran principalmente dos:

- Los impactos son visibles normalmente durante fracciones de segundo.
- Con el objetivo de detectar impactos, es necesario obtener y manejar grandes cantidades de información en forma de vídeos digitales.

Teniendo en cuenta esto, la creación de un software como el propuesto solucionaría los dos problemas anteriormente planteados ya que el software mediante el análisis de los fotogramas del vídeo sería capaz de detectar los impactos aunque ocurran en fracciones de segundo ya que quedarán reflejados en los fotogramas del vídeo. En cuanto al segundo problema planteado, también es resoluble ya que con un software automatizado que realice la tarea de analizar los vídeos se evitará la intervención humana que analice los vídeos y únicamente será necesaria su intervención a la hora de recoger y analizar los resultados arrojados por el programa, lo cual hace que la importancia de la cantidad de información directamente relacionada con la cantidad de tiempo invertida por una persona pase a un segundo plano solucionando el segundo problema.

Lo siguiente que debemos plantearnos es si se puede desarrollar el software con la tecnología actual. Con la tecnología existente podemos llegar a resolver este problema ya que el problema se puede descomponer fácilmente en un sencillo problema de análisis de vídeo e imagen digital y como sabemos gracias a asignaturas cursadas, existen multitud de bibliotecas que nos permiten trabajar sobre imagen y vídeo aportando multitud de utilidades que pueden facilitar esta tarea. De esta forma, se deberá elegir cuidadosamente la biblioteca de tratamiento de imagen y vídeo para alcanzar el objetivo principal planteado.

En cuanto a las restricciones temporales, debería ser posible implementar los objetivos marcados como obligatorios en el periodo temporal establecido, además, debería ser posible incluso llegar a implementar algunos de los objetivosopcionales que mejorará la experiencia del usuario con este software.

Cuestiones	Respuesta
¿Este software soluciona los problemas planteados?	Sí
¿Es posible su desarrollo con la tecnología existente?	Sí
¿Es posible su desarrollo teniendo en cuenta las limitaciones temporales?	Sí

Cuadro 3.1: Resumen viabilidad

3.3. Obtención de información sobre el dominio del problema y el sistema actual

La siguiente fase del proceso de análisis, obtención y especificación de requisitos consiste en obtener un glosario de términos destacados y de necesario conocimiento antes de realizar las reuniones para obtener los objetivos y requisitos del sistema así como una breve introducción al sistema.

3.3.1. Glosario términos destacados del ámbito del problema

- **Meteoroide:** Cuerpo del sistema solar de un tamaño comprendido entre 10 µm y 50 m de diámetro.
- **NEOs:** Siglas de objeto cercano a la tierra.
- **Cráter de impacto:** Depresión producida por el impacto de un cuerpo en la superficie de otro cuerpo de mayor tamaño.
- **Destello de impacto:** Energía luminosa liberada tras el impacto de un cuerpo contra otro a altas velocidades. Visible normalmente por fracciones de segundo.
- **Impacto lunar:** Colisión producida por meteoroides normalmente en la superficie de la Luna y visible gracias a la energía luminosa desprendida.
- **Falso positivo:** En este ámbito se refiere a detectar como impacto algo que realmente no lo es.
- **Falso positivo exterior a la superficie Lunar:** Cualquier posible detección de un impacto en el exterior de la superficie lunar deberá ser descartado, es un falso positivo ocasionado posiblemente por la aparición de cuerpos como estrellas.
- **Falso positivo zona iluminada:** Cualquier posible detección de un impacto en la zona iluminada deberá ser descartado. Serán falsos positivos ya que el brillo generado por los impactos es menor que el propio brillo de la zona.
- **Zona no iluminada de la Luna:** Zona de la Luna actualmente no iluminada directamente por el Sol pero visible utilizando telescopios. Estas regiones cobran suma importancia en este problema a resolver ya que son las únicas regiones en las que será posible detectar impactos debido a los destellos observables de los mismos.
- **Zona iluminada de la Luna:** Zona de la Luna directamente iluminadas por el Sol, en estas zonas no es posible detectar impactos, estas

zonas deberán ser ignoradas. El destello producido por el impacto en estas zonas es de menor intensidad que la intensidad de brillo de la propia zona iluminada.

- **Fotograma:** Cada una de las imágenes que compone un vídeo.
- **Dilatación fotogramas:** Proceso por el cual los píxeles más brillantes de una imagen 'crecen' ocupando más píxeles, más concretamente ocupando píxeles más oscuros.
- **Diferencia fotogramas:** Proceso por el cual dos fotogramas sufren una operación aritmética de resta que da como resultado las diferencias de los valores de los píxeles coordenada a coordenada de una imagen con respecto a otra. De esta forma dos imágenes iguales restadas darán como resultado una imagen totalmente en negro.

3.3.2. Introducción al sistema

Se plantea el desarrollo de un software de detección de impactos lunares, es decir un software que permita a un usuario dada una grabación de una zona no iluminada de la superficie lunar, detectar si han tenido o no lugar impactos.

De esta forma un usuario introducirá vídeos de grabaciones de la superficie lunar, las grabaciones para poder detectar con éxito impactos deberán ser de zonas no iluminadas aunque pueden aparecer zonas iluminadas pero en una proporción menor a las zonas no iluminadas que es donde se podrán detectar dichos impactos. Una vez introducido el vídeo el programa deberá analizar los fotogramas que componen dicho vídeo en busca de variaciones luminosas, es decir destellos luminosos producidos por los impactos de meteoroides u otros NEOs utilizando técnicas como la diferencia de fotogramas para encontrar dichos destellos.

Adicionalmente el sistema deberá tratar de minimizar en la medida de lo posible la detección de falsos positivos ocasionados por otros cuerpos exteriores a la superficie Lunar debido a su brillo variante, u otros falsos positivos ocasionados por ejemplo por la variación de brillo en las zonas iluminadas de la superficie Lunar.

3.4. Preparación de reuniones de elicitación y negociación

En esta sección se muestra, tras las primeras reuniones de elicitación, las tareas de identificación de los implicados, conocimiento las necesidades de los clientes y usuarios y por último resolución, en caso de que existan, los posibles conflictos.

3.4.1. Descripción general implicados

A continuación se presenta la descripción general de los implicados, en este caso serán dos perfiles, el astrónomo, encargado de utilizar directamente el sistema y un cliente intermedio encargado de recibir y valorar los prototipos para obtener feedback de forma constante:

Nombre	Descripción	Tipo	Responsabilidad
Astrónomo	Representa un científico experto en astronomía	Usuario sistema	Introducir vídeo para analizar. Obtener los resultados del análisis del vídeo. Analizar los resultados obtenidos

Nombre	Descripción	Tipo	Responsabilidad
Cliente directo	Representa al cliente con el que se mantiene contacto directo mediante reuniones	Usuario sistema	Asistir a las reuniones concertadas. Evaluar los prototipos presentados. Transmitir posibles cambios y mejoras al desarrollador

3.4.2. Descripción perfil implicados

Representante	Indefinido
Descripción	Astrónomo
Tipo	Usuario del sistema, hace uso del sistema de forma directa.
Responsabilidades	Introducir vídeos en el sistema para poder procesarlos, recoger los resultados y analizarlos.
Criterios de éxito	Hay éxito si se consigue procesar el vídeo suministrado obteniendo los fotogramas donde se han producido los impactos. Hay éxito si se produce el menor número posible de falsos positivos.
Implicación	Es el responsable de introducir el vídeo a procesar y recoger los resultados una vez procesado el vídeo.
Comentarios/Cuestiones	Está familiarizado con sistemas informáticos pero no tiene por qué conocer detalles técnicos del sistema

Representante	Dr. Sergio Alonso Burgos
Descripción	Cliente directo
Tipo	Experto, hace uso del sistema de forma directa.
Responsabilidades	Asistir a las reuniones concertadas. Evaluar los prototipos presentados. Transmitir posibles cambios y mejoras al desarrollador.
Criterios de éxito	Hay éxito si se consigue evaluar el prototipo propuesto por el desarrollador.
Implicación	Es el responsable de asistir a las reuniones y evaluar los prototipos propuestos por el desarrollador.
Comentarios/Cuestiones	Está familiarizado con sistemas informáticos. Conoce detalles internos del sistema.

3.5. Casos de Uso

Con el objetivo de obtener los requisitos funcionales, se presentan a continuación los diagramas de caso de uso obtenidos. Cómo bien sabemos los casos de uso son acciones que un sistema realiza al interactuar con un determinado agente externo, en este caso cualquiera de los dos actores definidos previamente. De esta forma, un caso de uso es iniciado por uno de los actores con el objetivo de describir funcionalidades de nuestro sistema fruto de dichas interacciones.

La obtención de los distintos casos de uso de nuestro sistema nos permitirá determinar de forma más sencilla los requisitos funcionales ya que a partir de los casos de uso que obtengamos podremos determinar uno o más requisitos funcionales por caso de uso como se verá en el siguiente apartado.

Como consecuencia, en nuestro programa si nos limitamos a la definición aportada de caso de uso previa, encontraríamos de acuerdo a los objetivos los siguientes casos de uso:

- **CU-1: ANALIZAR VÍDEO:** Analizar un determinado vídeo aportado por un usuario con el objetivo de encontrar los posibles impactos que han ocurrido durante la grabación.
- **CU-2: ELECCIÓN DE PARÁMETROS:** Elegir los parámetros para analizar el vídeo de forma sencilla e intuitiva.
- **CU-3: EXPORTAR PARÁMETROS:** Debido a la gran cantidad de parámetros que se pueden llegar a utilizar en programas como el que se va a desarrollar, se podrán exportar los parámetros utilizados para ser utilizados en otras ejecuciones.
- **CU-4: IMPORTAR PARÁMETROS:** Importar los parámetros exportados de una configuración previa para partir desde ese punto a la hora de seleccionar parámetros.
- **CU-5: VERIFICAR IMPACTOS:** Verificar/Validar los impactos de acuerdo a la existencia de un impacto capturado por dos vídeos en el mismo instante temporal
- **CU-6: RECORTAR VÍDEOS:** Recortar vídeos para no tener que tratar con partes no útiles previas al ajuste de los parámetros de la propia cámara o para evitar tratar con vídeos demasiados largos.

A continuación se muestra el diagrama de casos de uso resultante donde se se pueden observar los **casos de uso** anteriormente mencionados junto a los **actores** descritos en el apartado previo así como las distintas **asociaciones actor/caso de uso**, donde en este caso cada asociación indica que un actor participa en dicho caso de uso activándolo, y la **frontera del sistema**:

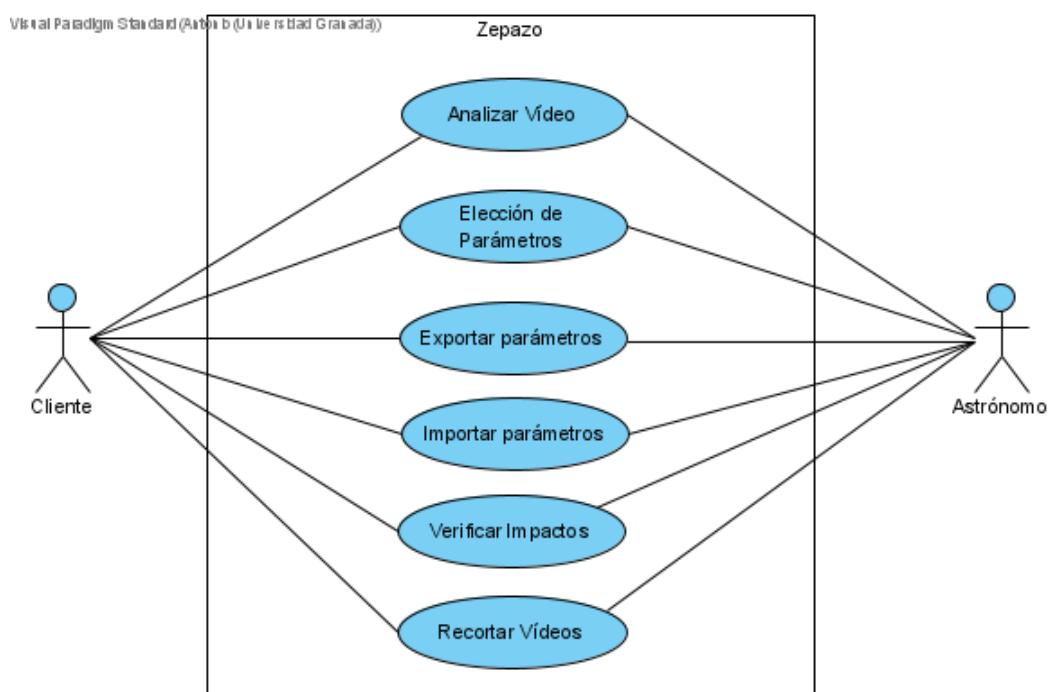


Figura 3.1: Diagrama de casos de Uso

3.6. Lista estructurada de requisitos

Tal y como se observa en las listas siguientes, se va a establecer la clasificación de los requisitos en **tres categorías principales, funcionales** que son aquellos requisitos que describen la interacción entre el sistema y su entorno, requisitos **no funcionales** que describen cualidades y restricciones del sistema y por último los **requisitos de información** donde se describirán las propiedades relacionadas con el almacenamiento de información de nuestro software.

3.6.1. Requisitos funcionales

A continuación se presenta la lista estructurada de requisitos funcionales obtenidos a través del modelado de casos de uso presentados en el apartado anterior.

Caso de uso 1

- **RF 1 -INDICAR PARÁMETROS:** El sistema deberá poder aceptar parámetros para ajustar el análisis a las condiciones de cada vídeo.
 - **RF 1.1 -MOSTRAR AYUDA ARGUMENTOS:** El sistema deberá mostrar al usuario cuando lo indique o falle en la especificación de un argumento ayuda sobre un argumento en concreto o todos los que utiliza el programa.
 - **RF 1.2 -INDICAR VÍDEO ANALIZAR:** El sistema aceptará como parámetro la ruta del vídeo a analizar.
 - **RF 1.3 -INDICAR LÍMITE DETECCIÓN:** El sistema aceptará como parámetro un valor que permitirá ajustar el análisis a las condiciones luminosas del vídeo.
 - **RF 1.4 -INDICAR DILATACIÓN FOTOGRAMAS:** El sistema aceptará como parámetro un valor que indicará cuánto dilatar las imágenes para evitar falsos positivos en las zonas luminosas.
 - **RF 1.5 -INDICAR MÁSCARAS:** El sistema aceptará como parámetro un conjunto de coordenadas en las que establecer máscaras.
 - **RF 1.6 -INDICAR LÍMITE CONTORNO LUNAR:** El sistema aceptará como parámetro un valor que permitirá ajustar el contorno lunar a las condiciones del vídeo.
 - **RF 1.7 -INDICAR CARPETA GUARDADO FOTOGRAMAS:** El sistema aceptará como parámetro una ruta de un directorio donde guardar los fotogramas donde se han detectado impactos.

- **RF 1.8 -INDICAR NÚMERO FOTOGRAMAS PREVIOS Y POSTERIORES A GUARDAR:** El sistema aceptará como parámetro un número que indicará el número de fotogramas previos y posteriores al impacto que se desean guardar.
 - **RF 1.9 -INDICAR ARCHIVO CONFIGURACIÓN:** El sistema aceptará como parámetro la ruta de un fichero de configuración a través del cual importar todos los parámetros.
 - **RF 1.10 -INDICAR RUTA ARCHIVO LOG VÍDEO:** El sistema aceptará como parámetro la ruta de un archivo de log resultado de analizar un vídeo para poder compararlo con otro.
 - **RF 1.11 -INDICAR FOTOGRAMA INICIAL:** El sistema aceptará como parámetro el número de fotograma inicial para analizar desde ese punto.
 - **RF 1.12 -INDICAR FOTOGRAMA FINAL:** El sistema aceptará como parámetro el número de fotograma final bien para analizar hasta ese punto.
- **RF 2 -ANALIZAR VÍDEO:** El sistema tratará de analizar el vídeo en busca de impactos.
 - **RF 3 -CARGAR VÍDEO:** El sistema cargará el vídeo para poder analizarlo.
 - **RF 4 -OBTENER ESTADÍSTICAS VÍDEO:** El sistema obtendrá estadísticas del vídeo necesarias para su procesamiento o identificación de los fotogramas guardados de impactos como los fotogramas, fotogramas por segundo, duración de la grabación, etc.
 - **RF 5 -APLICAR MÁSCARAS:** El sistema deberá aplicar a cada fotograma las máscaras indicadas por parámetro.
 - **RF 6 -CALCULAR DIFERENCIAS FOTOGRAMAS:** El sistema deberá calcular las diferencias entre los fotogramas como parte fundamental del proceso de detección de impactos.
 - **RF 7 -MOSTRAR PREVISUALIZACIÓN:** El sistema dará la posibilidad de mostrar una previsualización del proceso de análisis donde se muestre el contorno lunar calculado y los posibles impactos que se producirían con los argumentos indicados.
 - **RF 7.1 -TERMINAR PREVISUALIZACIÓN:** El sistema deberá dar la posibilidad de terminar el proceso de previsualización en cualquier momento.

- **RF 8 -DILATAR IMAGEN:** El sistema dará la posibilidad de dilatar los fotogramas con el objetivo de evitar falsos positivos producidos por la aparición de zonas luminosas en la grabación.
- **RF 9 -CALCULAR CONTORNO LUNAR:** El sistema deberá calcular el contorno lunar con el objetivo de evitar detectar como impactos aquellos destellos luminosos que no se encuentren sobre la superficie lunar.
- **RF 10 -MARCAR IMPACTOS:** El sistema deberá marcar sobre el fotograma el lugar donde ha tenido lugar un impacto para poder identificarlo de forma rápida en la imagen.
- **RF 11 -ENCONTRAR POSIBLES IMPACTOS:** El sistema deberá encontrar los impactos producidos en una grabación.
 - **RF 11.1 - DESCARTAR IMPACTOS EXTERIORES:** El sistema deberá descartar aquellos impactos que se produzcan en el exterior de la superficie lunar.
 - **RF 11.2 - OBTENER IMPACTOS INTERIORES:** El sistema deberá mantener aquellos impactos que se produzcan en el interior de la superficie lunar.
- **RF 12 -ALMACENAR IMPACTOS:** El sistema deberá almacenar el fotograma donde se ha producido un impacto.
 - **RF 12.1 -ALMACENAR IMPACTOS PREVIOS:** El sistema dará la posibilidad de almacenar un número determinado de fotogramas previos a la detección del impacto.
 - **RF 12.2 -ALMACENAR IMPACTOS POSTERIORES:** El sistema dará la posibilidad de almacenar un número determinado de fotogramas posteriores a la detección del impacto.
- **RF 13 - INFORME ANÁLISIS:** El sistema deberá generar un archivo donde queden reflejados los impactos que han tenido lugar en la grabación de forma resumida y esquemática.

Caso de uso 2

- **RF 14 -AYUDA SELECCIÓN PARÁMETROS:** El sistema deberá proporcionar alguna forma ajustar los parámetros de forma sencilla como por ejemplo a través de una interfaz.
 - **RF 14.1 AYUDA SELECCIÓN LÍMITE DETECCIÓN:** El sistema deberá ayudar proporcionar una herramienta al usuario como una interfaz para seleccionar el límite de detección.
 - **RF 14.2 AYUDA SELECCIÓN DILATACIÓN FOTOGRAMAS:** El sistema deberá ayudar proporcionar una herramienta al usuario como una interfaz para seleccionar el valor de dilatación de fotogramas.
 - **RF 14.3 AYUDA SELECCIÓN MÁSCARAS:** El sistema deberá ayudar proporcionar una herramienta al usuario como una interfaz para añadir o eliminar máscaras.
 - **RF 13.4 AYUDA SELECCIÓN LÍMITE CONTORNO LUNAR:** El sistema deberá ayudar proporcionar una herramienta al usuario como una interfaz para seleccionar el valor del límite para el cálculo del contorno lunar.
 - **RF 14.5 AYUDA SELECCIÓN CARPETA GUARDADO FOTOGRAMAS:** El sistema deberá ayudar proporcionar una herramienta al usuario como una interfaz para seleccionar la carpeta de guardado.
 - **RF 14.6 AYUDA SELECCIÓN FOTOGRAMAS PREVIOS Y POSTERIORES A GUARDAR:** El sistema deberá ayudar proporcionar una herramienta al usuario como una interfaz para seleccionar el número de fotogramas a guardar antes y después de producirse un impacto.
- **RF 15 - MOSTRAR PARÁMETROS APLICADOS A FOTOGRAMA:** El sistema de ayuda para la selección de parámetros contará además con una opción para mostrar los parámetros establecidos sobre un fotograma del vídeo.
- **RF 16 - RESTAURAR VALORES POR DEFECTO PARÁMETROS:** El sistema de ayuda para la selección de parámetros contará además con una opción para restablecer los valores de los parámetros a los valores por defecto.
- **RF 17 - OBTENER COMANDO A UTILIZAR:** El sistema de ayuda para la selección de parámetros contará además con la opción de obtener el comando que sería necesario utilizar para analizar el vídeo con la configuración establecida en el sistema de ayuda de parámetros.

Caso de uso 3

- **RF 18 - EXPORTAR ARCHIVO DE CONFIGURACIÓN:** El sistema de ayuda para la selección de parámetros contará además con una opción para exportar la configuración establecida a un archivo de configuración que lo acepte el sistema de análisis de vídeos.

Caso de uso 4

- **RF 19 - IMPORTAR ARCHIVO DE CONFIGURACIÓN:** El sistema de ayuda para la selección de parámetros contará además con una opción para importar una determinada configuración establecida previamente.

Caso de uso 5

- **RF 20 - VERIFICAR IMPACTOS:** El sistema deberá poder verificar los impactos de dos vídeos a partir del momento temporal donde tomaron lugar.
 - **RF 20.1 - INDICAR RUTA LOG VÍDEO 1:** El sistema aceptará como parámetro la ruta del log del primer vídeo.
 - **RF 20.2 - INDICAR RUTA LOG VÍDEO 2:** El sistema aceptará como parámetro la ruta del log del segundo vídeo.
 - **RF 20.3 - INDICAR MARGEN TEMPORAL:** El sistema aceptará como parámetro el máximo número de segundos para que dos impactos se emparejen.
- **RF 21 - INFORME VERIFICACIÓN:** El sistema deberá generar un archivo donde queden reflejados los impactos que han sido verificados.
 - **RF 21.1 - INDICAR RUTA LOG VERIFICACIÓN:** El sistema aceptará como parámetro la ruta del log producido tras el proceso de verificación.

Caso de uso 6

- **RF 22 - RECORTAR VÍDEO:** El sistema deberá dar la posibilidad de recortar un vídeo para no tener que trabajar con el vídeo completo inicialmente por ejemplo para establecer unos parámetros adecuados.
 - **RF 22.1 - INDICAR VIDEO RECORTAR:** El sistema aceptará como parámetro la ruta del vídeo a recortar.
 - **RF 22.2 - INDICAR FOTOGRAMA INICIO Y FINAL:** El sistema aceptará como parámetro el número de fotograma inicial y final para recortar el vídeo.

- **RF 22.3 - INDICAR RUTA VIDEO RECORTADO:** El sistema aceptará como parámetro la ruta donde guardar el vídeo recortado.

3.6.2. Requisitos no funcionales

En este apartado se presentan los requisitos no funcionales, es decir aquellas restricciones o cualidades que no tienen una relación directa con la funcionalidad del sistema. De esta forma los clasificamos haciendo uso del esquema FURPS que clasifica los requisitos en funcionales (anteriormente definidos), facilidad de uso, fiabilidad, rendimiento, soporte:

Facilidad de uso

- **RNF1 -MANUAL USUARIO:** El sistema debe contar con manuales de usuario para facilitar su aprendizaje.
- **RNF2 -TIEMPO APRENDIZAJE:** Un usuario experimentado debe ser capaz de entender y poder utilizar todas las funciones del sistema tras un aprendizaje no superior a 3 horas.
- **RNF3 -DOCUMENTACIÓN CÓDIGO:** El código fuente del programa debe estar documentado de tal manera que sea posible generar una documentación de forma automática.

Fiabilidad

- **RNF4 -MENSAJES DE ERROR:** El sistema debe contar con mensajes de error y estos deben ser claros e intuitivos.
- **RNF5 -FALSOS POSITIVOS:** El sistema debe minimizar la detección de falsos positivos.
- **RNF6 -RECUPERACIÓN FALLOS:** Ante un fallo en la aplicación, deberá poder reiniciarse de forma inmediata.

Rendimiento

- **RNF7 -TIEMPO PROCESAMIENTO:** El sistema debe analizar el vídeo en un tiempo igual o inferior a la duración del mismo.

Soporte

- **RNF8 -TESTS CÓDIGO:** Todas las funciones utilizadas deben estar correctamente probadas mediante tests unitarios.

- **RNF8.1 -CONTENEDOR TESTS:** Tener un entorno donde poder ejecutar los tests unitarios.
- **RNF9 -INTEGRACIÓN CONTINUA:** Antes de incorporar nuevos cambios será necesario pasar todos los tests unitarios.
- **RNF10 -CONTENEDOR APLICACIÓN:** Tener un entorno donde poder ejecutar la aplicación.
 - **RNF10.1 -INSTALACIÓN APLICACIÓN:** Tener un entorno donde poder instalar la aplicación.
 - **RNF10.2 -DEPENDENCIAS APLICACIÓN:** Todas las dependencias deberán estar definidas con una versión concreta y gestionadas por un gestor de dependencias.
- **RNF11 -IDIOMA:** El idioma por defecto de toda documentación y manuales debe ser el inglés.

3.6.3. Requisitos de información

- **RI 1. -FOTOGRAMAS IMPACTOS:** El sistema almacenará los fotogramas donde han tenido lugar los impactos. En el nombre de la imagen a guardar se debe añadir información significativa que permita identificarlo.

Contenido: Imagen, Número de impacto.

Requisitos asociados: RF 12

- **RI 2. -LOGS IMPACTOS:** El sistema almacenará en formato JSON un log donde se pueda ver un resumen de los impactos que han tenido lugar en la grabación aportada.

Contenido: Número de Impacto, número de fotograma, momento temporal.

