



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de una herramienta de ayuda a la colimación de telescopios

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Nicolás García Sastre

Tutor: Alberto José Pérez Jiménez

Curso 2020-2021

Resumen

En este proyecto se desarrolla un programa para la ayuda a la colimación de telescopios reflectores con el fin de hacer más fácil el proceso, permitiendo al usuario conectar su telescopio o cámara con el programa.

Para ello se desarrolló una librería para la realización de los cálculos necesarios para la colimación y la conexión con el telescopio, y un programa de escritorio que utiliza dicha librería para mostrar al usuario los pasos a realizar. Todo ello se desarrolló mediante C++ y Qt.

Palabras clave: Telescopio, colimación, reflector, librería, C++, Qt

Resum

En aquest projecte es desenvolupa un programa per a l'ajuda a la col·limació de telescopis reflectors per tal de fer més fàcil el procés, permetent a l'usuari connectar el seu telescopi o càmera amb el programa.

Per a això es va desenvolupar una llibreria per a la realització dels càlculs necessaris per a la col·limació i la connexió amb el telescopi, i un programa d'escriptori que utilitza aquesta llibreria per mostrar a l'usuari els passos a realitzar. Tot això es va desenvolupar mitjançant C++ i Qt.

Palabras clave: Telescopi, col·limació reflector, llibreria, C++, Qt

Abstract

In this project, a program is developed to help the collimation of reflecting telescopes in order to make the process easier, allowing the user to connect their telescope or camera with the program.

For this, a library was developed to carry out the necessary calculations for collimation and connection with the telescope, and a desktop program that uses this library to show the user the steps to be carried out. All of this was developed using C++ and Qt.

Key words: Telescope, collimation, reflecting, library, C++, Qt

Índice general

Índice general	V
Índice de figuras	VII
Índice de tablas	VIII

1	Introducción	1
1.1	Motivación	2
1.2	Objetivos	2
1.3	Impacto esperado	2
1.4	Metodología	3
1.5	Estructura de la memoria	3
2	Estado del arte	5
2.1	Herramientas	5
2.2	Programas relacionados	6
2.2.1	Astro-Snap	6
2.2.2	MetaGuide	6
2.2.3	CCD Inspector	7
2.3	Crítica al contexto tecnológico	8
2.4	Propuesta	8
3	Análisis del problema	9
3.1	Especificación de requisitos	9
3.1.1	Requisitos específicos	12
3.1.2	Requisitos no funcionales	14
3.2	Solución propuesta	15
4	Diseño de la solución	17
4.1	Arquitectura del sistema	17
4.2	Diseño detallado	17
4.2.1	Esquema relacional	17
4.2.2	CollimatorLib	19
4.2.3	CollimatorDesktop	19
4.2.4	Base de datos	21
4.3	Tecnología utilizada	22
4.3.1	Lenguaje de programación	23
4.3.2	Qt	23
4.3.3	SQLite	24
4.3.4	Docker	24
4.3.5	Linux	25
4.3.6	Subversion (SVN)	25
4.3.7	Doxygen	25
4.3.8	OpenMP	26
5	Desarrollo de la solución propuesta	27
5.1	Procesado de imágenes	27
5.1.1	Optimización de los cálculos	31

5.2	Persistencia	31
5.3	Vistas	32
5.4	Conexión con el servidor INDI	35
5.4.1	Obtención de dispositivos	35
5.4.2	Obtención de imágenes	36
5.5	Documentación	36
6	Implantación	39
6.1	Generación del instalable	39
6.2	Instalación	41
7	Pruebas	43
7.1	Pruebas unitarias	43
7.1.1	Persistencia	43
7.1.2	Conexión con INDI	43
7.1.3	Colimación	43
7.2	Pruebas de integración	44
7.3	Pruebas de validación	44
8	Conclusiones	45
8.1	Conclusiones finales	45
8.2	Relación con los estudios cursados	46
8.3	Trabajos futuros	46
	Bibliografía	49
<hr/>		
	Apéndices	
A	Funcionamiento del programa	51
A.1	Listado de telescopios	51
A.2	Vista de un Telescopio	52
A.3	Vista de la configuración de procesado de imágenes	56
A.4	Vista de la configuración de tornillos	57
A.5	Vista de la configuración de conexión	58
B	Doxygen	61
C	Estudio de la paralelización de los cálculos	65
D	Diagramas	69

Índice de figuras

1.1	Espejos de un telescopio reflector	1
2.1	Astro-Snap	6
2.2	MetaGuide	7
2.3	CCD-Inspector	7
3.1	Diagrama de dominio	9
3.2	Caso de uso de configuración y persistencia	10
3.3	Caso de uso de colimación	11
3.4	Estrella desenfocada	15
4.1	Diagrama de clases reducido	18
4.2	Diagrama de base de datos	18
4.3	Qt designer	24
4.4	Funcionamiento de Subversion	25
5.1	Centro de masa y cuadro delimitador	28
5.2	Rectas calculadas para la colimación	29
5.3	Circunferencia Giniométrica	30
5.4	Resultado de los cálculos de la colimación	30
5.5	Gráfica de los tiempos de ejecución de la normalización	31
5.6	Diálogo de conexión con servidor remoto en Qt Designer	33
5.7	Vista de telescopio	34
5.8	Diagrama de navegación	34
5.9	Conexión con servidor INDI	35
6.1	Estructura de ficheros para dpkg-deb	40
6.2	Estructura de ficheros tras la instalación	41
A.1	Vista de lista de Telescopios	51
A.2	Vista de la adición de un telescopio	52
A.3	Vista principal de un Telescopio	53
A.4	Vista de depuración de imagen	54
A.5	Vista de colimación automática	55
A.6	Vista de colimación manual	56
A.7	Vista de la configuración de Procesado de imágenes	57
A.8	Vista de la configuración de tornillos	58
A.9	Vista de la adición de tornillo	58
A.10	Vista de la configuración de conexión	59
B.1	Portada de Doxygen	61
B.2	Lista de clases en Doxygen	62
B.3	Clase en Doxygen	63
B.4	Call graph en Doxygen	63
B.5	Caller graph en Doxygen	63

C.1	Tiempos de ejecución de la normalización	66
C.2	Resultados de la paralelización del cálculo del centro de masa	67
D.1	Diagrama de clases de la aplicación de escritorio	69
D.2	Diagrama de clases de la librería	70
D.3	Diagrama BPMN de la conexión con INDI	71

Índice de tablas

3.1	Requisito funcional 1: Creación de configuración	12
3.2	Requisito funcional 2: Configuración de procesado de imagen	12
3.3	Requisito funcional 3: Configuración de visualización de la imagen	13
3.4	Requisito funcional 4: Uso de un modo manual o automático	13
3.5	Requisito funcional 5: Configuración de conexión	13
3.6	Requisito funcional 6: Configuración de conexión	14
3.7	Requisito funcional 7: Configuración de conexión	14
3.8	Requisito funcional 7: Configuración de conexión	14
3.9	Requisito no funcional 1: Mantenibilidad	15
3.10	Requisito no funcional 2: Eficiencia	15
7.1	Pruebas de integración	44
C.1	Resultados de la paralelización de la normalización	66
C.2	Tiempos de ejecución del cálculo del centro de masa	66
C.3	Speed-Up y eficiencia de la normalización	67
C.4	Speed-Up y eficiencia del cálculo del centro de masa	67

CAPÍTULO 1

Introducción

La colimación de un telescopio reflector [1] consiste en la alineación de los espejos de este para lograr que la imagen esté perfectamente enfocada.

Un telescopio reflector se compone de tres componentes ópticos o espejos: el espejo primario, que es un espejo esférico parabólico que hace de superficie principal de captación de luz; el espejo secundario, que es un espejo situado en un plano diagonal a 45° que dirige y extiende la luz al tercer espejo; y por último, el espejo llamado ocular de aumento, por donde mira directamente el observador.

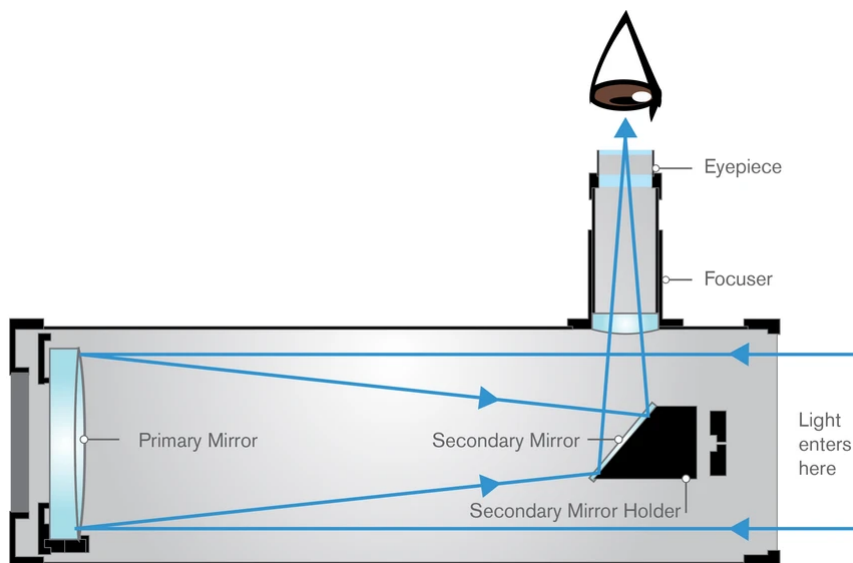


Figura 1.1: Espejos de un telescopio reflector. (Imagen extraída de ¹)

Cuando los componentes ópticos del telescopio no se encuentran bien calibrados, difícilmente se puede enfocar un objetivo, lo que a veces imposibilita el visionado a través del telescopio.

Para la realización de la correcta colimación del telescopio se deben ajustar los tornillos de los espejos, de manera que se logra que la imagen sea nítida y perfectamente visible, proceso que puede suponer un reto para las personas que se acaban de iniciar en la astronomía.

Este proyecto trata de facilitar el proceso, mostrando al usuario cada paso que debe llevar a cabo para que su telescopio quede colimado con la precisión que desee. Le permite configurar los distintos parámetros para obtener los pasos que le sirvan para colimar

¹<https://www.celestron.com/blogs/knowledgebase/the-ultimate-guide-to-celestron-optical-tubes>

su telescopio de forma automática, o dibujar sobre las fotografías para realizar la colimación de forma manual. Todo esto, mediante imágenes tomadas previamente o en tiempo real conectándose a la cámara de su telescopio.

1.1 Motivación

El principal problema que se desea abordar con este proyecto es la dificultad que conlleva colimar un telescopio, además de la gran cantidad de tiempo que se puede llegar a invertir en esta tarea si no se posee experiencia previa. Se trata, pues, de hacer mas fácil el proceso de la colimación de telescopios reflectores con ayuda de software, de forma que la colimación sea rápida, sencilla e intuitiva para el usuario.

1.2 Objetivos

El objetivo principal es desarrollar una herramienta que ayude a los usuarios que utilicen telescopios reflectores a realizar la colimación de los mismos. Dicha herramienta está enfocada tanto para aquellos usuarios experimentados, como para los principiantes, y únicamente precisarían de un telescopio y una cámara.

Los objetivos específicos que debe cumplir la herramienta son los siguientes:

- Proporcionar un sistema de guiado para la colimación de telescopios, que de forma automática, detecte que tornillos son necesarios ajustar.
- Posibilitar la configuración de varios telescopios o un mismo telescopio con varias configuraciones.
- Realizar una interfaz de usuario que sea intuitiva y fácil de utilizar para un usuario inexperto.
- Guiar al usuario durante la colimación mediante imágenes tomadas previamente.
- Guiar al usuario durante la colimación mediante imágenes en tiempo real.
- Posibilitar colimación de forma manual.
- Posibilitar la realización de capturas de pantalla durante la colimación.
- Despliegue automático de la aplicación.

1.3 Impacto esperado

Se espera que el proyecto en su totalidad sea útil para la comunidad aficionada a la astronomía y para aquellos usuarios, tanto principiantes como avanzados, que posean un telescopio reflector y quieran colimarlo de forma fácil.

También se espera que la librería desarrollada para el proyecto pueda ser de utilidad para otros desarrolladores, y pueda usarse para desarrollar otros programas o páginas web para la colimación de telescopios, tanto para programas de ordenador como para aplicaciones móviles.

1.4 Metodología

Para la realización del trabajo hemos aplicado la metodología ágil, en concreto el modelo SCRUM. Ello nos permitió adaptarnos rápidamente a los requisitos especificados, e ir aumentando las funcionalidades del programa con cada sprint, sin necesidad de rediseñar ni reimplementar partes o la totalidad del programa.

Durante cada sprint, se realizaron diagramas, modelos, y recursos necesarios para el correcto análisis de requisitos y su posterior implantación para poder llevar a cabo el desarrollo para la solución del problema planteado.

Concretamente, se plantearon tres sprints. En el primero, abordamos el comportamiento principal del problema: el reconocimiento de las imágenes. En el segundo, se realizó la interfaz de la aplicación, y durante el tercero, la conexión con servidores INDI.

1.5 Estructura de la memoria

La memoria de este trabajo se compone de nueve capítulos, cuyo contenido se explica a continuación:

- **Capítulo 1, Introducción:** En este capítulo se expone la motivación del proyecto, así como los objetivos que se quieren cumplir y junto a la metodología que se va a seguir para conseguirlo.
- **Capítulo 2, Estado del arte:** En este apartado se exponen las herramientas disponibles actualmente para la realización de este proyecto, además de un análisis de los programas relacionados que existen actualmente y una crítica al contexto tecnológico. Finalmente se explica la propuesta que se hace en este proyecto.
- **Capítulo 3, Análisis del problema:** Se analiza el problema que se plantea en este proyecto, así como la solución propuesta y el plan de trabajo a seguir.
- **Capítulo 4, Diseño de la solución:** Se explica el diseño del proyecto para abordar el problema planteado, así como las tecnologías utilizadas para ello.
- **Capítulo 5, Desarrollo de la solución propuesta:** Se detalla cómo se ha abordado la implementación del diseño planteado en el anterior capítulo.
- **Capítulo 6, Implantación:** Se explica el despliegue del sistema en los equipos.
- **Capítulo 7, Pruebas:** Se explica cómo se han realizado una serie de pruebas para la correcta verificación de la implementación realizada.
- **Capítulo 8, Conclusiones:** En este capítulo se han resumido los temas abarcados en el trabajo, junto a los objetivos cumplidos y su solución. Además, se ha relacionado el proyecto con los estudios cursados y se han propuesto trabajos futuros que se pueden realizar sobre este proyecto.

A continuación, se incluye una serie de referencias bibliográficas que han sido utilizadas como estudio y referencia para abordar la investigación necesaria para la realización del proyecto.

Finalmente, se incluyen los siguientes apéndices:

- **Funcionamiento del programa:** En este apéndice, se explica cada una de las funcionalidades del programa, navegando por cada una de las interfaces y analizando el cometido de cada una de ellas.
- **Doxygen:** En este apéndice se explica el funcionamiento de la documentación generada a través de la herramienta doxygen.
- **Estudio de la paralelización de los cálculos:** Aquí se desarrolla una explicación detallada de cómo se han paralelizado algunos cálculos para optimizar el programa, además de cómo afecta esto al mismo.
- **Diagramas:** En este apartado se incluyen los diagramas completos del proyecto.

CAPÍTULO 2

Estado del arte

En este apartado hablaremos sobre el avance de la tecnología de las últimas décadas, imprescindible para la realización de este proyecto, de las herramientas de software necesarias y sobre las aplicaciones relacionadas ya existentes.

Actualmente existen varios métodos manuales para la colimación de telescopios reflectores [2], sin embargo el software que automatice este proceso es limitado.

Existe mucha diversidad de telescopios, muchos son digitales y permiten la conexión inalámbrica con programas y aplicaciones[3], tanto para la visualización como para el control directo de las monturas de los telescopios.

Además, gracias al avance de la fotografía digital, hay muchas cámaras capaces de conectarse con ordenadores, tanto mediante cable como de forma inalámbrica. Existen, además, adaptadores universales que permiten insertar cámaras digitales en los telescopios, reemplazando o adicionando al ocular, incluso cámaras especiales CCD capaces de acoplarse directamente en el telescopio.

2.1 Herramientas

Hoy en día, el software relacionado con el procesamiento y edición de imágenes está muy avanzado. Por ende, también se dispone de grandes librerías con muchas funcionalidades que permiten cargar, visualizar, procesar y editar imágenes de forma muy sencilla.

Entre las más conocidas estaría OpenCV[4], una librería de código abierto disponible para C, C++ y Python desarrollada originalmente por Intel en 1999 [5], que permite una amplia gama de funciones para el procesamiento de imágenes. Dispone de mucha documentación y es muy utilizada para detección de objetos e inteligencia artificial.

También está disponible Generic Image Library[6], una librería puesta en marcha por Adobe para la edición y procesamiento de imágenes. Esta librería es utilizada en la mayoría de proyectos de Adobe, hecho que demuestra su potencial y gran alcance.

CFITSIO [7] es una librería desarrollada por la NASA disponible para C y Fortran que proporciona rutinas de alto nivel para la lectura y escritura en archivos de tipo FITS (Flexible Image Transport System), que es el formato estándar que se utiliza para las fotografías astronómicas. Cuenta con multitud de ejemplos, referencias y documentación para su uso.

Para la conexión directa con telescopio o cámara, está disponible la librería INDI [8] que permite la conexión una amplia variedad de cámaras y telescopios, además de con diversos controladores e incluso es compatible con Raspberry Pi.

2.2 Programas relacionados

Contamos con gran cantidad de software para el control y visualización de las imágenes que aporta un telescopio, sin embargo se ha procedido a analizar exclusivamente el software que sea de ayuda para la colimación de telescopios:

2.2.1. Astro-Snap

Astrosnap es un software para la fotografía estelar que permite dibujar radios alrededor de los anillos de difracción en forma de mira telescópica para ayudar a la colimación de forma manual. Permite la conexión con cámaras o telescopios remotos, además de con imágenes. Sin embargo, no indica cuales serían los pasos a seguir para la colimación.

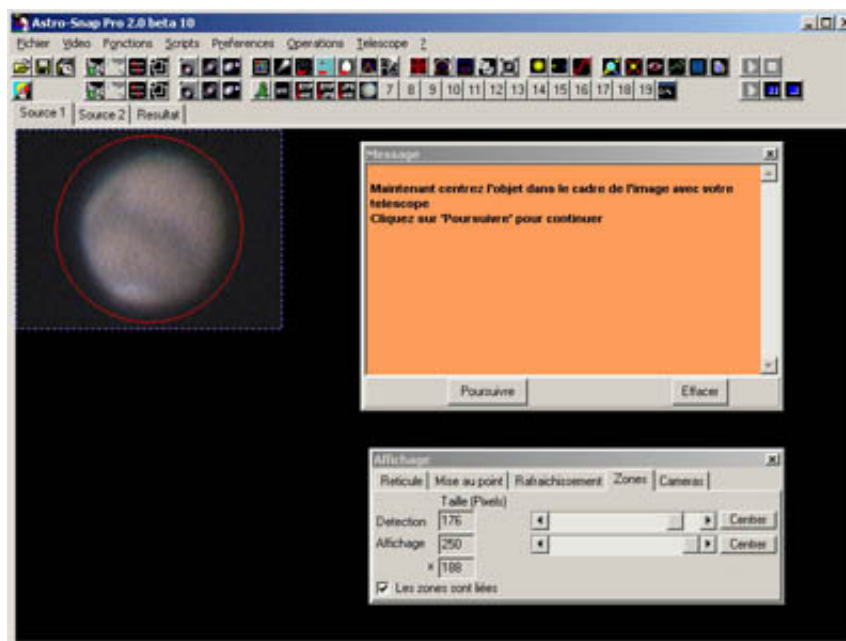


Figura 2.1: Astro-Snap.

2.2.2. MetaGuide

MetaGuide es un programa de escritorio para la corrección y guiado de monturas telescópicas. También dispone de ayuda a la colimación mostrando lo ajustada que está la difracción de la estrella.

