Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Monitorización de los controles y gráficas de un sistema de Óptica Adaptativa

Monitoring of the controls and graphics of an Adaptive Optics system

Esther Jorge Paramio

La Laguna, 30 de junio de 2021

D. **Jonay Tomás Toledo Carrillo**, con N.I.F. 78.698.554-Y profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Roberto López López**, con N.I.F. 36.026.296-Q ingeniero óptico adscrito al Departamento de Óptica en el Instituto Astrofísico de Canarias, como cotutor

D. **Esther Soria Hernández**, con N.I.F. 72.896.041-W estudiante de doctorado perteneciente al Departamento de Enseñanza en el Instituto Astrofísico de Canarias, como cotutora

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Monitorización de los controles y gráficas de un sistema de Óptica Adaptativa*”

ha sido realizada bajo su dirección por D. **Esther Jorge Paramio**, con N.I.F. 43.835.757-B.

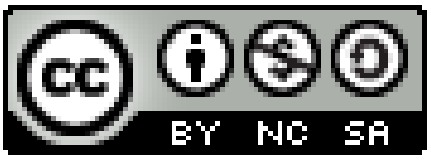
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 30 de junio de 2021

Agradecimientos

En primer lugar agradecer tanto a mi tutor, Jonay Tomás Toledo Carrillo, como a mis cotutores de la empresa, Roberto López López y Esther Soria Hernández, por su ayuda, orientación y apoyo durante la realización de este proyecto.

También agradecer a mi familia, amigos y compañeros que me han ayudado y apoyado a lo largo de estos meses de trabajo.

Licencia



Resumen

Este trabajo se ha basado en crear un módulo con interfaz gráfica para el Instituto de Atrofísica de Canarias (IAC) [18] con el fin de representar gráficos e imágenes del proceso de control y que permiten verificar el funcionamiento de un sistema de Óptica Adaptativa (OA) [38].

Los gráficos que se han generado son la imagen de la cámara de sensado, la variación de los spots mediante un diagrama de fases y la representación del frente de ondas [22] reconstruido en términos de una serie de polinomios conocimos como Polinomios de Zernike [31], los cuales nos permiten graficarlo tanto en 2D como en 3D.

Finalmente para desarrollar este proyecto se han empleado diversas herramientas como QtCreator [10] usando python y Github [26] como sistema de control de versiones.

Palabras clave: Óptica Adaptativa, Gráficas, Frente de Ondas, Polinomios de Zernike, Interfaz Gráfica, IAC

Abstract

This work has been based on creating a module with a graphical interface for the Instituto de Astrofísica de Canarias (IAC) in order to represent graphs and images of the control process and that allow to verify the operation of an Adaptative Optics system (AO).

The graphics that have been generated are the image of the sensor camera, the variation of the spots by means with a phase diagram and the representation of the wavefront reconstructed in terms of a series of poynomials known as Zernike Polynomials, which allow us to graph it both in 2D and 3D.

Finally, to develop this project, various tools have been used, such as QtCreator using python and Github as version control system.

Keywords: Optic Adaptive, Graphics, Wavefront, Zernike's Polynomials, Graphic Interface, IAC

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Un telescopio es un instrumento óptico que los astrónomos usan para ver objetos lejanos, a través del manejo de la luz y de sus propiedades. Éstos funcionan utilizando espejos curvos para captar y enfocar la luz del cielo nocturno [30].

Los telescopios reflectores están compuestos generalmente por dos espejos, uno grande llamado el espejo primario, que colecta la luz, y uno más pequeño denominado espejo secundario, que actúa como un alargador de la focal y por lo tanto de la potencia de aumento. El espejo primario se suele ubicar en un extremo del tubo del telescopio, mientras que el espejo secundario se coloca en la línea de visión ocular.