Requisitos asociados: RF 13

- **RI 3. -LOGS VERIFICACIÓN:** El sistema almacenará en formato JSON un log donde se pueda observar de forma sencilla si algún impacto ha sido verificado

Contenido: Número de impacto del vídeo 1, número de impacto del vídeo 2, diferencia temporal, ubicación del archivo de log del vídeo 1, ubicación del archivo de log del vídeo 2.

Requisitos asociados: RF 21

Capítulo 4: Diseño

4.1. Introducción al diseño

En este capítulo se mostrará el proceso de diseño de la aplicación a desarrollar, para ello se muestra el proceso de diseño realizado a lo largo del desarrollo de la aplicación, ya que el diseño ha surgido al mismo tiempo que el propio desarrollo de la aplicación con el objetivo de poder ajustar el diseño tanto a los requisitos y objetivos iniciales como a los que han surgido gracias a las reuniones realizadas con mi tutor. De esta forma el diseño no ha estado definido desde el inicio del proyecto y se muestra el proceso de diseño a continuación en las siguientes secciones.

4.1.1. Prototipo Inicial

De esta forma tal y como se mencionó en el apartado '1.1 Introducción', mi tutor *Dr. Sergio Alonso Burgos había realizado diversos prototipos previos* a la aplicación que se presenta en este documento donde se incluía una aproximación inicial para resolver el problema de la detección de impactos. En este primer diseño, el prototipo anteriormente mencionado, del cual se ha partido, la aplicación estaba estructurada en dos bloques:

- **Zepazo.py:** Fichero donde (de forma general) se definen los argumentos del programa necesarios para empezar el análisis de la grabación como pueden ser el propio vídeo a analizar, el rango de fotogramas a analizar, etc. A continuación una vez capturados los argumentos se realiza una comprobación de los mismos donde se producen informes de error y se llama a una función del fichero 'functions.py' que se encarga de analizar el vídeo y devuelve los fotogramas donde se han encontrado los impactos y se procede al almacenamiento de dichos fotogramas en una determinada carpeta. A continuación haciendo uso de un sistema GPS se estima el momento temporal del impacto con el objetivo de crear un archivo con información producida de los impactos producidos.
- **functions.py:** En este archivo encontrábamos métodos para guardar fotogramas, una clase llamada 'VideoParser' y otra clase llamada BeepsParser.
 - La clase VideoParser consta de un método principal llamado 'analyze' se encargaba de intentar detectar los impactos contenido la mayor parte de la lógica de la aplicación. Adicionalmente encontrábamos funciones relacionadas con la determinación del

momento temporal del impacto basándose en una señal producida cada cierto tiempo determinada por un GPS y que aparecía en los videos.

- La clase BeepsParser se encargaba de encontrar en una pista de audio extraída del video los zumbidos que aparecían en el video y parsearlos con el objetivo de obtener los momentos temporales en los que ocurrían.

De esta forma, el prototipo previo mencionado contaba con un diseño de clases y paquetes como la que se muestra a continuación:

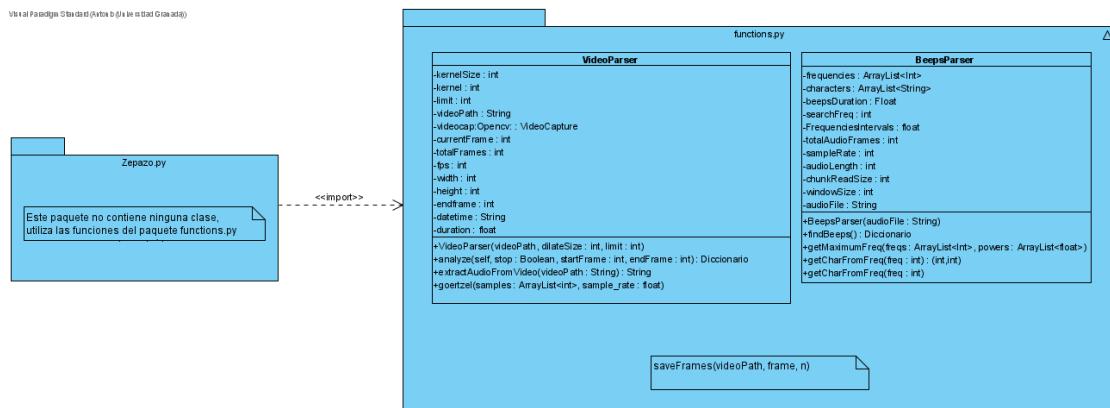


Figura 4.1: Diagrama de clases y paquetes prototipo inicial

Donde como podemos ver, encontramos lo anteriormente mencionado, un paquete (zepazo) contiene el punto de acceso que será con el que interactúe el usuario y que importa todas las funcionalidades definidas en el paquete functions que consta de dos clases, una dedicada principalmente al análisis de video y otra clase dedicada al análisis del sonido de los videos para extraer los momentos temporales de los impactos a partir de una señal sonora generada cada cierto tiempo de forma intermitente de acuerdo a una señal GPS.

4.2. Diseño de la aplicación

Tras el estudio de los diversos requisitos obtenidos durante la fase de análisis en el apartado 3.6 se ha decidido que **Zepazo sea un conjunto de programas** donde cada programa tenga una funcionalidad muy concreta. En total atendiendo a los requisitos generados tras el estudio de los distintos casos de uso, obtendríamos un total de **cuatro programas** agrupando los casos de uso (y en consecuencia los requisitos obtenidos) de la siguiente forma:

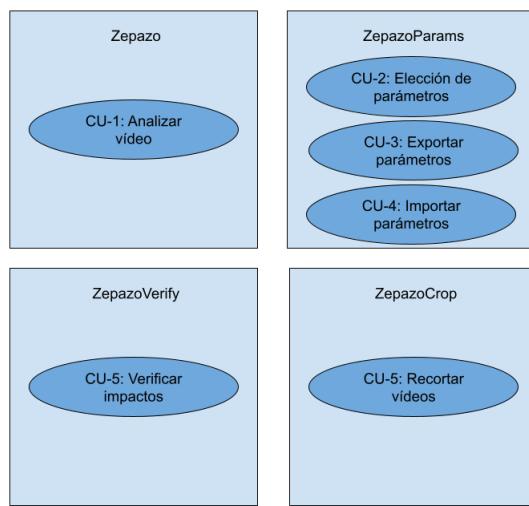


Figura 4.2: Diagrama agrupación casos de uso en programas

En consecuencia encontramos un total de cuatro programas:

- **Zepazo**: Es el programa principal encargado de analizar un determinado vídeo en busca de los posibles impactos que hayan podido tomar lugar, deberá responder ante los requisitos funcionales **RF1 - RF13**
- **ZepazoParams**: Este programa servirá de ayuda para seleccionar los parámetros de tal forma que a través de una sencilla interfaz podrán ajustarse los mismos de forma visual, deberá responder ante los requisitos funcionales **RF14 - RF19**
- **ZepazoVerify**: Este programa permitirá verificar/validar los impactos a partir de los archivos de log generados por el análisis de dos vídeos, decimos que un impacto ha sido verificado/validado cuando se ha detectado por el programa Zepazo en dos grabaciones distintas en el mismo momento temporal, deberá responder ante los requisitos funcionales **RF20 - RF21**

- **ZepazoCrop:** Este programa permitirá recortar vídeos para ser utilizados por ZepazoParams con el objetivo de no tener que cargar el vídeo entero así como el usuario descargar de los servidores el vídeo completo que puede llegar a ser de gran peso y duración, deberá responder ante los requisitos funcionales **RF22**

De esta forma conseguimos que cada aplicación nos permita realizar una tarea diferente y obtenemos entonces la siguiente estructura:

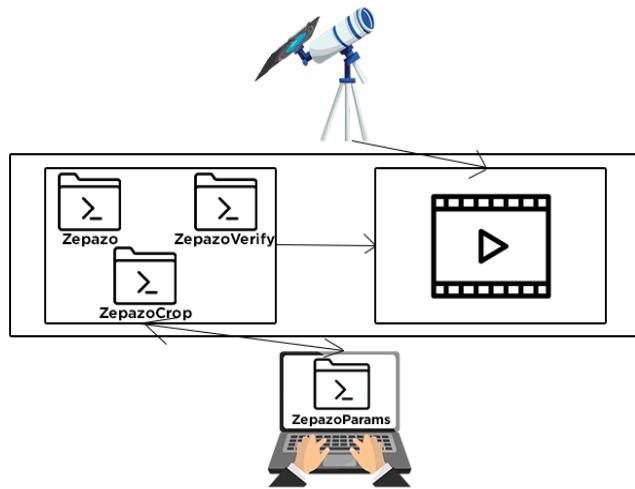


Figura 4.3: Diagrama objetivo del diseño

En la figura anterior podemos observar como funcionaría el sistema completo de detección de impactos:

- En primer lugar los vídeos capturados a través de telescopios serían almacenados en un servidor donde adicionalmente encontraríamos tres de los cuatro programas, Zepazo, ZepazoVerify, ZepazoCrop.
- El conjunto de programas que se encuentra en el servidor tiene acceso a los vídeos completos.
- El usuario accederá al servidor para ejecutar cualquiera de los programas indicados.
- El usuario en su estación de trabajo contará con el programa ZepazoParams. Para hacer uso de este programa accederá al servidor, cortará un trozo de vídeo con el programa ZepazoCrop y se lo descargará, lo cargará en ZepazoParams y ajustará los parámetros. Una vez configurados los parámetros ejecutará el programa principal Zepazo indicando únicamente el archivo de configuración exportado de ZepazoParams.

Adicionalmente si se trata de un usuario experimentado no será necesario utilizar ZepazoParams y podrá ejecutarse con parámetros el programa Zepazo.

- Una vez analizado el vídeo el usuario podrá validar los impactos de varios vídeos haciendo uso del programa ZepazoVerify.

Con este diseño basado en diversos programas divididos entre el cliente y el servidor conseguimos las siguientes **ventajas**:

- **Separación de funcionalidades:** Cada programa realiza una tarea única:
 - Zepazo: Analizar.
 - ZepazoParams: Elección asistida de parámetros.
 - ZepazoVerify: Verificar/validar impactos
 - ZepazoCrop: Recortar vídeos.
- **Menor sobrecarga del equipo del cliente:** El cliente únicamente ejecutará en su equipo el programa ZepazoParams el cual consta de una sencilla interfaz de apoyo para la selección de parámetros y no consta de funciones complejas que tomen mucho tiempo de procesamiento.
- **Menor sobrecarga de red:** El usuario no necesita descargar los vídeos completos, con un fragmento del mismo, recortado con el programa ZepazoCrop, es capaz de seleccionar los parámetros en el programa ZepazoParams.
- Reducción del tiempo de uso del equipo del cliente: El cliente como se ha comentado localmente hará uso de ZepazoParams únicamente. No será necesario que ejecute en su equipo el programa Zepazo encargado del análisis que puede llegar a demorarse mucho tiempo en función de las duraciones de las grabaciones que se procesen.
- **Unificación de la información:** Toda la información, como son los vídeos como los logs obtenidos tras el análisis y verificación se encuentran en el servidor, y podrán ser accedidos por los programas que los necesiten que son justamente los que se encuentran en el servidor.

4.2.1. Diseño Zepazo

Zepazo es el nombre que recibe el programa principal **encargado de analizar un determinado vídeo** en busca de los posibles impactos que hayan podido tomar lugar en el mismo. De esta forma con el desarrollo de este programa principal se pretenden abordar los requisitos planteados durante el análisis relacionados con el **primer caso de uso**. En este apartado no entraremos en detalles concretos de la implementación, esto se hará en el capítulo siguiente 'Implementación'.

De esta forma para el diseño de este sistema se partieron de los siguientes **conceptos**:

- **Analizador de vídeo:** Representa un analizador de vídeo donde encontramos todas las funciones relacionadas con el propio análisis del vídeo así como aplicación de máscaras, generación del resumen final, obtención del tiempo real donde toman lugar impactos entre otras funcionalidades.
- **Analizador de imagen:** Representa un analizador de imágenes y es utilizado para realizar operaciones sobre cada uno de los fotogramas del vídeo incluyendo funciones relacionadas con la dilatación, resta, marcado del contorno lunar en los diversos fotogramas que forman un vídeo, etc., que serán utilizados por el analizador de vídeo.
- **Impacto:** Representa un impacto que pueda tener lugar dentro de una grabación y servirá para almacenar información acerca del mismo.

De esta forma la arquitectura de este software va a estar formada por **tres paquetes** principalmente, un **primer paquete** formado por los analizadores donde encontramos los anteriormente mencionados analizadores de vídeo e imagen. Un **segundo paquete** formado por la clase impacto así como el programa principal que se encuentra en el fichero zepazo.py que hará uso del bloque de analizadores para llevar a cabo la tarea de obtener los impactos así como de las clases que encontramos en el **tercer paquete** encargado de la persistencia donde se intentan conseguir los dos principios principales de la persistencia como son:

- **Inyección de dependencias:** Se inyecta una dependencia a la clase encargada de realizar el análisis con el objetivo de acceder a los datos persistentes.
- **Única fuente de verdad:** Dicho objeto que es inyectado será el único que acceda a los datos.

De esta forma encontramos un diagrama de clases como el siguiente:

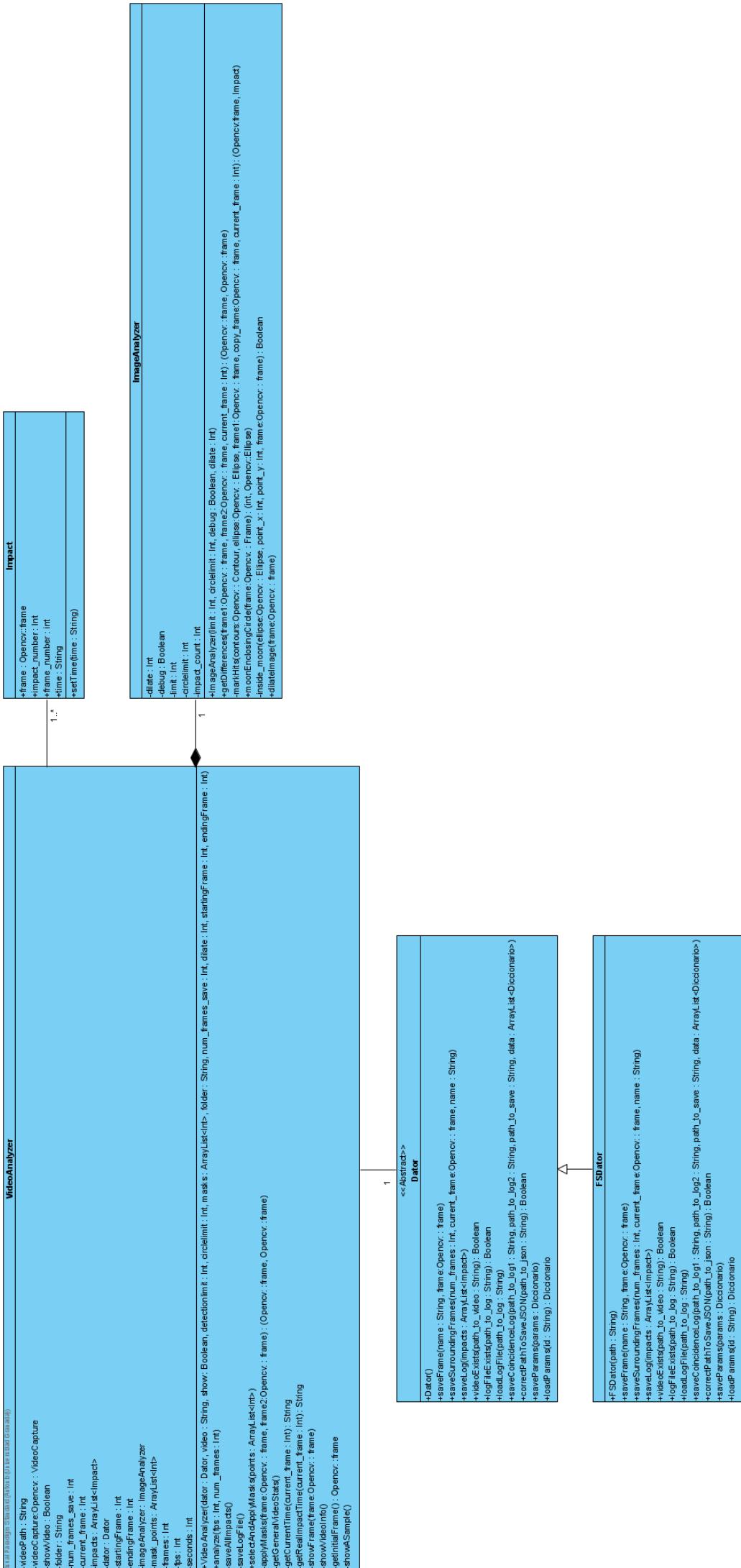


Figura 4.4: Diagrama Clases Zepazo

Y el diagrama de paquetes tal y como se ha comentado anteriormente sería el siguiente:

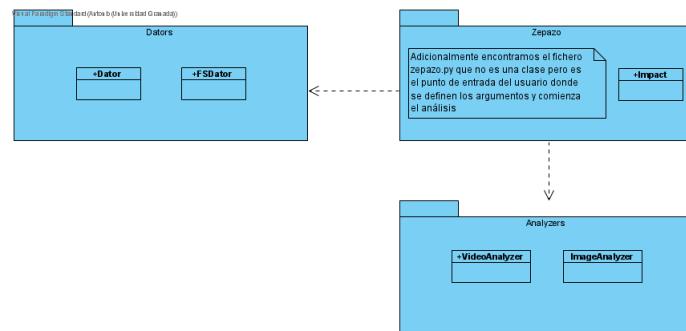


Figura 4.5: Diagrama Paquetes Zepazo

Diagrama actividad script 'zepazo.py':

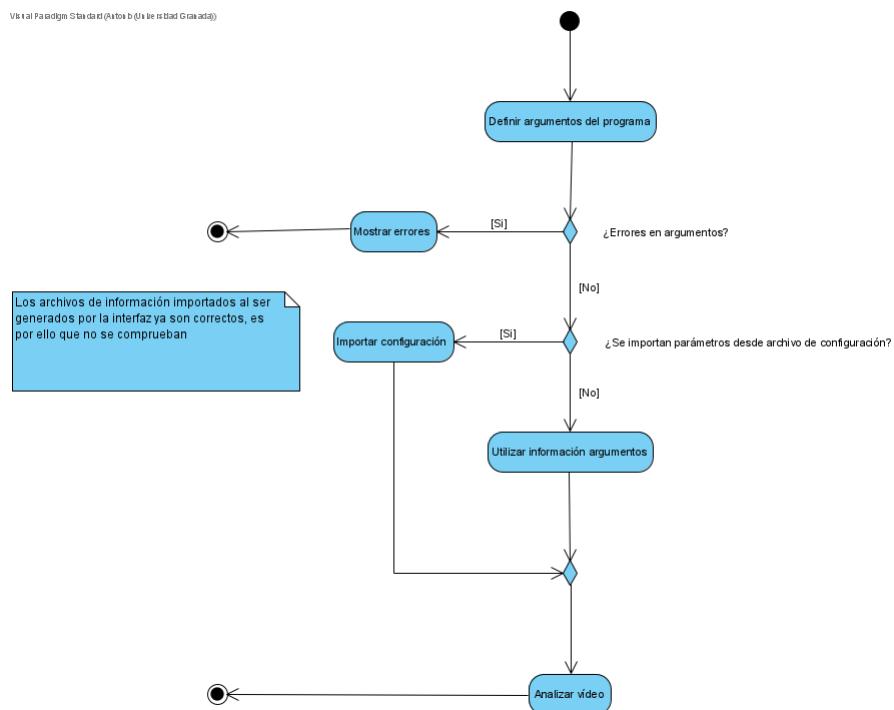


Figura 4.6: Diagrama Actividad Script zepazo.py

4.2.2. Diseño interfaz selección de parámetros

Este segundo programa llamado *ZepazoParams* pretende dar solución al problema de la dificultad por parte de un usuario de seleccionar de forma correcta los parámetros necesarios para poder ejecutar el análisis de un vídeo de forma exitosa. La elección de parámetros como se verá en el capítulo de implementación es uno de los aspectos más complejos de la aplicación ya que se requiere mucha exactitud para que el análisis sea exitoso y suelen ser valores enteros difíciles de determinar para un nuevo usuario, es por ello que este programa de forma muy sencilla permite a través de una interfaz elegir dichos parámetros e ir viendo en tiempo real como afectarían a la fase de análisis para posteriormente utilizar dichos parámetros establecidos en el programa 'Zepazo' comentado en la sección anterior.

De esta forma encontraremos una clase nueva llamada *ZepazoParams*, donde se encuentra definida la interfaz con todos sus componentes así como los métodos que son llamados cuando se generan ciertos eventos (entraremos más en detalle en el capítulo siguiente). De esta forma, en primer lugar se presenta el boceto realizado inicialmente para la creación de la interfaz:

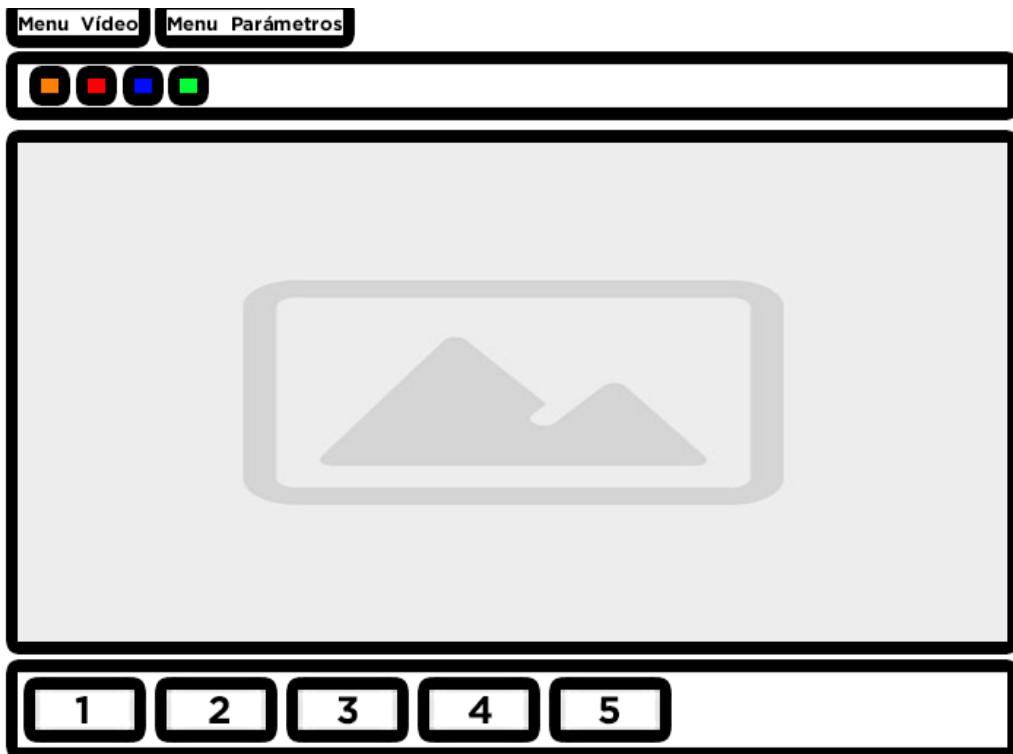


Figura 4.7: Diseño Interfaz

Como se puede ver en la figura anterior, la interfaz **consta de cuatro bloques** principales, en la parte superior, el **primer bloque** está formado por dos menús desplegables, el menú de vídeo para seleccionar un vídeo y el menú de parámetros para exportar los parámetros a JSON e importar desde JSON.

En el **segundo bloque** donde encontramos los botones con códigos de colores, donde irán botones relacionados con funciones como la previsualización de todos los parámetros aplicados a un frame, restablecer el valor de todos los parámetros, mostrar la previsualización del comando generado y mostrar un fragmento de visualización del proceso de análisis.

En el **tercer bloque**, donde se encuentra el símbolo de una imagen se mostraría el primer fotograma del vídeo seleccionado donde se verían los resultados de las variaciones de los parámetros aplicados a dicho fotograma.

En el **cuarto bloque**, el que se encuentra en la parte inferior, encontraremos un total de cinco secciones cada una dedicada al ajuste de uno de los parámetros de nuestro programa. A continuación se detalla el contenido de cada una de las secciones numeradas:

- En la sección numerada como **1**, encontraremos una casilla para indicar numéricamente el valor del índice de detección que podrá previsualizarse con el botón para mostrar una muestra del análisis del segundo bloque.
- En la sección numerada como **2**, encontraremos otra casilla para indicar numéricamente el valor del índice de detección de la elipse así como un botón para intentar ajustarlo automáticamente.
- En la sección numerada como **3**, encontraremos una sección dedicada a ajustar el valor de la dilatación de fotogramas, de esta forma contaremos con un botón para activarla y entonces se habilitará una casilla para indicar un valor numérico de dilatación.
- En la sección numerada como **4**, encontraremos una sección dedicada a las máscaras donde contaremos con botones para añadir, eliminar una o eliminar todas las máscaras.
- En la sección numerada como **5**, encontraremos una sección dedicada a todo lo relacionado con el guardado de fotogramas como por ejemplo un botón para seleccionar una carpeta donde guardar los fotogramas así como una casilla numérica para indicar el número de fotogramas previos y posteriores a guardar alrededor del impacto.