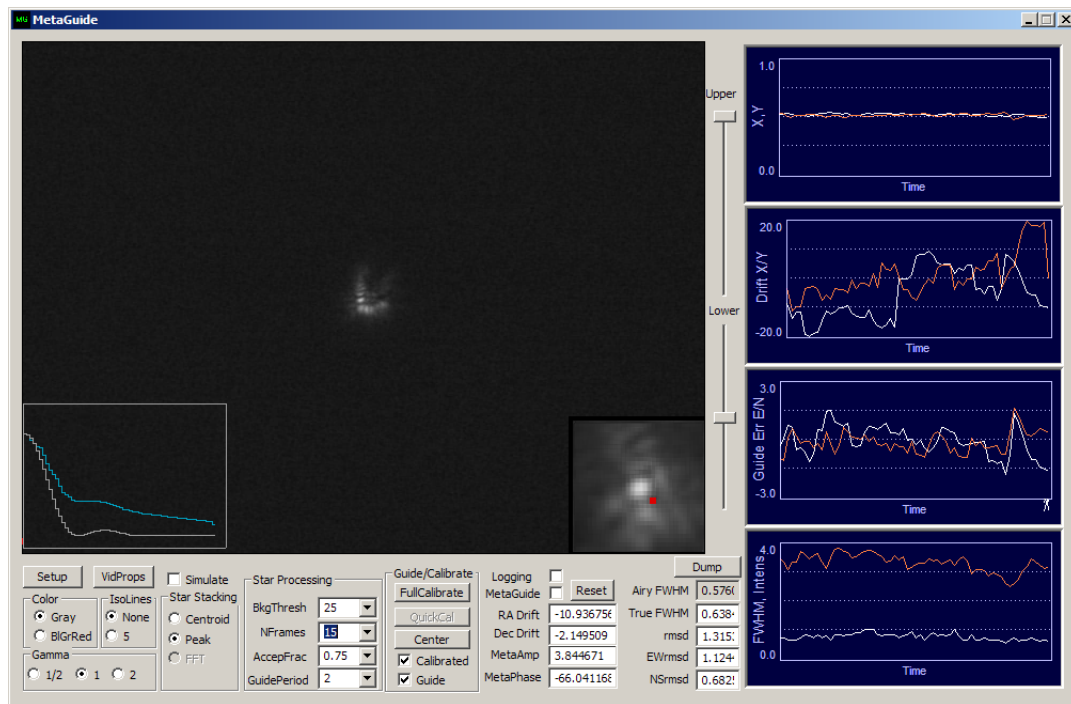


Figura 2.2: MetaGuide.

2.2.3. CCD Inspector

CCD Inspector es un software para colimación de telescopios mediante cámaras CCD[9]. Permite monitorizar en tiempo real el estado de la colimación del telescopio, sin embargo, tanto su configuración como su uso son complejos.

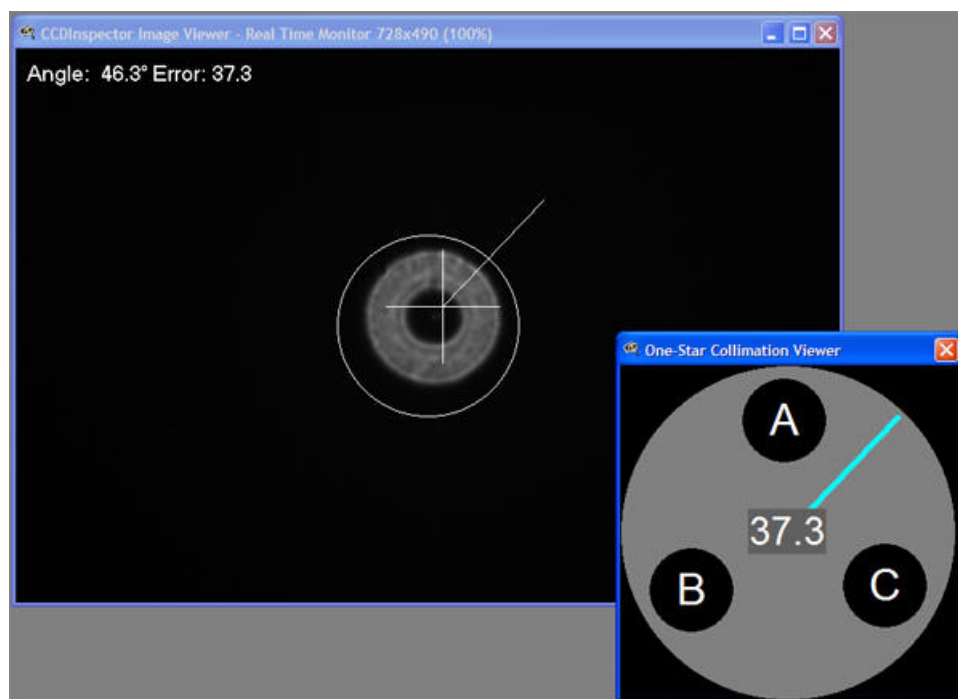


Figura 2.3: CCD-Inspector.

2.3 Crítica al contexto tecnológico

Después de investigar cada uno de los programas, se llegó a la conclusión de que se dispone poco software enfocado a la ayuda a la colimación de telescopios. Además, aquellos ya existentes no son automáticos o son de difícil uso.

2.4 Propuesta

Una vez vistas las similitudes y diferencias entre las aplicaciones homólogas a este proyecto, se puede observar cual es la necesidad del desarrollo.

En este proyecto se propone el desarrollo de una aplicación capaz de guiar al usuario a través de una colimación automática y sencilla. Durante esta colimación, el usuario podrá configurar fácilmente aquellos parámetros que crea que son necesarios para llevar a cabo la colimación. Además, permitirá al usuario conectarse de forma inalámbrica con su cámara o telescopio.

CAPÍTULO 3

Análisis del problema

Hoy en día, cuando alguien necesita usar su telescopio, tanto reflector como refractor, necesita utilizar parte de su tiempo en la calibración de sus componentes. Dichos componentes pueden variar dependiendo del tipo del telescopio: la montura, el buscador, o en el caso que nos interesa, los espejos.

Esta tarea de calibración de los componentes del telescopio conlleva un tiempo previo al uso del telescopio, aunque con la experiencia se atenúa, al principio puede llegar a ser tedioso y desesperante. Tanto como para que, aquellas personas que se inician con gran fascinación por la astronomía, acaben desinteresándose.

El objetivo de este proyecto es conseguir que el proceso de la colimación de los espejos del telescopio sea fácil y rápida, tanto para usuarios con experiencia como para aquellos que se acaban de iniciar.

En este documento se han definido los requisitos necesarios para el desarrollo del programa. Esta especificación de requisitos ha sido realizada siguiendo el estándar IEEE 830 [10].

3.1 Especificación de requisitos

Con el fin de obtener un mayor conocimiento sobre el dominio del problema y adaptarse a los términos del proyecto, se elaboró un diagrama de dominio (Figura 3.1).

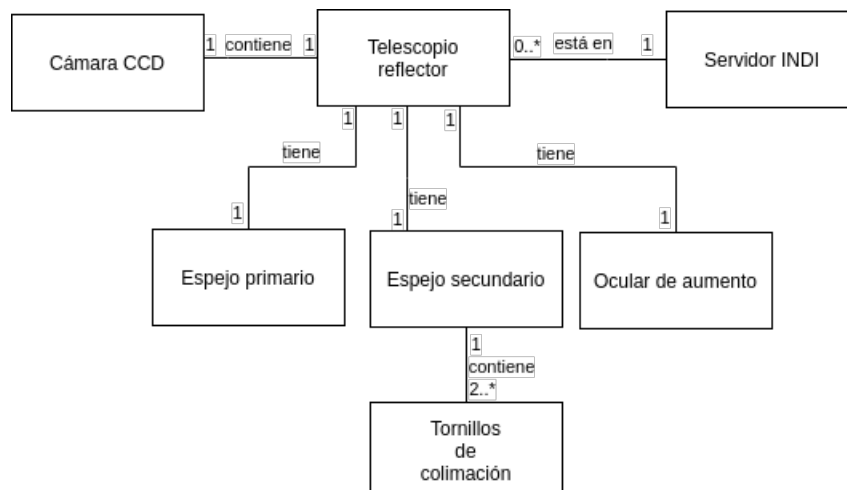


Figura 3.1: Diagrama de dominio (Elaboración propia)

La versión final del software debe cumplir con las siguientes funcionalidades:

- El sistema debe posibilitar el registro de varias configuraciones de telescopios, de forma que el usuario pueda elegir más adelante cual prefiere usar. Un mismo telescopio o cámara puede utilizar dos configuraciones diferentes.
- El sistema debe permitir a los usuarios configurar su telescopio a medida, dando la posibilidad de configurar las variables con las que el programa calculará la colimación, de manera que esta se adecúe al telescopio y a la precisión deseada.
- El sistema debe permitir a los usuarios configurar aquello que desea ver sobre la imagen. Debe facilitar la elección de si se desea o no ver los tornillos del telescopio y las referencias que crea la colimación automática.
- El sistema debe almacenar las configuraciones del usuario para reutilizarlas cuando desee colimar su telescopio de nuevo.
- El sistema debe permitir a los usuarios cargar imágenes desde su ordenador para que el sistema calcule los pasos para colimar el telescopio.
- El sistema debe permitir a los usuarios conectarse con su servidor para recibir imágenes automáticamente desde su cámara CCD o telescopio, para que el sistema calcule los pasos para colimar el telescopio.
- El sistema debe permitir a los usuarios seleccionar colimación manual con la que puedan dibujar una guía en forma de circunferencia para la colimación, o colimación automática, donde el sistema guía al usuario calculando como debe proceder.
- El sistema debe permitir a los usuarios tomar capturas de pantalla del procesado de las imágenes cargadas, con los resultados de la colimación automática o manual.

Durante la especificación de requisitos, se realizaron diagramas de casos de uso para incluir todas estas funciones. En este diagrama (figuras 3.2, 3.3) se definen todas las acciones que el usuario podrá realizar en el programa.

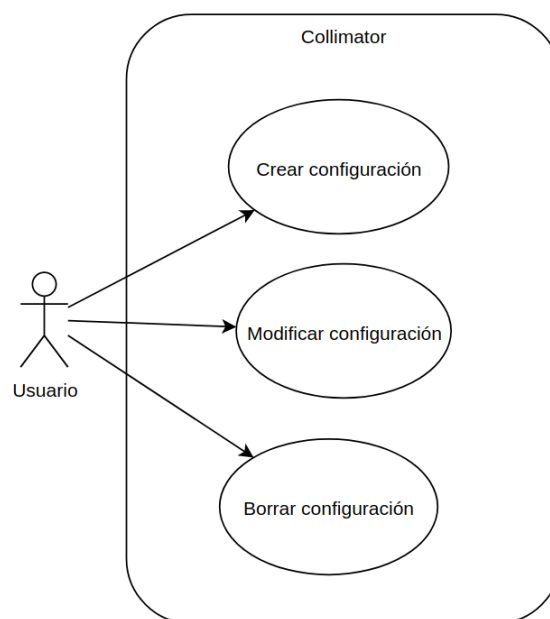


Figura 3.2: Caso de uso de configuración y persistencia (Elaboración propia)

- **Crear configuración:** El usuario debe poder crear configuraciones para sus telescopios.
- **Modificar configuración:** El sistema debe permitir que el usuario modifique las configuraciones anteriormente creadas.
- **Borrar configuración:** El usuario puede borrar una configuración.

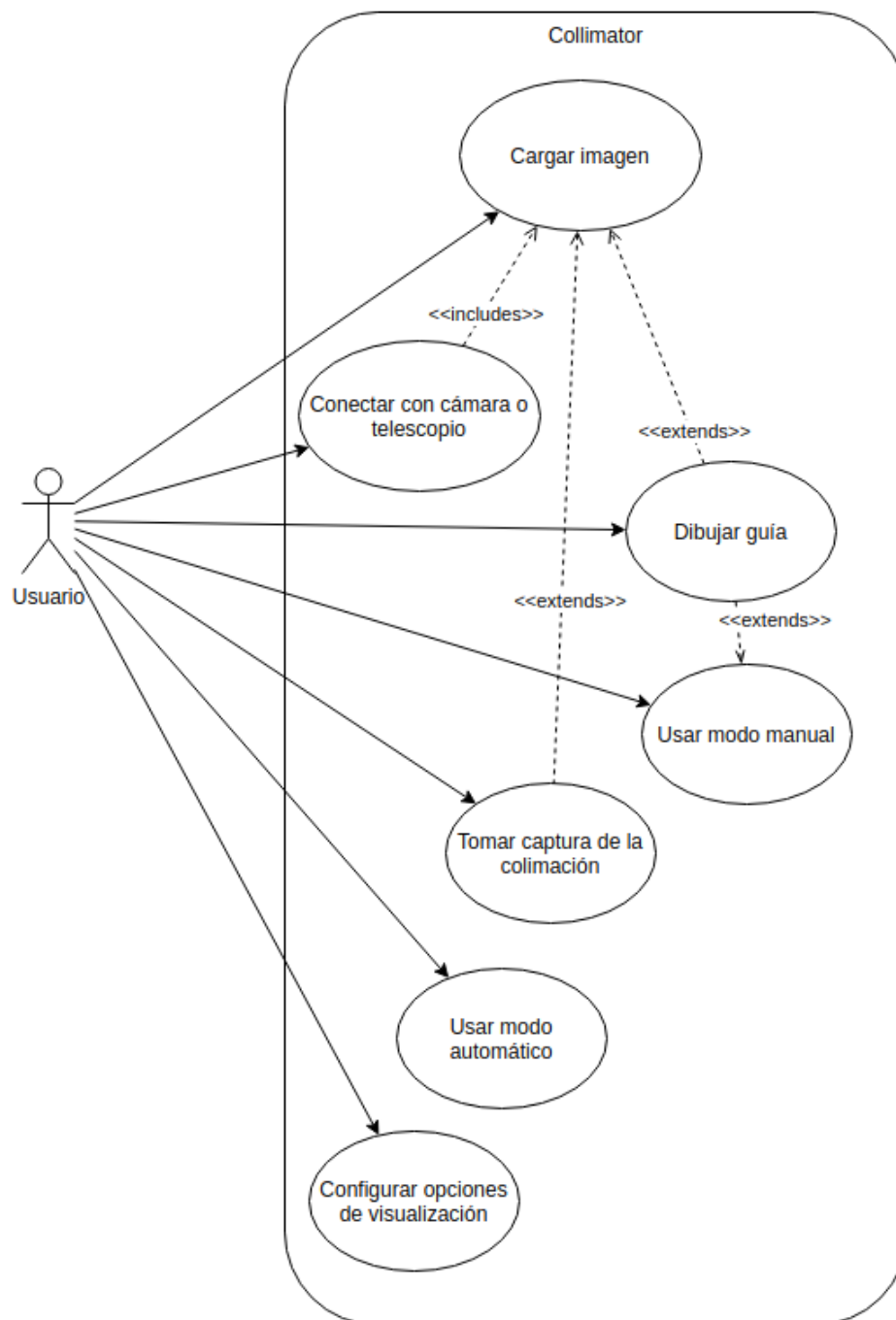


Figura 3.3: Caso de uso de colimación (Elaboración propia)

- **Cargar imagen:** El usuario debe poder cargar una imagen desde su ordenador para realizar la colimación.

- **Conectar con cámara o telescopio:** El usuario debe poder conectar con una cámara para obtener imágenes y realizar la colimación.
- **Usar modo automático:** El sistema debe permitir al usuario poder utilizar el modo automático de colimación que le enseña que pasos debe realizar para colimar su telescopio.
- **Usar modo manual:** El usuario debe poder utilizar el modo automático de colimación que le permite dibujar una guía en forma de mira sobre la imagen.
- **Dibujar guía:** El sistema debe permitir al usuario dibujar una guía en forma de mira sobre la imagen cuando esté utilizando el modo manual.
- **Tomar captura de la colimación:** El usuario debe poder tomar capturas de la imagen que recibe con la guía que haya dibujado en caso de que utilice el modo manual o el paso de colimación que esté observando en caso de que utilice el modo automático.
- **Configurar opciones de visualización:** El sistema debe permitir elegir al usuario aquellas cosas que desea ver sobre la imagen, ya sean los tornillos configurados o aquello que calcula el sistema durante la colimación automática.

3.1.1. Requisitos específicos

Los requisitos funcionales que construyen el alcance del proyecto son los siguientes:

Identificador del requisito	RF-01
Nombre de requisito	Creación de configuración
Descripción de requisito	Se debe permitir la creación de nuevas configuraciones para telescopios.
Requisitos relacionados	-
Precondición	-
Postcondición	-

Tabla 3.1: Requisito funcional 1: Creación de configuración

Identificador del requisito	RF-02
Nombre de requisito	Configuración de procesado de imagen
Descripción de requisito	Se debe permitir al usuario editar la configuración del procesado de imágenes para la colimación automática.
Requisitos relacionados	RF-01
Precondición	El usuario debe haber creado una configuración para su telescopio
Postcondición	-

Tabla 3.2: Requisito funcional 2: Configuración de procesado de imagen

Identificador del requisito	RF-03
Nombre de requisito	Configuración de visualización de la imagen
Descripción de requisito	Se debe permitir al usuario configurar aquello que desea ver sobre la imagen. En el modo manual debe permitir visualizar una mira y en el automático los cálculos que la aplicación haya hecho sobre la imagen.
Requisitos relacionados	RF-01, RF-04
Precondición	El usuario debe haber creado una configuración para su telescopio.
Postcondición	-

Tabla 3.3: Requisito funcional 3: Configuración de visualización de la imagen

Identificador del requisito	RF-04
Nombre de requisito	Uso de un modo manual o automático
Descripción de requisito	Se debe permitir al usuario elegir entre un modo manual donde el usuario dibuje una mira sobre la imagen para llevar a cabo la colimación o un modo automático donde el programa calcule automáticamente los pasos para la colimación.
Requisitos relacionados	RF-01
Precondición	El usuario debe haber creado una configuración para su telescopio.
Postcondición	-

Tabla 3.4: Requisito funcional 4: Uso de un modo manual o automático

Identificador del requisito	RF-05
Nombre de requisito	Configuración de conexión
Descripción de requisito	Se debe permitir al usuario configurar un servidor INDI desde donde recibir imágenes de su telescopio para colimarlos, así como la frecuencia con la que debe recibir dichas imágenes.
Requisitos relacionados	RF-01
Precondición	El usuario debe haber creado una configuración para su telescopio.
Postcondición	El usuario podrá conectarse con el servidor para recibir imágenes y colimar el telescopio.

Tabla 3.5: Requisito funcional 5: Configuración de conexión

Identificador del requisito	RF-06
Nombre de requisito	Cargar imagen manualmente
Descripción de requisito	Se debe permitir al usuario cargar una imagen guardada en su ordenador, ya sea para la colimación automática o manual.
Requisitos relacionados	RF-01
Precondición	El usuario debe haber creado una configuración para su telescopio.
Postcondición	-

Tabla 3.6: Requisito funcional 6: Configuración de conexión

Identificador del requisito	RF-07
Nombre de requisito	Cargar imagen desde servidor
Descripción de requisito	Se debe permitir al usuario cargar imágenes desde un servidor INDI, de forma que reciba cada cierto tiempo configurado las imágenes de su telescopio en el servidor.
Requisitos relacionados	RF-01, RF-05
Precondición	El usuario debe haber creado una configuración para su telescopio y haber configurado la conexión con el servidor INDI.
Postcondición	-

Tabla 3.7: Requisito funcional 7: Configuración de conexión

Identificador del requisito	RF-08
Nombre de requisito	Dibujar guía
Descripción de requisito	Se debe permitir al usuario dibujar una guía en forma de mira durante la colimación automática.
Requisitos relacionados	RF-01, RF-06/RF-07, RF-04
Precondición	El usuario debe haber creado una configuración para su telescopio y haber cargado una imagen, ya sea a través de INDI o desde su ordenador.
Postcondición	-

Tabla 3.8: Requisito funcional 7: Configuración de conexión

3.1.2. Requisitos no funcionales

El programa deberá contar con los siguientes atributos de calidad:

Identificador del requisito	RNF-01
Nombre de requisito	Mantenibilidad
Descripción de requisito	El software debe estar preparado para ser altamente modificable debido a necesidades evolutivas, correctivas o perfectivas.

Tabla 3.9: Requisito no funcional 1: Mantenibilidad

Identificador del requisito	RNF-02
Nombre de requisito	Eficiencia
Descripción de requisito	Los cálculos que se realicen sobre la imagen deben tardar menos de 1 segundo en completarse.

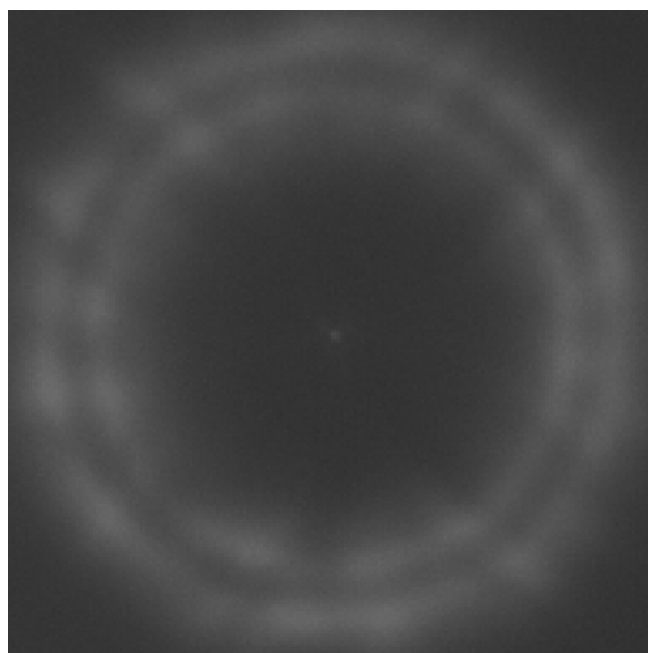
Tabla 3.10: Requisito no funcional 2: Eficiencia

3.2 Solución propuesta

La solución que se propone es crear una aplicación de escritorio capaz de mostrar al usuario imágenes de cómo debe colimar su telescopio tras haberlo configurado en la aplicación.

Para ello se utilizará el método de colimación mediante el desenfoque de una estrella, ya sea real o preferiblemente artificial.

Este método consiste en centrar el campo de visión del telescopio sobre una estrella y desenfocar la imagen. Al hacer esto, aparecen los llamados anillos de difracción. Estos, mediante su forma y brillo, nos indican el estado de la colimación del telescopio.

**Figura 3.4:** Estrella desenfocada (Imagen proporcionada por Alberto Pérez Jiménez)

Dados los anillos de difracción, se procede a atornillar o aflojar los tornillos de los espejos del telescopio. Esto se decide dependiendo de donde esté la zona con mayor cantidad de brillo. Si hubiera más brillo en la zona superior derecha y un tornillo en esta zona, habría que apretar el mismo.

Siguiendo estos pasos, apretando y aflojando los tornillos de colimación del telescopio, finalmente se consigue un brillo uniforme y una forma circular de los anillos de difracción.

Así pues, se propone que el programa utilice esta técnica. El usuario deberá enfocar el telescopio a una estrella y desenfocarla para que se visualicen los anillos de difracción. Una vez hecho esto, cargará su configuración en el programa con los tornillos y propiedades de imagen y conectará con su telescopio. Mientras se visualicen las imágenes en tiempo real, el usuario podrá ajustar su cámara, si hiciera falta, para hacer coincidir los dibujos de los tornillos con los tornillos reales.

Si el programa no fuera capaz de indicar correctamente cómo proceder, el usuario podrá realizar los ajustes necesarios en tiempo real, ya sea cambiando la configuración de la aplicación o moviendo el telescopio o cámara.

Finalmente, el programa deberá indicarle qué tornillos debe ir apretando o aflojando en el modo automático, hasta que su telescopio quede colimado.

CAPÍTULO 4

Diseño de la solución

4.1 Arquitectura del sistema

El proyecto desarrollado cuenta con una separación entre la lógica y las interfaces y la persistencia, mediante la creación de una librería separada de la propia aplicación de escritorio, dónde se consulta para llevar a cabo las operaciones sobre las imágenes, para así poder llevar más adelante parte de este proyecto a otras aplicaciones o ámbitos.

Esta librería es aquella encargada del cálculo sobre las imágenes y de la conexión con los servidores INDI, mientras que la aplicación, se encarga de la persistencia y de las interfaces.

4.2 Diseño detallado

Conforme a la organización de la aplicación, contamos con dos proyectos que separan la capa lógica de la persistencia y las interfaces. A la librería que contiene la lógica la llamamos CollimatorLib, mientras a la aplicación de escritorio, CollimatorDesktop.

4.2.1. Esquema relacional

A continuación se detalla el diagrama de clases reducido de la aplicación en la figura 4.1. El diagrama completo de clases se muestra en las figuras D.2 y D.1

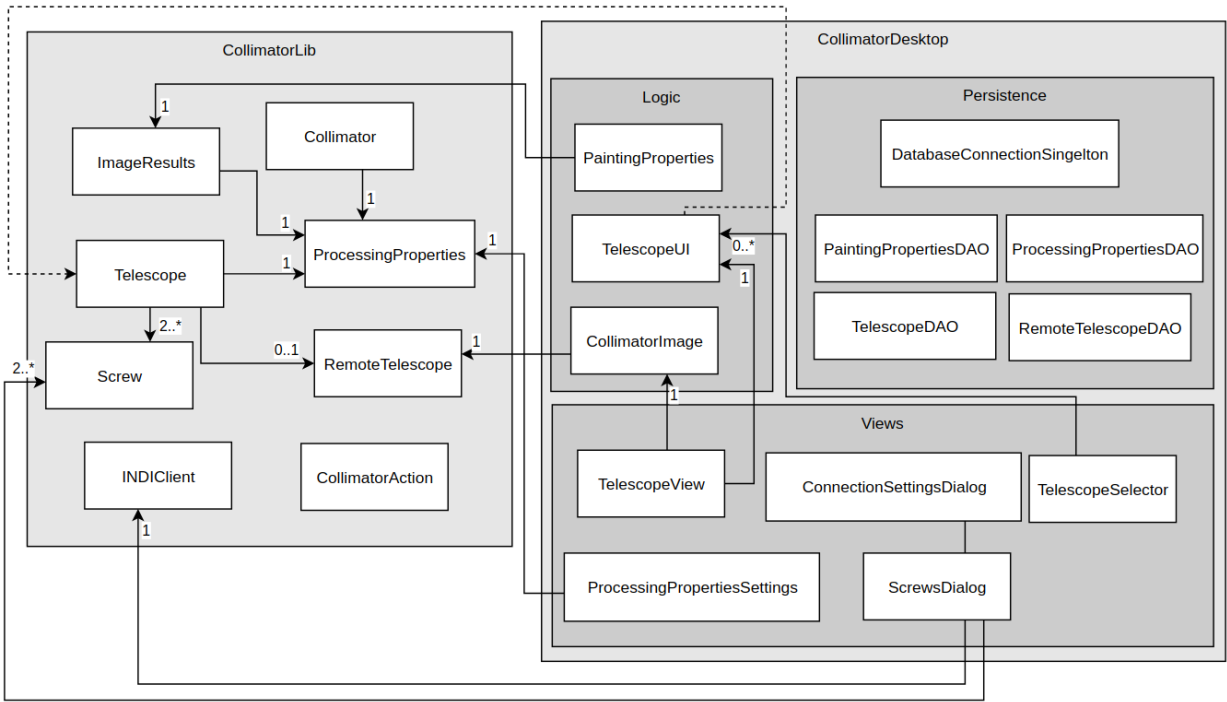


Figura 4.1: Diagrama de clases reducido (Elaboración propia)

Además, para la persistencia, al utilizar un motor SQL, detallamos el diagrama de la base de datos relacional.

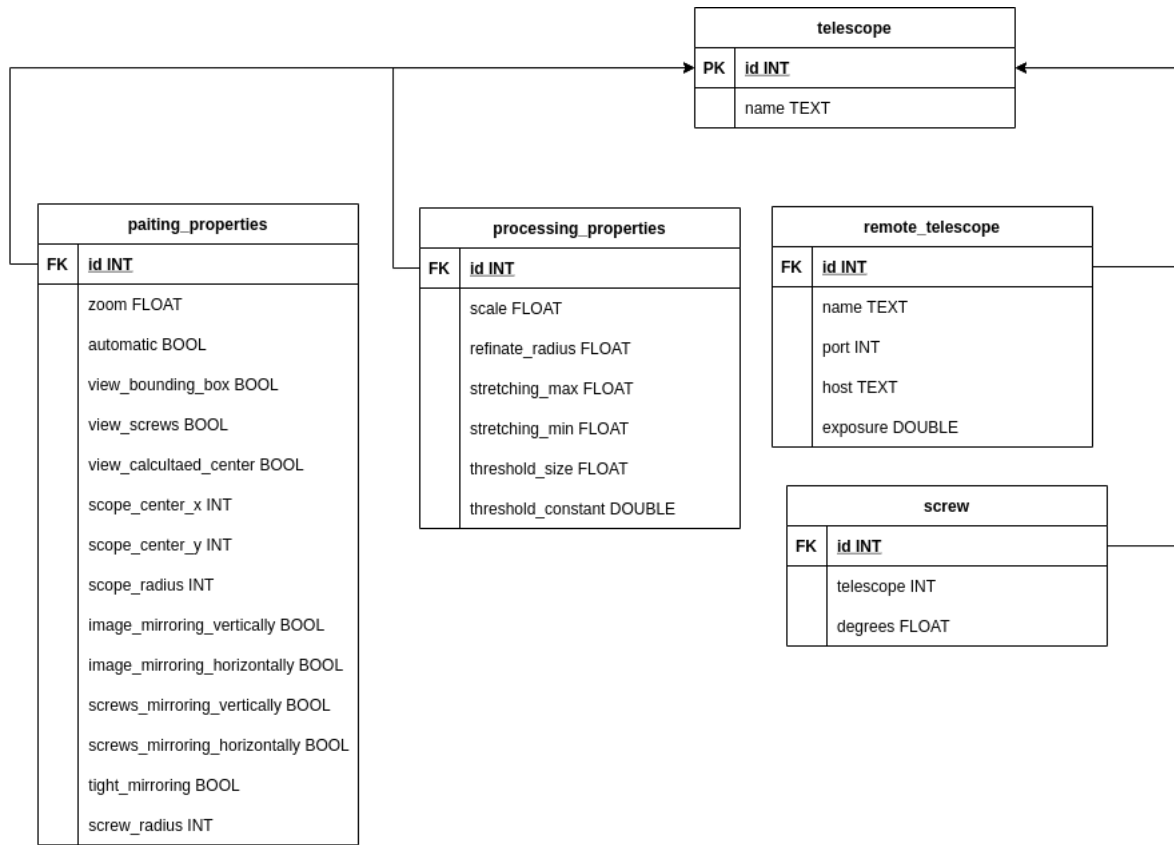


Figura 4.2: Diagrama de base de datos (Elaboración propia)

4.2.2. CollimatorLib

CollimatorLib es una librería que contiene la parte lógica de la aplicación. Contiene tanto los métodos que realizan cálculos para obtener la colimación, como los objetos necesarios para llevarla a cabo. Las clases que contiene son las siguientes:

- **Collimator:** Es la clase principal de la librería. Esta se encarga de realizar las transformaciones y cálculos sobre la imagen, además, permite cargar las imágenes en formato FITS directamente a MATS, el formato utilizado por OpenCV.
- **ProcessingProperties:** Es un objeto que contiene todas las propiedades que tendrá en cuenta la clase Collimator a la hora de realizar los cálculos.
- **ImageResults:** Se trata de un objeto que contiene los resultados de una colimación. Contiene tanto los aspectos técnicos (como sería el centro de brillo de la imagen) como la propia acción que debe realizar el usuario para colimar el telescopio.
- **CollimatorAction:** Es un objeto que contiene el paso de colimación que debe realizar el usuario. Contiene el tornillo que debe utilizar y la acción de apretar o aflojar dicho tornillo.
- **Telescope:** Se trata del objeto principal de la aplicación. Dicho objeto contiene las propiedades del telescopio, así como los tornillos y las propiedades de colimación de dicho telescopio.
- **Screw:** Objeto que representa un tornillo del telescopio. Contiene cálculos para obtener su posición dado un punto de referencia.
- **RemoteTelescope:** Objeto que contiene los datos necesarios para la conexión con el telescopio remoto.
- **INDIClient:** Se trata de una clase auxiliar para la implementación de un cliente con la librería INDI. Es necesario que esta clase extienda de la clase BaseClient de la librería INDI, y facilita los métodos abstractos que realiza el programa durante la conexión.

4.2.3. CollimatorDesktop

CollimatorDesktop corresponde con el programa de escritorio de la aplicación. Contiene tanto las vistas como los controladores y la persistencia. Este se divide a su vez en tres subproyectos. Uno contiene la persistencia, otro las vistas y otro la lógica de la visualización de la imagen, que se encarga de dibujar tanto la imagen recibida como los resultados de los cálculos de la colimación.

Para la persistencia, se realizó una clase DAO (Data Access Object) para cada objeto que se guardaba en base de datos. Cada uno de los objetos, tiene los métodos públicos get, para obtener un objeto con un parámetro id, getAll, para obtener todas las instancias de dicho objeto guardadas en base de datos, save, para guardar una instancia en base de datos, update, para actualizarlo y remove para borrarlo.

Las clases son las siguientes:

- **TelescopeDAO:** Esta clase se corresponde con el Data Access Object del objeto TelescopeUI. Desde la vista principal se consulta al método getAll para obtener todas las configuraciones de telescopios del sistema.

- **RemoteTelescopeDAO:** Esta clase se corresponde con el Data Access Object del objeto RemoteTelescope.
- **PaintingPropertiesDAO:** Esta clase se corresponde con el Data Access Object del objeto PaintingProperties.
- **ProcessingPropertiesDAO:** Esta clase se corresponde con el Data Access Object del objeto ProcessingProperties.
- **DatabaseConnectionSingleton:** Se trata de un singleton con la conexión de base de datos. Puesto que se utiliza el gestor SQLite, que únicamente permite una conexión simultáneamente con la base de datos, es necesario que se instancie una única vez dentro del programa, por ello se optó por el patrón singleton.

Lógica:

- **TelescopeUI:** Se trata de una clase que hereda del objeto Telescope de la librería, para añadirle el objeto PaintingProperties, necesario para el dibujado de los atributos del telescopio sobre la imagen.
- **CollimatorImage:** Se trata de una clase que hereda de QLabel, una clase de Qt que proporciona poder visualizar texto o imágenes en interfaces. Se hereda para realizar una mayor abstracción en la implementación a la hora de cargar imágenes o dibujar sobre estas y reducir la cantidad de código de los controladores. Interactúa directamente con la mayoría de clases de la librería.
- **PaintingProperties:** Es la clase que realiza el dibujado sobre una imagen y contiene las propiedades del mismo. Contiene tanto las propiedades del dibujo de los elementos del telescopio, como de los cálculos realizados.

Vistas:

- **TelescopeSelector:** Corresponde con el controlador de la primera vista al abrir la aplicación. Carga las configuraciones de todos los telescopios y permite tanto añadir como borrar las mismas.
- **TelescopeView:** Corresponde con el controlador de la vista principal de un telescopio. Contiene una instancia de CollimatorImage, con la que se cargan y visualizan en tiempo real las imágenes. Desde aquí, se controla el abrir las ventanas de configuración de las propiedades del procesado de imágenes, así como también las propiedades de conexión.
- **ConnectionSettings:** Corresponde con el controlador de la vista que configura la conexión con el servidor de imágenes remoto. Esta vista interactúa directamente con la librería para conectarse al servidor y permite elegir los dispositivos disponibles.
- **ProcessingProperties:** Corresponde con el controlador de la vista para configurar los parámetros del procesado de imágenes.
- **ScrewsDialog:** Corresponde con el controlador de la vista para configurar los tornillos que tiene un telescopio.

4.2.4. Base de datos

Puesto que se utiliza una base de datos relacional, se crearon las siguientes tablas con las siguientes columnas:

- **telescope:** Es la tabla que representa la configuración de un telescopio. Contiene las siguientes columnas:
 - **id:** Clave primaria.
 - **name:** Nombre del telescopio.
- **painting_properties:** Esta es la tabla que contiene la configuración que define que se pintará y cómo se pintará durante la colimación del telescopio sobre la imagen. Contiene las siguientes columnas:
 - **id:** Esta columna es a su vez clave primaria y clave ajena a id de telescope. Esto se hace así puesto todos los atributos de esta tabla pertenecen a un telescopio.
 - **zoom:** Esta columna representa el zoom que realiza el usuario sobre la imagen.
 - **automatic:** Esta columna declara si la colimación se hace en modo automático o manual.
 - **view_bounding_box:** Esta columna declara si se debe visualizar el cuadro delimitador en la colimación automática.
 - **view_screws:** Esta columna declara si se deben visualizar los tornillos sobre la imagen.
 - **view_calculated_center:** Esta columna declara si se debe visualizar el centro calculado en la colimación automática.
 - **scope_center_x:** Esta columna declara la posición sobre el eje x del centro de la mira en la colimación manual.
 - **scope_center_y:** Esta columna declara la posición sobre el eje y del centro de la mira en la colimación manual.
 - **scope_radius:** Esta columna declara el radio de la mira en la colimación manual.
 - **image_mirroring_vertically:** Esta columna declara si la imagen se debe reflejar verticalmente durante la colimación.
 - **image_mirroring_horizontally:** Esta columna declara si la imagen se debe reflejar horizontalmente durante la colimación.
 - **screws_mirroring_vertically:** Esta columna declara si los tornillos se deben reflejar verticalmente durante la colimación.
 - **screws_mirroring_horizontally:** Esta columna declara si los tornillos se deben reflejar horizontalmente durante la colimación.
 - **tight_mirroring:** Esta columna declara si la acción de apretar y aflojar tornillos debe ser revertida.
 - **screw_radius:** Esta columna declara el radio que deben tener los tornillos desde el centro de la mira del telescopio.
- **processing_properties:** Esta tabla contiene todos aquellos datos necesarios para hacer los cálculos de la colimación automática sobre la imagen.
 - **id:** Esta columna es a su vez clave primaria y clave ajena a id de telescope. Esto se hace así puesto todos los atributos de esta tabla pertenecen a un telescopio.

- **scale:** Esta columna declara la escala que se realiza a la imagen antes de realizar los cálculos.
 - **refinate_radius:** Esta columna declara el radio máximo con el que se recortará la imagen una vez encontrado el centro.
 - **stretching_max:** Esta columna declara el valor máximo del estiramiento del histograma de la imagen.
 - **stretching_min:** Esta columna declara el valor mínimo del estiramiento del histograma de la imagen.
 - **threshold_size:** Esta columna declara el tamaño del umbral adaptativo que se realiza sobre la imagen.
 - **threshold_constant:** Esta columna declara la constante del umbral adaptativo que se realiza sobre la imagen.
- **remote_telescope:** Esta tabla contiene los datos necesarios para conectarse a un dispositivo en un servidor INDI
 - **id:** Esta columna es a su vez clave primaria y clave ajena a id de telescope. Esto se hace así puesto que todos los atributos de esta tabla pertenecen a un telescopio.
 - **host:** Esta columna declara el nombre o ip del servidor al cual debe conectarse para recibir imágenes.
 - **port:** Esta columna declara el puerto del servidor.
 - **name:** Esta columna declara el nombre del dispositivo al cual debe conectarse dentro del servidor.
 - **exposure:** Esta columna declara el tiempo de exposición al que deben someterse las imágenes. Este dato se envía directamente al servidor, y cuanto menor sea, más rápido es el flujo de imágenes.
 - **screw:** Representa cada uno de los tornillos que contiene un telescopio.
 - **id:** Clave primaria.
 - **telescope:** Clave ajena a id de telescope.
 - **degrees:** Esta columna representa los grados que se utilizan para calcular la posición del tornillo mediante razones trigonométricas.

4.3 Tecnología utilizada

Una de las primeras decisiones que se ha de tomar a la hora de desarrollar cualquier software, es en qué entorno de desarrollo se considera mas óptimo para realizar el proyecto.

Nosotros hemos desarrollado un cliente que podría estar sobre cualquier plataforma. Hemos elegido Linux porque nos ha interesado y es una plataforma muy usada entre los aficionados. Además, migrarlo a OSX sería relativamente fácil en un futuro. El servidor INDI, el que contiene los drivers y se conecta con los dispositivos, sí que está en Linux o OSX (muchas veces sobre una Raspberry. Por ejemplo Stella Mate). Nuestro cliente tomará "prestadas" las cámaras que el servidor ofrezca, simplificando así su uso para tu herramienta.

4.3.1. Lenguaje de programación

Las siguientes decisiones a tomar son el entorno de desarrollo y el lenguaje de programación. Observadas las limitaciones, se barajaron las siguientes opciones:

- **Python:** Python es un lenguaje de programación interpretado de alto nivel.