El diagrama de clases diseñado para este programa de selección de parámetros es el siguiente:

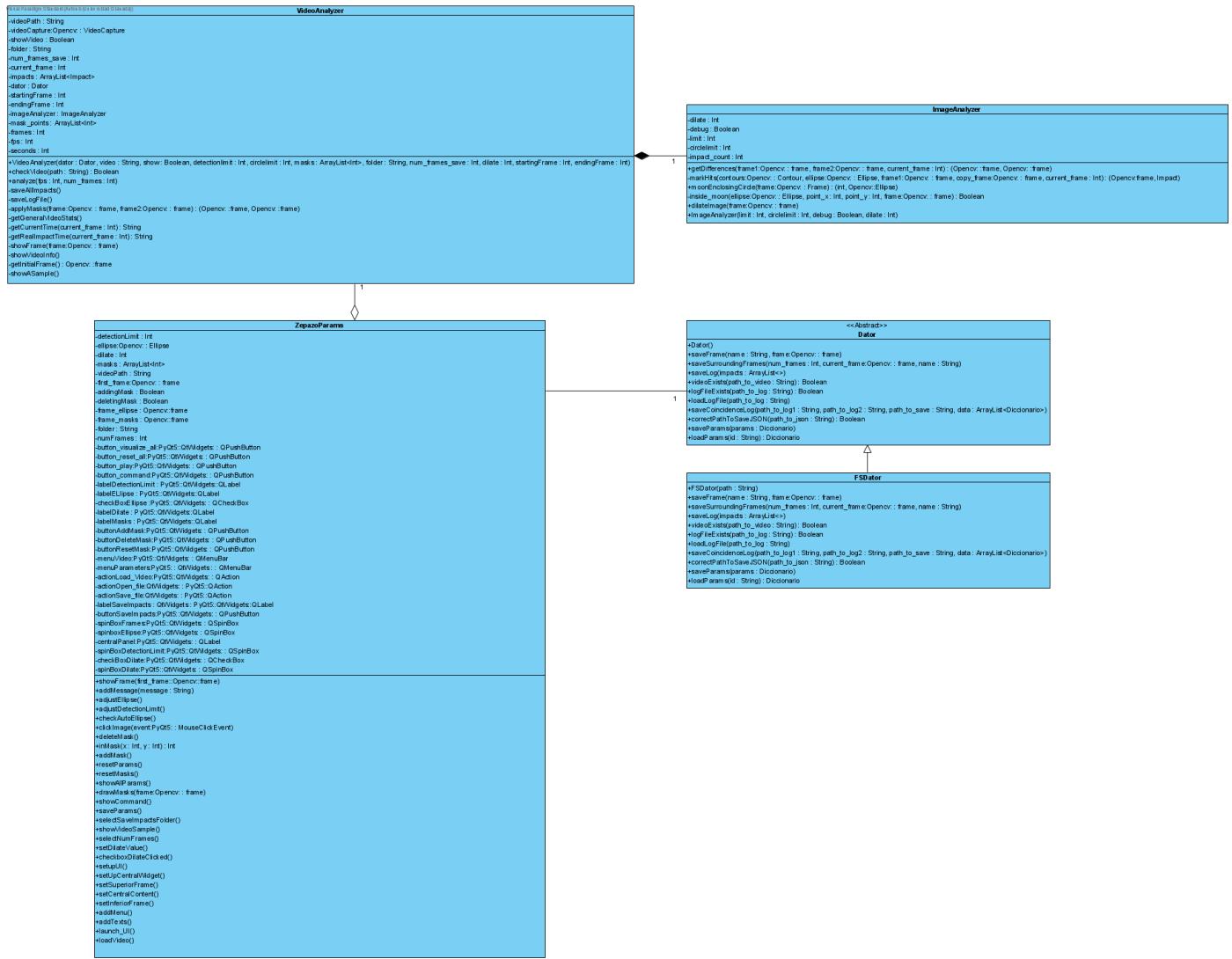


Figura 4.8: Diagrama Clases Zepazo Parameter Interface

Como podemos ver en la figura anterior, se ha diseñado una nueva clase que va a contener toda la interfaz y su funcionalidad. Es por ello que en esta clase encontramos dos tipos de **atributos**, el primer tipo está relacionado con los parámetros que se ajustarán de tal forma que encontramos atributos como el límite de detección, una lista de máscaras, la carpeta donde guardar los fotogramas, etc. El segundo bloque de parámetros están relacionados con elementos de la interfaz como pueden ser botones, cajas de verificación, menús, etc.

En cuando a los **métodos** se han definido los necesarios tanto para definir la propia interfaz como aquellos necesarios para enlazarse a los diversos eventos que aparecen consecuencia de las acciones realizadas por los usuarios y cuya lógica reside mayormente en métodos de la clase 'VideoAnalyzer'.

Y el diagrama de paquetes es el siguiente donde podemos ver el mismo paquete que en el programa 'Zepazo' llamado 'Analyzers' que contiene ambos gestores de vídeo e imagen así como un paquete nuevo que encapsula la interfaz y que hace uso de las clases del paquete 'Analyzers':

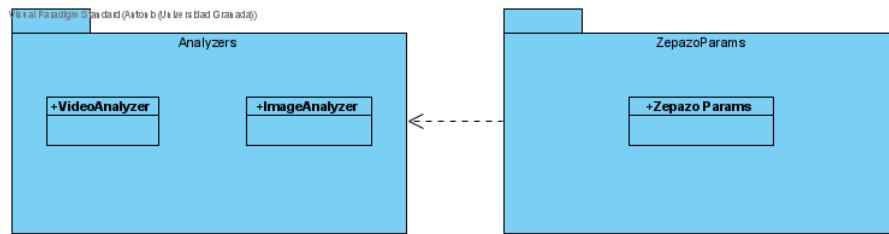


Figura 4.9: Diagrama Paquetes Zepazo Parameter Interface

4.2.3. Diseño validador de impactos

Este programa que recibe el nombre de **ZepazoVerify** pretende dar solución al problema de la verificación/validación de impactos. Esto es dados los resultados de dos vídeos analizados distintos pero grabados en momentos temporales similares intentar detectar si existe alguna coincidencia temporal entre los impactos encontrados en ambos vídeos, de esta forma si conseguimos identificar un impacto en ambos vídeos en un mismo momento temporal diremos que dicho impacto ha sido validado ya que se descarta que la detección sea fruto de un falso positivo al ocurrir en dos grabaciones distintas simultáneamente.

El objetivo principal de este programa es que fuera una herramienta muy sencilla, de esta forma se ha creado un pequeño programa que se puede encontrar en el fichero 'ZepazoVerify.py'. De esta forma no se ha necesitado utilizar un diseño orientado a objetos y en consecuencia no encontramos clases para realizar un diagrama de clases. Es por ello que se presenta a continuación el diseño de su comportamiento mediante un **diagrama de actividad**.

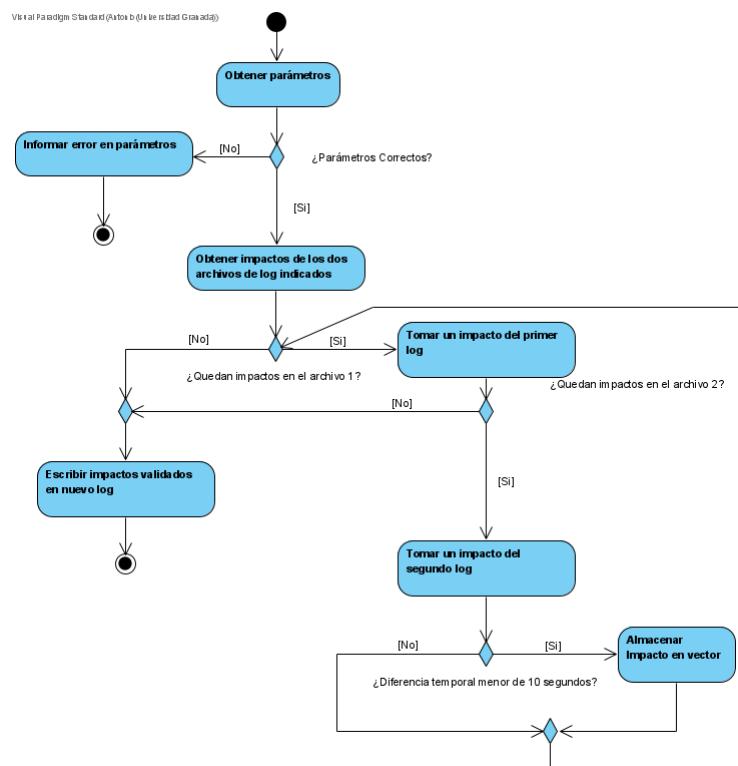


Figura 4.10: Diagrama Actividad Zepazo Verify

4.2.4. Diseño cortador de vídeos

Este último programa del conjunto de programas Zepazo llamado **ZepazoCrop** pretende aportar solución para el hecho de que los vídeos se suelen encontrar en un servidor de ficheros almacenados y descargar constantemente los vídeos completos de largas horas de duración puede suponer un coste en tiempo importante. Así esta sencilla herramienta nos permitirá recortar vídeos desde un determinado fotograma hasta otro lo cual puede ser útil para obtener también un pequeño vídeo alrededor de un determinado impacto detectado por el programa principal Zepazo.

Al igual que ocurría con el programa anterior, al ser un programa tan sencillo no ha sido necesario utilizar/diseñar ninguna clase de tal forma que el diseño del programa se muestra a continuación a través de un **diagrama de actividad**:

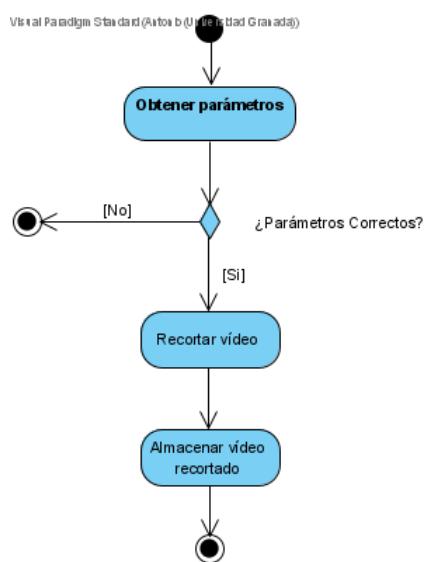


Figura 4.11: Diagrama Actividad Zepazo Crop

Capítulo 5: Implementación

En este capítulo se presenta en detalle la implementación de cada uno de los programas, es por ello que este capítulo se divide en cuatro bloques principales donde se discutirán las aspectos principales de la implementación de cada una de ellas.

5.1. Zepazo

Zepazo es el programa principal del conjunto de programas desarrollado y su objetivo es el análisis de un determinado vídeo aportado por un usuario con el fin de obtener los posibles impactos que hayan tomado lugar durante la grabación. Con el desarrollo de esta clase se pretenden abordar los **requisitos RF1 -RF13 3.6.1**. Para ello, tal y como se mostró en la figura 4.4 encontramos un total de **cinco clases desarrolladas y un script** que se presentan a continuación junto a detalles concretos de la implementación.

Este programa se ejecuta a través de la línea de comandos de la siguiente forma:

```
python3 src/zepazo.py [lista de argumentos]
```

Donde **lista de argumentos** en un conjunto de argumentos a seleccionar por el cliente entre los que encontramos:

- **-h, --help**: Ayuda de parámetros. Se muestran los posibles parámetros que el programa puede utilizar.
- **-v, --video**: Ruta del vídeo a analizar. **Tipo**: Cadena de caracteres.
- **-d, --debug**: Mostrar una simulación del proceso de análisis con los parámetros indicados, no se guardan los posibles impactos detectados. **Tipo**: Boolean.
- **-cm, --coordinatesmask**: Lista de puntos indicando las coordenadas de las máscaras a añadir. Cada dos elementos de la lista indicará una máscara, de esta forma, el primer elemento de cada par de coordenadas será la esquina superior izquierda del rectángulo y el segundo será la esquina inferior derecha. **Tipo**: Lista de enteros.
- **-l, --detectionlimit**: Índice de detección de impactos, este valor depende del brillo del vídeo introducido. **Tipo**: entero positivo.
- **-cl, --circlelimit**: Valor que sirve para ajustar la elipse al contorno lunar, si no se establece el argumento, se calculará un valor por defecto. **Tipo**: entero positivo.

- **-f, --folder:** Ruta de la carpeta donde almacenar los fotogramas de los impactos detectados. **Tipo:** Cadena de caracteres.
- **-ssf, --saveSurroundingFrames:** Número de fotogramas previos y posteriores al impacto que se desean almacenar, por defecto únicamente se almacena el fotograma del impacto. **Tipo:** entero positivo
- **-dt, --dilate:** Valor del kernel a utilizar para dilatar los fotogramas previos y evitar falsos positivos derivados de movimiento y zonas luminosas de la luna. **Tipo:** entero positivo.
- **-cf, --configFile:** Ruta de un archivo de configuración generado por ZepazoParams para importar valores de los parámetros. **Tipo:** Cadena de caracteres.
- **-sf, --startingFrame:** Número de fotograma desde el que comenzar el análisis. **Tipo:** Entero positivo.
- **-ef, --endingFrame:** Número de fotograma hasta el que realizar el análisis. **Tipo:** Entero positivo.

Nota: Cuando se hace uso del siguiente comando para obtener ayuda acerca de los parámetros se le mostrará en inglés:

```
python3 src/zepazo.py --help
```

Para poder utilizar argumentos a través de la línea de comandos se ha hecho uso de un módulo de Python llamado **Argparse** que nos permite de forma muy sencilla escribir interfaces de línea de comandos así como parsear dichos argumentos a valores usables por un programa escrito en Python [10].

Así el flujo de **actividad de este script** consiste en que una vez el usuario introduce por línea de órdenes el comando completo con los parámetros necesarios se procederá a realizar lo siguiente en el script:

- Se comprueban si son válidos los parámetros, si no son válidos se informa de error y se sale del programa.
- Se crea un objeto de tipo VideoAnalyzer con los parámetros seleccionados.
- Se llama al método que 'analyze' de dicha clase para analizar el vídeo.
- Termina la ejecución.

Para poder comprender el proceso de análisis iniciado por el método 'analyze' es necesario detallar los diferentes atributos y métodos de las clases utilizadas durante dicho proceso, se muestran a continuación.

5.1.1. Clase Impact

La clase Impact es una sencilla clase que contiene la información necesaria para identificar los impactos en una determinada grabación.

Atributos

- **frame**: Fotograma en el que ha tenido lugar el impacto. **Tipo**: Opencv::Frame.
- **impact_number** : Número de impacto. Empieza en 0 y se incrementa en una unidad con cada impacto encontrado. **Tipo**: Int.
- **frame_number** : Número de fotograma en el que ha tenido lugar el impacto. **Tipo**: Int.
- **time** : Momento temporal real en el que se produjo el impacto. **Tipo**: String.

Métodos

- **__init__(self, frame, impact_number, frame_number)**:
Constructor de la clase que inicializa los valores de los atributos gracias a los pasados como parámetro menos el valor time, ya que en la construcción del objeto no es posible saberlo, se calcula posteriormente como se verá en las siguientes secciones.
- **setTime(self, time)**:
Método que nos permite dotar de valor al atributo 'time' de la clase.

Esta clase nos permitirá crear un objeto de tipo 'Impact' e ir almacenándolos en un vector de tal forma que al terminar el análisis se guarden en memoria los impactos así como se genere un archivo de log como se verá en la descripción de la implementación de la clase 'ImageAnalyzer' y 'VideoAnalyzer'.

5.1.2. Clase ImageAnalyzer

La clase ImageAnalyzer es una clase que nos permite realizar operaciones sobre imágenes y es utilizada para realizar tratamientos sobre cada uno de los fotogramas del vídeo. De esta forma, *esta clase contiene la mayor parte de métodos importantes de este programa* que serán utilizados por la clase 'VideoAnalyzer' y serán explicadas en detalle en esta sección.

Esta clase principalmente trata de obtener las **diferencias entre dos fotogramas** mediante una simple operación de resta píxel a píxel entre dos fotogramas consecutivos (haciendo uso del método principal de la clase 'getDifferences') y aplicando una serie de operaciones que se describirán en el apartado de explicación de métodos para evitar en la medida de lo posible la detección de falsos positivos.

Atributos

- **dilate:** Valor obtenido por argumentos como 'Dilate', que indica el tamaño del kernel de dilatación a utilizar. **Tipo:** Int.
- **debug:** Valor obtenido por argumentos como 'Debug', que indica si se está ejecutando el programa en modo debug, si se ejecuta en modo debug no se almacenarán fotogramas y se devolverán fotogramas con marcas de impactos cuando ocurran así como con el contorno lunar dibujado para ver si se está realizando todo de forma correcta. **Tipo:** Boolean.
- **limit:** Es el límite de detección, valor obtenido por argumentos del programa como 'DetectionLimit' y es un valor que debe ajustarse en función del brillo que presente el vídeo. Por defecto si no se indica un valor se establece a 50 que por lo general ofrece buenos resultados.
- **circlelimit:** Valor obtenido por argumentos como 'CircleLimit' y es un valor que debe establecerse para poder ajustar una elipse irregular a la superficie lunar que aparece en el vídeo. Si no se indica un valor se intentará calcular por defecto, se verá a continuación en el apartado métodos. **Tipo:** Int.
- **impact_count:** Números de impactos detectados por este analizador de imágenes, útil para numerar los impactos ocurridos en el vídeo. **Tipo:** Int.

Métodos

- **__init__(self, limit, circlelimit, debug, dilate):**

Es el constructor de la clase que se encarga de asignar los valores recibidos por parámetros a los atributos de la clase a excepción de 'impact_count' que será calculado dinámicamente.

- **dilateImage(self, frame):**

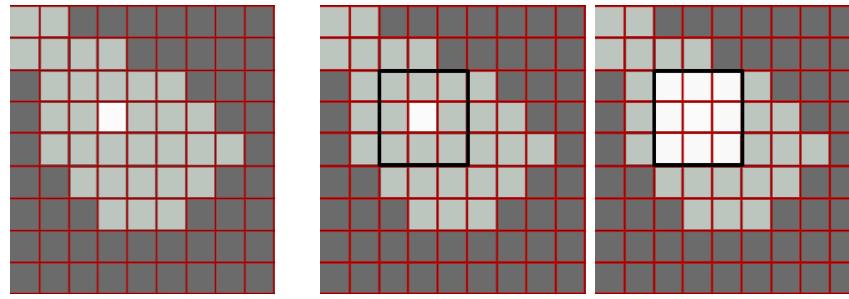
Este método permite dilatar los píxeles con valor '1', es decir, aquellos píxeles de color blanco de un determinado fotograma que coincidirán con las zonas iluminadas que aparezcan así como otros cuerpos brillantes como estrellas con el objetivo de evitar falsos positivos derivados de estas zonas.

Los falsos positivos ocurridos debido tanto zonas iluminadas de la luna como otros cuerpos brillantes que aparezcan en escena (estrellas normalmente) son consecuencia de su brillo irregular, lo que hace que se produzcan parpadeos entre diversos fotogramas y que al realizar la resta de fotogramas queden zonas marcadas en dichas partes, produciéndose la posterior detección de un falso impacto.

La **dilatación de imágenes** es una operación que incluye OpenCV mediante la operación 'dilate' [11] y que consiste en una iteración sobre los píxeles de la imagen donde en cada iteración se toma una matriz NxN de píxeles y se comprueba si en esa región existe algún píxel con valor 1 (píxel de color blanco) y en caso afirmativo todos los píxeles de dicha matriz se ponen a 1 (blanco), produciéndose un efecto de uniformidad en aquellas zonas blancas de la imagen que podrían parpadear. En nuestro caso la zona iluminada de la luna constantemente quedará de color blanco uniforme sin parpadeos al igual que el resto de cuerpos brillantes que aparezcan en la imagen eliminando su efecto de parpadeo.

La matriz NxN es denominada **kernel** y N en nuestro caso es el valor del atributo 'dilate', de tal forma que a mayor sea este valor mayor será la matriz y en consecuencia si utilizásemos valores muy altos, podríamos llegar incluso a tener una imagen totalmente blanca, se recomiendan valores cercanos a 4 cuando sea necesario utilizar esta opción.

Un ejemplo gráfico de dilación sería el siguiente, donde podemos ver una parte de una imagen con una zona iluminada donde podemos ver un píxel blanco:



(a) Píxel blanco

(b) Píxel blanco con ker-
nel 3x3 marcado

(c) Píxel blanco dilatado

Figura 5.1: Proceso dilatación píxel

Gracias a este método, si aparece alguna zona iluminada de la superficie lunar en la grabación, lo cual podría dar problemas ya que las zonas iluminadas poseen brillo intermitente y pueden producir falsos positivos. De esta forma observamos el mismo fotograma pero en dos casos diferentes, el primero sin aportar un valor de dilatación y en consecuencia no dilatando la imagen y el segundo dilatando la imagen (obsérvense detecciones de impacto marcadas en la primera imagen en la esquina superior izquierda):

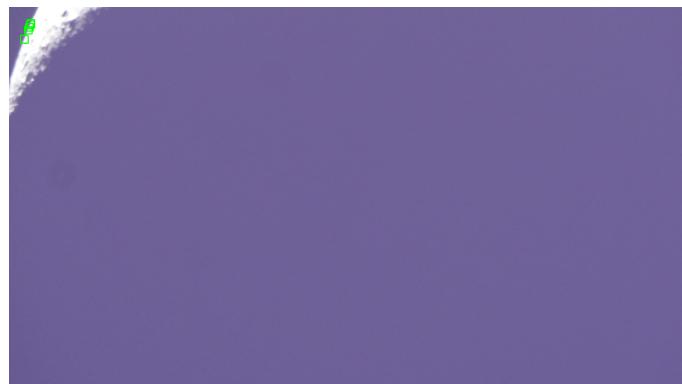


Figura 5.2: Fotograma detección impactos, falsos positivos en zona iluminada

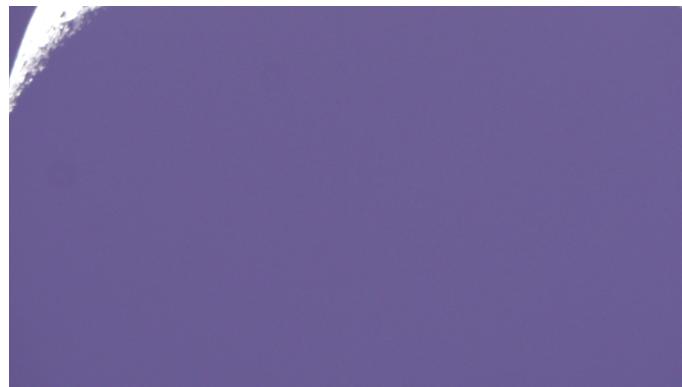


Figura 5.3: Mismo fotograma, dilatación aplicada, falsos positivos eliminados

Donde la codificación de esta función ha sido la siguiente:

```
1 def dilateImage( self , frame ):  
2     kernel = np . ones(( self . dilate , self . dilate ) , dtype=np  
3                         . float32 )  
4     dilated_frame = cv2 . dilate( frame , kernel , iterations  
5                                     = 1)  
6     return dilated_frame
```

Podemos observar que se define un kernel que es una matriz de dimensión (`self.dilate` x `self.dilate`) relleno a unos en la línea 2.

En la línea 3 podemos observar como se usa el método `dilate` de OpenCV sobre el fotograma pasado como parámetro haciendo uso del kernel anterior con una única iteración.

- **moonEnclosingCircle(self, frame):**

Este método se encarga de dado un fotograma del vídeo intentar obtener una elipse (puede ser con ejes rotados), que se ajuste al contorno lunar con el objetivo de hacer uso del método `'inside_moon'` que nos permitirá descartar falsos positivos fuera de la superficie lunar.

En la figura anterior podemos observar de forma esquemática a través de un boceto como se ha ajustado una elipse (contorno amarillo alrededor de la superficie lunar) en un determinado fotograma del vídeo. Esto se realizará para posteriormente descartar aquellas detecciones de impactos fuera de la superficie lunar tal y como se comentó previamente.

Para poder llegar al objetivo final de este método que es obtener la elipse que se ajusta al contorno lunar debemos realizar una serie de pasos que explicaremos apoyándonos en el código:

```

1 def moonEnclosingCircle(self, frame):
2     grayFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
3
4     if(self.circlelimit != None):
5         umbral = int(self.circlelimit)
6     else:
7         umbral = np.average(grayFrame) - np.std(
8             grayFrame)
9
10    _, threshed_moon = cv2.threshold(np.array(grayFrame),
11                                     dtype=np.uint8), umbral, 255, cv2.
12                                     THRESH_BINARY)
13
14    kernel = np.ones((5,5),np.uint8)
15
16    closing = cv2.morphologyEx(threshed_moon, cv2.
17                               MORPH_CLOSE, kernel)
18
19    moon_contour, _ = cv2.findContours(closing, cv2.
20                                       RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
21
22    c = []
23    if(len(moon_contour) > 0):
24        c = max(moon_contour, key=cv2.contourArea)
25
26    if(len(c) > 5):
27        ellipse = cv2.fitEllipse(c)
28
29    if(umbral < 5 or umbral > 70):
30        ellipse = None
31
32    if(len(c) > 5 and ellipse != None):
33        return umbral, ellipse
34    else:
35        return 0, None

```

Como podemos ver el método recibe un determinado fotograma del vídeo sobre el que debemos calcular la elipse que se ajusta a la superficie lunar que aparece en el fotograma y procede de la siguiente forma:

- **Paso 1:** En primer lugar convertimos a escala de grises el fotograma que hemos obtenido por parámetros tal y como se muestra en la **Línea 2** haciendo uso del método cvtColor indicándole el fotograma y el nuevo espacio de color [12]. Esto es necesario para poder utilizar los métodos de OpenCV que se muestran en los siguientes pasos.
- **Paso 2:** A continuación si el usuario ha definido un valor para el argumento 'CircleLimit' se usará dicho valor a continuación en las siguientes líneas y si no se hace una estimación que se aproxima relativamente bien a este valor como es la media de los valores de color de los píxeles del fotograma en escala de grises - desviación típica de los valores de color de los píxeles. Esto se observa en las **Líneas 4-7**.
- **Paso 3:** En siguiente lugar se hace uso de la función de OpenCV threshold [13] que tiene el siguiente comportamiento: Dado un determinado valor entero como umbral, se itera sobre cada píxel de la imagen de tal manera que todos los píxeles que superen al umbral (valor anteriormente obtenido) se establece su valor a un valor máximo indicado por el tercer argumento de la función y en caso de que sea menor que el umbral se establece a 0. **Línea 9**. Podría representarse por una función como la siguiente:

$$\text{threshold}(x) = \begin{cases} 0 \text{ (Blanco)} & \text{si Color Píxel(x)} < \text{umbral} \\ \text{Tercer Argumento (Negro)} & \text{si Color Píxel(x)} \geq \text{umbral} \end{cases}$$

El valor del umbral deberá ser un valor cercano a la media de los valores de los colores de los píxeles de la zona de la superficie lunar y normalmente será seleccionado por el usuario. Esta operación obtiene un fotograma donde únicamente hay dos colores, o blancos o negro, de tal forma que debería quedar la luna en blanco y la parte exterior en negro si se ha ajustado correctamente el valor umbral.