Python tiene como ventaja principal su facilidad, tanto en su sintaxis, como a la hora de descargar, enlazar y utilizar las librerías. Además, debido a su reciente crecimiento en el mercado, contiene una gran cantidad de documentación que facilita su uso.

Como desventaja, encontramos que al ser un lenguaje interpretado, es más lento que otras opciones propuestas.

- **C:** Se trata de un lenguaje de programación de propósito general desarrollado entre 1969 y 1972.

Las grandes ventajas de C son su rapidez y la libertad que ofrece al programador a la hora de gestionar los recursos.

En contraposición, el enlace con las librerías puede resultar tedioso y es complejo a la hora de desarrollar programas de media o gran magnitud y con interfaz gráfica.

- **C++:** Es un lenguaje de programación de propósito general creado en 1979 como subconjunto de C. Introduce conceptos como la programación orientada a objetos.

Al ser una extensión de C, la rapidez de este es muy similar. Además, al introducir la programación orientada a objetos, nos facilita bastante la creación de proyectos grandes, contando además con muchos frameworks que facilitan la programación de interfaces y el enlazado con librerías.

Debido a la valoración realizada de los lenguajes de programación, se consideró que C++ era el lenguaje ideal para el desarrollo del proyecto.

4.3.2. Qt

Se decidió utilizar Qt Creator como IDE, puesto que integra el desarrollo con Qt¹, un framework multiplataforma orientado a objetos que facilita el desarrollo de programas que utilicen interfaz gráfica, así como múltiples herramientas como QT Designer para el diseño de interfaces gráficas, librerías para persistencia en base de datos y ayudas para la programación en C++.

¹<https://www.qt.io/>

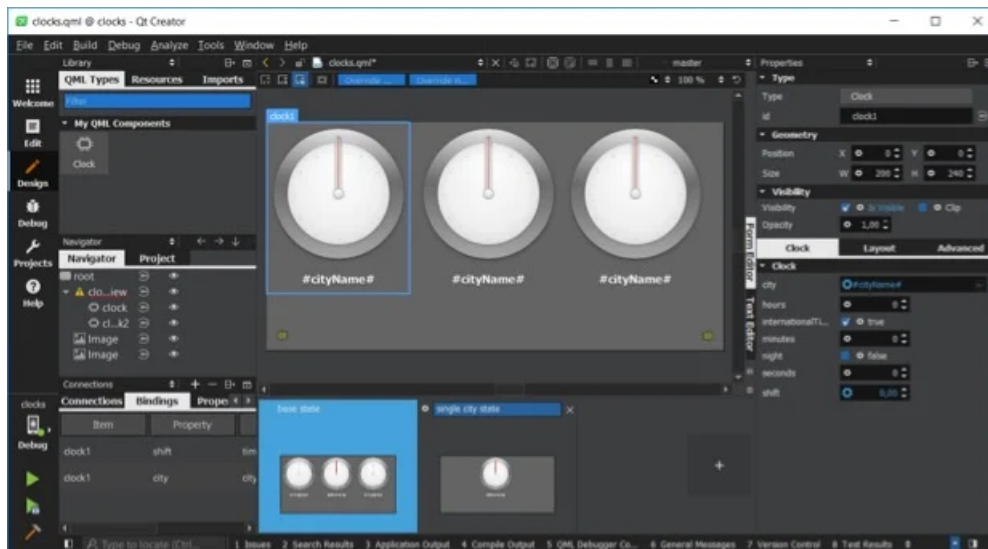


Figura 4.3: Qt designer (Imagen extraída de [11])

4.3.3. SQLite

Para lograr la persistencia, se utilizará SQLite [12] para guardar las configuraciones del usuario en un archivo en el ordenador, que implementa un motor de bases de datos relacional SQL rápido y altamente fiable.

Dado que no se requiere un sistema distribuido, ni consultar los datos desde diferentes ordenadores y la base de datos no tendrá gran tamaño, se pensó que la mejor forma de lograr la persistencia era utilizando SQLite, puesto que permite guardar una base de datos en un archivo. Dicho archivo se guarda en la carpeta Documentos del usuario que utiliza la aplicación.

4.3.4. Docker

Puesto que se utiliza linux para el desarrollo de la aplicación, se decidió compilar el proyecto para Ubuntu. Para hacer la compilación y generación de paquetes .deb de forma automática, se utilizó la tecnología Docker[13].

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

Para realizar el despliegue, se creó un contenedor en el cual se inicializa una imagen con base Ubuntu y se instalan todas las dependencias necesarias para la compilación del proyecto. Además, crea un script que compila y genera el .deb de la aplicación.

Una vez creado el contenedor, se puede ejecutar dicho script, que monta las carpetas con el código fuente del proyecto, lo compila y genera el .deb listo para instalar en versiones de Ubuntu mayores o iguales a 20.04, y en distribuciones basadas en esta, debido a las restricciones de las librerías dependientes.

Esto facilita poder editar el código de la aplicación e inmediatamente generar el instalador, sin necesidad de introducir los comandos manualmente. Además, permite generar el instalable para Ubuntu desde cualquier Sistema Operativo, sin necesidad de tener que instalar Ubuntu en el ordenador ni en una máquina virtual. De hecho, la aplicación se desarrolló en Arch Linux, mientras que se desplegó para distribuciones Ubuntu y derivadas.

4.3.5. Linux

Se decidió hacer el programa compatible con Ubuntu, puesto que de las distribuciones Linux para escritorios, es de las más utilizadas actualmente.

El programa se desarrolló utilizando Arch Linux, así que compilando e instalando las librerías OpenCV 4, INDI 1.8 y CFITSIO desde el código fuente, puede utilizarse en cualquier sistema operativo.

4.3.6. Subversion (SVN)

Subversion [14] es un sistema de control de versiones centralizado de código abierto desarrollado conjuntamente entre la comunidad y desarrolladores de CollabNet, Elego, VisualSVN y WANDisco. Fue lanzado inicialmente el 20 de Octubre de 2000 y actualmente se encuentra bajo la licencia Apache.

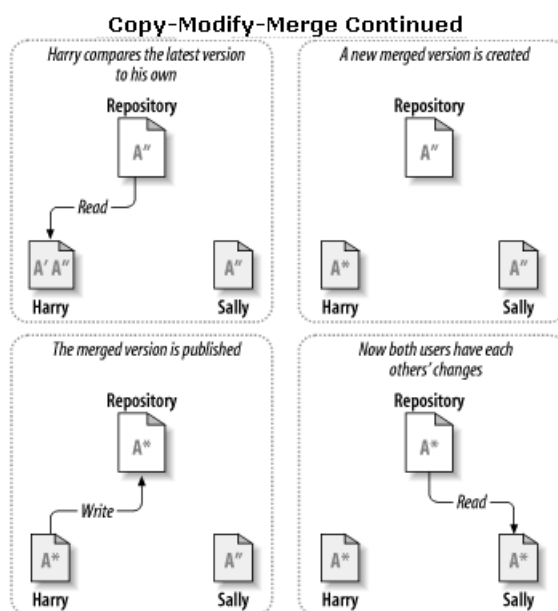


Figura 4.4: Funcionamiento de Subversion (Imagen extraída de [15])

Esta herramienta nos permite administrar las versiones de un sistema de ficheros, ya sea tanto código fuente, como documentación y cualquier tipo de archivos. Con Subversion podemos ver cambios entre versiones y retroceder en el histórico de estas.

4.3.7. Doxygen

Doxygen[16] es un generador de documentación para C++, C, Java y otros lenguajes. Funciona en la mayoría de sistemas Unix y en Windows y Mac OS X.

Esta es la herramienta que se ha utilizado para generar la documentación del proyecto. A la hora de programar, se introducían los comentarios en todos los métodos y clases, en los archivos .h de cada clase. Doxygen permite generar y modificar un doxyfile para la generación de la documentación de forma personalizada, y se modificó para generar una web en html para la documentación del proyecto.

4.3.8. OpenMP

OpenMP[17] (Open Multi-Processing) es una interfaz de programación de aplicaciones (API) que admite la programación multiprocesamiento de memoria compartida multiplataforma en C, C++ y Fortran. Lo conforma un conjunto de directivas del compilador, rutinas de biblioteca y variables de entorno que influyen en el comportamiento en tiempo de ejecución.

Esta es la herramienta que se ha utilizado para la paralelización de los cálculos de la colimación automática.

CAPÍTULO 5

Desarrollo de la solución propuesta

En este capítulo se va a mostrar el proyecto desarrollado, con el fin de proporcionar una solución al problema expuesto en los capítulos anteriores, explicando todas las etapas por las que ha pasado el desarrollo del proyecto.

5.1 Procesado de imágenes

Como parte esencial de este proyecto, en primer lugar encontramos el procesado de las imágenes que son introducidas, ya sea por parte del usuario o recibidas del servidor INDI. Este procesado de imágenes debe dar respuesta al problema principal de la aplicación. Para ello, se ha implementado la clase *Collimator*, cuyo funcionamiento se explica a continuación.

Aquí es donde participan principalmente *CFitsio* y *OpenCV*. *CFitsio* es una librería que nos permite cargar las imágenes, puesto que estas se suelen recibir en el tipo de imagen *.fits*. Así, cuando un usuario carga una imagen en formato *fits*, el programa llama a la función estática *loadFitsImage* que se encarga de leer la imagen y transformarla al formato *Mat*, el formato utilizado por la librería *OpenCV*.

Una vez la imagen se ha cargado en un formato reconocible por *OpenCV*, se ha implementado la función *processImage*, que realiza distintas transformaciones sobre la misma para devolver un objeto que contiene el resultado de la colimación. Las transformaciones, paso por paso, son las siguientes:

1. **Escalado:** La imagen se escala para hacerla más grande o, en su defecto, más pequeña.
2. **Generación del binario:** Una imagen binaria es una imagen que contiene únicamente dos valores posibles para cada píxel. En este caso, los colores utilizados son el blanco y el negro. Para ello, realizamos sobre la imagen:
 - a) **Normalización:** La normalización en el procesamiento de imágenes consiste en cambiar el rango de valores de intensidad de píxeles. En el caso que nos ocupa, realizamos la normalización para hacer que aquellos píxeles que se encuentren por encima del máximo sean completamente blancos, en el caso contrario de aquellos que se encuentren por debajo del mínimo sean negros.
Por último, aquellos píxeles que se encuentren dentro del rango mínimo-máximo, se normalizan de acuerdo con la fórmula de la normalización lineal de una

imagen digital en escala de grises:

$$pixel = \frac{pixel - v_{min}}{v_{max} - v_{min}} \cdot 255$$

- b) **Método del valor umbral adaptativo:** Es el método que crea el binario. Lo que realiza este proceso es reemplazar cada píxel por blanco o negro en base a las características locales del entorno en que se evalúa cada píxel.
3. **Obtención del centro de masa y cuadro delimitador:** Para la obtención del cuadro delimitador que rodea los anillos de difracción, y el centro de masa que depende de los blancos, se realizan los siguientes pasos:
- a) **Cálculo del centro de masa:** Se calcula el centro de masa a partir de la escala de grises; cuando es blanco se suma el valor de la posición para obtener la media.
 - b) **Cálculo del cuadro delimitador:** Se calcula el cuadro delimitador obteniendo los blancos que se encuentran en las esquinas de la imagen dado el binario, el centro calculado en el paso anterior y un radio límite introducido por el usuario.
 - c) **Cálculo del centro de masa:** Se calcula nuevamente el centro de masa, pero esta vez únicamente dentro del cuadro delimitador. Este cálculo nos da como resultado el centro de masa de los anillos de difracción, que nos permite más adelante, junto al centro del cuadro delimitador, calcular la distancia relativa para obtener cómo debe realizarse la colimación.

Este proceso nos da como resultado el centro de masa y el cuadro delimitador de los anillos de difracción. En el proyecto desarrollado, estos se pueden visualizar en rojo el cuadro delimitador y en verde el centro de masa, como se ve en la figura 5.1.

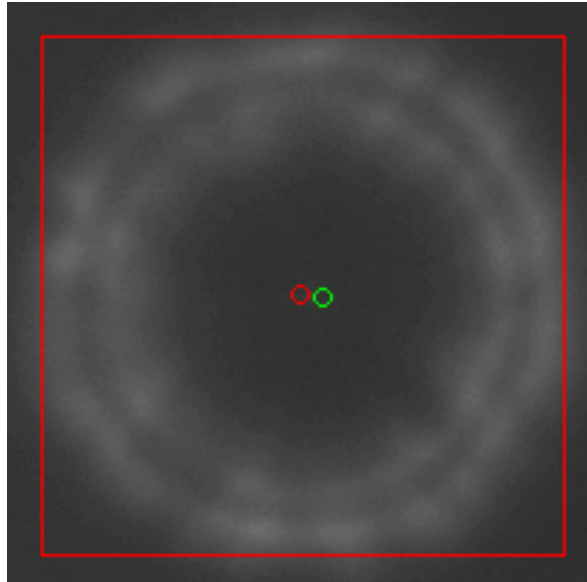


Figura 5.1: Centro de masa y cuadro delimitador (Elaboración propia)

Acto seguido el sistema calcula, dados estos resultados, cuál es el tornillo que debe apretar o aflojar. Para ello, realiza los siguientes pasos:

- **Cálculo de rectas desde cada tornillo hasta el centro del cuadro delimitador:** Se calcula la ecuación de las rectas con los puntos de cada uno de los tornillo junto con el centro del cuadro delimitador.
- **Cálculo de la recta perpendicular entre las rectas calculadas anteriormente y el centro de masa:** Se calculan las rectas perpendiculares de las rectas calculadas anteriormente para obtener la distancia entre el centro de masa y dichas rectas. La recta con menor distancia, será la recta cuyo tornillo se aflojará o atornillará.
- **Cálculo de acción con el tornillo:** Si la distancia entre el tornillo y el centro de masa es mayor que la distancia entre el tornillo y el centro del cuadro delimitador, habrá que aflojar el tornillo. En caso contrario, habrá que apretarlo. Dicho comportamiento se puede invertir desde las opciones del programa.

El proceso explicado se ve con más claridad en la figura 5.2, con las líneas calculadas en color amarillo.

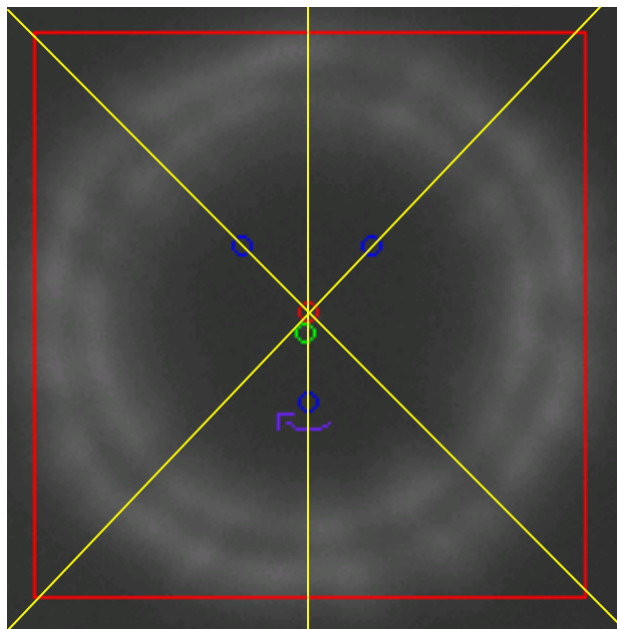


Figura 5.2: Rectas calculadas para la colimación (Elaboración propia)

A continuación, se realizan cálculos para mostrar los resultados de la imagen al usuario. Dichos cálculos son necesarios para mostrar los tornillos configurados por el usuario sobre la imagen y la flecha que indica en qué dirección debe girar el tornillo. Estos se configuran mediante su ángulo y radio sobre la circunferencia goniométrica [18] para realizar los cálculos.

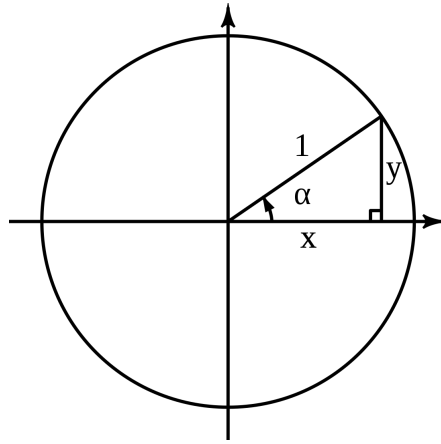


Figura 5.3: Circunferencia Giniométrica (Image extraída de [18])

Para el cálculo de la posición del tornillo simplemente se utilizan las razones trigonométricas, donde P_x y P_y definen el centro previamente calculado, α define el ángulo que el usuario configura del tornillo, y X e Y definen la posición del tornillo sobre la imagen:

$$X = C_x + \cos(\alpha) \cdot R$$

$$Y = C_y + \sin(\alpha) \cdot R$$

Con esto, se obtiene la posición del tornillo, y mediante esta misma fórmula y rotaciones aplicadas sobre el tornillo, se dibuja la flecha que indica en qué dirección el usuario debe girar el tornillo.

Todos estos cálculos se invierten si el usuario utiliza la opción de reflejar la imagen o la posición de los tornillos.

En conjunto, todo lo calculado resulta en una imagen como la de la figura 5.4

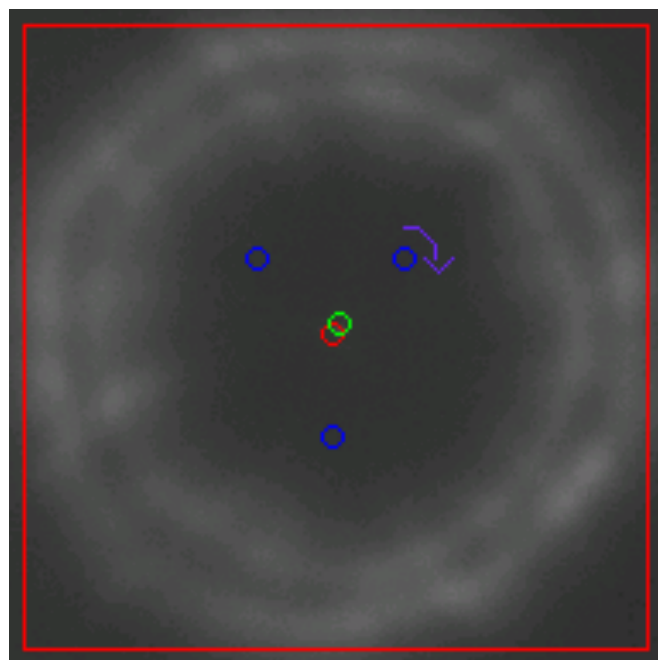


Figura 5.4: Resultado de los cálculos de la colimación (Elaboración propia)

5.1.1. Optimización de los cálculos

Los cálculos que se realizan para la obtención del resultado de la colimación se han paralelizado para obtener mejores tiempos, como se muestra en la figura 5.5.

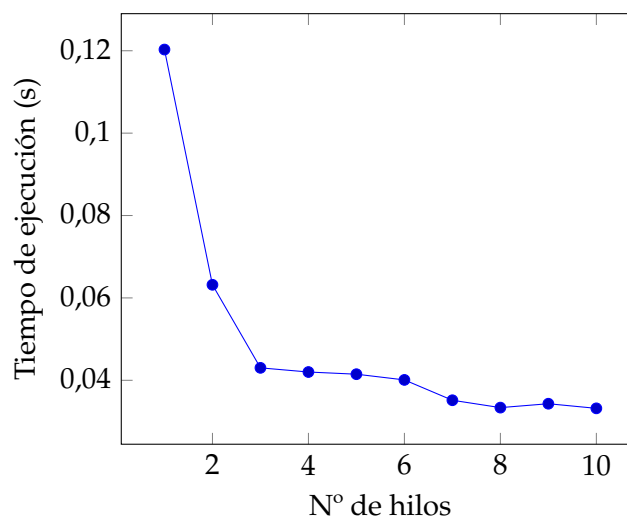


Figura 5.5: Gráfica de los tiempos de ejecución de la normalización (Elaboración propia)

Esta paralelización se llevó a cabo mediante la librería OpenMP, paralelizando los bucles for de las funciones que realizan los cálculos sobre las imágenes.

```

1 #pragma omp parallel for private(i) reduction(+:ncx,ncy,count)
2 for(i = starty; i < heigth * width; i++) {
3     uchar* gray = &(img->ptr<uchar>(i / width))[i % width];
4     if(*gray == 255) {
5         ncx += i % width;
6         ncy += i / width;
7         count++;
8     }
9 }

```

OpenMP distribuye automáticamente cada iteración del bucle entre los hilos disponibles del procesador. Como se observa, se utiliza *private(i)* para que la variable *i* sea privada para cada hilo de ejecución, mientras que se utiliza *reduction* para aquellas variables sobre las que se debe realizar una operación aritmética, en este caso sumar las variables *ncx*, *ncy* y *count*. El resultado de estas variables al final del bucle será la suma de los resultados de cada hilo de ejecución.