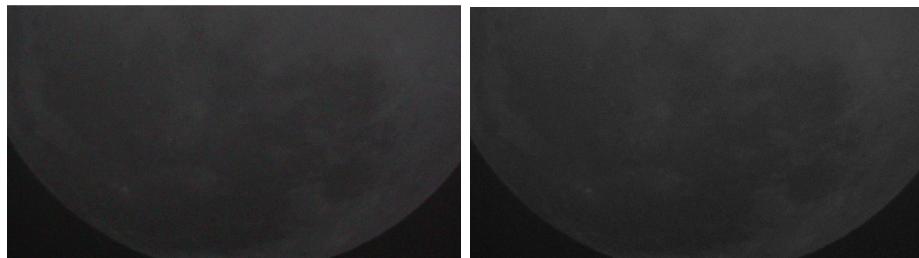
- **Paso 4:** Por motivos de rendimiento únicamente, se aplica una operación adicional de OpenCV llamada closing [11] que es una especie de dilatación que rellena la zona determinada por un kernel de color blanco pero únicamente cuando la mayor parte de los píxeles seleccionados por el kernel del fotograma son de color blanco. Con esto conseguimos adicionalmente que el resultado obtenido por la operación anterior de thresholding sea más preciso adicionalmente llenando huecos negros de la luna como blancos.

Líneas: 11-13.

- **Paso 5:** Se hace uso de otra función de OpenCV llamada findContours [14] que nos permite encontrar el contorno de una figura en una determinada imagen uniendo todos los puntos continuos que tienen el mismo color o intensidad. Recordemos que en este punto gracias a los pasos anteriores tenemos un fotograma con dos colores, blanco y negro, donde en blanco tenemos la superficie lunar y en negro el resto. De esta forma, si aplicamos la función de búsqueda de contornos obtendremos una lista de coordenadas con el contorno de la luna. **Línea: 15.** En este paso y en adelante ya no obtendremos nuevos fotogramas. Gracias a la operación de closing realizada en el paso anterior, la función findContours tiene que operar sobre una menor cantidad de puntos, de ahí que se mejore el rendimiento.

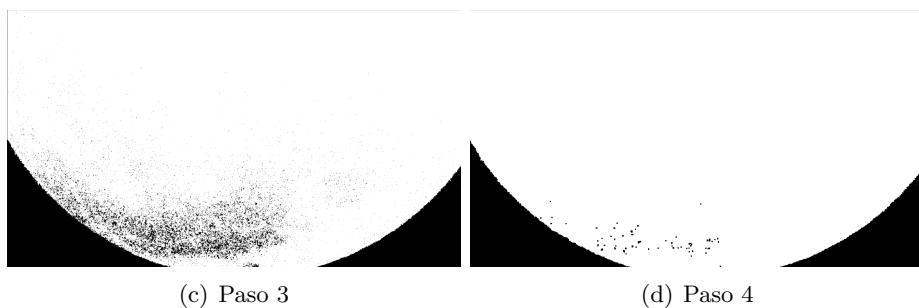
- **Paso 6:** Se comprueban ciertos valores como que al menos haya 5 puntos en el contorno que es mínimo necesario para construir una elipse con el método proporcionado por OpenCV o que el umbral no sea atípico (ni muy alto ni muy bajo) y si todo esto es correcto se devuelve tanto el umbral calculado como la elipse obtenida con el método fitEllipse al cual hay que pasarle una lista de puntos, los contornos. (**Nota:** el método findContours devuelve varias listas de posibles contornos, nos quedamos con la mayor y esta es la que recibe el método fitEllipse). El método fitEllipse devuelve un objeto de tipo OpenCV::Ellipse que podrá ser utilizado para ver si un impacto pertenece o no a dicha elipse. **Líneas 17-30.**

Se ilustra de forma gráfica con un fotograma real el funcionamiento paso a paso del método anterior (se muestran los pasos que generan frames):



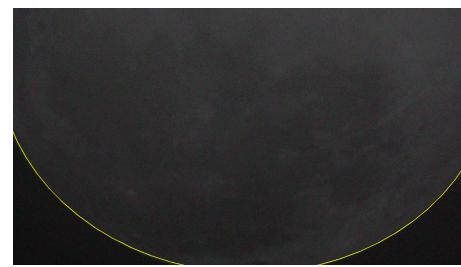
(a) Parámetro recibido: Fotograma

(b) Paso 1



(c) Paso 3

(d) Paso 4



(e) Elipse obtenida en paso 6 dibujada sobre fotograma

Figura 5.4: Funcionamiento método moonEnclosingCircle: Ejemplo real.

■ **inside_moon(self, ellipse, point_x, point_y):**

Este método como vemos recibe por parámetros una elipse, la cual ha sido obtenida por el método anterior y un punto de la imagen definido por sus coordenadas X,Y.

OpenCV a la hora de calcular la elipse con la función anteriormente vista 'fitEllipse' utilizado por el método 'moonEnclosingCircle' de nuestra clase devuelve una **elipse rotada**, esto es importante tenerlo en cuenta ya que la ecuación para comprobar si un punto pertenece a una elipse o no difiere si la elipse está o no rotada:

$$\frac{(\cos(\alpha)(x_p-x_0)+\sin(y_p-y_0))^2}{a^2} + \frac{(\sin(\alpha)(x_p-x_0)+\cos(y_p-y_0))^2}{b^2} \leq 1$$

Figura 5.5: Fórmula pertenencia punto a elipse rotada

Donde:

- y_p : Coordenada Y del punto a comprobar. Parámetro del método.
- x_p : Coordenada X del punto a comprobar. Parámetro del método
- y_0 : Coordenada Y del centro de la elipse. Se obtiene como $\text{ellipse}[0][0]$.
- x_0 : Coordenada X del centro de la elipse. Se obtiene como $\text{ellipse}[0][1]$.
- a : longitud del eje mayor. Se obtiene como $\text{ellipse}[1][0]$.
- b : longitud del eje menor. Se obtiene como $\text{ellipse}[1][1]$.

De esta forma, haciendo uso de la fórmula anterior el método es el siguiente:

```

1 def inside_moon( self , ellipse , point_x , point_y ) :
2     axis1 = ellipse [ 1 ][ 0 ]
3     axis2 = ellipse [ 1 ][ 1 ]
4
5     centerX = ellipse [ 0 ][ 0 ]
6     centerY = ellipse [ 0 ][ 1 ]
7
8     angle = math . radians ( ellipse [ 2 ])
9
10    x = ( pow (( math . cos ( angle )) * ( point_x - centerX ) +
11           math . sin ( angle )) * ( point_y - centerY ) , 2 ) / pow (
12             axis1 / 2 , 2 ) + ( pow (( math . sin ( angle )) * ( point_x -
13               centerX ) - math . cos ( angle )) * ( point_y - centerY ) ,
14             2 ) / pow ( axis2 / 2 , 2 )
15
16    return x <= 1

```

- **markHits(self, contours, ellipse, frame1, copy_frame, current_frame):**

Este método será el encargado de dada una lista de puntos (impactos), comprobar si cada uno de los puntos pertenece o no a la superficie lunar llamando al método anterior, y en caso afirmativo **dibuja un rectángulo alrededor del posible impacto**. El código es muy sencillo y se explica a continuación:

- Se calcula la elipse que envuelve la Luna haciendo uso del método moonEnclosingCircle
- Si se pudo calcular una elipse: Por cada punto de la lista de puntos, se hace uso del método 'inside_moon':
 - Si el punto pertenece a la elipse, se dibuja un rectángulo de color rojo alrededor del mismo, si no está activado el modo debug se incrementa el atributo 'impact_count' y se crea un objeto de tipo impact que se añade a una lista.
 - Si el punto no pertenece a la elipse, se dibuja un rectángulo de color morado alrededor del mismo (útil para el modo debug).
- Si no se pudo calcular una elipse no podemos descartarlo: Se dibuja un rectángulo de color verde alrededor del mismo, si no está activado el modo debug se incrementa el atributo 'impact_count' y se crea un objeto de tipo impact que se añade a una lista.
- Se devuelve la lista de impactos así como el fotograma dibujado.

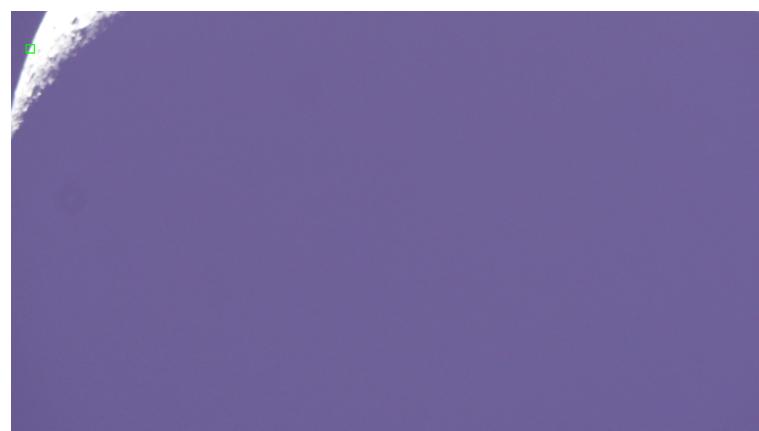
Nota: ¿Por qué se pasan por parámetros dos fotogramas? Es necesario para en uno de ellos ir dibujando impactos y sobre el otro se van calculando las elipses. Si calculásemos la elipse sobre un fotograma donde hay dibujado algún impacto podría producirse una alteración en el cálculo de dicha elipse.



(a) Contorno calculado: Posible impacto detectado (b) Zoom sobre posible impacto detectado



(c) Contorno calculado: Estrella exterior a la luna descartada



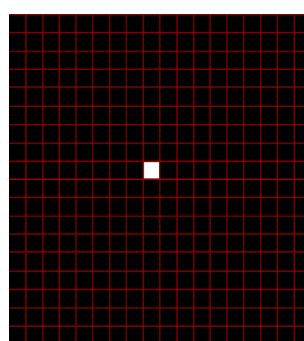
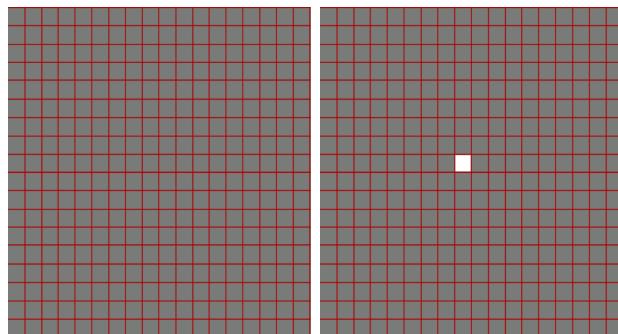
(d) Contorno no posible de calcular: Detectado posible impacto (esquina superior izquierda)

Figura 5.6: Funcionamiento método markHits: Ejemplo real.

- `getDifferences(self, frame1, frame2, current_frame):`

Este es el método principal de la clase 'ImageAnalyzer' y se encarga de la comparación de dos fotogramas con el objetivo de encontrar algún impacto en esos dos fotogramas, para ello realiza la resta de ambos.

Cuando hablamos de restar dos fotogramas nos referimos a la resta de los valores de color píxel a píxel entre ambas imágenes. Por ejemplo si tomamos un fotograma y lo restamos consigo mismo, al no existir diferencias entre ambas imágenes, la resta píxel a píxel produce una imagen totalmente negra. Esto puede ser utilizado para detectar impactos ya que las grabaciones se realizan con fast frames cameras, un tipo especial de cámaras que graban con una tasa de captura de entre 100 y 200 fotogramas por segundo lo que hace que la diferencia entre dos fotogramas consecutivos sea prácticamente nula y podamos conseguir detectar los impactos que suelen tomar lugar en décimas de segundo o menos, obteniendo el siguiente efecto:



(c) Resultado resta de fotogramas anteriores

Figura 5.7: Funcionamiento método getDifferences.

El código del método que nos ayudará a su explicación es el siguiente:

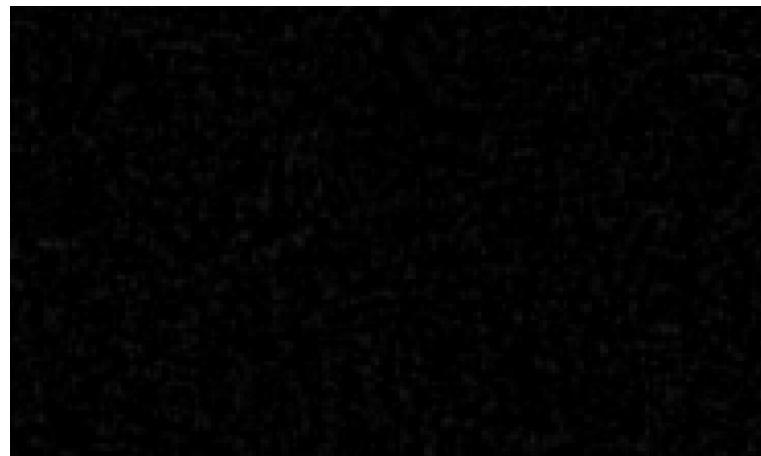
```
1 def getDifferences(self, frame1, frame2, current_frame):
2     difference = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
3
4     grayFrame1 = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)
5     grayFrame2 = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
6
7     if(self.dilate != None):
8         grayFrame2 = self.dilateImage(grayFrame2)
9
10    difference = cv2.subtract(grayFrame2, grayFrame1,
11                             difference)
12
13
14    _, threshed_img = cv2.threshold(np.array(difference,
15                                     dtype=np.uint8), self.limit, 255, cv2.THRESH_BINARY)
16
17    contours, _ = cv2.findContours(threshed_img, cv2.
18                                   RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
19
20    if(self.debug):
21        _, ellipse = self.moonEnclosingCircle(frame1)
22    else:
23        ellipse = None
24
25    copy_frame = frame1.copy()
26    if(self.debug == True and ellipse != None):
27        cv2.ellipse(copy_frame, ellipse,(0, 255, 255), 2)
28
29    copy_frame, detected_impact = self.markHits(contours,
30                                                 ellipse, frame1, copy_frame, current_frame)
31
32    if(self.debug):
33        return copy_frame, detected_impact
34    else:
35        return frame1, detected_impact
```

Observamos que este método que será llamado desde la clase 'VideoAnalyzer' que se explicará en la siguiente sección, recibe como **parámetros** dos fotogramas consecutivos y el número del primer fotograma.

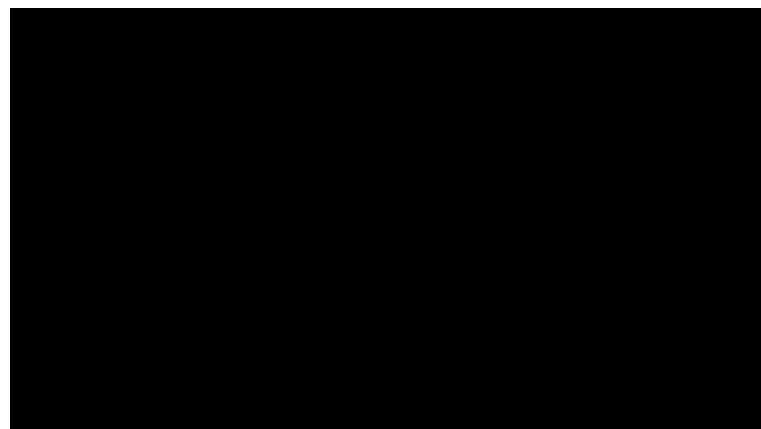
Los pasos que se siguen en este método son los siguientes:

- **Paso 1:** Se reserva memoria para el fotograma diferencia que se obtendrá en pasos posteriores. **Línea:** 2.
- **Paso 2:** Se convierten ambos fotogramas recibidos por parámetros del espacio de color RGB al espacio de color BGR2GRAY (escala de grises) con el objetivo de tener menos información en las imágenes y acelerar el proceso completo, ya que como sabemos en las imágenes RGB cada píxel almacena tres valores, uno por cada banda de color, mientras que en BGR2GRAY (escala de grises) únicamente almacenamos 1 que es un valor que va desde 0 a 255. **Líneas:** 4-5.
- **Paso 3:** Se comprueba si el usuario ha introducido el argumento 'Dilate', si lo ha elegido se dilata el segundo fotograma haciendo uso del **método 'dilateImage'** ya que la resta que se producirá más adelante será la del segundo fotograma menos la del primero. **Líneas:** 7-8.
- **Paso 4:** Se realiza la diferencia entre ambos fotogramas almacenándose el resultado en el fotograma del paso 1 donde se reservó memoria para ello. Para hacer la resta de los valores de cada píxel entre ambas imágenes se ha hecho uso del método de OpenCV 'subtract' [15] operación que para realizarse es necesario que ambas imágenes posean el mismo espacio de color así como dimensiones, aspectos que se cumplen siempre en nuestro programa. **Línea:** 11.
- **Paso 5:** Haciendo uso del método de OpenCV threshold sobre la imagen obtenida como diferencia de dos fotogramas consecutivos, pero esta vez tomando como **umbral el argumento 'Limit' que actúa como valor que indica la sensibilidad en la detección de impactos**, por defecto es 50 si no ha elegido nada el usuario ya que suele dar buenos resultados. Esto elimina ruido que aparece en la resta de fotogramas.

Con esto conseguimos quedarnos únicamente con aquellas partes de la imagen que superen dicho umbral. Recordemos que el threshold se ha aplicado sobre la imagen diferencia con lo cual esa imagen contenía a parte de posibles impactos ruido que puede interferir en la correcta detección de impactos, con esta operación eliminamos dicho ruido. **Línea:** 14.



(a) Resta de dos fotograma con zoom aplicado (obsérvese el ruido)



(b) Resta de dos fotogramas + threshold aplicado (ruido eliminado)

Figura 5.8: Eliminación de ruido

- **Paso 6:** Tras el paso anterior, si hubiese ocurrido un impacto quedaría un fotograma totalmente negro donde podría observarse una marca blanca, el impacto. Es por ello que ahora mediante la función de OpenCV 'findContours' que devuelve una lista de puntos que son las coordenadas del contorno del impacto. **Línea:** 16.
- **Paso 7:** En las **líneas** 18-25 en caso de que el modo debug esté activado se calcula para cada fotograma la elipse para que el usuario pueda verla. La elipse si no está el modo debug activado no se calcula en cada fotograma, únicamente cuando se detecta un impacto en el método 'markHits' para comprobar si el impacto se encuentra dentro de la superficie lunar, lo que supone una gran mejora de rendimiento con respecto a estar calculando para cada fotograma.

- **Paso 8:** Se llama al **método 'markHits'** pasándole la lista de puntos del contorno para que en caso de que la lista tenga un tamaño mayor que 0 (hay posibles impactos) se marquen en los fotogramas y devuelva la lista de impactos encontrados en dicho fotograma, normalmente un impacto si la lista de contornos contiene puntos. **Línea:** 27.
- **Paso 9:** Se devuelve el fotograma con el impacto marcado así como la lista de objetos impactos (normalmente uno) que han tomado lugar en dicha imagen. **Líneas:** 29-32.

5.1.3. Clase VideoAnalyzer

La clase VideoAnalyzer nos permite trabajar con vídeos realizando ciertas operaciones sobre los mismos. De esta forma, esta clase nos permitirá *cargar un determinado vídeo y extraer sus fotogramas para analizarlos con métodos de la clase 'ImageAnalyzer'* para finalmente generar un log que nos permita tener un resumen de lo que ha pasado en el vídeo entre otras funciones como aplicar máscaras, guardar en memoria las imágenes de los posibles impactos, etc.

Atributos

- **videoPath:** Ruta del vídeo que se va a analizar en busca de impactos obtenido a través del argumento 'Video'. **Tipo:** String.
- **videoCapture:** Objeto de OpenCV [16] que nos permitirá ir leyendo los fotogramas de un determinado vídeo. **Tipo:** OpenCV::VideoCapture.
- **showVideo:** Atributo que indica si el análisis se está ejecutando o no en modo debug. Indicado por el argumento 'Debug'. **Tipo:** Boolean.
- **folder:** Ruta de la carpeta donde guardar los impactos, si no se indica haciendo uso del parámetro 'Folder' se guardan por defecto en la misma carpeta donde se encuentra el vídeo a analizar. **Tipo:** String.
- **num_frames_save:** Número de fotogramas previos y posteriores a guardar, por defecto ninguno. Indicado por el parámetro 'SaveSurrounding Frames'. **Tipo:** Int
- **current_frame:** Número del fotograma que actualmente se está analizando. **Tipo:** Int.
- **impacts:** Lista de objetos de tipo 'Impact' que representa la lista de posibles impactos detectados en el vídeo que se está analizando. **Tipo:** ArrayList<Impact>.

- **dator:** Objeto de tipo FSDator aunque podría ser cualquier clase que implemente los métodos de la clase abstracta 'Dator' que nos sirve como interfaz de acceso a la persistencia. **Tipo:** Dator.
- **startingFrame:** Número de fotograma desde el que empezar a analizar, por defecto el primero. Se recibe mediante el argumento 'StartingFrame'. **Tipo:** Int.
- **endingFrame:** Número de fotograma hasta el que analizar, por defecto el último. Se recibe mediante el argumento 'EndingFrame'. **Tipo:** Int.
- **imageAnalyzer:** Objeto de la clase 'ImageAnalyzer' que nos servirá para aplicar operaciones sobre cada uno de los fotogramas. **Tipo:** ImageAnalyzer.
- **mask_points:** Vector de coordenadas de tal forma que cada cuatro posiciones se forma una máscara rectangular. Ejemplo: [0,1,2,3], se establecerá un rectángulo donde la esquina superior izquierda se encuentra en las coordenadas (0,1) y la esquina superior derecha en las coordenadas (2,3). Se recibe mediante el argumento 'CoordinateMask'. **Tipo:** ArrayList<Int>.
- **frames:** Número de fotogramas del vídeo, se calcula dinámicamente. **Tipo:** Int.
- **fps:** Número de fotogramas por segundo del vídeo, se calcula dinámicamente. **Tipo:** Int.
- **seconds:** Duración del vídeo en segundos, se calcula dinámicamente. **Tipo:** Int.

Métodos

- `__init(dator, video, show, detectionlimit, circlelimit, mask, folder, num_frames_save, dilate, startingFrame, endingFrame)`:

Este método es el **constructor** de la clase VideoAnalyzer y se encarga en primer lugar de dotar de valor a los atributos de la clase que no son calculados dinámicamente a través de los argumentos recibidos, es decir se dota de valores a toda la lista anterior de atributos menos a los atributos fps, frames y seconds.

En cuanto a la asignación de los valores mencionados anteriormente destacamos algunos casos donde la asignación es peculiar:

- **videoCapture:** Como podemos indicarle desde que fotograma empezar a extraer del vídeo en función de si se ha asignado un fotograma de inicio por parámetros se creará a partir de dicho fotograma y en caso contrario comenzará desde el primer fotograma.
- **mask_points:** Si el usuario ha indicado una lista de enteros para establecer las máscaras se convierten haciendo uso del método 'selectAndApplyMasks' a una lista de parejas de puntos donde cada pareja es una coordenada de tal forma que dos parejas ahora definen una máscara.

Posteriormente, llamando al método `getGeneralVideoStats` obtiene y asigna valores a los atributos fps, grames y seconds.

Nota: Como puede verse se recibe un objeto de tipo **Dator** utilizado para llevar a cabo todas las operaciones relacionadas con persistencia. A esta clase se inyecta dicha dependencia consiguiendo así los principios de *inyección de dependencias* y *única fuente de verdad* comentados en el apartado de diseño.

- **getGeneralVideoStats():**

Este método haciendo uso del objeto OpenCV::VideoCapture almacenado en el atributo de la clase 'videoCapture' y su método 'get' se encarga de obtener y asignar los valores de los atributos fps, frames y seconds.

Para obtener dichos valores al método 'get' del objeto OpenCV::VideoCapture debemos pasarle un determinado flag [17]:

- **Obtener fps:** Se obtienen usando el flag CAP_PROP_FPS:

```
1 self.fps = int(cap.get(cv2.CAP_PROP_FPS))
```

- **Obtener frames:** Se obtienen usando el flag CAP_PROP_FRAME_COUNT:

```
1 self.frames = cap.get(cv2.CAP_PROP_FRAME_COUNT)
```

- **Obtener segundos:** Se calcula haciendo uso de los dos valores anteriores:

```
1 self.seconds = int(self.frames / self.fps)
```

- **showVideoInfo():**

Este método imprime por pantalla información del propio vídeo y será mostrada al iniciar el análisis de un vídeo. La información que se muestra es la siguiente:

- Ruta del vídeo a analizar.
- Fotogramas por segundo.
- Número total de fotogramas.
- Duración del vídeo en horas minutos y segundos.

- **selectAndApplyMask(points):**

Este método recibe como entrada una lista de enteros, tal y como se reciben por argumentos, que representan las máscaras a añadir. Este método agrupa los puntos en coordenadas para facilitar su utilización por el resto de métodos.

Por ejemplo, la entrada podría ser la siguiente:

```
1 [0, 1, 2, 3]
```

Y tal y como se ha comentado, la salida será una lista de coordenadas que facilita su posterior utilización:

```
1 [[0, 1], [2, 3]]
```

Con la lista de puntos anteriores conseguiríamos establecer una máscara con las siguientes coordenadas:

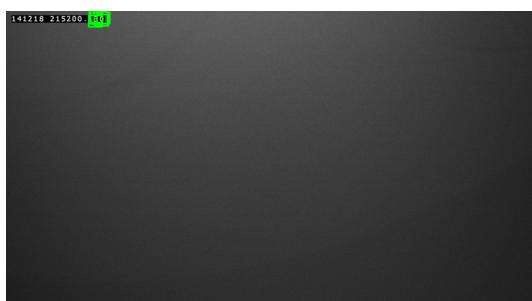
- Esquina superior izquierda con coordenadas (0,1).
- Esquina inferior derecha con coordenadas (2,3).