5.2 Persistencia

Para la persistencia, se ha utilizado SQLite. SQLite nos permite guardar una base de datos relacional en un archivo, por lo tanto, la base de datos que contiene todas las configuraciones son almacenadas en el ordenador del usuario.

La primera vez que el usuario arranca el programa, se comprueba si el archivo de persistencia existe en el ordenador del usuario. Dicho archivo se guarda dentro de la carpeta Documentos del usuario que utiliza el programa. Una vez se crea el archivo, se crean también todas las tablas explicadas anteriormente.

Todo esto, se realiza mediante una clase Singleton, puesto que SQLite únicamente permite una conexión simultánea con la base de datos. La clase `DatabaseConnectionSin`

gleton es la encargada de esto. Abre el archivo y si no existe, crea el archivo junto con sus tablas empleando consultas SQL.

Una vez establecida la conexión con la base de datos, son las clases DAO las encargadas de realizar consultas sobre la base de datos.

Cada clase DAO contiene los siguientes métodos:

- **get:** Este método requiere un entero como argumento que representa el id del objeto a obtener en dicho DAO. Realiza un SELECT en base de datos y construye y devuelve el objeto.
- **getAll:** Este método devuelve una lista que contiene todas las instancias del objeto del DAO que existen en la base de datos.
- **update:** Este método requiere un objeto del DAO como argumento. Realiza un UPDATE en base de datos para actualizar el objeto tal y como se pasa por parámetro.
- **remove:** Este método requiere un entero como argumento que representa el id del objeto a borrar. Realiza un DELETE en la base de datos.
- **save:** Este método requiere un objeto del DAO como argumento. Realiza un INSERT del objeto en la base de datos.

También cabe explicar que para realizar la persistencia, el programa carga todos los objetos Telescope junto con los objetos de los que depende al inicio del programa. Para ello, se llama al método getAll de TelescopeDAO. Dicho método realiza la siguiente consulta:

```
1 SELECT *, t.id AS telescopeid, sc.id AS screwid, t.name AS telescopename ,  
2     rt.name AS remotetelescopename  
3 FROM telescope t  
4 LEFT JOIN processing_properties processingProperties  
5     ON processingProperties.id = t.id  
6 LEFT JOIN painting_properties paintingProperties  
7     ON paintingProperties.id = t.id  
8 LEFT JOIN screw sc ON sc.telescope = t.id  
9 LEFT JOIN remote_telescope rt ON rt.id = t.id  
10 ORDER BY telescopeid
```

Este comportamiento es así, puesto que el número de telescopios que tendrá un usuario será reducido. De esta manera es más óptimo realizar una única consulta al principio que realizar muchas conexiones y consultas conforme se utiliza el programa, dado que tiene un coste lineal que depende del número de telescopios guardados.

Además, las operaciones de guardado (save y update) se realizan cuando el usuario lleva a cabo un cambio de configuración en el DAO de la configuración correspondiente que se modifica.

5.3 Vistas

Para la realización de las vistas de la aplicación, se han realizado todas dentro de la carpeta views, donde cada vista tiene su archivo .ui que contiene un archivo en formato xml que define las vistas. Cada archivo .ui es cargado por su clase de C++, teniendo un archivo .cpp y su correspondiente .h.

Las vistas se realizaron utilizando la herramienta Qt Designer, que permite crear interfaces gráficas arrastrando los componentes a utilizar. Un ejemplo sería la interfaz para configurar el servidor remoto:

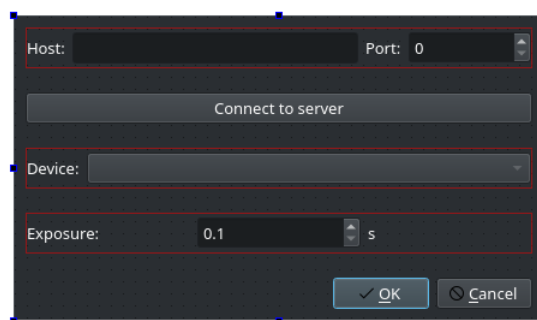


Figura 5.6: Diálogo de conexión con servidor remoto en Qt Designer (Elaboración propia)

Los eventos que ocurren dentro de las vistas, como los botones, son controlados dentro de la clase mediante los conectores de Qt, conectándolos con SLOTS si se trata de una función compleja o con una función lambda si es sencilla:

```

1 connect(zoomInBtn, &QPushButton::clicked,
2         this, [this] { image->zoom(0.5); });
3 connect(zoomOutBtn, &QPushButton::clicked,
4         this, [this] { image->zoom(-0.5); });
5
6 connect(ui->loadImageButton, SIGNAL(clicked()),
7         this, SLOT(openImageFromFile()));
8 connect(ui->actionImage_processing_settings, SIGNAL(triggered()),
9         this, SLOT(openImageProcessing()));
10 connect(ui->actionSettingsScrews, SIGNAL(triggered()),
11          this, SLOT(screwsSettings()));
12 connect(ui->actionSave_screenshot, SIGNAL(triggered()),
13          this, SLOT(saveScreenshot()));
14 connect(ui->actionClose_2, &QAction::triggered,
15          this, &QWidget::close);
16 connect(ui->actionConnection, &QAction::triggered,
17          this, &TelescopeView::connectionSettings);

```

Además, se ha creado una clase que extiende de QLabel, *collimatorimage*, que se inserta dentro de la vista en la cual se visualiza la colimación, en la vista de telescopio. Esta clase se encarga de manejar la imagen, añadiendo abstracción y teniendo dos métodos principales: *loadImage*, que dada una localización de una imagen, la muestra en pantalla con los ajustes de colimación cargados, y *connectTelescope* que se conecta a un servidor INDI para cargar las imágenes recibidas. Además, esta clase permite tomar capturas de la imagen actual, guardando su contenido en un archivo.

Las vistas que se han creado son las siguientes:

- **Listado de telescopios:** Esta es la vista que se abre al iniciar la aplicación. Contiene un listado de cada una de las configuraciones de telescopios que el usuario ha creado.
- **Vista de telescopio:** Esta es la vista principal de la aplicación. En esta vista se puede ver la colimación y contiene apartados y enlaces a vistas de configuración. Desde esta vista se ven las imágenes recibidas por el servidor o las cargadas por el usuario.
- **Vista de la configuración de procesamiento de imágenes:** Esta vista permite configurar las opciones del procesamiento de imágenes durante la colimación automática.
- **Vista de la configuración de tornillos:** Esta vista permite configurar los tornillos de la configuración de un telescopio.

- **Vista de la configuración de conexión:** Esta vista permite configurar la conexión con el servidor INDI.

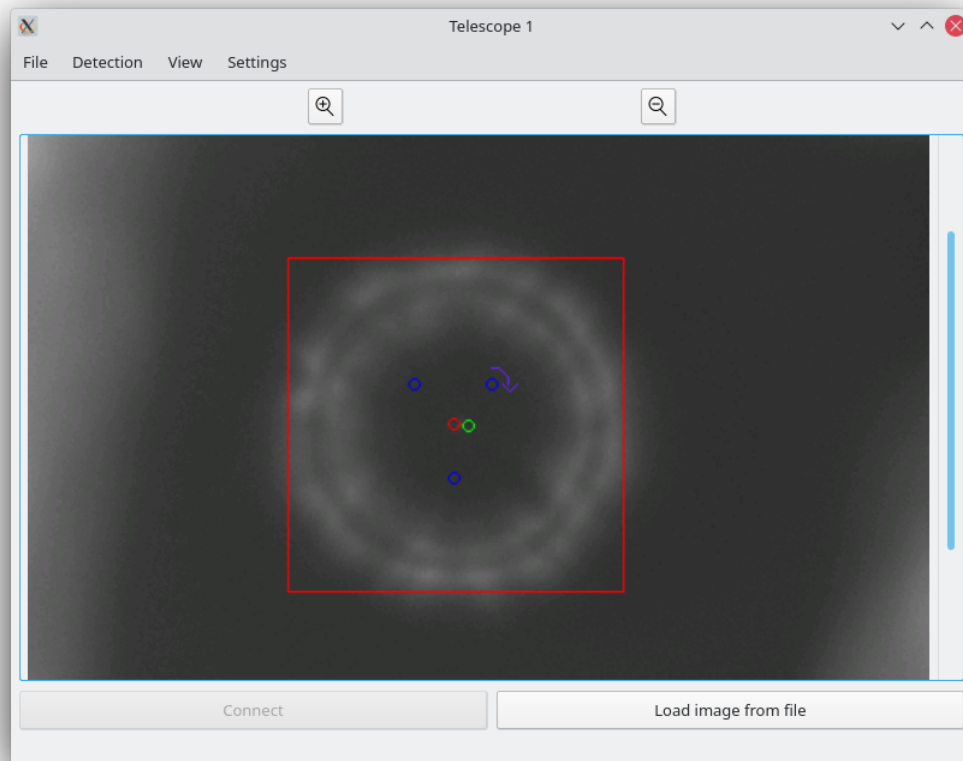


Figura 5.7: Vista de telescopio (Elaboración propia)

Como se observa en la figura 5.7, se ha realizado la interfaz utilizando Qt Designer, mientras que la imagen central es la clase `CollimatorImage`, que extiende de `QLabel` y se inserta dentro de dicha vista, siendo un componente utilizable.

En la figura 5.8 podemos observar la navegación entre vistas.

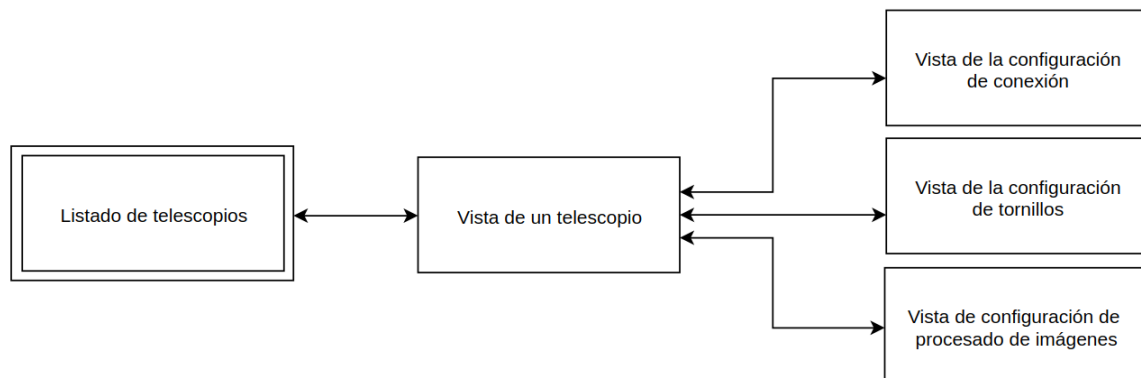


Figura 5.8: Diagrama de navegación (Elaboración propia)

Al abrir la aplicación vemos la lista de telescopios, desde la cual podemos acceder a la vista de un telescopio. Una vez dentro de la vista de un telescopio podemos realizar la

colimación y configurar los parámetros necesarios para llevarla a cabo navegando entre los menús que nos llevan a las vistas de configuración.

5.4 Conexión con el servidor INDI

El programa utiliza la librería INDI para conectarse con los servidores INDI. Estos pueden contener cámaras o telescopios compatibles.

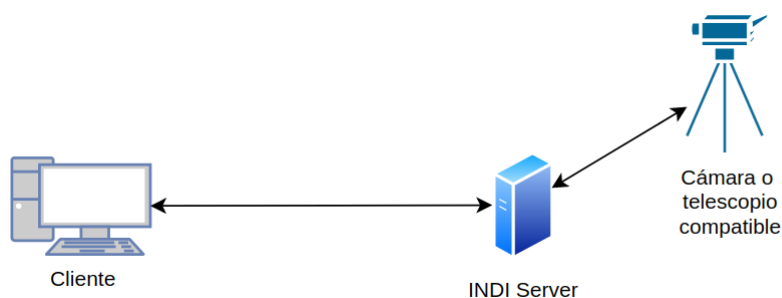


Figura 5.9: Conexión con servidor INDI (Elaboración propia)

Los servidores INDI son muy empleados tanto por astrónomos profesionales como aficionados, ya que son fáciles de instalar en cualquier ordenador, incluso en ordenadores portátiles o Raspberries PI.

Estos servidores permiten una conexión fácil y rápida con telescopios y cámaras compatibles, ya que disponen de drivers propios en el servidor, y no es necesario que el usuario se preocupe por instalar drivers para sus cámaras. Esto abre ampliamente el campo de utilidad del proyecto, ya que consigue el principio de fácil uso que este persigue.

Esta funcionalidad al ser compleja, requirió de un sprint completo para completarse.

Para la conexión con el servidor INDI, se han implementado dos clases, `INDIClient`, que se encarga de conectarse con el servidor, y `RemoteTelescope`, que representa un dispositivo en el servidor INDI.

Para la implementación de la clase `INDIClient`, es necesario que la clase extienda de `INDI::BaseClient`, y que implemente las funciones necesarias. Además, como también utiliza las señales que implementa Qt, esta clase hace doble herencia, heredando también de `QObject`.

Para la implementación del programa, fue necesario sobrescribir los métodos de `INDI::BaseClient` `newProperty` y `newBlob`.

5.4.1. Obtención de dispositivos

Para la obtención de dispositivos, se sobrescribe el método `newProperty`. Este método es llamado cada vez que se detecta una nueva propiedad en el servidor. En el programa, se utiliza para detectar los dispositivos del servidor, y guarda y notifica únicamente de aquellos que dispongan de cámara:

```

1 void newProperty(INDI::Property* property) override {
2     if (property->getName() == "CCD_EXPOSURE") {
3         INDI::BaseDevice* device = property->getBaseDevice();
4         devices.insert(device->getDeviceName(), device);
5         emit deviceDetected(RemoteTelescope(device, this));
6     }
7 }

```

Como se puede observar, cuando recibe una propiedad "CCD_EXPOSURE", el sistema sabe que el dispositivo con dicha propiedad es capaz de realizar fotografías, por ello, lo guarda en memoria.

5.4.2. Obtención de imágenes

Para la obtención de imágenes, se sobrescribe el método newBlob. Este método es llamado cada vez que se recibe una imagen del servidor. El sistema guarda la imagen temporalmente y emite una señal para notificarlo. Esta señal se utiliza posteriormente por la aplicación para leer la imagen y mostrarla al usuario.

Para la obtención repetida de imágenes, desde la clase CollimatorImage de CollimatorDesktop, se pide una imagen una vez, y a continuación, cada vez que se recibe una imagen, se pide la siguiente llamando al método getImageFromTelescope de la clase INDIClient de la librería, hasta que el usuario desee detener la colimación.

La velocidad con la que las imágenes se muestren dependerá directamente de la exposición que se configure en las propiedades de conexión, puesto que cuanto mayor sea la exposición, mayor será el tiempo que tarde la cámara INDI en tomar la fotografía.

El flujo de conexión con el servidor INDI para la obtención de imágenes se ve reflejado en el diagrama D.3.

5.5 Documentación

Para la documentación, se ha utilizado la herramienta Doxygen, un generador de documentación para C++, C, Java y otros lenguajes.

En el código del programa, están documentados cada uno de los métodos del programa en los archivos .h de la siguiente forma:

```

1 /**
2  * @brief Method for obtaining a PaintingProperties by its id
3  * @param id of the Painting Properties to be obtained
4  * @return PaintingProperties matching the id
5  */
6 PaintingProperties get(int id);

```

Utilizando @brief, se especifica un resumen de la funcionalidad del método, utilizando @param para cada uno de los parámetros, se hace una descripción del parámetro, y utilizando @return, se explica aquello que devuelve el método.

Se ha utilizado el mismo método, pero únicamente con @brief para la explicación de cada clase.

Ejecutando Doxygen dentro de la carpeta del proyecto, genera una carpeta doc, donde dentro podemos encontrar la documentación tanto en LaTeX como en HTML, siendo HTML el caso de interés.

Al incluir un archivo README.md dentro del proyecto y configurarlo en el doxyfile, Doxygen genera la página principal de acuerdo con el archivo README como se muestra en la figura B.1

Dentro de este archivo README, se ha incluido una breve descripción del programa y las instrucciones de instalación y compilación.

CAPÍTULO 6

Implantación

Para la implantación del programa, se ha utilizado la herramienta Docker[13]. Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

6.1 Generación del instalable

El planteamiento que se le ha dado es crear una imagen de docker capaz de compilar el programa, y de esta forma, generar para el sistema operativo de la imagen, un instalador.

Para ello, se ha utilizado la imagen base de Ubuntu. Ubuntu dispone de todas las librerías y herramientas necesarias a través de los repositorios oficiales, por lo tanto basta con instalarlo todo a través del gestor de paquetes *apt*.

Una vez se instalan todos los paquetes necesarios, se crea un script para que, al iniciar la imagen del docker, se compile el proyecto y se genere el archivo .deb correspondiente.

Para hacer todo esto, se generó el siguiente *Dockerfile*:

```
1 FROM ubuntu:latest
2
3 #Creation of required folders
4 RUN mkdir -p /home/collimator/collimator/CollimatorDesktop \
5     && mkdir -p /root/Documents
6 #Dependencies
7 RUN apt-get update && DEBIAN_FRONTEND="noninteractive" apt-get install -y cmake
8     g++ build-essential qt5-default libindi-dev libopencv-dev libcfitsio-dev
9     zlib1g-dev
10 #start script
11 RUN echo "#!/bin/sh" >> /home/collimator/collimator.sh \
12     && echo "cd /home/collimator/collimator/CollimatorDesktop/CollimatorLib
13     && qmake CollimatorLib.pro && make -j$(nproc) && make install &&
14     cd .. && qmake CollimatorDesktop.pro && make -j$(nproc)" >> /home/
15     collimator/collimator.sh \
16     && echo "cd /root/Documents/deb && cp /home/collimator/collimator/
17     CollimatorDesktop/CollimatorDesktop CollimatorDesktop/bin/
18     CollimatorDesktop && cp /home/collimator/collimator/
19     CollimatorDesktop/CollimatorLib/libCollimatorLib.so
20     CollimatorDesktop/usr/lib/libCollimatorLib.so.1 && dpkg-deb --build
21     --root-owner-group CollimatorDesktop" >> /home/collimator/
22     collimator.sh \
23     && chmod +x /home/collimator/collimator.sh
```

Como se puede observar, se utiliza la instrucción *nproc* a la hora de compilar el proyecto para utilizar el máximo número de hilos posibles y realizar la compilación en el menor tiempo posible.

Al terminar la compilación del proyecto, el script se mueve a la ruta */root/Documents/deb*, puesto que esta ruta se utiliza como carpeta de enlace entre el ordenador desde el que se ejecuta la imagen de Docker y la propia imagen. Dentro de esta carpeta, genera el *.deb* con el comando *dpkg-deb*.

La ruta desde donde se ejecuta *dpkg-deb* contiene la siguiente estructura de ficheros:

```
CollimatorDesktop
├── DEBIAN
│   ├── control
│   ├── bin
│   │   └── CollimatorDesktop
│   └── usr
│       ├── lib
│       │   └── libCollimatorLib.so.1
│       ├── share
│       │   └── applications
│       │       └── collimator.desktop
```

Figura 6.1: Estructura de ficheros para *dpkg-deb* (Elaboración propia)

La misma estructura de carpetas será copiada cuando se ejecute el comando *apt install* con el instalador *.deb*, a excepción de la carpeta *DEBIAN* que contiene el archivo de control.

El archivo *control* es un archivo que contiene la información necesaria para la instalación del programa.

```
1 Package: CollimatorDesktop
2 Architecture: all
3 Maintainer: Nicolas Garcia Sastre
4 Priority: optional
5 Version: 1.0
6 Description: Program for easy telescope collimation
7 Depends: zlib1g, libopencv-dev, libqt5widgets5, libqt5gui5, libqt5sql5,
    libqt5core5a, libc6, libstdc++6, libgcc-s1, libindi-dev, libcfitsio-dev
```

El archivo contiene el nombre del paquete en *Package*, las arquitecturas soportadas en *Architecture*, el mantenedor de la aplicación, la prioridad, la versión y la descripción.

Además, es necesario incluir las dependencias, puesto que como se ha mencionado anteriormente, es necesario tener instalado Qt, CFITSIO, OpenCV4 e INDI. Todos estos paquetes se encuentran dentro de los repositorios oficiales de Ubuntu, así que se instalarán automáticamente al instalar la aplicación.

Para lanzar el docker que genera el instalador, se utiliza la siguiente orden desde la carpeta del proyecto:

```
1 docker run \
2 --mount type=bind,source="$(pwd)/CollimatorDesktop",target="/home/collimator/
   collimator/CollimatorDesktop" \
3 --mount type=bind,source="$(pwd)/shared",target="/root/Documents" \
4 -it collimator /home/collimator/collimator.sh
```


Esta orden monta tanto la carpeta que contiene el proyecto como la carpeta shared dentro de la ruta mencionada anteriormente para generar el instalador, y finalmente lanza el script.

La misma imagen de docker fue, además, utilizada durante el desarrollo para probar el programa, modificando el script para que en vez de generar el instalador, ejecute el programa una vez compilado.

6.2 Instalación

Para la instalación, desde una máquina con Ubuntu, hay que ejecutar el siguiente comando desde la carpeta que contiene el archivo generado CollimatorDesktop.deb:

```
1 sudo apt-get install ./CollimatorDesktop.deb
```

Dicho comando utiliza apt para instalar el programa, que generará los siguientes archivos en las siguientes rutas:

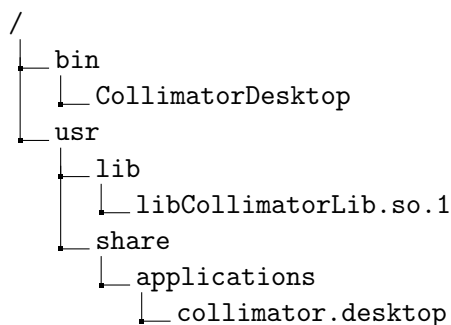


Figura 6.2: Estructura de ficheros tras la instalación (Elaboración propia)

- **CollimatorDesktop:** Es el ejecutable de la aplicación. Al encontrarse dentro de la carpeta /bin el usuario podrá ejecutarla desde cualquier directorio con la línea de comandos.
- **libCollimatorLib.so.1:** Es la librería de la aplicación. Al ser una librería compartida, debe instalarse en este directorio. De esta forma, sus funciones pueden ser utilizada por otras aplicaciones en un futuro.
- **collimator.desktop:** Este archivo contiene la definición de la aplicación para ser encontrado por Ubuntu. Al incluir este archivo en la carpeta /usr/share/applications, será reconocida por un programa de escritorio en Ubuntu y mostrado en el listado de aplicaciones. El contenido del fichero define lo siguiente:

```

1 [Desktop Entry]
2 Version=1.0
3 Name=CollimatorDesktop
4 Comment=Easy collimation of your telescope
5 GenericName=Telescope Collimator
6 Keywords=Collimate ; Telescope ; Collimation
7 Exec=/bin/CollimatorDesktop
8 Terminal=false
9 X-MultipleArgs=false
10 Type=Application
11 Categories=GNOME;GTK;Stars ;
12 StartupNotify=true
  
```

Como se puede observar en el contenido del fichero, además del nombre, comentarios y palabras clave, se define a través de *Exec* el ejecutable */bin/CollimatorDesktop*, y a través de *Terminal* que no es necesario abrir una terminal para ejecutarlo, pues es una aplicación visual.

CAPÍTULO 7

Pruebas

7.1 Pruebas unitarias

Para las pruebas de la aplicación, se ha utilizado la herramienta QTest que ofrece Qt[19].

Mediante esta herramienta, se pueden crear tests unitarios utilizando los slots de Qt. Los slots que se encuentren dentro de la clase de un proyecto de testing, se ejecutarán automáticamente, y mediante el método QVERIFY, se determina si el test pasa o falla.

Para hacer esto, se ha creado un proyecto de Qt que importa todas las clases necesarias de los proyectos CollimatorLib y CollimatorDesktop, y se ha creado una clase que realiza los tests unitarios.

7.1.1. Persistencia

Se ha creado un test unitario para la persistencia, donde se comprueba que las operaciones *CRUD* sobre la base de datos se realizan correctamente.

Dado que la clase TelescopeDAO utiliza las demás clases DAO, se ha considerado realizar los tests unitarios potencialmente sobre esta clase.

Sobre esta clase, se realiza primero una operacion save seguida de las operaciones get, update, getAll y remove. Tras cada una de estas operaciones, se comprueba que dicho objeto en base de datos se encuentra igual que en memoria para dar el test por válido.

7.1.2. Conexión con INDI

Se ha creado también un test para la conexión con el servidor INDI. Dicho test, se conecta a un servidor INDI, y si la conexión es correcta, espera un segundo para recibir los dispositivos que se encuentren en dicho servidor. En caso de que en este tiempo no encuentre ningún dispositivo, el test resultará fallido.

7.1.3. Colimación

Se ha creado un test que comprueba que para una muestra de imágenes, el programa resuelve correctamente la colimación, es decir, que devuelve correctamente el tornillo que se debe apretar o aflojar, dependiendo del caso.

Este se considera uno de los tests más importantes, no solo por ser la función principal de la aplicación, sino porque valida que los cálculos se realicen correctamente.

7.2 Pruebas de integración

A lo largo del desarrollo de la aplicación, se hacían pruebas de integración continuamente para cada funcionalidad implementada. Estas se realizaban e iban construyendo a medida que se desarrollaba el proyecto, dado que se ha utilizado metodología ágil, en concreto SCRUM.

De esta forma, las siguientes pruebas se realizaban una vez se implementaba un componente del programa.

Las pruebas de integración realizadas son las siguientes:

Objetivo	Descripción de la prueba	Resultados esperados
Validar la integración entre la lógica de la librería y la aplicación de escritorio.	Cargar una imagen en la aplicación y utilizar la colimación automática.	En el programa se visualizará el resultado de la colimación.
Validar la integración entre la base de datos y la aplicación.	Crear una configuración, modificarla y reiniciar el programa.	La aplicación carga los datos de la configuración creada en la sesión anterior.
Validar la integración entre la conexión remota y la aplicación.	Conectarse con un servidor INDI y cargar imágenes.	La aplicación muestra los resultados de la colimación como si fuera una imagen cargada desde el ordenador local.
Validar el comportamiento de la reflexión de la imagen.	Cargar una imagen y utilizar la opción de reflejado de imagen y colimación automática.	La imagen se reflejará y la colimación se hará de acuerdo a la imagen reflejada.
Validar el comportamiento de la reflexión de los tornillos.	Cargar una imagen y utilizar la opción de reflejado de tornillos y colimación automática.	Los tornillos se visualizarán reflejados y la colimación se hará de acuerdo con los tornillos reflejados.
Validar el comportamiento de captura de pantalla.	Cargar una imagen y realizar una captura de pantalla mediante el botón de la aplicación.	La captura de pantalla se guardará correctamente en una imagen png.

Tabla 7.1: Pruebas de integración

7.3 Pruebas de validación

Después de las pruebas de integración se realizaron distintas pruebas de validación. Durante estas pruebas de validación, probamos el funcionamiento de todos los botones y campos de formulario implementados, así como todas las funcionalidades del programa. Después de cada fase de pruebas se realizaba un informe con los defectos encontrados y sugerencias de mejora para las siguientes iteraciones del producto. Ejemplos de sugerencias encontradas en dichas pruebas de validación serían las opciones de reflejado de los tornillos o de la imagen.

CAPÍTULO 8

Conclusiones

Gracias al desarrollo de este proyecto, colimar el telescopio es más sencillo para el usuario medio. En este proyecto se ha proporcionado un software fácil de utilizar y con múltiples funciones que serán muy útiles para los usuarios finales.

Mediante un primer análisis del problema, se ha logrado limitar el alcance del proyecto a algo alcanzable dentro del mismo. Además de conseguir con satisfacción las expectativas iniciales que debía cumplir el software, se han logrado todos los objetivos especificados al inicio del mismo. A pesar de todas las funcionalidades del proyecto y de su desarrollo, todavía quedan opciones por explorar, como se puede consultar en el capítulo de trabajos futuros de esta memoria.

Durante la realización del proyecto se han encontrado distintas dificultades. La principal ha sido la obtención de datos dada una imagen. Mi falta de experiencia en el campo de visión por computador ha supuesto que se ralentizaran las primeras fases del proyecto. Poco a poco, gracias a este trabajo, mis conocimientos en dicho campo han mejorado notablemente haciendo, además, crecer mi interés. Ahora, conociendo más este campo, concibo muchos más proyectos que podrían ser interesantes y aportar utilidad a muchos ámbitos todavía inexplorados.

Por último, agradecer a mi tutor, Alberto José Pérez Jiménez, por brindarme la oportunidad, conocimiento y apoyo para realizar un proyecto tan interesante y didáctico como este, además de introducirme en el mundo de la astronomía.

8.1 Conclusiones finales

A continuación, vamos a repasar los objetivos presentados al inicio de la memoria, para ver si estos se han cumplido con éxito:

- ☑ **Proporcionar un sistema de guiado para la colimación de telescopios, que de forma automática, detecte que tornillos son necesarios ajustar:** Este es el objetivo principal de la aplicación, que se ha cumplido con éxito como se ha explicado en el capítulo cinco, en el apartado de procesamiento de imágenes.
- ☑ **Posibilitar la configuración de varios telescopios o un mismo telescopio con varias configuraciones:** Se ha programado la aplicación de forma que cada configuración sea independiente y persistente. Un ejemplo de esto sería la figura A.1.
- ☑ **Realizar una interfaz de usuario que sea intuitiva y fácil de utilizar para un usuario inexperto:** Se han programado las interfaces de forma que sean claras y con ico-

nos representativos. Además, el programa al crear una configuración, la crea con valores por defecto de forma que sea sencillo ajustar las mismas.

- ☑ **Guiar al usuario durante la colimación mediante imágenes tomadas previamente:** Este objetivo se ha abordado, pues el usuario puede elegir entre utilizar imágenes de su ordenador o del servidor INDI.
- ☑ **Guiar al usuario durante la colimación mediante imágenes en tiempo real:** Este objetivo ha sido cumplido gracias a la conexión con los servidores INDI.
- ☑ **Posibilitar colimación de forma manual:** Se ha programado la aplicación de forma que sea posible colimar manualmente el telescopio, dibujando una mira sobre las imágenes capturadas por el programa.
- ☑ **Posibilitar la realización de capturas de pantalla durante la colimación:** Este objetivo ha sido cumplido programando un botón capaz de realizar dicha función.
- ☑ **Despliegue automático de la aplicación:** Este objetivo se ha cumplido, mediante el uso de la herramienta docker, como se explica en el capítulo de implantación.

8.2 Relación con los estudios cursados

Desde el primer momento ha sido necesaria la puesta en práctica de los estudios cursados a lo largo de todo el grado en *Ingeniería Informática*. Desde la fase inicial del proyecto, se aplicaron los conocimientos de la asignatura *Análisis y especificación de requisitos*, necesaria para realizar la correcta especificación de requisitos que se iban a implementar en el proyecto.

Dentro de la fase de diseño, requerí de los conocimientos abordados en varias asignaturas para la realización de los diferentes diagramas. Además, los contenidos impartidos en la asignatura *Bases de Datos* fueron de gran utilidad, tanto para la creación de la base de datos, como para la posterior realización de sus consultas. También fue necesaria la implementación de algunos diseños arquitectónicos aprendidos en la mención de Ingeniería del software, que ayudaron a agilizar la estructura del proyecto.

También cabe destacar la utilización de los conocimientos aprendidos en la asignatura *Diseño de software*, con el fin de intentar que el código implementado sea lo más legible y de mayor calidad posible. Así mismo, también se aplicaron los conceptos de la asignatura *Mantenimiento de software*, que me permitió desarrollar las competencias necesarias para llevar a cabo la documentación y pruebas unitarias de este proyecto.

8.3 Trabajos futuros

En lo relativo al trabajo futuro, este proyecto ha abierto varias puertas para próximos proyectos que se recogen a continuación, dado que se puede utilizar la librería que se ha desarrollado para otros proyectos:

- **Aplicación web:** Se propone utilizar la librería para crear un programa cliente-servidor, donde el servidor reciba imágenes o direcciones a servidores INDI y realice y la colimación, devolviéndola a un cliente web donde se muestren los resultados al usuario.

- **Aplicación para dispositivos móviles:** Se propone utilizar la librería para hacer un cliente móvil, adaptando las interfaces del programa desarrollado a dispositivos móviles, utilizando la librería de la misma forma que lo hace la aplicación de escritorio.

Bibliografía

- [1] Wikipedia. Reflecting telescope — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Reflecting%20telescope&oldid=1030733544>, 2021. [Online; accessed 28-June-2021].
- [2] *Collimation and Adjustment Techniques*, pages 153–178. Springer London, London, 2005.
- [3] Matthew Gates. Guía de usuario de stellarium.
- [4] Gary Bradski and Adrian Kaehler. Opencv. *Dr. Dobbs's journal of software tools*, 3, 2000.
- [5] Wikipedia. OpenCV — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=OpenCV&oldid=1011992803>, 2021. [Online; accessed 28-June-2021].
- [6] Lubomir Bourdev and Hailin Jin. Generic image library. *Software Developer's Journal*, page 4252, 2007.
- [7] William D Pence. Cfitsio: a fits file subroutine library. *Astrophysics Source Code Library*, pages ascl-1010, 2010.
- [8] Wikipedia. Instrument Neutral Distributed Interface — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Instrument%20Neutral%20Distributed%20Interface&oldid=1017379350>, 2021. [Online; accessed 28-June-2021].
- [9] J.R. Janesick. Ccd transfer method: standard for absolute performance of ccds and digital ccd camera systems. In *Proceedings of SPIE*, volume 3019, page 70, 1997.
- [10] John Doe. Recommended practice for software requirements specifications (ieee). *IEEE, New York*, 2011.
- [11] Qt Design Studio, desarrollado por Qt. <https://www.qt.io/product/development-tools>.
- [12] Mike Owens and Grant Allen. *SQLite*. Springer, 2010.
- [13] Charles Anderson. Docker [software engineering]. *Ieee Software*, 32(3):102–c3, 2015.
- [14] Wikipedia. Subversión — Wikipedia, the free encyclopedia. <http://es.wikipedia.org/w/index.php?title=Subversi%C3%B3n&oldid=136266920>, 2021. [Online; accessed 28-June-2021].
- [15] Copy-Modify-Merge Continued. <https://www.cl.cam.ac.uk/local/sys/web/subversion/introduction/subversionguide.html>.

- [16] Dimitri Van Heesch. Doxygen: Source code documentation generator tool. URL: <http://www.doxygen.org>, 2008.
- [17] Rohit Chandra, Leo Dagum, David Kohr, Ramesh Menon, Dror Maydan, and Jeff McDonald. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.
- [18] Wikipedia. Circunferencia goniométrica — Wikipedia, the free encyclopedia. <http://es.wikipedia.org/w/index.php?title=Circunferencia%20goniom%C3%A9trica&oldid=133506893>, 2021. [Online; accessed 26-June-2021].
- [19] Qt Test Overview. <https://doc.qt.io/qt-5/qtest-overview.html>.

APÉNDICE A

Funcionamiento del programa

Al tratarse de una aplicación de escritorio, la aplicación puede ejecutarse desde la línea de comandos mediante el comando "CollimatorDesktop" o desde el menú de aplicaciones.

A.1 Listado de telescopios

Esta es la vista que se abre al iniciar la aplicación. Contiene un listado de cada una de las configuraciones de telescopios que el usuario ha creado.

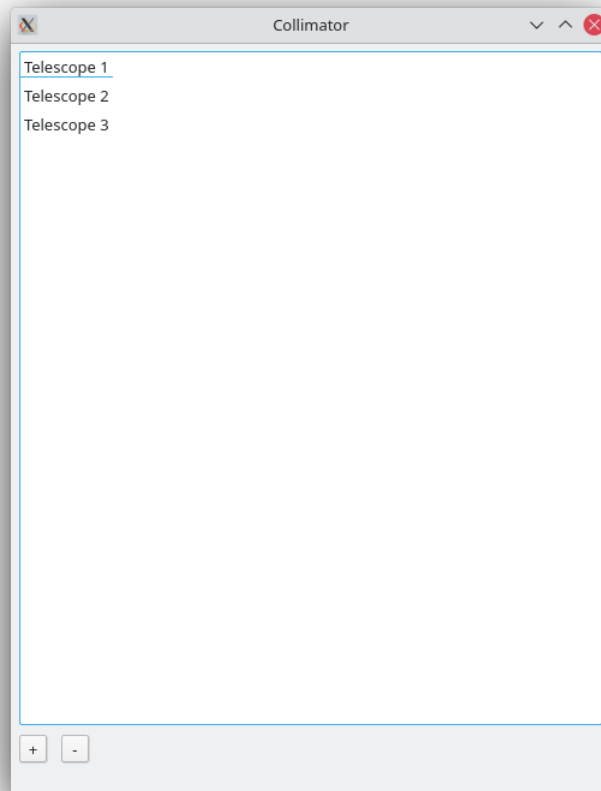


Figura A.1: Vista de lista de Telescopios (Elaboración propia)

Desde esta vista, podemos crear o eliminar configuraciones con los botones "+" y "-" que se sitúan debajo de la lista. Al pulsar el botón para crear, se abrirá la vista A.2, que nos preguntará el nombre de la configuración que el usuario desee introducir y lo añadirá a la lista.

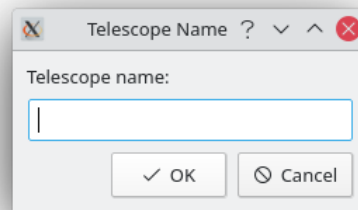


Figura A.2: Vista de la adición de un telescopio (Elaboración propia)

Por el lado contrario, si se selecciona un telescopio y se pulsa el botón de eliminar, se borrará la configuración de la aplicación.

Cuando el usuario realice doble clic sobre una de las configuraciones, se abrirá en una nueva ventana la vista A.3.

A.2 Vista de un Telescopio

Esta es la vista principal de la aplicación. En esta vista se visualiza la colimación.

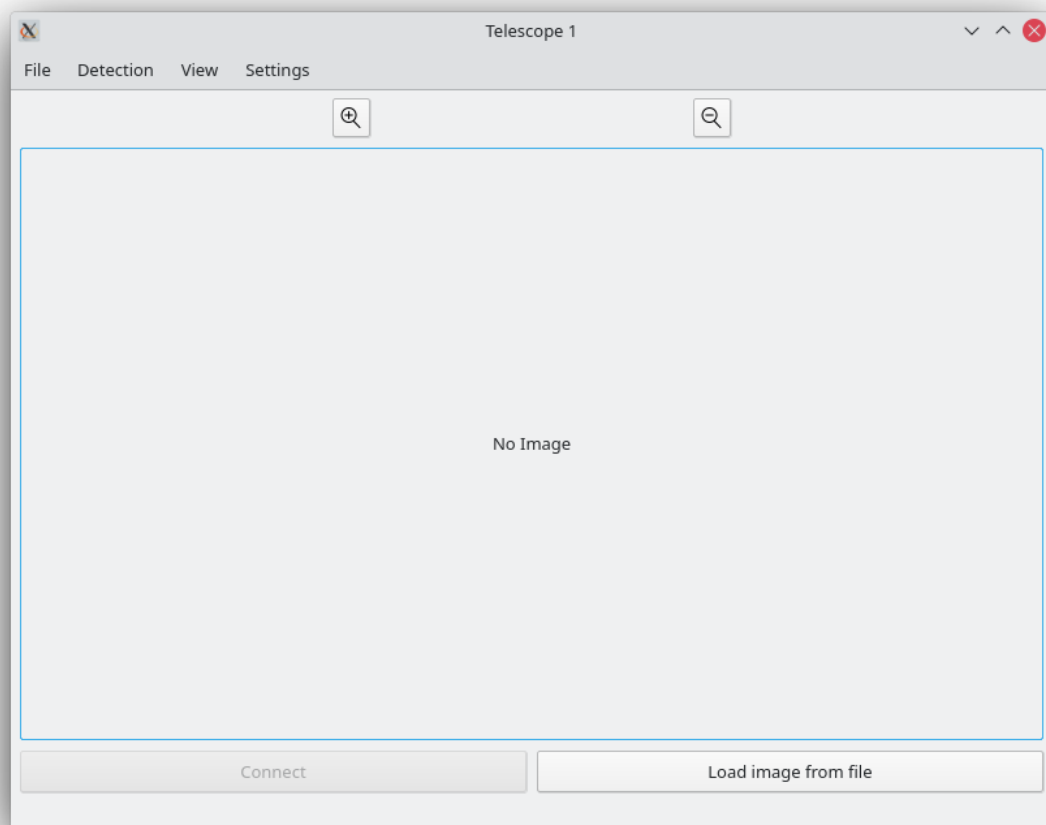


Figura A.3: Vista principal de un Telescopio (Elaboración propia)

La vista contiene los siguientes menús para editar la configuración que el usuario ha seleccionado:

- **File**

- **Save screenshot:** Permite al usuario guardar una imagen de aquello que esté visualizando durante la colimación. En cuanto se aprieta, se toma una imagen de lo que se está visualizando y se abre un diálogo en el que el usuario elige dónde desea guardar la captura.
- **Close:** Permite al usuario cerrar la vista del telescopio.