- **applyMasks(frame, frame2):**

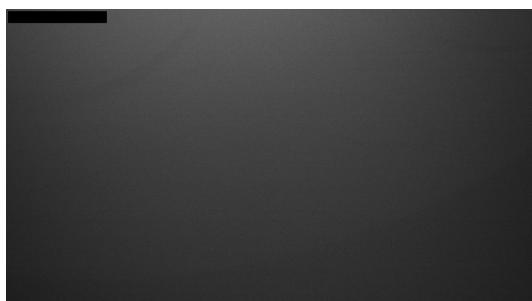
Dados dos fotogramas consecutivos de un vídeo se dibujan tantos rectángulos como máscaras hay definidas en el atributo de la clase `mask_points`, para ello se hace uso de la función de OpenCV 'rectangle' que dibuja un rectángulo pasándole las coordenadas de la esquina superior izquierda y la inferior derecha. Por último devuelve los dos fotogramas con la máscara aplicada.

Nota: *El color de la máscara es importante.* Esto es debido a que si por ejemplo tuviésemos una máscara de un color tal que al pasarlo a escala de grises y posterior thresholding (esto se hacía en la clase `ImageAnalyzer`) pasase a ser blanca, a la hora de dilatar la imagen se dilataría la máscara produciendo problemas de detección de falsos positivos en la propia máscara.

Un ejemplo muy útil del uso de máscaras es cuando contamos con temporizadores en el propio vídeo, al ser elementos que cambian cada cierto tiempo se detectarían como impactos si además se encuentran dentro del contorno lunar (o si no ha sido posible detectar el contorno) como ocurre en el siguiente ejemplo:



(a) 'Impactos' detectados en el temporizador



(b) Máscara aplicada, no se detectan impactos

Figura 5.9: Uso de máscaras: Ejemplo práctico

■ `getCurrentTime(current_frame):`

Este método se encarga de dado un número de fotograma como parámetro, calcular el momento temporal dentro del vídeo en el que se encuentra.

Este cálculo es muy sencillo y puede verse de la siguiente forma: Si sabemos que cada segundo tenemos una determinada cantidad de fotogramas (fps) la cual sabemos ya que fue calculada, entonces si nos encontramos con el fotograma número X, el momento temporal en el que se encuentra se calculará como:

$$X \text{ fotograma} * \frac{1s}{\text{fotogramas}}$$

■ `getRealImpactTime(current_frame):`

Este método se encarga de para un determinado número de fotograma, haciendo uso del método 'getCurrentTime' así como del método 'os.path.getmtime' del módulo OS que nos devuelve el tiempo en el que fue creado el archivo, calcular el momento real temporal (no con respecto al vídeo) donde tomó lugar el impacto.

Procede de la siguiente forma:

- Obtiene el momento en el que se creó el archivo.
- Obtiene el momento en el que ha ocurrido el impacto dentro del vídeo.
- Obtiene la duración del vídeo.
- Se resta el momento temporal de la creación del archivo menos la duración del vídeo. Esto se hace ya que el momento temporal de creación del archivo obtenido por 'os.path.getmtime' nos sitúa cuando la grabación ha terminado y no cuando empezó.
- Se suma a la diferencia anterior el momento del impacto dentro del vídeo.
- Se devuelve el resultado de la suma anterior.

■ `saveAllImpacts()`:

Este método se encarga de para cada objeto de tipo 'Impact' almacenado en el vector atributo de la clase 'impacts':

- Calcular el tiempo real en el que tuvo lugar el impacto haciendo uso del método 'getRealTimeImpact' y asignárselo gracias al método de la clase 'Impact' 'setTime'.
- Establecer un nombre para el impacto que es del tipo:

nombre_video.extensión_númeroImpacto.númeroFotogramaPrevioPosterior

- Guardar el fotograma en memoria haciendo uso de la clase Dator y de su método 'saveFrame' (se explicará más adelante) pasándole el fotograma y el nombre de la imagen a guardar.
- Si se ha establecido por línea de órdenes un valor para el argumento 'SaveSurroundingFrames' entonces se llama al método 'saveSurroundingFrames' de la clase dator para almacenar tantos fotogramas previos y posteriores como el usuario haya indicado.

■ `saveLogFile()`:

Este método haciendo uso de la clase 'Dator' y su método 'saveLog' almacenará información de los posibles impactos ocurridos durante la grabación. Más detalles serán aportados en secciones siguientes sobre la clase 'Dator'.

■ `showFrame(frame)`:

Este método muestra un fotograma del vídeo que es pasado como parámetro haciendo uso de la función de OpenCV llamada 'imshow' [18].

- **analyze(fps, num_frames):**

Este método se encarga de analizar el vídeo indicado por parámetros estableciendo las máscaras indicadas por el usuario gracias al método 'applyMasks' de la propia clase y llamando al método 'getDifferences' de la clase 'ImageAnalyzer' comentado en la sección anterior. El código es el siguiente:

```
1 def analyze(self, fps, num_frames_show):  
2     cap = self.videoCapture  
3     play = True  
4  
5     if(fps == None and num_frames_show == None):  
6         fps = 10  
7         num_frames_show = self.frames  
8     while(cap.isOpened() and play):  
9         if(self.endingFrame != None and self.endingFrame  
10            <= self.current_frame):  
11             play = False  
12         num_frames_show = num_frames_show - 1  
13         ret, frame=cap.read()  
14         ret, frame2=cap.read()  
15  
16         frame, frame2 = self.applyMasks(frame, frame2)  
17  
18         if ret == True:  
19             frame, impact = (self.imageAnalizer).  
20                 getDifferences(frame, frame2, self.  
21                     current_frame)  
22  
23             if(impact != None and len(impact) > 0):  
24                 for i in impact:  
25                     self.impacts.append(i)  
26             if self.showVideo:  
27                 resized_frame = cv2.resize(frame,  
28                     (1600,900))  
29                 self.showFrame(resized_frame)  
30                 if(cv2.waitKey(fps) & 0xFF == ord('q')):  
31                     play = False  
32             else:  
33                 play = False  
34             self.current_frame = self.current_frame + 2  
35  
36             cap.release()  
37             cv2.destroyAllWindows()  
38  
39             if(self.showVideo == False):  
40                 self.saveLogFile()  
41                 self.saveAllImpacts()
```

Los pasos que se siguen son los siguientes:

- **Paso 1:** Se dota de valor a la variable local fps de tal forma que si ha sido por parámetros se le da dicho valor y si no se pone a 10. Esta variable será utilizada para que si un usuario quiere terminar la previsualización del análisis pulsando una tecla, dicha tecla se comprueba si ha sido pulsada cada fps milisegundos, además sirve para mostrar a cámara más o menos rápida dicha visualización. **Líneas:** 5-7.
- **Paso 2:** Haciendo uso del método que comprueba que comprueba que la capturadora de OpenCV sigue abierta (leyendo fotogramas) 'isOpened' y la variable local play es verdadera se toman dos fotogramas consecutivos mientras que el fotograma actual sea menor que el indicado como final por un usuario si es que lo ha indicado en la línea de órdenes. **Líneas:** 8-13.
- **Paso 3:** Haciendo uso del método '*applyMasks*' se aplican las máscaras seleccionadas por el usuario a los fotogramas anteriores que se actualizan tras esta operación. **Línea:** 15.
- **Paso 4:** Si se ha conseguido leer exitosamente el último fotograma que se ha intentado obtener (indicado por la variable ret), entonces obtenemos si ha ocurrido un impacto y su fotograma con el impacto marcado haciendo uso del método '*getDifferences*' de la clase *ImageAnalyzer*. **Línea:** 18.
- **Paso 5:** Se comprueba si se ha obtenido algún impacto y si han tenido lugar se añaden al atributo de la clase 'impacts' que era una lista de objetos de tipo 'Impact'. **Líneas:** 20-22.
- **Paso 6:** Se comprueba si está activo el modo 'debug' y en caso afirmativo se llama al método de la clase 'showFrame' que mostrará en cadena cada fotograma formando así un vídeo. **Líneas:** 23-25.
- **Paso 7:** Se comprueba si el usuario ha pulsado la tecla 'q' para salir de la previsualización en caso de estar activo el modo debug, para ello espera durante 10 ms. **Líneas:** 26-27.
- **Paso 8:** Si no quedaban más fotogramas que leer pone la variable local 'ret' a false para no seguir iterando . **Líneas:** 28-29.
- **Paso 9:** Con cada vuelta del bucle se actualiza a 1 la variable current_frame sumándole 2 ya que se leen dos fotogramas. **Línea:** 30.
- **Paso 10:** Se liberan los recursos de la capturadora de OpenCV y se cierran la pestaña donde se mostraba el vídeo. **Líneas:** 32-33.
- **Paso 11:** Llamando al método 'saveAllImpacts' se guardan en memoria todos los posibles impactos detectados. **Línea:** 34.

- **Paso 12:** Si no está activo el modo debug se guardan en memoria los impactos así como el log file llamando a los métodos 'saveAllImpacts' y 'saveLogFile' de la clase. **Línea:** 35-37.
- **getInitialFrame():**
Método muy sencillo que devuelve el primer fotograma del vídeo, será utilizado por la interfaz.
- **showASample():**
Se llama al método 'analyze' pasándole un número pequeño de fotogramas para que muestre una pequeña preview del proceso de análisis. Utilizado por la interfaz.

5.1.4. Clase Dator

Esta clase intenta representar una clase abstracta que defina los métodos que deberán implementar las clases que se basen en su definición. Esta clase pretende definir pero no implementar todas las operaciones relacionadas con la persistencia como pueden ser el almacenamiento de fotogramas, un resumen de impactos, comprobar si existen archivos, etc.

De esta forma, esta clase no podrá ser instanciada y todos sus métodos presentan la siguiente forma:

```
1 class Dator:  
2     def __init__(self):  
3         if(isinstance(self, Dator)):  
4             raise Exception("Non-instantiable class")  
5  
6     def saveFrame(self, name, frame):  
7         raise Exception("The method must be implemented by  
8             another class")  
9  
10    ...
```

Con el uso de esta clase se pretende poder aportar una forma de acceder a los datos de tal forma que sea totalmente al programa como lo haga, es decir, no importe si es basado en sistemas de ficheros, bases de datos, etc. ya que cualquier clase que herede de Dator deberá implementar los métodos definidos por dator y un programa llamará a estos métodos con tan solo cambiar en una línea el tipo de dator a utilizar, en su definición.

En nuestro programa por ejemplo en el script 'zepazo.py' bastaría con cambiar la siguiente línea para alternar entre diferentes formas de persistencia que se puedan implementar:

```
1 dator = FSDator( folder )
```

Métodos

Al ser una clase abstracta no implementa los métodos que define, es por ello que a continuación se muestra lo que deben hacer los métodos de las clases que los implementen:

- **saveFrame(name, frame):**

Este método cuando se implemente deberá almacenar persistentemente el fotograma recibido por el parámetro 'frame' con el nombre 'name'.

- **saveSurroundingFrames(num_frames, current_frame, name):**

Este método cuando se implemente deberá almacenar persistentemente num_frames anteriores y posteriores al número de fotograma indicado por 'current_frame' teniendo como nombre raíz 'name'.

- **saveLog(impacts):**

Este método cuando se implemente deberá almacenar persistentemente datos de los posibles impactos ocurridos al analizar el vídeo.

- **videoExists(path_to_video):**

Este método cuando se implemente deberá comprobar si existe un determinado vídeo.

- **logFileExists(path_to_log):**

Este método cuando se implemente deberá comprobar si existe un determinado resumen de posibles impactos de un vídeo.

- **loadLogFile(self, path_to_log):**

Este método cuando se implemente deberá obtener el resumen de impactos de un determinado análisis de un vídeo.

- **saveCoincidenceLog(path_to_log1, path_to_log2, path_to_save, data):**

Este método cuando se implemente deberá almacenar persistentemente las coincidencias temporales de impactos entre dos vídeos almacenados en el parámetro data.

- **correctPathToSaveJSON(path_to_json):**

Este método cuando se implemente deberá comprobar si es posible almacenar persistentemente un determinado registro identificado por el parámetro recibido.

- **saveParams(params):**

Este método cuando se implemente deberá almacenar persistentemente los parámetros exportados por la interfaz.

- **loadParams(id):**

Este método cuando se implemente deberá obtener un conjunto de parámetros almacenado previamente identificado por 'id'.

5.1.5. Clase FSDator

La clase FSDator hereda de la clase Dator y pretende aportar una solución de almacenamiento y consulta persistente basado en sistemas de ficheros convencionales. De esta forma, implementará todos los métodos de la clase Dator con este objetivo.

Atributos

Esta clase contiene un único atributo que es la ruta donde se encuentra el vídeo sobre el que se va a almacenar información.

Métodos

Al ser una clase abstracta no implementa los métodos que define, es por ello que a continuación se muestra lo que deben hacer los métodos de las clases que los implementen:

- **saveFrame(name, frame):**

Este método almacena persistentemente el fotograma recibido por el parámetro 'frame' con el nombre 'name'.

- **saveSurroundingFrames(num_frames, current_frame, name):**

Este método almacena persistentemente num_frames anteriores y posteriores al número de fotograma indicado por 'current_frame' teniendo como nombre raíz 'name'.

- **saveLog(impacts):**

Este método almacena persistentemente datos de los posibles impactos ocurridos al analizar el vídeo.

- **videoExists(path_to_video):**

Este método comprueba si existe un determinado vídeo.

- **logFileExists(path_to_log):**

Este método comprueba si existe un determinado resumen de posibles impactos de un vídeo.

- **loadLogFile(self, path_to_log):**
Este método obtiene el resumen de impactos de un determinado análisis de un vídeo.
- **saveCoincidenceLog(path_to_log1, path_to_log2, path_to_save, data):**
Este método almacena persistentemente las coincidencias temporales de impactos entre dos vídeos almacenados en el parámetro data.
- **correctPathToSaveJSON(path_to_json):**
Este método comprueba si es posible almacenar persistentemente un determinado registro identificado por el parámetro recibido.
- **saveParams(params):**
Este método almacena persistentemente los parámetros exportados por la interfaz.
- **loadParams(id):**
Este método obtiene un conjunto de parámetros almacenado previamente identificados por 'id'.

5.1.6. Script 'zepazo.py'

Este es un sencillo script que sirve de punto de entrada al usuario y donde se realizan los siguientes pasos tal y como se mostró en el diagrama de actividad de la figura 4.4 en el apartado de diseño:

- Se definen los argumentos descritos al inicio de esta sección.
- Se comprueban si los argumentos son correctos.
- Se comprueba si se va a importar un archivo de configuración de parámetros, en caso afirmativo se cargan los parámetros en variables. En caso negativo se usarán los valores definidos por argumentos.
- Se analiza el vídeo haciendo uso del método 'analyzer' de la clase 'VideoAnalyzer'.

5.2. ZepazoParams

Este programa pretende dar respuesta a los requisitos **RF14-RF19** [3.6.1](#). Es por ello que este *programa destinado a que los usuarios ajustasen de una forma más sencilla los parámetros de cara al análisis* de un vídeo mediante una sencilla interfaz tal y como se propuso en el diseño mostrado en la figura [4.7](#).

Tal y como se comentó en el apartado [2.2.1](#) de herramientas, para construir la interfaz se ha utilizado el módulo **PyQt5** [\[19\]](#) que me ha permitido construir una interfaz de forma muy sencilla y de forma muy parecida a como se haría en Java haciendo uso de Netbeans pero en este caso con PyQt contamos con la herramienta **QT 5 Designer** que a la hora de diseñar interfaces es muy parecido.

En la figura [4.8](#) donde encontramos el diagrama de clases de este programa, podemos ver que se hacen uso de las **clases** anteriormente comentadas **VideoAnalyzer**, **ImageAnalyzer**, **Dator** y **FSDator**, además de una clase nueva que se comentará en esta sección llamada **ZepazoParams**.

El programa se ejecuta de la siguiente forma:

```
1 python3 src/Interface/zepazo-params.py
```

De esta forma, este programa muestra la siguiente interfaz visual cuando se abre un vídeo:

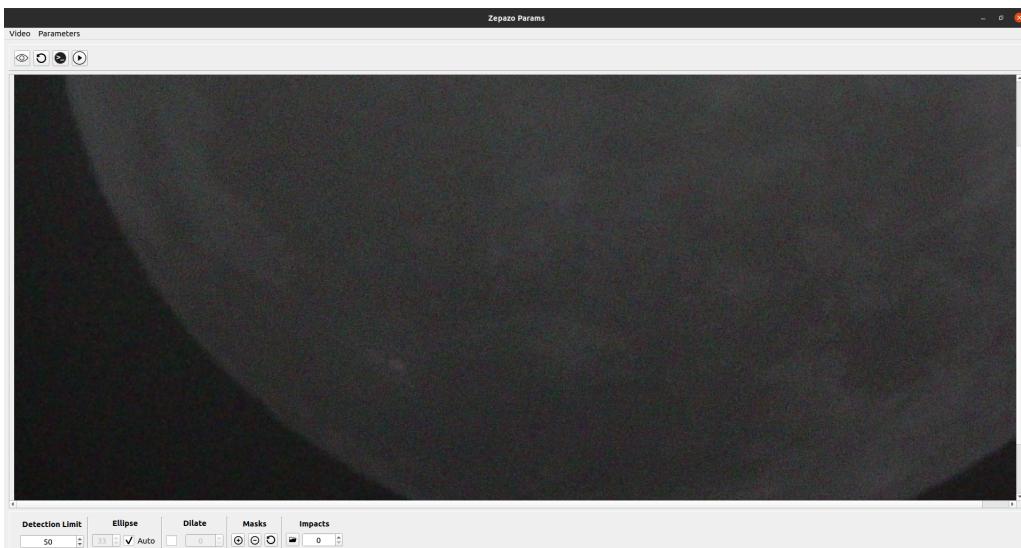


Figura 5.10: Interfaz programa ZepazoParams

5.2.1. Clase ZepazoParams

En la clase ZepazoParams encontramos código destinado principalmente a dos aspectos, en primer lugar código destinado a realizar acciones con la interfaz para ajustar los parámetros necesarios y en segundo lugar código relacionado con la creación de la propia interfaz como puede ser código para la creación de cada uno de los botones, layouts y paneles.

El primer bloque de código tal y como se ha mencionado irá destinado a definir el comportamiento de la interfaz, es decir, la vinculación de los distintos elementos visuales de la interfaz como por ejemplo botones para tengan una determinada funcionalidad. En dichos métodos como se podrá ver en el apartado donde se explique cada método se harán uso principalmente de los métodos de las clase ImageAnalyzer y VideoAnalyzer para conseguir previsualizar los cambios resultantes de modificar los valores de los parámetros.

En cuanto al segundo bloque de código, destinado a la creación de los elementos visuales de la interfaz, se ha construido gracias a la herramienta **QT 5 Designer**. Esta herramienta de forma gráfica nos permite diseñarla, y nuestro diseño definido a través de este programa mostraba la siguiente forma dentro de esta herramienta:

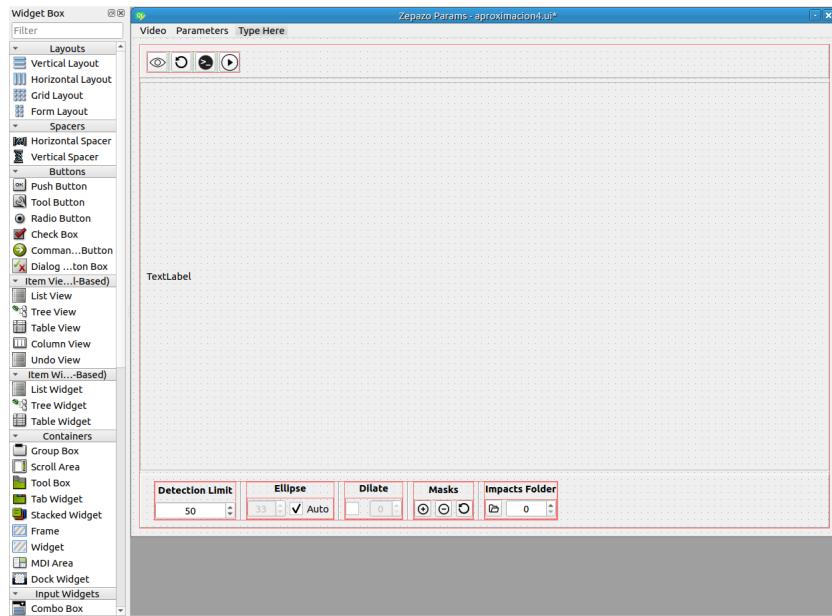


Figura 5.11: QT 5 Designer, creación Interfaz

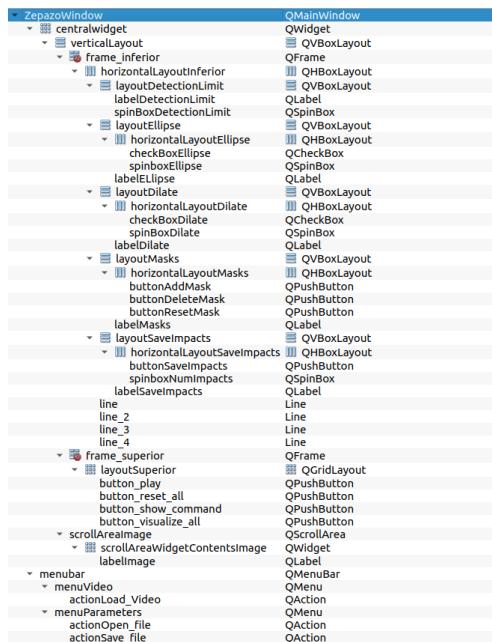


Figura 5.12: Interfaz, jerarquía de elementos visuales

Cuando hemos creado la interfaz haciendo uso de QT 5 Designer y la guardamos, generamos un archivo con extensión .ui que es legible por dicha aplicación, pero no es un fichero .py donde podamos programar la funcionalidad de la misma. Para convertir el fichero .ui en un fichero de tipo .py debemos ejecutar el siguiente comando:

```
1 pyuic5 -x archivo.ui -o archivo.py
```

Al ejecutar el comando anterior obtenemos una clase con dos métodos:

- **setupUi(ZepazoWindow):** Encargado de definir cada componente visual de la interfaz de acuerdo al esquema de la figura 5.12.
- **retranslateUi(ZepazoWindow):** Encargado de establecer los textos y tooltips de la interfaz.

No obstante, el código generado ha sido entendido y modularizado en diversas funciones para que el programa sea más legible y cada función se encargue de generar una parte de la interfaz como se verá en el apartado donde se comentan los métodos de esta clase.

Atributos

Atributos relacionados con parámetros:

- **detectionLimit:** Valor indicado por el usuario del argumento 'DetectionLimit'. **Tipo:** Int.
- **ellipse:** Contiene la elipse calculada que se ajusta al contorno lunar. **Tipo:** Opencv: : Ellipse.
- **dilate:** Valor indicado por el usuario del argumento 'Dilate'. **Tipo:** Int.
- **masks:** Lista de las máscaras que el usuario ha seleccionado. **Tipo:** Array<Int>.
- **videoPath:** Ruta del vídeo a analizar. **Tipo:** String.
- **first_frame:** Primer fotograma del vídeo. **Tipo:** Opencv: : frame.
- **addingMask:** Flag que indica si un usuario está añadiendo una máscara. **Tipo:** Boolean.
- **deletingMask:** Flag que indica si un usuario está eliminando una máscara. **Tipo:** Boolean.
- **frame_ellipse:** Fotograma que contiene dibujada la elipse alrededor del contorno lunar. **Tipo:** Opencv: : frame.

- **frame_masks:** Fotograma que contiene dibujadas las máscaras seleccionadas por el usuario hasta el momento. **Tipo:** Opencv: : frame.
- **folder:** Carpeta seleccionada por el usuario para guardar los fotogramas. **Tipo:** String.
- **numFrames:** Número de fotogramas a guardar antes y después de un impacto. **Tipo:** Int.

Atributos relacionados con los elementos visuales de la interfaz:

- **button_visualize_all:** Botón que al ser pulsado muestra todos los argumentos aplicados sobre un fotograma. **Tipo:** PyQt5::QtWidgets: : QPushButton.
- **button_reset_all:** Botón que al ser pulsado restablece todos los valores de los argumentos. **Tipo:** PyQt5::QtWidgets: : QPushButton.
- **button_play:** Botón para reproducir el vídeo indicado mostrando todos los parámetros aplicados. **Tipo:** PyQt5::QtWidgets: : QPushButton.
- **button_command:** Botón para mostrar una previsualización del comando a utilizar con los parámetros elegidos hasta el momento. **Tipo:** PyQt5::QtWidgets: : QPushButton.
- **labelDetectionLimit:** Texto que indica a qué hacen referencia los botones asociados. **Tipo:** PyQt5::QtWidgets::QLabel.
- **labelELipse:** Texto que indica a qué hacen referencia los botones asociados. **Tipo:** PyQt5::QtWidgets::QLabel.
- **checkBoxEllipse:** Botón que al ser marcado indica que para calcular la elipse se intente hacer de forma automática sin introducir ningún valor. **Tipo:** PyQt5::QtWidgets: : QCheckBox.
- **labelDilate:** Texto que indica a qué hacen referencia los botones asociados. **Tipo:** PyQt5::QtWidgets::QLabel.
- **labelMasks:** Texto que indica a qué hacen referencia los botones asociados. **Tipo:** PyQt5::QtWidgets::QLabel.
- **buttonAddMask:** Botón que al ser pulsado permitirá al usuario añadir una máscara. **Tipo:** PyQt5::QtWidgets: : QPushButton.
- **buttonDeleteMask:** Botón que al ser pulsado permitirá al usuario eliminar una máscara. **Tipo:** PyQt5::QtWidgets: : QPushButton.

- **buttonResetMask:** Botón que al ser pulsado eliminará todas las máscaras. **Tipo:** PyQt5::QtWidgets: : QPushButton.
- **menuVideo:** Menú que contiene la opción de abrir un vídeo. **Tipo:** PyQt5QtWidgets: : QMenuBar.
- **menuParameters:** Menú que contiene las opciones de importar y exportar parámetros. **Tipo:** PyQt5: : QtWidgets : : QMenuBar.
- **actionLoad_Video:** Opción del menú de vídeo para cargar un vídeo. **Tipo:** PyQt5::QtWidgets: : QAction.
- **actionOpen_file:** Opción del menú de parámetros para importar parámetros. **Tipo:** QtWidgets: : PyQt5::QAction.
- **actionSave_file:** Opción del menú de parámetros para exportar parámetros. **Tipo:** QtWidgets: : PyQt5::QAction.
- **labelSaveImpacts:** Texto que indica a qué hacen referencia los botones asociados. **Tipo:** QtWidgets : PyQt5::QtWidgets:: QLabel.
- **buttonSaveImpacts:** Botón que al ser pulsado dejará seleccionar una carpeta donde guardar los fotogramas. **Tipo:** PyQt5::QtWidgets: : QPushButton.
- **spinBoxFrames:** Selector numérico que permite indicar cuantos fotogramas guardar antes y después del impacto. **Tipo:** PyQt5::QtWidgets: : QSpinBox.
- **spinboxEllipse:** Selector numérico que permite ajustar el valor del parámetro 'CircleLimit' para la detección de la elipse. **Tipo:** PyQt5::QtWidgets: : QSpinBox.
- **centralPanel:** Panel central que contendrá el primer fotograma del vídeo. **Tipo:** PyQt5::QtWidgets: : QLabel.
- **spinBoxDetectionLimit:** Selector numérico que permite ajustar el valor del argumento 'DetectionLimit'. **Tipo:** PyQt5::QtWidgets: : QSpinBox.
- **checkBoxDilate:** Casilla marcable que permite indicar si se va a utilizar la dilatación de fotogramas. **Tipo:** PyQt5::QtWidgets: : QCheckBox.
- **spinBoxDilate:** Selector numérico que permite ajustar el valor del argumento 'Dilate'. **Tipo:** PyQt5::QtWidgets: : QSpinBox.

Métodos

Métodos relacionados con creación de componentes visuales:

■ **setUpCentralWidget():**

Este método se encarga de establecer un layout para toda la pestaña basado en el uso de tres paneles:

- El **panel superior** que contiene herramientas relacionadas con la previsualización de parámetros tanto en un vídeo como en un fotograma, previsualización del comando así como para restablecer todos los parámetros.
- El **panel central** que contendrá el primer fotograma del vídeo sobre el que se van reflejando los cambios de los diversos parámetros.
- El **panel inferior** que contiene las herramientas necesarias para ajustar los distintos parámetros.

■ **setSuperiorFrame():**

Se encarga de establecer los componentes visuales del panel superior y es el siguiente:



Figura 5.13: Interfaz, panel superior

Donde de izquierda a derecha vemos:

- Botón para ver todos los parámetros aplicados a un fotograma.
- Botón para restablecer los valores de los parámetros.
- Botón para obtener el comando a utilizar con los parámetros establecidos hasta el momento.
- Botón para obtener una previsualización del análisis con los parámetros actuales aplicados.

- **setCentralContent():**

Se encarga de establecer los componentes visuales del panel central como son el área de scroll y la zona donde se muestra el primer fotograma:



Figura 5.14: Interfaz, panel central

- **setInferiorFrame():**

Se encarga de establecer los componentes visuales del panel inferior y es el siguiente:



Figura 5.15: Interfaz, panel inferior

Encontramos los siguientes elementos:

- Zona de selección del parámetro DetectionLimit. La cual cuenta con una casilla numérica para su selección.
- Zona de selección del parámetro CircleLimit relacionado con la ellipse, puede elegirse numéricamente o seleccionar la opción que lo intenta calcular de forma automática.
- Zona de selección del argumento dilate contando con una casilla numérica para su elección así como un botón para activar o desactivar la dilatación.
- Zona de selección de las máscaras contando con botones para añadir o eliminar una máscara así como otro botón para restaurar todas.
- Zona de selección de la carpeta donde guardar los fotogramas así como el número de fotogramas previos o posteriores a guardar.

- **addMenu():**

Se encarga de establecer los menús superiores donde encontramos la opción de cargar videos o importar y exportar parámetros.

- **addTexts():**

Se encarga de establecer todos los textos y tooltips de la aplicación así como de vincular cada evento con su correspondiente método.

- **setupUI():**

Se encarga de llamar a los métodos anteriores para formar toda la interfaz.

- **launch_UI():**

Al ser llamado mostrará la interfaz al usuario, este es el método que creará una pestaña en el escritorio mostrando la interfaz.

Métodos relacionados con el comportamiento de la interfaz:

- **showFrame(first_frame):**

Este método se encarga de dado un fotograma leído por los métodos de OpenCV la transforma en una imagen que puede ser utilizada por PyQt para mostrarla y la muestra en el panel central.

- **addMessage(message):**

Este método será utilizado por otros para mostrar una notificación al usuario en forma de mensaje cuando ocurra un error. Un ejemplo de mensaje generado por este método es el siguiente:

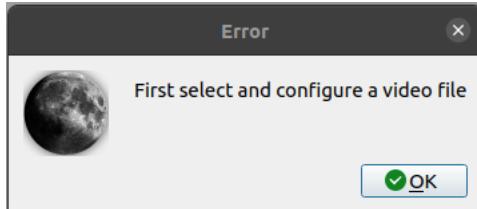
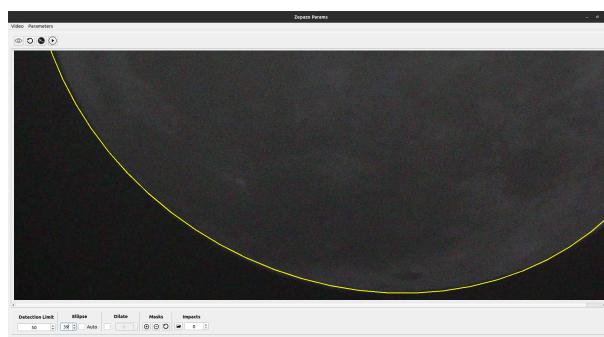


Figura 5.16: Interfaz, mensaje error

- **adjustEllipse():**

Este método es llamado cuando se cambia el valor del argumento 'CircleLimit' y haciendo uso del método 'selectAndApplyCircleLimitArgument' de la clase VideoAnalyzer que haciendo uso del atributo 'imageAnalyzer' obtiene la elipse para el primer fotograma del vídeo y lo muestra.

Un ejemplo es el siguiente:



(a) Valor óptimo: 39, elipse perfectamente ajustada



(b) Prueba de otro valor: 48

Figura 5.17: Funcionamiento método adjustEllipse.

- **adjustDetectionLimit():**

Este método es llamado cuando se varía el valor del argumento 'DetectionLimit' y se encarga de variar el valor del atributo 'detectionLimit' para que cuando se previsualice el análisis tome efecto.

■ **checkAutoEllipse():**

Se encarga de comprobar si está o no marcada la casilla 'Auto' para la determinación del valor del argumento 'CircleLimit' de tal manera que cuando se marca obtiene la elipse sin de forma automática sin que el usuario tenga que indicar ningún valor haciendo uso de nuevo del método 'selectAndApplyCircleLimitArgument' de la clase VideoAnalyzer. Por ejemplo esta es la aproximación que devuelve para este fotograma:

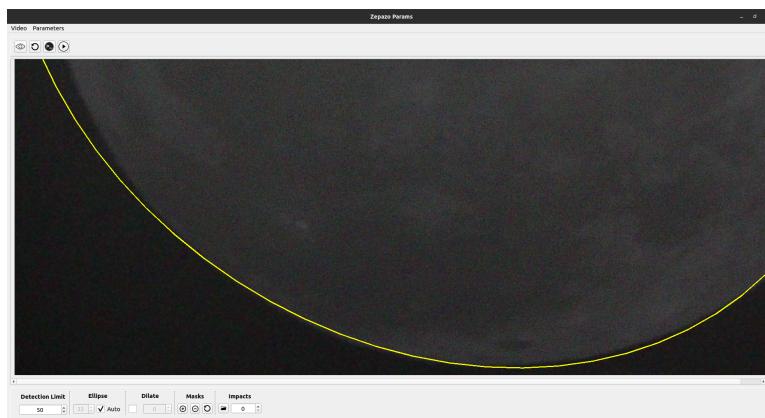


Figura 5.18: Interfaz, elipse calculada automáticamente

Donde como podemos ver la elipse calculada se ajusta casi perfectamente, únicamente si queremos la elipse perfecta deberemos subir un par de puntos el valor del argumento 'CircleLimit'.

■ **clickImage(event):**

Este método se llama cada vez que un usuario pulsa sobre la pantalla y entonces se distinguen varios casos:

- Se está añadiendo una máscara: Se esperan dos clicks en la imagen, uno para la esquina superior izquierda de la máscara y otro para la esquina inferior derecha de la misma, se añaden los puntos al vector de máscaras de la clase. Con el segundo click se le muestra al usuario la máscara dibujada sobre el fotograma.
- Se está eliminando una máscara: Si hace click en una máscara se elimina.

■ **deleteMask():**

Este método pone a true el atributo 'deletingMask' para que a la hora de detectar clicks en la imagen el programa distinga que está en el caso de eliminar una máscara.

- **inMask(x, y):**

Este método comprueba dada una coordenada (x,y) donde un usuario ha hecho click si existe alguna máscara del vector de máscaras de la clase que contenga dicho punto, si es así elimina dicha máscara.

- **addMask():**

Este método pone a true el atributo 'addingMask' para que a la hora de detectar clicks en la imagen el programa distinga que está en el caso de añadir una máscara. Es llamado cuando se pulsa el botón de añadir máscara. Se muestran todas las máscaras añadidas hasta ahora aplicadas sobre el fotograma. Ejemplo de máscara añadida en la interfaz:

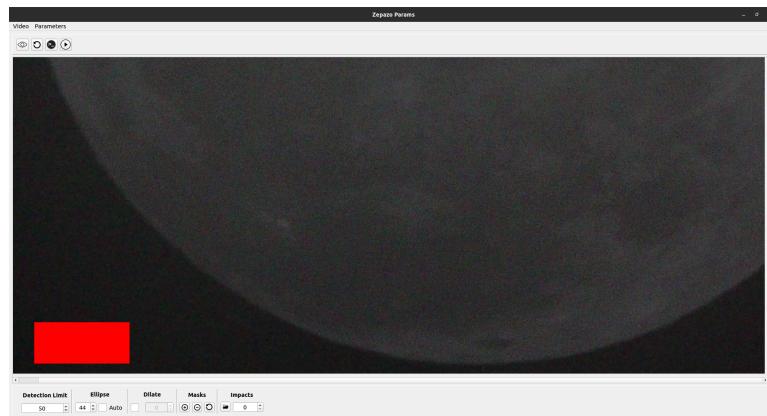


Figura 5.19: Interfaz, máscara añadida

- **resetParams():**

Este método restablece a sus valores por defecto todos los atributos de la clase y muestra el fotograma inicial sin ningún parámetro aplicado en consecuencia. Es llamado cuando se pulsa el botón de restablecer parámetros.

- **resetMasks():**

Este método elimina todas las máscaras. Es llamado cuando se pulsa el botón de eliminar todas las máscaras.

- **showAllParams():**

Este método muestra todos los argumentos aplicados sobre un mismo fotograma. Es llamado cuando se pulsa el botón de mostrar todos los parámetros. En el ejemplo siguiente vemos tanto la elipse calculada como una máscara añadida:

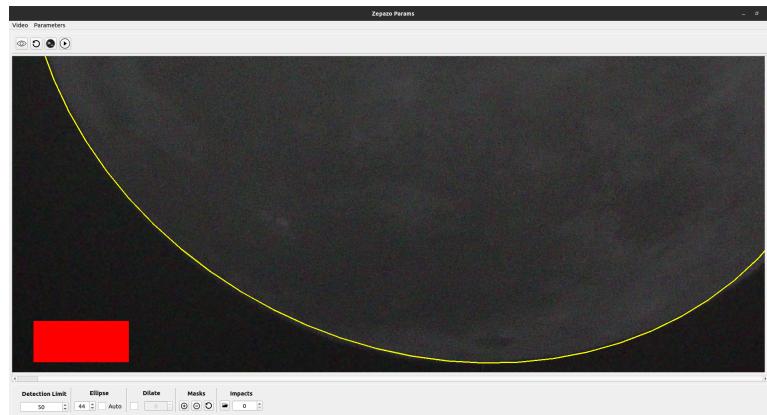


Figura 5.20: Interfaz, funcionamiento botón mostrar parámetros

- **drawMasks(frame):**

Este método se encarga de dibujar todas las máscaras que el usuario ha definido hasta el momento, es utilizado tanto por el método addMask, deleteMask y showAllParams para mostrar las máscaras.

- **showCommand():**

Este método teniendo en cuenta los parámetros que el usuario ha definido hasta el momento muestra y copia al portapapeles el comando que sería necesario utilizar para ejecutar el programa con la configuración seleccionada. Un ejemplo sería el siguiente:

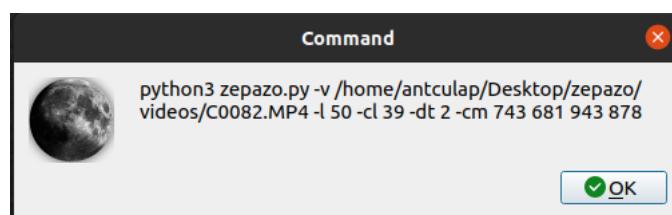


Figura 5.21: Interfaz, mostrando el comando a utilizar

- **saveParams():**

Este método cuando en el menú de parámetros se selecciona la opción 'save file' se exportan los parámetros en este caso en JSON (por el tipo de Dator utilizado) generando un archivo exportable a través de una pestaña de dialogo que nos permite seleccionar la ubicación del archivo a guardar para utilizarlos en el programa 'Zepazo' para cargar dicha configuración. Un ejemplo de parámetros exportados es el siguiente:

```
1 {  
2     "detectionlimit": 50,  
3     "dilate": 2,  
4     "coordinatesmask": [743, 681, 943, 878],  
5     "circlelimit": 39,  
6     "folder": null,  
7     "saveSurroundingFrames": null  
8 }
```

- **selectSaveImpactsFolder():**

Este método es llamado cuando se pulsa el botón para elegir una carpeta donde guardar los fotogramas y abre un diálogo para seleccionar dicha carpeta.

- **showVideoSample():**

Este método se encarga de mostrar una previsualización del proceso de análisis con todos los parámetros seleccionados aplicados.

- **selectNumFrames():**

Este método se encarga de cambiar el valor del atributo numFrames que indica el número de fotogramas previos y posteriores a guardar. Es llamado cuando se varía el valor del mismo en la interfaz.

- **setDilateValue():**

Este método se encarga de cambiar el valor del atributo dilate. Es llamado cuando se varía el valor del mismo en la interfaz.

- **checkboxDilateClicked():**

Este método se encarga de activar o desactivar la dilatación de fotogramas cuando se activa la casilla de dilatación de la interfaz.

- **loadVideo():**

Este método se encarga mediante una ventana de diálogo de seleccionar un vídeo para ajustar los parámetros sobre dicho vídeo.

5.3. ZepazoVerify

Este programa pretende dar respuesta a los requisitos RF20-RF21 3.6.1. Es por ello que este programa estará destinado a validar/verificar impactos.

La verificación de impactos tal y como se comentó en el apartado 1.3 de introducción consiste en contar con grabaciones de la luna del mismo momento temporal de tal forma que contemos con dos grabaciones de la luna parecidas para posteriormente lanzar el proceso de análisis en ambos vídeos, una vez obtenidos los resultados del análisis de cada vídeo se validarán los impactos mediante la comparación de los resultados del análisis buscando por *impactos que han tomado lugar en un momento temporal similar en ambos videos* descartando que sean falsos positivos ya que han ocurrido en ambas grabaciones.

El programa se ejecuta de la siguiente forma:

```
1 python3 src/zepazoVerify.py [lista de argumentos]
```

Donde **lista de argumentos** en un conjunto de argumentos a seleccionar por el cliente entre los que encontramos:

- **-lgf1, --logFile1**: Log resultado del análisis del primer vídeo. **Tipo**: Cadena de caracteres.
- **-lgf2, --logFile2**: Log resultado del análisis del segundo vídeo. **Tipo**: Cadena de caracteres.
- **-rlfg, --resultingLogFile**: Ruta donde guardar el log resultante. **Tipo**: Cadena de caracteres.
- **-mt, --marginTime**: Margen temporal para que dos impactos con una diferencia indicada por este parámetro sean emparejados. **Tipo**: Entero positivo.

De esta forma, una vez indicados los parámetros para cada impacto del primer archivo de log se comparará con cada impacto del segundo archivo de log, con el objetivo de encontrar impactos que hayan ocurrido con como mucho una diferencia temporal (en segundos) indicada por el argumento 'marginTime'.

Un ejemplo con un valor del argumento 'marginTime' de 10 sería el siguiente:

Archivo log vídeo 1:

```

1 [
2   {
3     "impact_frame_number": 5,
4     "impact_number": 0,
5     "impact_time": "17:7:11"
6   },
7   {
8     "impact_frame_number": 30,
9     "impact_number": 1,
10    "impact_time": "17:7:20"
11  }
12 ]

```

Archivo log vídeo 2:

```

1 [
2   {
3     "impact_frame_number": 12,
4     "impact_number": 0,
5     "impact_time": "17:7:25"
6   },
7   {
8     "impact_frame_number": 6,
9     "impact_number": 1,
10    "impact_time": "17:7:50"
11  }
12 ]

```

Archivo log resultante:

```

1 [
2   {
3     "difference": 5,
4     "log_1_impact": 1,
5     "log_2_impact": 0,
6     "path_to_log1": "../test/log.json",
7     "path_to_log2": "../test/log2.json"
8   }
9 ]

```

Donde podemos ver que efectivamente se ha emparejado un impacto del primer log con otro del segundo donde la diferencia temporal es menos de 10 segundos, en este caso tal y como indica el log resultante es de 5 segundos.

5.4. ZepazoCrop

Este programa pretende dar respuesta al requisito RF22 [3.6.1](#). Es por ello que este programa estará destinado a simplemente recortar vídeos. Esta tarea es muy importante de cara a no sobrecargar la red del servidor ya que un usuario que quiera acceder a los vídeos para poder ajustar los parámetros mediante el programa 'zepazoParams' gracias a esta herramienta recortará un fragmento de un vídeo y se descargará del servidor únicamente dicho fragmento.

El programa se ejecuta de la siguiente forma:

```
1 python3 src/zepazoCrop.py [lista de argumentos]
```

Donde **lista de argumentos** en un conjunto de argumentos a seleccionar por el cliente entre los que encontramos:

- **-vo, --videoOriginal:** Ruta del vídeo a recortar. **Tipo:** Cadena de caracteres.
- **-vc, --videoCropped:** Ruta donde guardar el vídeo recortado. **Tipo:** Cadena de caracteres.
- **-ss, --secondStart:** Segundo desde el que recortar el vídeo. **Tipo:** Entero positivo.
- **-se, --secondEnd:** Segundo hasta el que recortar el vídeo. **Tipo:** Entero positivo.

De esta forma una vez el programa recibe los argumentos indicados por el usuario comprueba que sean correctos y utilizando el módulo de Python 'moviepy.editor' [\[20\]](#) recorta un vídeo de la siguiente forma:

```
1 cropped = editor.VideoFileClip(args.videoOriginal)
2 cropped_video = cropped.subclip(args.secondStart, args.
3     secondEnd)
4 cropped_video.write_videofile(args.videoCropped)
5 cropped_video.close()
```

Donde como podemos ver se crea un objeto de tipo VideoFileClip que nos permitirá extraer un fragmento del vídeo y acto seguido indicando el momento de inicio y el momento final se recorta el vídeo para posteriormente escribirlo en la ruta indicada por el argumento 'videoCropped'.

Capítulo 6: Pruebas

En este capítulo se pretenden mostrar las distintas pruebas realizadas con el objetivo de tanto asegurar la mayor calidad posible del software como la determinación de si nuestro producto cumple con las necesidades y requisitos establecidos por el cliente. De esta forma, durante toda la realización del proyecto se han ido probando todas las funcionalidades conforme se incluían y podemos clasificar las pruebas realizadas en dos bloques:

- **Pruebas Unitarias:** Pruebas cuyo objetivo principal es la determinación del correcto comportamiento de una función, método o pieza de código por separado.
- **Pruebas de aceptación:** Pruebas que pretenden comprobar si el software desarrollado cumple con los requisitos y necesidades de un determinado cliente.

No obstante, a parte de los dos bloques de pruebas mencionados anteriormente, durante todo el desarrollo de este software se han ido comprobando, haciendo uso de vídeos de ejemplo típicos a procesar con este programa, el correcto funcionamiento de cada una de las funcionalidades implementadas haciendo uso del modo 'Debug' implementado en el programa, el cual se comentará brevemente en este apartado.

6.1. Pruebas unitarias

Las pruebas unitarias son un tipo de pruebas que nos permiten asegurar el correcto funcionamiento de piezas de código tales como funciones o métodos permitiéndonos encontrar fallos en nuestro código y consecuentemente mejorando la calidad del mismo.

En consecuencia, en nuestro proyecto durante el desarrollo se han elaborado pruebas unitarias con el objetivo de probar la mayor parte de código que nos ha sido posible haciéndonos descubrir en algunos casos numerosos fallos en el software que de otra forma hubiesen sido complejos de detectar.

De esta forma, en el desarrollo de este proyecto se ha buscado automatizar el desarrollo y ejecución de este tipo de pruebas y para ello hemos contado con las siguientes herramientas:

- **Docker** [21]: Herramienta que nos ha permitido encapsular en un contenedor todo lo necesario para ejecutar las pruebas unitarias como es el propio código y las dependencias necesarias para ejecutar el programa y pasar las pruebas.
- **Pytest** [22]: Framework que nos ha permitido escribir pruebas unitarias gracias al conjunto de aserciones provistas por el mismo.
- **Poetry** [23]: Gestor de dependencias que nos ha permitido encapsular todas las dependencias de nuestro programa de tal forma que al crear el contenedor es sencillo instalarlas con un único comando.
- **GitHub**: Plataforma de control de versiones utilizada para alojar el proyecto así como para automatizar la ejecución de tests mediante GitHub Actions.
- **GitHub Actions** [24]: Característica de la plataforma GitHub que nos ha permitido ejecutar las pruebas unitarias con cada cambio subido a nuestro repositorio.

6.1.1. Dockerfile

Como ya se ha mencionado, se ha hecho uso de Docker con el objetivo de construir un contenedor que contenga todas las dependencias necesarias así como código para ejecutar las pruebas unitarias independientemente de la plataforma.

Gracias a la elaboración de este contenedor se pueden ejecutar las pruebas unitarias con los siguientes pasos:

- Construir el contenedor:

```
1 docker build . -t zepazo
```

- Ejecutar los tests:

```
1 docker run -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix zepazo
```

Es por ello que se ha desarrollado un archivo Dockerfile, el cual no es más que un conjunto de instrucciones que indican a Docker como construir un contenedor y presenta la siguiente forma:

```
1  FROM python:3.8-slim
2
3  LABEL version="1.0.0" maintainer="antculap@gmail.com"
4
5  RUN apt-get update -y
6
7  #Install opencv dependencies
8  RUN apt install --no-install-recommends build-essential -y
9  RUN apt install libgl1-mesa-glx -y && \
10    apt install libglib2.0-0 -y && \
11    apt install libgtk2.0-dev -y && \
12    apt install libqt5x11extras5 -y && \
13    apt install x11-xserver-utils -y && \
14    apt install xvfb -y
15
16  #Needed to avoid errors in opencv
17  ENV LANG=C.UTF-8
18
19  #Upgrade PIP, if not, can cause problems installing opencv-
20  #python
21  RUN pip install --upgrade pip
22
23  #Install poetry to install dependencies
24  RUN pip install poetry
25
26  #Copy dependency files
27  COPY poetry.lock pyproject.toml /home/test_user/
28
29  #We are in a container, it is isolated, virtualenv is not
30  #needed
31  RUN poetry config virtualenvs.create false
32
33  RUN useradd test_user
34  RUN su test_user
35
36  #Change directory
37  WORKDIR /home/test_user/
38
39  #Install dependencies
40  RUN poetry install
41
42  #Install a specific opencv version in order to avoid problems
43  RUN pip install opencv-python==4.3.0.36
44
45  COPY . .
46
47  #Dependency files no longer needed
48  RUN rm -r /home/test_user/poetry.lock
49
50  #Execute tests
51  CMD ["poetry", "run", "task", "test"]
```

Donde podemos ver que se siguen los siguientes pasos destacados:

- Se parte de una imagen base de Python, la cual es la 'python:3.8-slim' incluyendo instaladas todas las herramientas básicas necesarias para trabajar con Python siendo esta una versión muy ligera. **Línea:** 1.
- Se obtienen todas las actualizaciones de las listas de paquetes disponibles. **Línea:** 5.
- Se instalan paquetes necesarios para poder trabajar con OpenCv posteriormente. **Líneas:** 8-14.
- Se copian los archivos de dependencias. **Línea:** 26.
- Se añade un nuevo usuario sin permisos de administración y cambiamos a él, ya que dichos permisos no son necesarios para instalar dependencias o ejecutar las pruebas siguiendo así las buenas prácticas de Docker. **Líneas:** 31-32.
- Se instalan las dependencias haciendo uso de Poetry. **Línea:** 38.
- Se copia tanto el código como las pruebas. **Línea:** 43.
- *Se ejecutan las pruebas unitarias haciendo uso del gestor de tareas Poetry con la extensión Taskipy.* **Línea:** 49.

6.1.2. GitHub Action

Gracias a esta herramienta de GitHub hemos podido *automatizar el proceso de construcción del contenedor y ejecución de las pruebas* dentro del mismo en un entorno controlado. Para ello, simplemente hemos tenido que crear una acción y es la siguiente:

```
1      #GitHub action to build container and run unit tests
2      name: 'Unit tests'
3      on: [push]
4
5      jobs:
6          build_container_and_run_tests:
7              runs-on: ubuntu-latest
8              steps:
9                  - uses: actions/checkout@v2
10                 - name: build docker container
11                     run: docker build . -t zepazo && docker run -e DISPLAY=
$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix zepazo
```

Donde como podemos ver, es una acción llamada 'Unit tests' en el cual tiene un único trabajo y consiste en construir el contenedor anteriormente explicado y ejecutarlo, ya que como vimos al ejecutarlo se ejecutarán las pruebas unitarias.