- **Detection:** Desde aquí se puede seleccionar si el usuario desea realizar una colimación manual o automática.

- **Manual:** Permite al usuario cambiar al modo de colimación manual.
- **Automatic:** Permite al usuario cambiar al modo de colimación automática.

- **View:** Desde aquí el usuario puede seleccionar aquello que desea visualizar sobre la imagen que se carga.

- **Bounding Box:** Si esta opción está activada, el usuario podrá ver en rojo el cuadro delimitador cuando haya una imagen cargada. Esta opción únicamente es utilizable durante la colimación automática.

- **Calculated Center:** Si esta opción está activada el usuario podrá ver en verde el centro de masa calculado cuando haya una imagen cargada. Esta opción únicamente es utilizable durante la colimación automática.
- **Screws:** Si esta opción está activada el usuario podrá ver los tornillos que haya configurado anteriormente desde la vista A.8.
- **Mirror:** Desde este menú, el usuario puede elegir aquello que desea reflejar en la imagen
 - **Image**
 - ◊ **Horizontally:** Si esta opción está activada, la imagen se mostrará espejada horizontalmente.
 - ◊ **Vertically:** Si esta opción está activada, la imagen se mostrará espejada verticalmente.
 - **Screws**
 - ◊ **Horizontally:** Si esta opción está activada, los tornillos se mostrarán espejados horizontalmente y se calculará la colimación conforme al espejo.
 - ◊ **Vertically:** Si esta opción está activada, los tornillos se mostrarán espejados verticalmente y se calculará la colimación conforme al espejo.
- **Settings:** Desde este menú el usuario puede configurar las opciones de colimación y la conexión con el servidor INDI remoto.
 - **Image processing:** Esta opción abre la vista A.7, que permite configurar las opciones del procesamiento de imágenes.
 - **Screws:** Esta opción abre la vista A.8, que permite configurar los tornillos.
 - **Connection:** Esta opción abre la vista A.10, que permite configurar la conexión con el servidor INDI.
 - **Debug Image:** Esta opción permite al usuario abrir una ventana durante la colimación automática que muestra la depuración de la imagen procesada, tal y como se muestra en la figura A.4

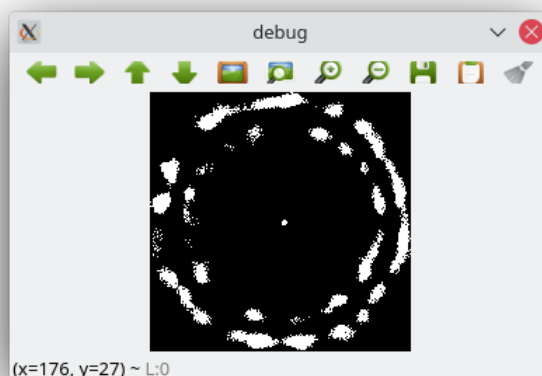


Figura A.4: Vista de depuración de imagen (Elaboración propia)

Además de los menús, en la vista existen cuatro botones: dos en la parte superior para realizar zoom y deshacerlo y dos en la parte inferior; uno para conectarse con el servidor

INDI configurado en la vista A.10 y obtener las imágenes del servidor en tiempo real para realizar su colimación y otro para cargar una imagen desde un archivo. Una vez se conecta con el servidor o se carga una imagen, esta se muestra como en la figura A.5 si es automática, o A.6 si es manual.

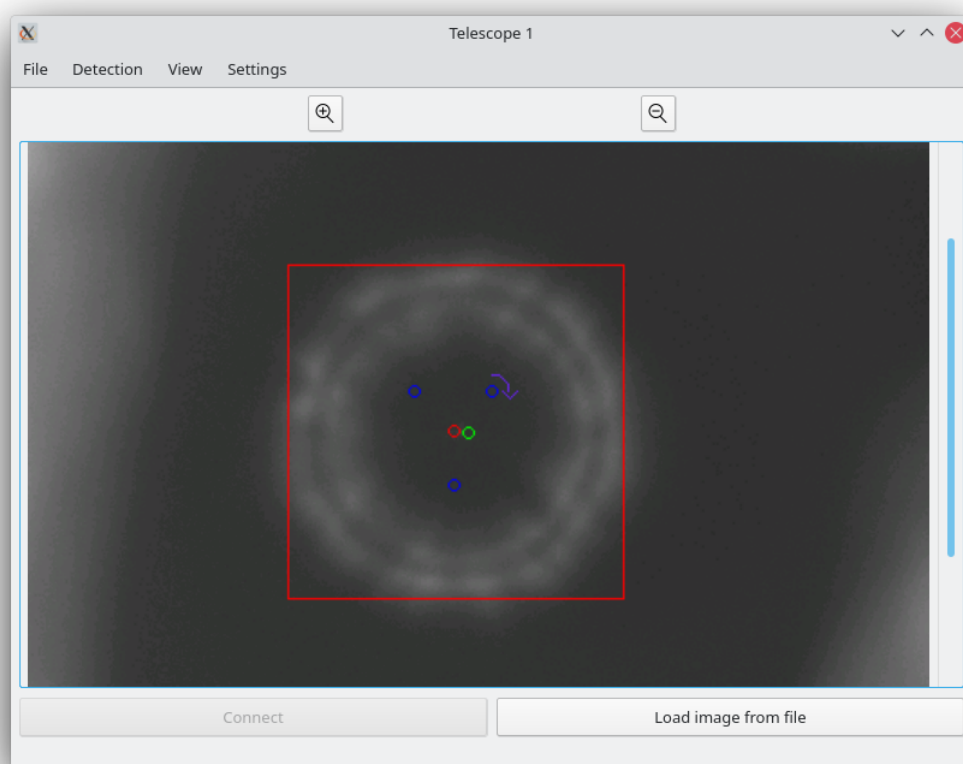


Figura A.5: Vista de colimación automática (Elaboración propia)

En el modo de colimación automático se puede visualizar, si está configurado, el centro de masa en verde, el cuadro delimitador en rojo, los tornillos en azul, y se muestra sobre uno de los tornillos la acción que se debe realizar para llevar a cabo la colimación mediante una flecha morada que apunta en la dirección que se debe girar el tornillo.

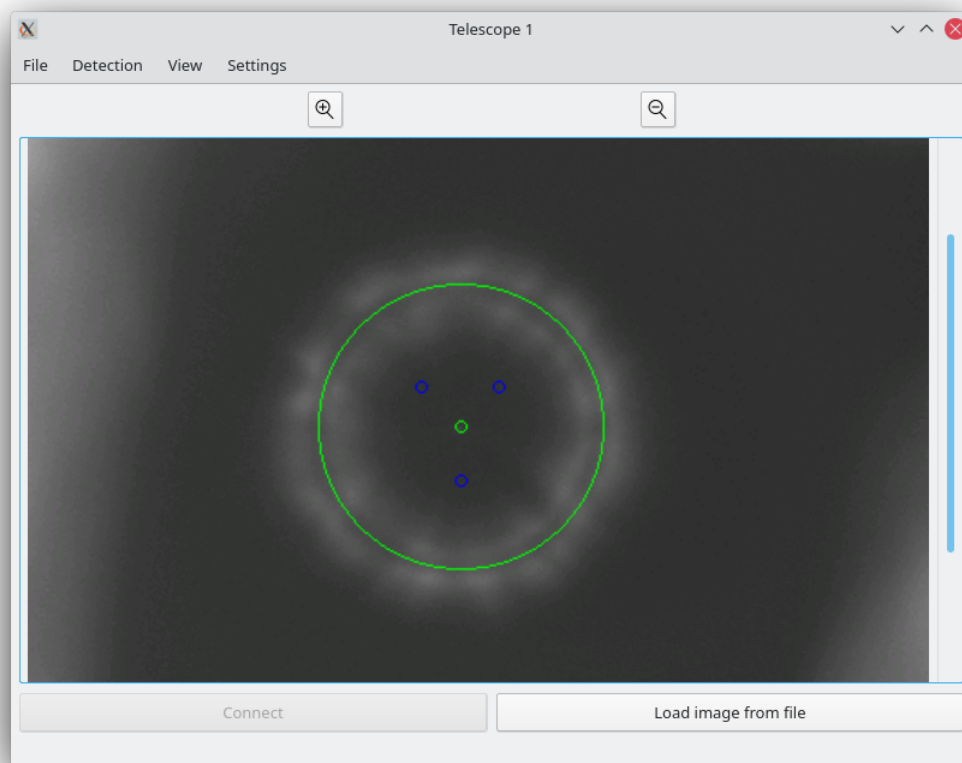


Figura A.6: Vista de colimación manual (Elaboración propia)

En el modo de colimación manual se pueden visualizar los tornillos configurados junto a una mira de color verde que el usuario puede ajustar clicando en centro y arrastrando el ratón sobre la imagen para ajustar el radio.

De esta forma, el usuario puede comparar los anillos de difracción con el radio que ha dibujado para llevar a cabo la colimación.

A.3 Vista de la configuración de procesamiento de imágenes

Esta vista permite configurar las opciones del procesamiento de imágenes durante la colimación automática.

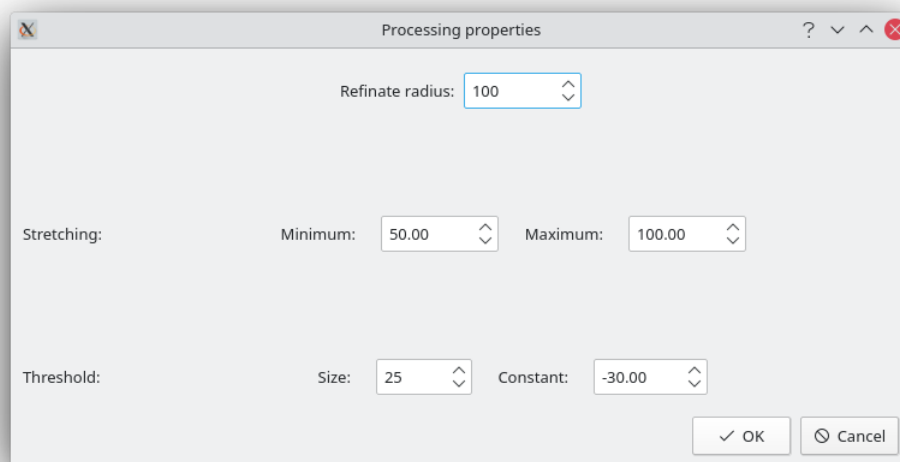


Figura A.7: Vista de la configuración de Procesado de imágenes (Elaboración propia)

Los parámetros configurables son los siguientes:

- **Refinate radius:** Este valor es el que se utiliza para el cálculo del cuadro delimitador.
- **Stretching:** Estos valores son los que se utilizan a la hora de calcular el binario, durante la normalización de la imagen.
 - **Minimum:** Valor mínimo para la normalización.
 - **Maximum:** Valor máximo para la normalización.
- **Threshold:** Estos valores son los que se utilizan a la hora de calcular el binario, durante el método del umbral adaptativo.
 - **Size:** Tamaño para el umbral adaptativo.
 - **Constant:** Constante para el umbral adaptativo.

Estos son los parámetros que inciden directamente en el procesado del modo automático de la colimación. El usuario puede observar cómo influyen dichos parámetros en la vista de depuración de imagen A.4.

A.4 Vista de la configuración de tornillos

Esta vista permite configurar los tornillos de la configuración de un telescopio.

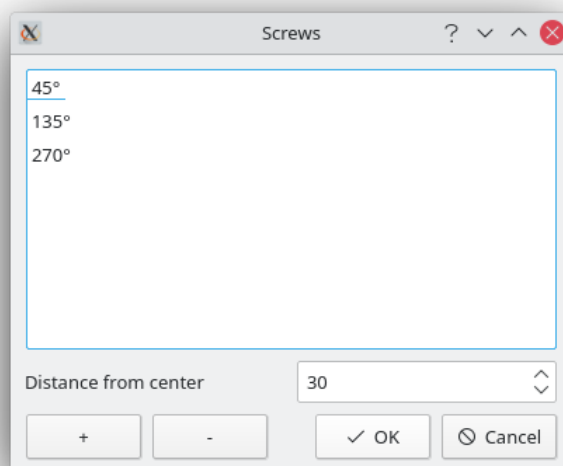


Figura A.8: Vista de la configuración de tornillos (Elaboración propia)

El usuario puede añadir y quitar tornillos con los botones inferiores. También puede determinar la distancia en píxeles a la que se encuentran los tornillos del centro de la vista del telescopio. Los tornillos son representados por su correspondiente ángulo en la circunferencia goniométrica.

El usuario no puede introducir menos de dos tornillos, ni introducir todos los tornillos en un único cuadrante en la circunferencia goniométrica.

Cuando el usuario pulsa sobre el botón para añadir un tornillo, se muestra el diálogo A.9, donde puede introducir un ángulo de cero a trescientos sesenta grados.

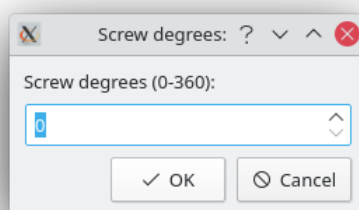


Figura A.9: Vista de la adición de tornillo (Elaboración propia)

A.5 Vista de la configuración de conexión

Esta vista permite configurar la conexión con el servidor INDI.

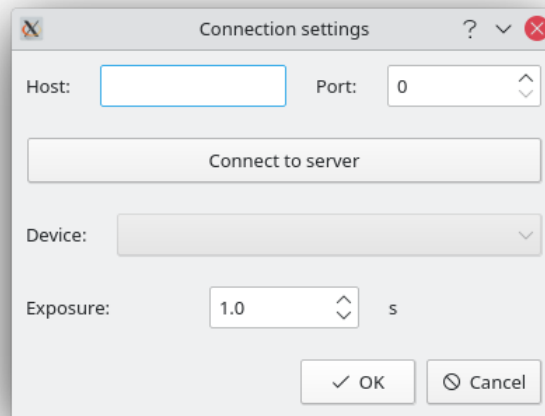


Figura A.10: Vista de la configuración de conexión (Elaboración propia)

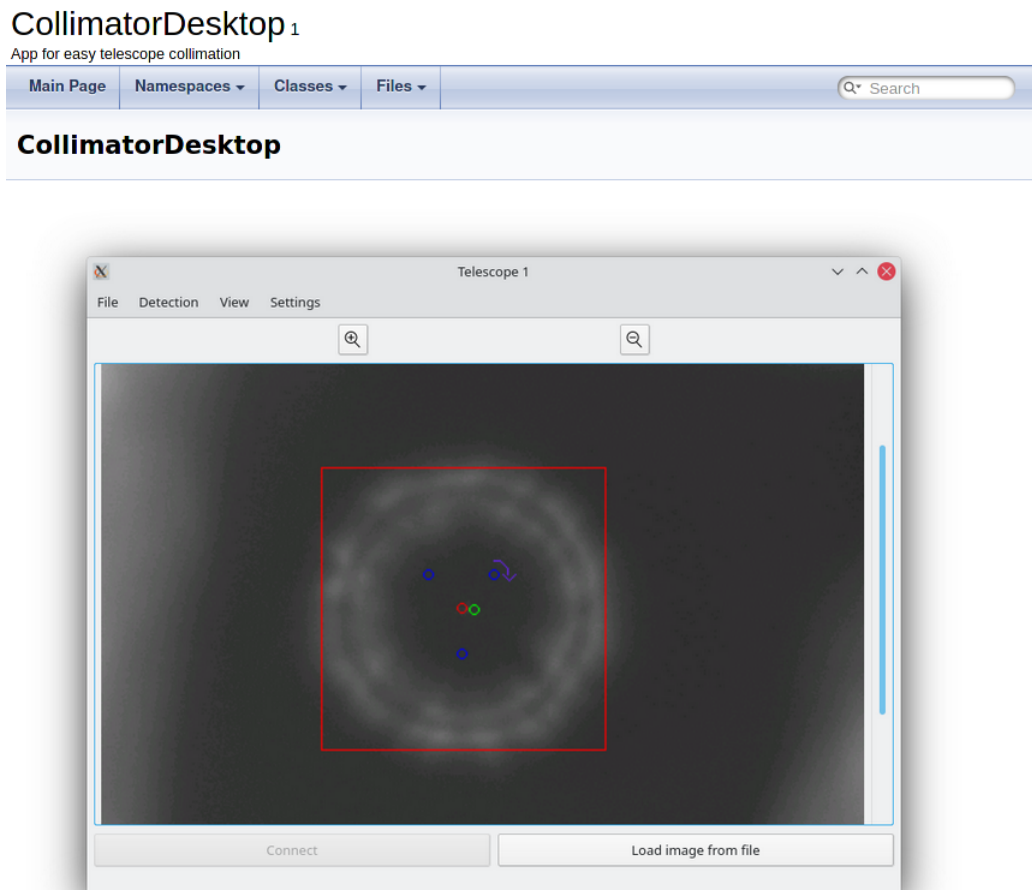
En esta vista, el usuario introduce el host y el puerto al que quiere conectarse. Una vez el usuario pulsa sobre el botón para realizar la conexión, se carga la lista de dispositivos que contiene una cámara en el servidor al cual se ha conectado.

Desde aquí el usuario puede, además, configurar la exposición que se le pedirá al servidor. Cuanto menor sea, con mayor frecuencia se recibirán imágenes, sin embargo, más oscuras serán estas.

APÉNDICE B

Doxygen

Al utilizar Doxygen para generar la documentación, se genera una página web que contiene toda la documentación. La portada de dicha página web, contiene una descripción de la aplicación como se muestra en la figura B.1



CollimatorDesktop is a program for easy telescope collimation for Linux.

Figura B.1: Portada de Doxygen (Elaboración propia)

Además, doxygen genera la documentación para cada una de las clases, que se puede observar dentro de *Classes*, como en la figura B.2

CollimatorDesktop¹






















App for easy telescope collimation

Main Page
Namespaces ▾
Classes ▾
Files ▾

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

[detail level 1 2]

 CollimationAction	Object that contains the collimation step to be performed by the user. It contains the screw to use and the action of tightening or loosening the screw
 Collimator	Main class of the library. This class is in charge of carrying out the transformations and calculations on the image, in addition, it allows uploading the images in FITS format directly to MATS, the format used by OpenCV
 CollimatorImage	That inherits from QLabel, a Qt class that provides the ability to display text or images. It is inherited to further shrink the implementation when loading images or drawing on them and to improve the code quality of the controllers. Interacts directly with most classes in the library
 ConnectionSettingsDialog	Corresponds to the view controller that configures the connection to the remote image server. This view interacts directly with the library to connect to the server and allows you to choose the available devices
 DatabaseConnectionSingleton	Singleton with the database connection. Since the SQLite manager is used, which only allows a simultaneous connection with the database, it is necessary to instantiate it only once within the program, for this reason the singleton pattern was chosen
 ImageResults	Object that contains the results of a collimation. It contains both the technical aspects (such as the center of brightness of the image) and the action that the user must perform to collimate the telescope
 INDIClient	Auxiliary class for the implementation of a client with the INDI library. It is necessary that this class extends BaseClient, and facilitates the abstract methods that the program performs during connection
 PaintingProperties	Class that makes the drawing on an image and contains its properties. It contains both the properties of the drawing of the elements of the telescope, as well as of the calculations performed
 PaintingPropertiesDAO	Corresponds to the Data Access Object of the PaintingProperties object
 ProcessingProperties	Object that contains all the properties that the Collimator class will take into account when performing the calculations
 ProcessingPropertiesDAO	Corresponds to the Data Access Object of the ProcessingProperties object
 ProcessingPropertiesSettings	Corresponds to the view controller to configure the image processing parameters
 RemoteTelescope	Object containing the data necessary for connection to the remote telescope
 RemoteTelescopeDAO	Corresponds to the Data Access Object of the RemoteTelescope object
 Screw	Object that represents a telescope screw. It contains calculations to obtain your position given a reference point
 ScrewsDialog	Corresponds to the view controller to configure the screws that a telescope has
 Telescope	Main object of the library. This object contains the properties of the telescope, as well as the screws and collimation properties of the telescope
 TelescopeDAO	Corresponds to the Data Access Object of the TelescopeUI object. From the main view, the getAll method is consulted to obtain all the telescope configurations of the system
 TelescopeSelector	Corresponds to the controller of the first view when opening the application. Loads the configurations of all telescopes and allows both to add and delete them
 TelescopeUI	That inherits from the Telescope library to add the PaintingProperties object, necessary for drawing the telescope attributes on the image
 TelescopeView	Corresponds to the main view controller of a telescope. It contains an instance of CollimatorImage , with which the images are loaded and displayed in real time. From here, you control the opening of the configuration windows of the image processing properties, as well as the connection properties


Generated by  1.9.2

Figura B.2: Lista de clases en Doxygen (Elaboración propia)

Dentro de esta página, se lista cada una de las clases del programa con su descripción, pudiendo acceder a cada una de ellas pulsando, para obtener una descripción mas detallada de la clase y cada uno de sus métodos.