Como consecuencia, con cada 'Push' realizado para incluir cambios a nuestro repositorio todas las pruebas unitarias codificadas se ejecutarán y obtendremos información acerca de su éxito o fallo a través de una forma muy clara y sencilla a través de un tick verde si se han pasado o una aspa roja si no pudiendo entrar en detalle acerca del error. Podemos observarlo en la siguiente figura:

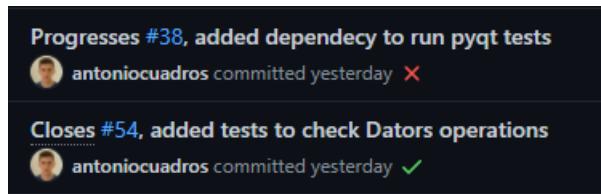


Figura 6.1: GitHub Commits History

6.1.3. Pruebas

Las pruebas unitarias, que como hemos visto se ejecutan con cada cambio incorporado a nuestro repositorio en la plataforma GitHub, se encuentran divididas en diversos ficheros encontrando de esta forma tantos ficheros como clases deseamos probar y se detallan a continuación:

Pruebas unitarias clase ImageAnalyzer

Nombre prueba	test_if_difference_image_is_ok
Pasos realizados	<ul style="list-style-type: none">■ Se obtiene un fotograma de un vídeo de prueba.■ Se obtiene la diferencia de ese fotograma consigo mismo haciendo uso del método ImageAnalyzer::getDifferences() pasándole dichos fotogramas.■ Se obtiene el fotograma resultado.
Comprobaciones	<ul style="list-style-type: none">■ Al ser el mismo fotograma restado consigo mismo el color de cada píxel debe ser 0, es decir negro.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_difference_images_working
Pasos realizados	<ul style="list-style-type: none">■ Lee dos fotogramas con pequeñas diferencias.■ Llama al método ImageAnalyzer::getDifferences() pasándole dichos fotogramas.
Comprobaciones	<ul style="list-style-type: none">■ Al ser fotogramas con ciertas diferencias, se han tenido que detectar impactos, en consecuencia el vector de impactos devuelto tiene que tener una longitud mayor que 0.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_dilate_is_working
Pasos realizados	<ul style="list-style-type: none">■ Lee dos fotogramas con pequeñas diferencias detectables y anulables mediante dilatación.■ Se aplica un valor alto de dilatación de imágenes.■ Llama al método ImageAnalyzer::getDifferences() pasándole dichos fotogramas.
Comprobaciones	<ul style="list-style-type: none">■ Al dilatar la imagen se espera obtener un vector de impactos de tamaño 0, es decir, no se detectan impactos.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_moon_outline_is_obtained
Pasos realizados	<ul style="list-style-type: none">■ Se lee una imagen a la que es posible calcular el contorno de la superficie lunar.■ Se calcula la elipse que recubre la superficie lunar haciendo uso del método ImageAnalyzer::moonEnclosingCircle().
Comprobaciones	<ul style="list-style-type: none">■ Se ha obtenido una elipse ajustada a la superficie lunar.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_moon_outline_is_not_obtained
Pasos realizados	<ul style="list-style-type: none">■ Se lee una imagen a la que no es posible calcular el contorno de la superficie lunar.■ Se calcula la elipse que recubre la superficie lunar haciendo uso del método ImageAnalyzer::moonEnclosingCircle().
Comprobaciones	<ul style="list-style-type: none">■ No se ha obtenido una elipse ajustada a la superficie lunar.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_defined_threshold_is_used
Pasos realizados	<ul style="list-style-type: none"> ■ Se lee un fotograma. ■ Se define un valor umbral para el cálculo de la elipse. ■ Se calcula la elipse que recubre la superficie lunar haciendo uso del método ImageAnalyzer::moonEnclosingCircle() y el valor aportado.
Comprobaciones	<ul style="list-style-type: none"> ■ El umbral devuelto debe coincidir con el especificado.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_threshold_is_calculated
Pasos realizados	<ul style="list-style-type: none"> ■ Se lee un fotograma. ■ No se define un valor umbral para el cálculo de la elipse. ■ Se calcula la elipse que recubre la superficie lunar haciendo uso del método ImageAnalyzer::moonEnclosingCircle().
Comprobaciones	<ul style="list-style-type: none"> ■ Se espera obtener un valor umbral calculado a partir del fotograma aportado.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_marking_impacts_is_working
Pasos realizados	<ul style="list-style-type: none"> ■ Se leen dos fotogramas con ciertas diferencias. ■ Se llama al método ImageAnalyzer::getDifferences() para obtener un fotograma resultado de las diferencias.
Comprobaciones	<ul style="list-style-type: none"> ■ Vector de impactos obtenido con longitud mayor que 0. ■ Se compara la imagen diferencia con el segundo fotograma comprobando si se ha añadido la marca sobre el impacto.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_impact_takes_place_inside_moon_is_deected
Pasos realizados	<ul style="list-style-type: none"> ■ Se lee una imagen donde se puede calcular la elipse. ■ Se calcula la elipse llamando al método ImageAnalyzer::moonEnclosingCircle(). ■ Se llama al método ImageAnalyzer::inside_moon() pasándole un punto interior a la elipse calculada.
Comprobaciones	<ul style="list-style-type: none"> ■ Se comprueba que el método verifica que el impacto es interior a la elipse.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_impact_doesnt_place_inside_moon_is_detected
Pasos realizados	<ul style="list-style-type: none">■ Se lee una imagen donde se puede calcular la elipse.■ Se calcula la elipse llamando al método ImageAnalyzer::moonEnclosingCircle().■ Se llama al método ImageAnalyzer::inside_moon() pasándole un punto exterior a la elipse calculada.
Comprobaciones	<ul style="list-style-type: none">■ Se comprueba que el método verifica que el impacto es exterior a la elipse.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Pruebas unitarias clase VideoAnalyzer

Nombre prueba	test_if_object_is_created_ok
Pasos realizados	<ul style="list-style-type: none">■ Se crea un objeto de tipo VideoAnalyzer indicándole una ruta de un vídeo existente.
Comprobaciones	<ul style="list-style-type: none">■ El objeto contiene la ruta indicada.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_object_is_created_with_error
Pasos realizados	<ul style="list-style-type: none">■ Se crea un objeto de tipo VideoAnalyzer indicándole una ruta de un vídeo no existente.
Comprobaciones	<ul style="list-style-type: none">■ Se obtiene una excepción derivada de la ruta de vídeo errónea.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_video_is_loaded_correctly
Pasos realizados	<ul style="list-style-type: none">■ Se crea un objeto de tipo VideoAnalyzer indicándole una ruta de un vídeo existente.■ Se prueba a leer un fotograma del vídeo.
Comprobaciones	<ul style="list-style-type: none">■ El fotograma es correcto y no nulo.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_current_time_ok
Pasos realizados	<ul style="list-style-type: none">■ Se llama al método VideoAnalyzer::getCurrentTime() pasándole diversos números de fotogramas.
Comprobaciones	<ul style="list-style-type: none">■ El fotograma 0 tiene como tiempo actual '00:00:00'■ El fotograma 100 para vídeo cargado tiene de tiempo '0:00:04.347826'
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_video_shows
Pasos realizados	<ul style="list-style-type: none">■ Se lee un fotograma del vídeo.■ Se llama al método VideoAnalyzer::showFrame().
Comprobaciones	<ul style="list-style-type: none">■ El fotograma leído es mostrado.■ El fotograma leído es correcto y distinto de nulo.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_image_is_saved
Pasos realizados	<ul style="list-style-type: none">■ Se lee un fotograma del vídeo.■ Se crea un objeto de tipo impacto que incluye el fotograma anterior.■ Se llama al método VideoAnalyzer::saveAllImpacts().
Comprobaciones	<ul style="list-style-type: none">■ Existe un nuevo archivo de tipo imagen que muestra el impacto.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_stats_are_ok
Pasos realizados	<ul style="list-style-type: none"> ■ Se llama al método VideoAnalyzer::getGeneralVideoStats() sobre un vídeo.
Comprobaciones	<ul style="list-style-type: none"> ■ Número de fotogramas calculado igual a 29. ■ Fotogramas por segundo igual a 23. ■ Segundos de duración del vídeo igual a 1.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_mask_points_ok
Pasos realizados	<ul style="list-style-type: none"> ■ Se crea un vector unidimensional de puntos de máscaras. ■ Se llama al método VideoAnalyzer::selectAndApplyMask() pasándole dichos puntos.
Comprobaciones	<ul style="list-style-type: none"> ■ Ahora es un vector bidimensional. ■ La longitud del vector ahora es la mitad del original.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test.is_masks_are_being_placed
Pasos realizados	<ul style="list-style-type: none"> ■ Se define un vector de puntos para incluir una máscara. ■ Se aplica la máscara a un fotograma leído llamando al método VideoAnalyzer::selectAndApplyMask().
Comprobaciones	<ul style="list-style-type: none"> ■ Un punto interior a la máscara tiene el color de la máscara.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test.if_pre_post_images_are_saved
Pasos realizados	<ul style="list-style-type: none"> ■ Se define un objeto de tipo impacto. ■ Se indica que deseamos guardar 1 fotograma previo y posterior al impacto. ■ Se llama al método VideoAnalyzer::saveAllImpacts().
Comprobaciones	<ul style="list-style-type: none"> ■ Se han creado tres archivos de imagen, uno para el propio impacto, otro para el fotograma posterior y un último para el fotograma previo.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_log_is_saved
Pasos realizados	<ul style="list-style-type: none">■ Se llama al método VideoAnalyzer::saveLogFile()
Comprobaciones	<ul style="list-style-type: none">■ Se ha creado un archivo llamado log.json
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Nombre prueba	test_if_initial_frame_is_ok
Pasos realizados	<ul style="list-style-type: none">■ Se define un fotograma de inicio y otro de final para acotar el análisis.
Comprobaciones	<ul style="list-style-type: none">■ El fotograma donde inicia el análisis es el indicado.■ El fotograma donde termina el análisis es el indicado.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Pruebas unitarias clase Dator

Nombre prueba	test_if_dator_created_then_error
Pasos realizados	<ul style="list-style-type: none">■ Se intenta crear un objeto de tipo Dator.
Comprobaciones	<ul style="list-style-type: none">■ Excepción obtenida debido a clase no instanciable.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Pruebas unitarias clase FSDator

Esta clase como ya se mencionó en el apartado de implementación, pretendía implementar los métodos de la clase abstracta Dator con el objetivo de conseguir persistencia basada en sistemas de ficheros. Debido a la gran similitud de los tests realizados para probar dichos métodos se presenta el esquema general que siguen todos ellos en vez de entrar en detalle de cada uno:

Nombre prueba	prueba_genérica_FSDator
Pasos realizados	<ul style="list-style-type: none">■ Se define una estructura de datos a guardar (Frame, Log, ...) o se carga una estructura de datos.
Comprobaciones	<ul style="list-style-type: none">■ Se han guardado o cargado dichos datos y los valores son correctos.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Pruebas unitarias ZepazoParams

Para este programa también contamos con un conjunto de pruebas unitarias donde se modifican los valores de las cajas numéricas, sliders, checkboxes... y se comprueban si los valores calculados con métodos de la clase VideoAnalyzer e ImageAnalyzer son correctos.

De esta forma, como únicamente hacen pruebas que consisten en cambios en los valores en la interfaz y la comprobación de que el método de la clase VideoAnalyzer recibe bien los valores y hace los cálculos oportunos, se presenta un esquema genérico de las pruebas realizadas para esta clase:

Nombre prueba	prueba_genérica_ZepazoParams
Pasos realizados	<ul style="list-style-type: none">■ Se cambian valores haciendo uso de elementos de la interfaz.■ Se llaman a los métodos de la interfaz que operan con dichos valores que a su vez llaman a métodos de la clase VideoAnalyzer o ImageAnalyzer.
Comprobaciones	<ul style="list-style-type: none">■ Los valores obtenidos por las clases VideoAnalyzer e ImageAnalyzer son correctos.■ El cálculo realizado es correcto.
Criterio de éxito	Todas las comprobaciones correctas
Criterio de fallo	Alguna de las comprobaciones incorrectas

Pruebas unitarias ZepazoCrop y ZepazoVerify

Las pruebas realizadas para estos dos scripts consisten en lanzar ambos scripts con parámetros erróneos e ir comprobando que se lancen las excepciones oportunas hasta que en el último test de cada fichero se lanza el programa con los parámetros correctos y se comprueba:

- **ZepazoCrop:** El vídeo ha sido recortado correctamente comprobando su duración.
- **ZepazoVerify:** El log resultante muestra los datos esperados de acuerdo a los dos logs iniciales aportados.

6.2. Pruebas de aceptación

Las pruebas de aceptación son pruebas las cuales se llevan a cabo normalmente antes del lanzamiento de una nueva versión de un producto con el objetivo de determinar si el producto se ajusta a las necesidades de los clientes.

En nuestro caso, de cara a obtener pruebas de aceptación, durante todo el desarrollo del proyecto se realizaron reuniones con el cliente al cual, como se mencionó en el apartado de metodología, [2.2](#) se le presentaron diversos prototipos resultado del avance del proyecto. De esta forma, durante estas reuniones se le presentó al cliente un determinado prototipo que incluía un conjunto de características nuevas de acuerdo a los objetivos y requisitos iniciales así como modificaciones indicadas por el mismo cliente en la presentación de prototipos anteriores con el objetivo de obtener retroalimentación del mismo.

Gracias a estas reuniones se han podido incorporar modificaciones al proyecto para adecuarlo a las necesidades del propio cliente, como por ejemplo adición de nuevos requisitos así como modificaciones/matizaciones de los mismos. A continuación se presenta de forma resumida las reuniones realizadas con el cliente donde podremos ver tanto las características del prototipo presentado como la opinión del cliente acerca del mismo:

Fecha:	8-03-2021
Prototipo presentado:	<p>Este primer prototipo se centra en el programa principal Zepazo, encargado del análisis de vídeos. Para ello se presenta una primera aproximación a la interacción con el mismo proponiendo un sistema de parámetros para indicar los argumentos necesarios con el objetivo de lanzar el análisis.</p> <p>Este primer prototipo incluye adicionalmente:</p> <ul style="list-style-type: none"> ■ Carga de vídeos. ■ Primera aproximación a la detección de impactos mediante una simple resta de fotogramas. ■ Primera aproximación a la detección del contorno lunar mediante una aproximación a una forma circular en vez de elipse (incluida finalmente). ■ Previsualización del proceso de análisis. ■ Inclusión del parámetro CircleLimit para variar el índice de detección. ■ Cálculo de información relativa al vídeo útil para ubicar en cuestiones temporales posteriormente impactos.
FeedBack del cliente:	<p>Añadir: Deberían tratarse errores relacionados con la carga de vídeos en rutas erróneas. Debería añadirse una aclaración de cada tipo de argumento utilizado por el programa. El programa debería mostrar ayuda a la hora de seleccionar argumentos. Incluir valor por defecto cuando sea posible para los parámetros. El proceso de previsualización debe poder abortarse de forma sencilla pulsando una tecla.</p> <p>Modificar: En la previsualización deberán mostrarse todos los posibles impactos que tomen lugar para ver si los parámetros están bien ajustados.</p> <p>Otras consideraciones: El cliente indica que el programa debe contar con un rendimiento tal que el análisis no se demore mucho tiempo más que el que tomaría reproducirlo. Debe tenerse en cuenta cuestiones de rendimiento durante todo el desarrollo. El cliente indica que sería interesante mantener la aproximación de detección del contorno lunar como una figura circular debido a que podría estimarse el centro de la superficie de forma sencilla de cara a en algún momento ubicar impactos en un mapa lunar ayudándonos de dicho punto. No obstante la aproximación por elipse no debe suponer un problema si mejora la detección de impactos.</p>

Fecha:	6-04-2021
Prototipo presentado:	<p>Este segundo prototipo se sigue centrando en el programa principal dedicado al análisis de los vídeos en busca de impactos, Zepazo. Es por ello que se ha trabajado en incluir o mejorar lo siguiente:</p> <ul style="list-style-type: none"> ■ Se ha refactorizado el código de tal forma que ahora se cuenta con un fichero dedicado a las operaciones relacionadas con el tratamiento de vídeo y otro fichero para las funciones relacionadas con el tratamiento de imágenes. ■ Se controlan errores relacionados con valores erróneos de argumentos. ■ Se muestra ayuda a la hora de seleccionar argumentos. ■ Valores por defecto a argumentos añadidos. ■ La previsualización muestra toda la información disponible para facilitar la elección de argumentos así como puede abortarse en cualquier momento pulsando la letra 'Q'. ■ El contorno lunar finalmente se aproxima mediante una elipse. ■ Añadida la posibilidad de utilizar máscaras. ■ Los posibles impactos detectados se marcan en los fotogramas en los que toman lugar. ■ Si un impacto toma lugar fuera de la elipse que marca el contorno lunar, es descartado en cambio, dentro es válido y además añadido guardado de fotogramas donde toman lugar impactos..
FeedBack del cliente:	<p>Añadir: Posibilidad de seleccionar la ruta donde guardar las imágenes de los impactos. Considerar otras formas de añadir máscaras aunque se mantenga la opción del vector de coordenadas, como por ejemplo con una interfaz. Considerar la implementación de un mecanismo que permita evitar falsos positivos ocurridos debido a las zonas iluminadas de la luna.</p> <p>Modificar: Utilizar de cara al modo Debug, un código de color que nos permita diferenciar si un impacto ha sido o no descartado en función de si toma lugar fuera o dentro de la superficie lunar.</p> <p>Otras consideraciones: Debido a la complejidad creciente del número de parámetros, lo que da lugar a comandos muy extensos y a la dificultad de un nuevo usuario para su determinación, se recomienda elevar la prioridad de la tarea opcional de la interfaz de ayuda a la selección de parámetros.</p>

Fecha:	21-04-2021
Prototipo presentado:	<p>Este tercer prototipo debido a la creciente complejidad de los parámetros en cuanto a número y dificultad de determinación de los mismos, se ha centrado en la elaboración del programa ZepazoParams dedicado a ayudar al usuario a la elección y ajuste de los mismos. Es por ello que la interfaz permite:</p> <ul style="list-style-type: none"> ■ Ajustar el límite de detección. ■ Ajustar el valor del que depende el cálculo de la elipse mediante un cálculo automático o selección manual. ■ Ajustar el valor de dilatación. ■ Añadir máscaras. ■ Mostrar previsualización del análisis con los valores seleccionados aplicados. ■ Restablecer todos los parámetros. ■ Cargar un vídeo. ■ Exportar e importar parámetros. <p>Adicionalmente en el programa Zepazo se han incluido mejoras como la dilatación de fotogramas así como la importación de parámetros desde un archivo JSON generado por el programa ZepazoParams.</p>
FeedBack del cliente:	<p>Añadir:</p> <ul style="list-style-type: none"> Posibilidad de eliminar máscaras seleccionadas. Posibilidad de seleccionar la carpeta donde guardar fotogramas desde la interfaz. Posibilidad de seleccionar el número de fotogramas previos y posteriores a guardar. Controlar rangos de los valores de los argumentos para no permitir la selección incorrecta de los mismos desde este programa así como mostrar mensajes de error cuando sea necesario. Añadir un botón que permita previsualizar el comando que sería necesario utilizar para ejecutar el programa Zepazo con los argumentos seleccionados. <p>Modificar:</p> <ul style="list-style-type: none"> Intentar considerar la posibilidad de detectar de forma automática el valor del argumento CircleLimit mediante la selección por ejemplo de un área de la superficie lunar, investigar este tema. <p>Otras consideraciones:</p> <ul style="list-style-type: none"> Como hay suficiente tiempo, elevar la prioridad de los objetivos opcionales relacionados con el recorte de vídeos y verificación de impactos tratando de crear scripts sencillos.

Fecha:	25-05-2021
Prototipo presentado:	<p>Se presenta un prototipo donde se han incluido las mejoras indicadas por el cliente en la última reunión relacionadas con la interfaz como eran:</p> <ul style="list-style-type: none"> ■ Posibilidad de eliminar máscaras seleccionadas. ■ Posibilidad de seleccionar la carpeta donde guardar fotogramas desde la interfaz. ■ Posibilidad de seleccionar el número de fotogramas previos y posteriores a guardar. ■ Controlados rangos de los valores de los argumentos para no permitir la selección incorrecta de los mismos desde este programa así como ahora mostramos mensajes de error cuando es necesario. <p>En cuanto al programa Zepazo, se ha trabajado en mejoras generales de optimización. Y se han añadido las siguientes funciones:</p> <ul style="list-style-type: none"> ■ Posibilidad de guardar fotogramas previos y posteriores a la detección de un posible impacto. ■ Posibilidad de analizar un vídeo desde un fotograma hasta otro concreto. ■ Al terminar el análisis se genera un resumen adicionalmente en formato JSON que nos permite ver información sobre los posibles impactos detectados. <p>Adicionalmente se han creado dos scripts correspondientes con dos nuevos programas:</p> <ul style="list-style-type: none"> ■ ZepazoCrop: Nos permite recortar un vídeo desde un segundo hasta otro indicado. ■ ZepazoVerify: Haciendo uso de los resúmenes generados de dos vídeos trata de validar impactos utilizando información temporal de los mismos.
FeedBack del cliente:	<p>El cliente verifica que este último prototipo satisface sus necesidades indicando que no obstante podría retomarse el trabajo en un futuro añadiendo nuevas funcionalidades como la ubicación de impactos en un mapa lunar entre otros.</p> <p>El cliente indica que le gustaría disponer de un manual de usuario.</p>

6.3. Modo debug

Adicionalmente se han realizado también pruebas con lo que he denominado modo 'Debug' del programa principal Zepazo que ha sido un modo de ejecución del programa que nos permite realizar un seguimiento del proceso de análisis pudiendo comprobar si las nuevas funcionalidades operaban correctamente. Para ejecutar la aplicación Zepazo en modo Debug simplemente ejecutamos:

```
python3 src/zepazo.py [lista de argumentos] -d
```

De esta forma, el modo 'Debug' me ha permitido realizar una visualización del proceso de análisis del vídeo donde al ejecutarlo contábamos con la reproducción del vídeo que se pretendía analizar y además se muestra información como:

- Impactos detectados como válidos: Es decir aquellos posibles impactos interiores a la superficie lunar, enmarcados en un rectángulo de color rojo.
- Impactos detectados como no válidos: Es decir aquellos posibles impactos exteriores a la superficie lunar, enmarcados en un rectángulo de color morado.
- Impactos detectados como posibles: Es decir aquellos posibles impactos detectados que no se ha podido ubicar ni dentro ni fuera de la superficie lunar ya que la elipse no se ha podido calcular, enmarcados en un rectángulo de color amarillo.
- Elipse que recubre la superficie lunar, se muestra una elipse de color amarillo tal y como se haya ajustado por parámetros.
- Las diversas máscaras establecidas, se muestran los rectángulos de las máscaras de color negro sobre las coordenadas indicadas.

Y se puede deducir información como:

- Si la dilatación de fotogramas se aplica correctamente, ya que en vídeos donde aparecen zonas iluminadas de la luna, si no aplicamos ningún valor de dilatación podemos ver como se detectan posibles impactos (falsos positivos) en dichas zonas mientras que al aplicar dilatación el problema desaparece.
- Si el cálculo automático de la elipse que recubre la superficie lunar funciona mayormente bien al no indicar un valor para el argumento CircleLimit observando si se ajusta al contorno lunar.

Cuando la aplicación se ejecuta en modo Debug no se almacenan los fotogramas de los posibles impactos detectados así como tampoco se almacenan imágenes de fotogramas previos y posteriores en caso de indicarse ni el resumen final. Adicionalmente se puede finalizar la previsualización del análisis, es decir, salir del modo Debug pulsando la tecla 'Q'.

Capítulo 7: Conclusiones y trabajos futuros

El problema del cual partimos al inicio del documento y que daba origen a este proyecto era la enorme dificultad que suponía tratar con las grandes cantidades de información con las que se trata en este ámbito (largas horas de vídeo) que suponían un gran esfuerzo humano para ser analizadas. Por otra parte tal y como se planteó al inicio, los impactos normalmente dejaban un destello tras un impacto en la superficie lunar de normalmente décimas de segundo lo que añadía dificultad para su detección por parte de una persona encargada de analizar las grabaciones obtenidas.

En consecuencia, el objetivo principal de este proyecto era intentar eliminar o minimizar los problemas anteriores mediante la creación de un software que automatizase la detección de impactos de tal forma que la intervención humana fuese la menor posible y se intentasen obtener buenos resultados.

Como resultado, tal y como se ha propuesto en este documento, se ha desarrollado el conjunto de programas agrupados bajo el nombre de Zepazo el cual intenta proponer una solución a los problemas planteados anteriormente proporcionando programas para el análisis de vídeo, ajuste de parámetros, recorte de vídeos y verificación de impactos.

El programa tal y como se ha visto en el apartado de pruebas, es capaz de descartar una gran cantidad de falsos positivos ocasionados por situaciones analizadas a lo largo de este proyecto como pueden ser cuerpos brillantes que aparecen en escena (estrellas por ejemplo) gracias al cálculo de la posición de la superficie lunar mediante una aproximación a través de una elipse que recubre dicha superficie siendo capaces de descartar posibles detecciones exteriores a la misma. Así como zonas de la superficie lunar iluminadas que producían problemas debido a su alto brillo intermitente mediante la dilatación de fotogramas. Además el uso de máscaras para ocultar zonas no deseadas o problemáticas como temporizadores y el ajuste de parámetros de operaciones mediante argumentos nos ha permitido crear un software principal, Zepazo, encargado del análisis de vídeos que es capaz de funcionar correctamente y adaptarse a las condiciones casi únicas de cada vídeo.

Adicionalmente me gustaría comentar que el proyecto realizado tiene una aplicación directa en el proceso de investigación realizado por el Instituto de Astrofísica de Andalucía (IAA) y otros grupos del extranjero que están probando y utilizando satisfactoriamente el software desarrollado con el objetivo de detectar impactos de meteoroides sobre la superficie lunar de forma automática.

Con el desarrollo del conjunto de este conjunto de programas se han conseguido abordar los **objetivos 1.4 obligatorios**: OBJ0-OBJ6 (y sub-objetivos relacionados) así como una gran mayoría de los objetivos **opcionales** como son aquellos relacionados con la interfaz: OBJ7, OBJ8 donde se proponía el uso de máscaras, OBJ10 donde el sistema podría mostrar un log resumiendo el análisis del vídeo, OBJ11 relacionado con el recorte de vídeos que como hemos visto ha dado lugar al programa ZepazoCrop, OBJ12 donde se proponía la posibilidad de analizar los vídeos en un rango determinado de fotogramas y el OBJ13 donde se plantaba la posibilidad de validar impactos, conseguido mediante el programa ZepazoVerify, dejando únicamente sin realizar el OBJ9 relacionado con la ubicación espacial de impactos y el OBJ14 relacionado con contemplar la precisión temporal vía GPS.

En cuanto a la **temporización inicial** propuesta, *el orden de desarrollo de los distintos objetivos a cumplir se ha conseguido respetar mayormente* alterando únicamente la realización de un único objetivo opcional, el OBJ3.2 relacionado con la dilatación de fotogramas, cuyo desarrollo se pospuso hasta prácticamente el final del desarrollo de la interfaz debido a que en el momento en el que tocaba su desarrollo adicionalmente se proponían desarrollar varios sub-objetivos más coincidiendo con entregas de otras asignaturas, no obstante ante esta situación se pretendió dar prioridad a los objetivos obligatorios y es por ello que el desarrollo de este objetivo no ha seguido el orden establecido inicialmente.

Si nos fijamos en los **plazos establecidos por semanas de la temporización inicial**, también hemos podido ajustarnos casi perfectamente a excepción de que la realización del objetivo opcional relacionado con la creación del programa ZepazoParams, que proporcionaba una interfaz que ayudaba al usuario al ajuste de parámetros para el posterior análisis, cuya extensión temporal se había previsto de dos semanas, se prolongó una semana más debido a características extras propuestas por el cliente durante las reuniones. Adicionalmente una parte de los tests se han añadido al final con el objetivo de obtener una mayor cobertura del código intentando obtener un mayor porcentaje de código testeado para asegurar su calidad.

Las **herramientas** discutidas al inicio del documento que fueron detenidamente estudiadas de forma previa al desarrollo nos han permitido desarrollar el proyecto de una forma muy sencilla y no se ha modificado ninguna de ellas. Tal y como se comentó OpenCv, módulo de Python orientado al tratamiento de imágenes y vídeo ha sido una base muy importante para el proyecto que ha facilitado muchas tareas que de otra forma hubiesen sido mucho más complejas ya que gracias a la elección cuidadosa de ciertas operaciones que provee OpenCv hemos sido capaces de conseguir alcanzar objetivos proponiendo soluciones originales y funcionales.

Aunque como se ha comentado el conjunto de programas funciona de forma correcta y el programa principal Zepazo es capaz de adaptar de forma satisfactoria el análisis gracias al ajuste que hace el usuario de los mismos aún **existe margen de mejora desde un doble punto de vista:**

- ***Ajuste automático de parámetros:*** Con el objetivo de reducir la complejidad en la introducción y determinación de los argumentos se introdujo el programa ZepazoParams que permitía de forma intuitiva ajustar los parámetros, no obstante como trabajo futuro se debería trabajar en el estudio del cálculo automatizado de los parámetros de entrada que dependen de cada vídeo como por ejemplo el cálculo automático del valor de la elipse que realiza nuestro programa principal y arroja en numerosas ocasiones buenos resultados (posteriormente ajustable por el usuario) gracias a una sencilla operación matemática para el cálculo del valor umbral de la función threshold de OpenCv de la que depende principalmente el cálculo de la elipse. De esta forma, detectar estos valores clave y posterior estudio de su determinación automática con el objetivo de conseguir una mayor automatización deberá revisarse en un futuro para conseguir un programa menos complejo de entrada para el usuario, aunque siempre debería existir la posibilidad de su ajuste manualmente haciendo uso del programa ZepazoParams para asegurar la máxima perfección en la determinación de dichos argumentos y en consecuencia un mejor análisis.
- ***Nuevas funcionalidades:*** Con el objetivo de obtener un producto cada vez más completo podrían introducirse funcionalidades como la relacionada con el OBJ9, donde se propone la aproximación en un mapa lunar de la ubicación de un posible impacto cuando es detectado para su posterior estudio por expertos así como debería intentar realizarse el OBJ14 para contemplar la posibilidad de contar con un sistema GPS que nos permita poseer una mayor precisión temporal a la hora de ubicar temporalmente impactos. Adicionalmente podría añadirse un nuevo bloque de funcionalidades, ayudándonos de expertos en la materia, relacionado con la determinación de las propiedades físicas de los meteoroides como su masa, tamaño o incluso la velocidad a la que se produjo el impacto basándonos en elementos observables como el cráter producido así como propiedades físicas como la intensidad del destello producido tras el impacto que puede aportar información acerca de la energía liberada por el mismo.

Para terminar y como conclusión final me gustaría destacar que gracias a este proyecto he podido adquirir una gran cantidad de conocimientos principalmente acerca de dos campos que me encantan como son la informática y la astronomía. De esta forma, he sido capaz de adquirir un mayor conocimiento acerca de algoritmos y operaciones relacionados con el tratamiento de archivos multimedia digitales como son las imágenes y el vídeo pero también he sido capaz de apreciar e incorporar todo el conocimiento adquirido durante estos cuatro años en este proyecto. Así como he conseguido aprender mucho acerca de astronomía gracias a los artículos consultados y sobre todo a mi tutor Sergio que ha seguido y apoyado el proyecto en todo momento como un apasionado más de la astronomía y la informática.

Bibliografía

[1] **Autor:** Borovička, Jiří.

Título: Physical and chemical properties of meteoroids as deduced from observations.

Recurso: <https://www.cambridge.org/core/journals/proceedings-of-the-international-astronomical-union/article/physical-and-chemical-properties-of-meteoroids>

Año: 2005.

Páginas: Proceedings of the International Astronomical Union 249-271.

DOI: 10.1017/S1743921305006782.

[2] **Autor:** Stern, S. Alan.

Título: The lunar atmosphere: History, status, current problems and context.

Recurso: <https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/1999RG900005>

Año: 1999.

Páginas: Reviews of Geophysics 453-491.

DOI: 10.1029/1999RG900005.

[3] **Autores:** J. L. Ortiz, P. V. Sada, L. R. Bellot Rubio, F. J. Aceituno, J. Aceituno, P. J. Gutiérrez y U. Thielek.

Título: Optical detection of meteoroidal impacts on the Moon.

Recurso: <https://www.nature.com/articles/35016015>

Año: 2000.

Páginas: Nature 921-923.

DOI: 10.1038/35016015.

[4] **Autor:** J. L. Ortiz, F. J. Aceituno, J. Aceituno.

Título: A search for meteoritic flashes on the Moon.

Recurso: <http://adsabs.harvard.edu/pdf/1999A&26A...343L.570>

Año: 1999.

Páginas: Astron. Astrophys. 343, L57–L60.

- [5] **Autores:** elosh, Jay and Artemjeva, N. and Golub, A. and Nemchinov, I. and Shuvalov, V. and Trubetskaya, I..
Título: Remote visual detection of impacts on the lunar surface.
Recurso: <http://adsabs.harvard.edu/pdf/1993LPI....24..975M>
Año: 1993.
Páginas: 975-976.
- [6] **Autores:** Toro, A. D., Jiménez, B. B.
Título: Metodología para la elicitation de requisitos de sistemas software.
Recurso: <http://www.lsi.us.es/docs/informes/lsi-2000-10.pdf>
Año: 2000.
Documento: Informe Técnico LSI-2000-10. Facultad de Informática y Estadística Universidad de Sevilla.
- [7] **Nombre del sitio:** Neolita Website: Lunar Impact Events (NELIOTA - Home).
Recurso: <https://neliota.astro.noa.gr/>
- [8] **Autor:** J. M. Madiedo.
Nombre del sitio: The MIDAS Project
Recurso: https://www.cosmos.esa.int/documents/549247/643223/Madiedo_The_MIDAS_project.pdf University of Huelva, Spain.
- [9] **Nombre del sitio:** NELIOTA - Archive: Detected NEO Lunar Impact Event, ID: 20210317_180718, NEOLITA.
Recurso: <https://neliota.astro.noa.gr/DataAccess>
- [10] **Nombre del sitio:** Python3 Argparse Module.
Recurso: <https://docs.python.org/3/library/argparse.html>
- [11] **Nombre del sitio:** Python3 OpenCV Module, Morphological Transformations.
Recurso: https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html
- [12] **Nombre del sitio:** Python3 OpenCV Module, Color Space Conversions.
Recurso: https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html

- [13] **Nombre del sitio:** Python3 OpenCV Module, Image Thresholding.
Recurso: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html
- [14] **Nombre del sitio:** Python3 OpenCV Module, Contours.
Recurso: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html
- [15] **Nombre del sitio:** Python3 OpenCV Module, Arithmetic Operations on Images.
Recurso: https://docs.opencv.org/3.4/dd/d4d/tutorial_js_image_arithmetics.html
- [16] **Nombre del sitio:** Python3 OpenCV Module, VideoCapture Class Reference.
Recurso: https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html
- [17] **Nombre del sitio:** Python3 OpenCV Module, Flags for video I/O.
Recurso: https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html
- [18] **Nombre del sitio:** Python3 OpenCV Module, High-level GUI, imshow() method.
Recurso: https://docs.opencv.org/4.5.2/d7/dfc/group_highgui.html#ga453d42fe4cb60e5723281a89973ee563
- [19] **Nombre del sitio:** Python3 PyQt5 Module.
Recurso: <https://pypi.org/project/PyQt5/>
- [20] **Nombre del sitio:** Python3 Moviepy Module.
Recurso: <https://pypi.org/project/moviepy/>
- [21] **Nombre del sitio:** Docker docs Website
Recurso: <https://docs.docker.com/>
- [22] **Nombre del sitio:** Pytest Website.
Recurso: <https://docs.pytest.org/en/6.2.x/contents.html>
- [23] **Nombre del sitio:** Poetry Website.
Recurso: <https://python-poetry.org/>
- [24] **Nombre del sitio:** GitHub Actions Website.
Recurso: <https://github.com/features/actions>

Apéndice A: Manual de usuario

En este capítulo se muestra el manual de usuario el cual se incluye en el repositorio de GitHub en el Readme para que un usuario nuevo sea capaz de familiarizarse con los programas y la curva de aprendizaje sea sencilla. Adicionalmente, en el repositorio se incluye este mismo manual traducido al inglés, no obstante mostramos el manual en español a continuación:

A.1. Manual de usuario Zepazo

Zepazo es un conjunto de programas que pretende aportar una solución automatizada para la detección de impactos a través del análisis de grabaciones de zonas oscuras de la Luna. Con el objetivo de facilitar dicha tarea al usuario se proponen cuatro programas:

- **zepazo**: Programa principal encargado del análisis de las grabaciones.
- **zepazoParams**: Programa que aporta una sencilla interfaz para el establecimiento asistido de parámetros.
- **zepazoVerify**: Programa encargado de verificar/validar los impactos de dos grabaciones con el objetivo de encontrar impactos en momentos temporales similares y validarlos.
- **zepazoCrop**: Programa que pretende aportar una herramienta de recorte de vídeos con el objetivo de no tener que descargar los vídeos completos del servidor para ajustar los parámetros haciendo uso del programa ZepazoParams.

A.1.1. Instalación del conjunto de programas

Para instalar el conjunto de programas simplemente debemos tener instalado Poetry el cual se puede instalar de la siguiente forma:

```
1 pip install poetry
```

Una vez instalado poetry podemos instalar todas las dependencias de nuestro programa simplemente ejecutando:

```
1 poetry install
```

Nota: Si al ejecutar el programa ZepazoParams se obtiene un error de compatibilidad de PyQt con OpenCv deberemos bajar la versión de OpenCv de la siguiente forma:

```
1 pip install opencv-python==4.3.0.36
```

A.1.2. Zepazo

Es el programa principal cuyo objetivo es analizar un vídeo dado por un usuario en busca de los posibles impactos que hayan podido tomar lugar durante dicha grabación.

Este programa se ejecuta de la siguiente forma:

```
1 python3 src/zepazo.py [lista de argumentos]
```

Y contamos con la siguiente lista de argumentos para ajustar el proceso de análisis:

Abreviatura	Completo	Utilidad	Tipo	Uso	Valor por defecto
-h	-help	Ayuda de parámetros. Se muestran los posibles parámetros que el programa puede utilizar		Opcional	
-v	-video	Ruta del vídeo a analizar.	Cadena de caracteres	Obligatorio	
-d	-debug	Mostrar una simulación del proceso de análisis con los parámetros indicados, no se guardan los posibles impactos detectados	Boolean	Opcional	False

-cm	-coordinatesmask	Lista de puntos indicando las coordenadas de las máscaras a añadir. Cada dos elementos de la lista indicará una máscara, de esta forma, el primer elemento de cada par de coordenadas será la esquina superior izquierda del rectángulo y el segundo será la esquina inferior derecha.	Lista de enteros.	Opcional	
-l	-detectionlimit	Índice de detección de impactos, este valor depende del brillo del vídeo introducido	Entero positivo	Opcional	50
-cl	-circlelimit	Valor que sirve para ajustar la elipse al contorno lunar, si no se establece el argumento, se calculará un valor por defecto—Entero positivo	Opcional	Calculado dinámicamente	

-f	-folder	Ruta de la carpeta donde almacenar los fotogramas de los impactos detectados	Cadena de caracteres	Opcional	Ubicación del vídeo
-ssf	-saveSurrounding Frames	Número de fotogramas previos y posteriores al impacto que se desean almacenar, por defecto únicamente se almacena el fotograma del impacto.	Entero positivo	Opcional	0
-dt	-dilate	Valor del kernel a utilizar para dilatar los fotogramas previos y evitar falsos positivos derivados de movimiento y zonas luminosas de la luna	Entero positivo	Opcional	0
-cf	-configFile	Ruta de un archivo de configuración generado por ZepazoParams para importar valores de los parámetros.	Cadena de caracteres.	Opcional	Ubicación del vídeo
-sf	-startingFrame	Número de fotograma desde el que comenzar el análisis	Entero positivo	Opcional	0
-ef	-endingFrame	Número de fotograma hasta el que realizar el análisis	Entero positivo	Opcional	Último fotograma

Una vez ajustados los argumentos comenzará el análisis y una vez que este concluya obtendremos en la carpeta seleccionada por argumentos los fotogramas se ha detectado un posible impacto así como un archivo de log que resume los posibles impactos encontrados tras el análisis realizado.

A.1.3. ZepazoParams

Este programa pretende ayudar al usuario a ajustar los valores de los parámetros que se van a utilizar en el programa 'zepazo' a través de una sencilla interfaz.

Este programa se ejecuta de la siguiente forma:

```
1 python3 src/Interface/zepazo_params.py
```

Y al ejecutar lo anterior se nos abre la interfaz:

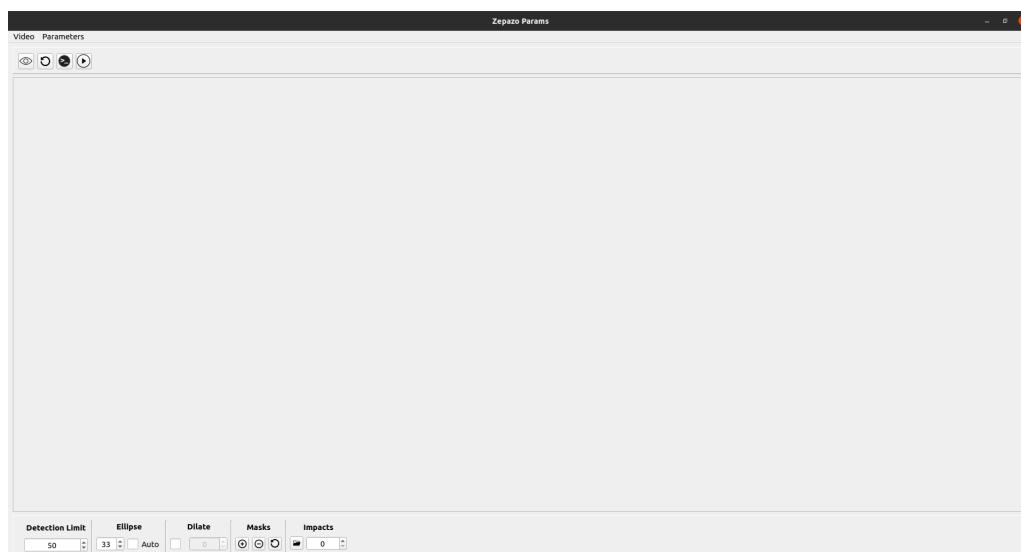


Figura A.1: Interfaz programa ZepazoParams sin vídeo seleccionado

En primer lugar para poder empezar a ajustar parámetros el usuario deberá seleccionar un vídeo haciendo uso del menú superior izquierdo 'Vídeo' y seleccionando la opción 'Open File' para elegir un vídeo. Una vez elegido un vídeo se mostrará el primer fotograma del mismo en la parte central de la interfaz:

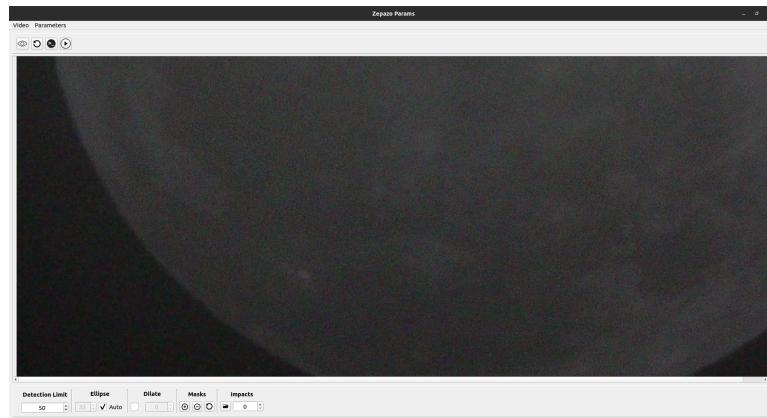


Figura A.2: Interfaz programa ZepazoParams, vídeo seleccionado

Una vez el vídeo ha sido seleccionado y contamos con una imagen del vídeo en pantalla, podemos empezar a ajustar parámetros de la siguiente forma (menú inferior):

- **Índice de detección:** por defecto presenta el valor de 50 y para vídeos con condiciones normales no debería variar mucho, no obstante puede aumentarse o disminuirse y pulsando al botón de la derecha con el símbolo de 'play' de la barra superior podremos ver como afecta el cambio de dicho valor.
- **Elipse:** En cuanto a la elipse que recubre la superficie lunar, por defecto viene activada la opción 'auto' la cual intenta obtener de forma automática una buena estimación. No obstante se puede desmarcar esta casilla y ajustar manualmente mediante el uso de la casilla numérica. Un ejemplo es el siguiente:

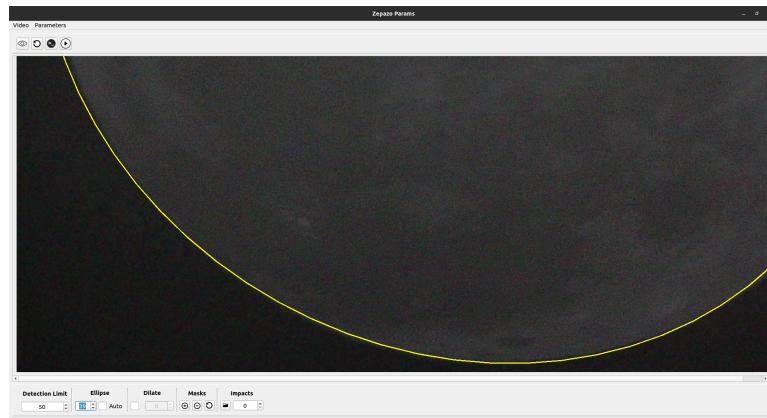


Figura A.3: Interfaz programa ZepazoParams, elipse ajustada

- **Dilate:** en este apartado podemos elegir si queremos o no dilatar los fotogramas de nuestro vídeo, es recomendable activar esta opción cuando en nuestro vídeo aparecen zonas iluminadas en la superficie lunar. Si se activa se podrá elegir el valor de dilatación en la casilla numérica y ver una previsualización mediante el botón 'play'.
- **Máscaras:** Este apartado nos permite añadir máscaras, eliminar o restablecer todas. Por ejemplo podríamos añadir las siguientes máscaras:



Figura A.4: Interfaz programa ZepazoParams, máscaras seleccionadas

- **Impactos:** Este apartado nos permite elegir tanto la carpeta donde guardar los impactos como adicionalmente elegir el número de impactos previos y posteriores a guardar cuando se detecta un impacto.

En cuanto al menú superior, encontramos las siguientes opciones (de izquierda a derecha):

- **Visualizar todos los parámetros:** Este botón que cuenta con el icono de un ojo, nos permite ver aplicados tanto las máscaras como la elipse al mismo tiempo ya que como hemos visto anteriormente, cuando se eligen máscaras únicamente se ven las máscaras y cuando se elige el valor de la elipse únicamente se ve la elipse, en este caso permite ver todo junto.
- **Restablecer todo:** Este segundo botón superior restablece a los valores por defecto todos los parámetros modificados.
- **Botón comando:** Al pulsar este botón obtenemos el comando completo con los parámetros seleccionados para lanzar el análisis con los valores elegidos así como se copia al clipboard. Un ejemplo es el siguiente:

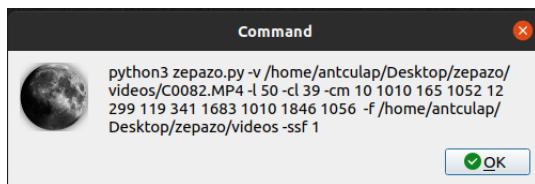


Figura A.5: Interfaz programa ZepazoParams, comando generado

- **Botón preview:** Este botón permite lanzar en modo debug el proceso de análisis para comprobar si los valores elegidos para los parámetros son correctos o no.

Adicionalmente, en el menú superior llamado 'Parameters' encontraremos la opción de exportar parámetros a JSON o importar desde JSON.

A.1.4. ZepazoVerify

Este programa pretende tras analizar dos vídeos que se han grabado en instantes temporales similares con el programa Zepazo, recoger los logs con el objetivo de identificar si existe alguna relación temporal entre los impactos del primer vídeo y los del segundo en cuanto a que ocurran en el mismo instante.

Este programa se ejecuta de la siguiente forma:

```
1 python3 src/zepazoVerify.py [ lista de argumentos ]
```

Y contamos con la siguiente lista de argumentos para ajustar el proceso de validación:

Abreviatura	Completo	Utilidad	Tipo	Uso
-lgf1	--logFile1	Log resultado del análisis del primer vídeo	Cadena de caracteres	Obligatorio
-lgf2	--logFile2	Log resultado del análisis del segundo vídeo	Cadena de caracteres	Obligatorio
-rlfg	--resultingLogFile	Ruta donde guardar el log resultante	Cadena de caracteres	Obligatorio
-mt	--marginTime	Margen temporal para que dos impactos con una diferencia indicada por este parámetro sean emparejados	Entero positivo	Obligatorio

De esta forma si ejecutamos el comando:

```
1 python3 src/zepazoVerify.py -lgf1 '/home/user/logs/log1.json' -lgf2 '/home/user/logs/log2.json' -rlfg '/home/user/logs/log_verify.json' -mt 10
```

Estaríamos intentando validar los impactos del vídeo que ha generado el log1 con los del vídeo que ha generado el log2 emparejando los mismos con una diferencia temporal máxima de 10 segundos y almacenando el resultado en el fichero log_verify.json.

De esta forma el resultado obtenido tras ejecutar este programa es un fichero como por ejemplo el siguiente:

```

1 [
2     {
3         "difference" : 5 ,
4         "log_1_impact" : 1 ,
5         "log_2_impact" : 0 ,
6         "path_to_log1" : ".../ test/log.json" ,
7         "path_to_log2" : ".../ test/log2.json"
8     }
9 ]

```

Donde podemos ver que se ha validado un impacto ya que la diferencia temporal ha sido de 5 segundos, en el log1 el impacto ha sido el número 1, en el log2 el impacto ha sido el número 0 y se aporta información adicional de la ubicación de los logs por si fuese necesario consultar información concreta de los impactos emparejados.

A.1.5. ZepazoCrop

Este programa nos permite recortar vídeos con el objetivo de no sobrecargar la red del servidor, ya que si un usuario desea descargar un vídeo para utilizar el programa zepazoParams con el objetivo de ajustar los parámetros para posteriormente lanzar el análisis no necesita tener el vídeo completo, con un fragmento breve es suficiente.

Este programa se ejecuta de la siguiente forma:

```

1 python3 src/zepazoCrop.py [ lista de argumentos ]

```

Y contamos con la siguiente lista de argumentos para ajustar el proceso de recorte:

Abreviatura	Completo	Utilidad	Tipo	Uso
-vo	--videoOriginal	Ruta del vídeo a recortar	Cadena de caracteres	Obligatorio
-vc	--videoCropped	Ruta donde guardar el vídeo recortado	Cadena de caracteres	Obligatorio
-ss	--secondStart	Segundo desde el que recortar el vídeo	Entero positivo	Obligatorio

-se	-secondEnd	Segundo hasta el que recortar el vídeo	Entero positivo	Obligatorio
-----	------------	--	-----------------	-------------

De esta forma si ejecutamos el comando:

```
1 python3 src/zepazoCrop.py -vo "/home/user/videos/video1" -vc "/home/user/videos/video1\_crop" -ss 120 -se 140
```

Estaríamos recortando el vídeo cuyo nombre es video1 desde el segundo 120 hasta el 140 y almacenando el resultado con el nombre video1_crop.