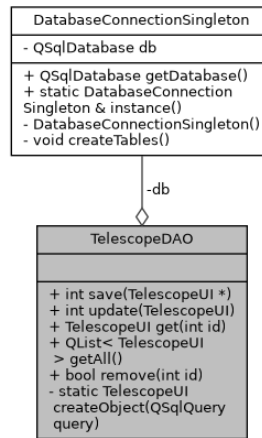
Dentro de cada clase, podemos encontrar una descripción de la propia clase, y acto seguido, el gráfico de colaboración de dicha clase. Mas adelante, la lista de métodos, tanto públicos como privados.

Además, para cada método, se genera un *call graph* (B.4), que indica a que métodos llama dicho método, y un *caller graph* (B.5), que indica desde que métodos se llama dicho método.

The **TelescopeDAO** class corresponds to the Data Access Object of the **TelescopeUI** object. From the main view, the `getAll` method is consulted to obtain all the telescope configurations of the system. [More...](#)

```
#include <telescopedao.h>
```

Collaboration diagram for TelescopeDAO:



Public Member Functions

int	save (TelescopeUI *)	TelescopeDAO::save Method for saving an instance of a Telescope . More...
int	update (TelescopeUI)	update Method for updating an instance of a telescope. Also updates or creates other objects that a telescope contains More...
TelescopeUI	get (int id)	get Method for obtaining a Telescope by id More...
QList< TelescopeUI >	getAll ()	getAll Method for obtaining all Telescopes in the database More...
bool	remove (int id)	remove Method for removing a Telescope by id More...

Static Private Member Functions

static TelescopeUI	createObject (QSqlQuery query)	createObject Private method for getting Telescope object using a row of a query More...
--------------------	---------------------------------------	--

Figura B.3: Clase en Doxygen (Elaboración propia)

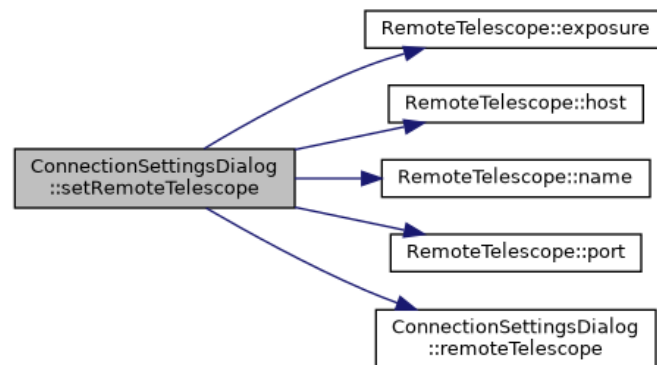


Figura B.4: Call graph en Doxygen (Elaboración propia)



Figura B.5: Caller graph en Doxygen (Elaboración propia)

APÉNDICE C

Estudio de la paralelización de los cálculos

Para hacer que los tiempos de los cálculos realizados sobre las imágenes sean menores, se ha utilizado OpenMP para paralelizar los bucles de las funciones que realizan el normalizado y el cálculo del centro de masa.

El bucle de la normalización:

```
1 #pragma omp parallel for private(i)
2 for(i = 0; i < img->cols * img->rows; i++)
3 {
4     uchar *v = &(img->ptr<uchar>(i / img->cols))[i % img->cols];
5     if(*v < vmin)
6         *v = 0;
7     else if(*v > vmax)
8         *v = 255;
9     else
10         *v = 255*(*v - vmin)/(vmax-vmin);
11 }
12 }
```

El bucle del cálculo de centro de masa:

```
1 #pragma omp parallel for private(i) reduction(+:ncx,ncy,count)
2 for(i = starty; i < heigth * width; i++)
3 {
4     uchar* gray = &(img->ptr<uchar>(i / width))[i % width];
5     if(*gray == 255)
6     {
7         ncx += i % width;
8         ncy += i / width;
9         count++;
10     }
11 }
```

Como se observa en el código, en ambas funciones se utiliza `pragma omp` para paralelizar el bucle `for`. En ambos bucles se privatiza la variable *i* para que dicha variable sea privada dentro de cada hilo, mientras que se ha utilizado *reduction* para realizar la suma de las variables *ncx*, *ncy* y *count* de cada hilo.

Tras realizar un análisis de ambas funciones, obtenemos los siguientes resultados. Las pruebas se han realizado sobre un ordenador capaz de utilizar ocho hilos y con una imagen de cuatro mil píxeles de ancho cuatro mil píxeles de alto.

Nº de hilos	Tiempo de ejecución (s)
1	0,120
2	0,063
3	0,043
4	0,042
5	0,041
6	0,040
7	0,035
8	0,033
9	0,034
10	0,033

Tabla C.1: Resultados de la paralelización de la normalización (Elaboración propia)

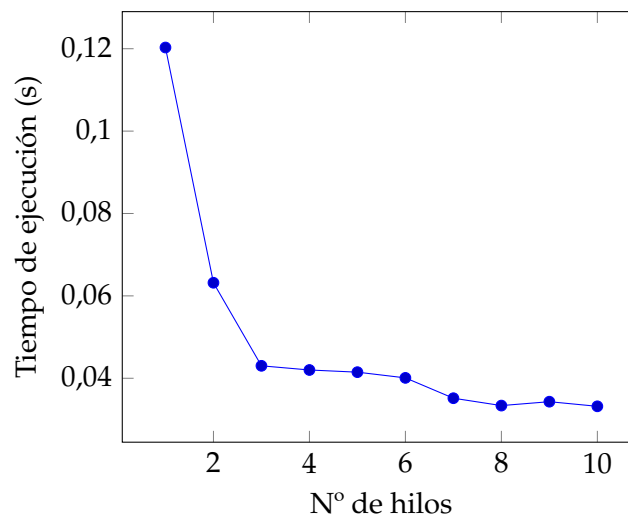


Figura C.1: Tiempos de ejecución de la normalización (Elaboración propia)

Nº de hilos	Tiempo de ejecución (s)
1	0,110
2	0,055
3	0,041
4	0,032
5	0,032
6	0,031
7	0,031
8	0,031
9	0,032
10	0,033

Tabla C.2: Tiempos de ejecución del cálculo del centro de masa (Elaboración propia)

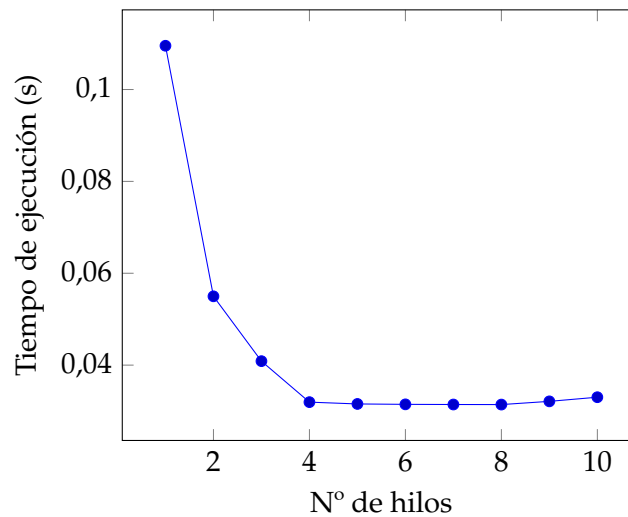


Figura C.2: Resultados de la paralelización del cálculo del centro de masa (Elaboración propia)

Dados estos datos, se calculan los *Speed-Up* y eficiencias:

Nº de hilos	Speed-Up	Eficiencia
1	0,00	0,00
2	1,90	0,95
3	2,80	0,93
4	2,86	0,72
5	2,90	0,58
6	3,00	0,50
7	3,42	0,49
8	3,61	0,45
9	3,51	0,39
10	3,62	0,36

Tabla C.3: Speed-Up y eficiencia de la normalización (Elaboración propia)

Nº de hilos	Speed-Up	Eficiencia
1	0,00	0,00
2	1,99	1,00
3	2,68	0,89
4	3,43	0,86
5	3,47	0,69
6	3,48	0,58
7	3,49	0,50
8	3,49	0,44
9	3,41	0,38
10	3,32	0,33

Tabla C.4: Speed-Up y eficiencia del cálculo del centro de masa (Elaboración propia)

A la vista de los resultados, se observa cómo el tiempo es menor cuantos más hilos hay (excepto cuando pase de ocho hilos), por lo tanto, mayor será el *Speed-Up*, sin embargo la eficiencia decrece.

APÉNDICE D

Diagramas

Diagrama de clases de la aplicación de escritorio:

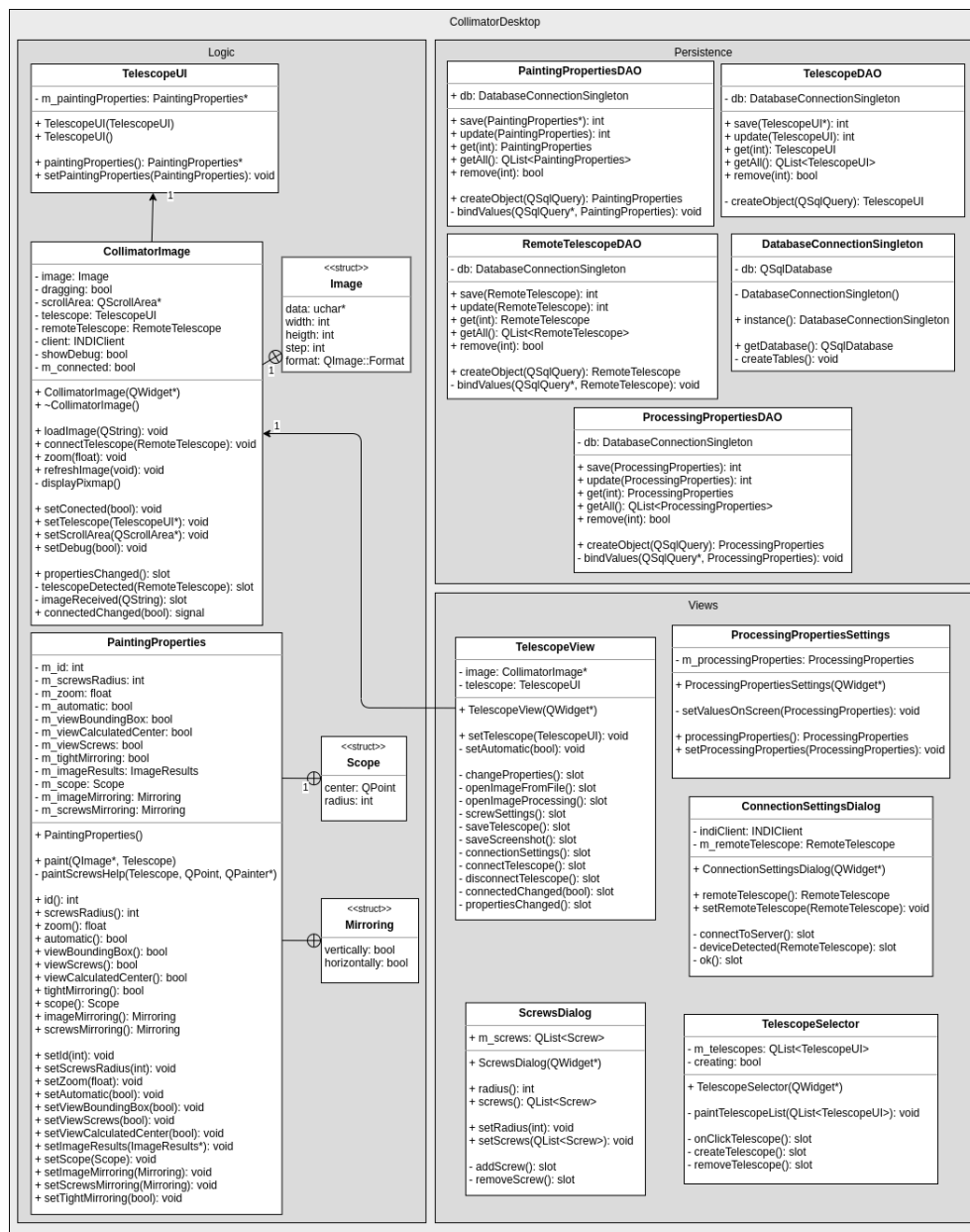


Figura D.1: Diagrama de clases de la aplicación de escritorio (Elaboración propia)

Diagrama de clases de la librería:

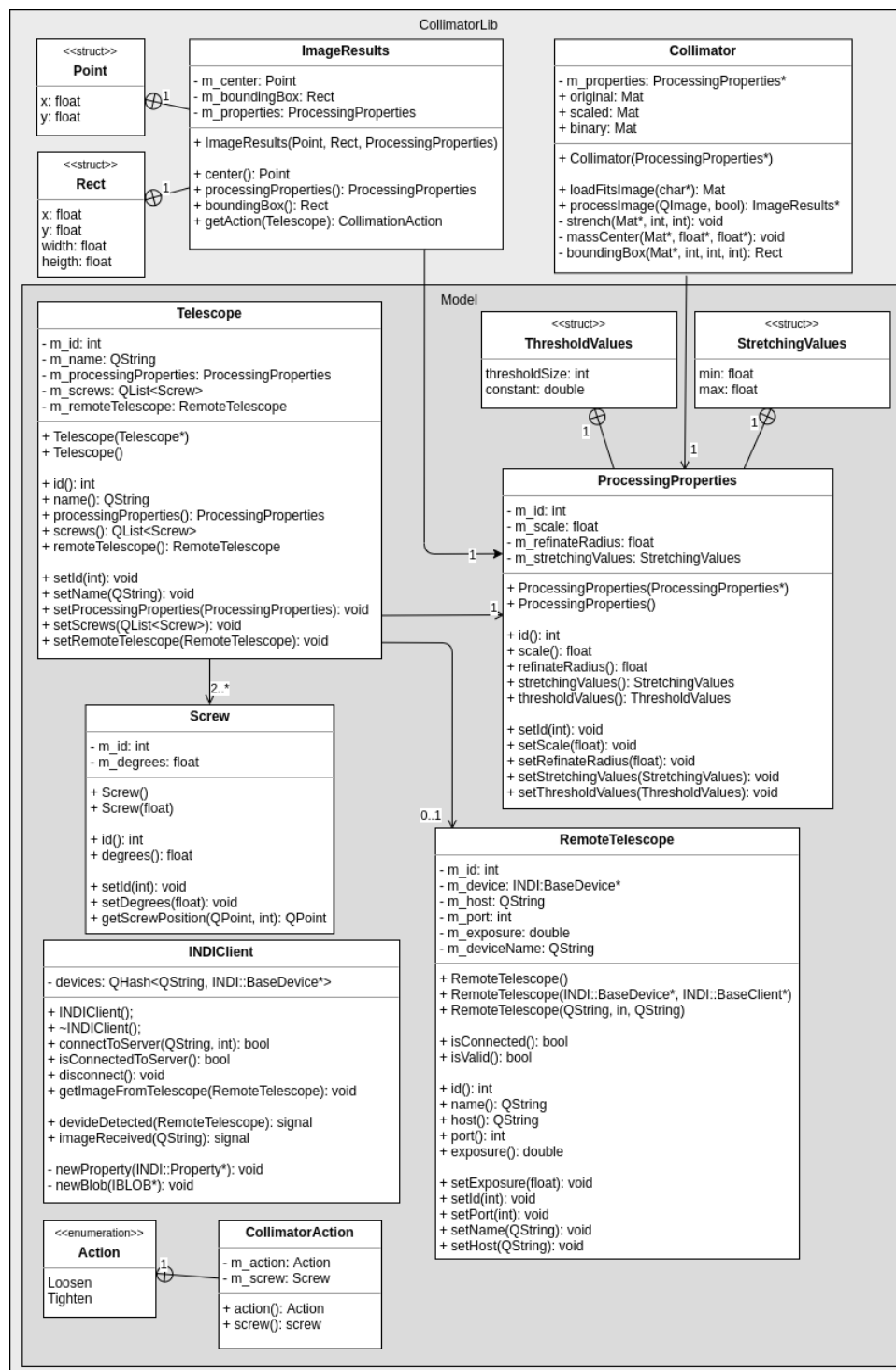


Figura D.2: Diagrama de clases de la librería (Elaboración propia)

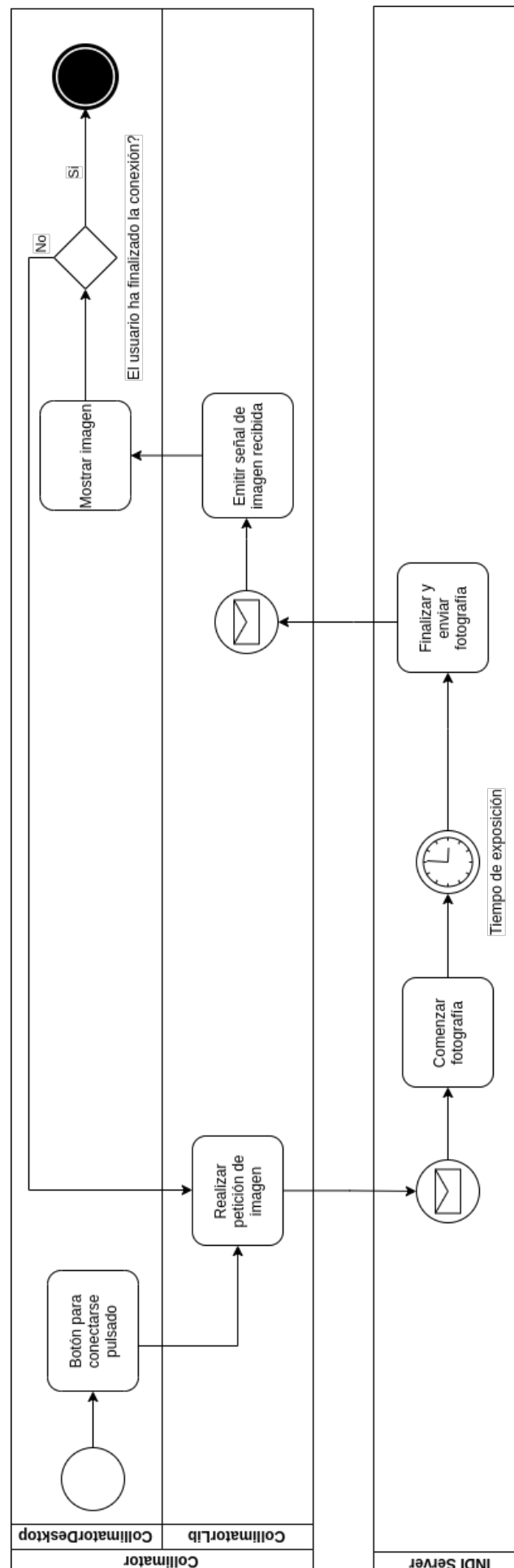


Figura D.3: Diagrama BPMN de la conexión con INDI (Elaboración propia)