Para obtener una imagen, el telescopio se dirige a un objeto y la luz entra en el tubo. La luz incide en el espejo primario y se refleja en el espejo secundario. A continuación, se refleja desde el espejo secundario y es enviada al defector [14].

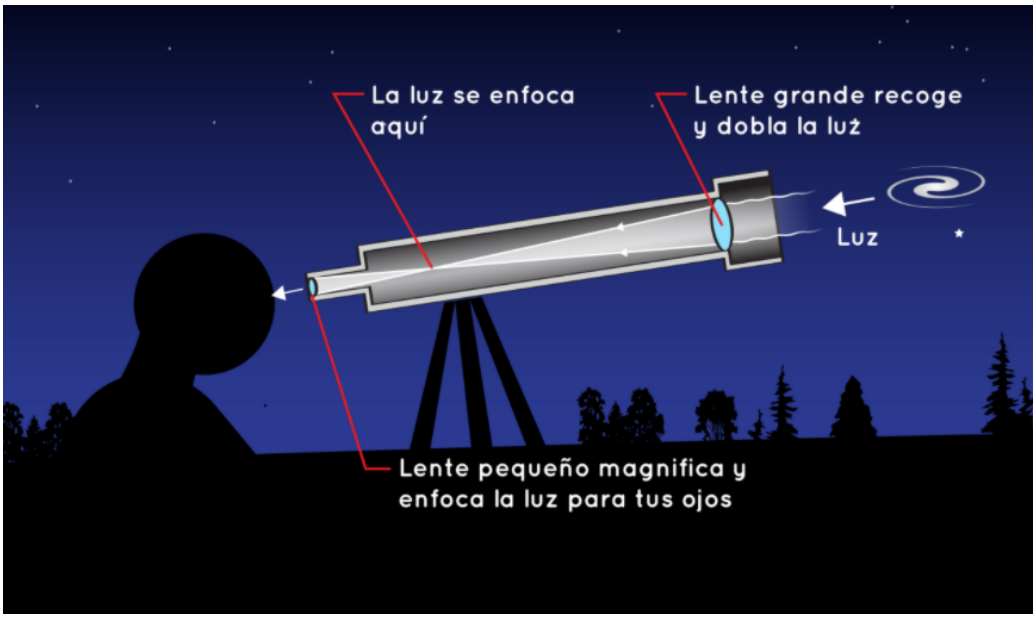


Figura 1.1: Un simple telescopio refractor utiliza lentes para hacer imágenes visibles y más grandes [16]

La expresión de la resolución de un telescopio es:

$$(\Delta\phi)_{min} = 1,22\lambda/D \text{ [25]}$$

donde λ es la longitud de onda y D es el diámetro del telescopio.

Podemos observar que es inversamente proporcional al diámetro del espejo primario, por lo que cuanto mayor es su diámetro, mayor es el poder resolutivo, siendo esta una razón por la que cada vez se buscan telescopios más grandes.

El problema que existe es que la atmósfera terrestre produce una distorsión en la imagen que llega al telescopio, por lo que la imagen de un punto (estrella) pasa a ser una mancha.

Posibles soluciones podrían ser enviar el telescopio a órbita, como en el caso del *Hubble* [27], donde se dispondría un telescopio más allá de la distorsión que produce la atmósfera terrestre eliminando de esta manera los efectos creados por la turbulencia durante la observación de las estrellas. También se puede realizar posteriormente un procesado de la captura final o una corrección de la imagen incidente en tiempo real.

Dentro de esta última solución se engloba lo que se conoce como Óptica Adaptativa, que se consigue acoplando al sistema un elemento activo, siendo comúnmente un espejo deformable [35], esto es un de espejo cuya superficie se puede deformar, con el fin de recrear el frente de ondas y corregir de esta forma las aberraciones producidas por la atmósfera, y es lo que utiliza el proyecto ALIOLI [13] del IAC.

El instrumento utilizado en la AO (*Adaptative Optic*) tendría el siguiente esquema.

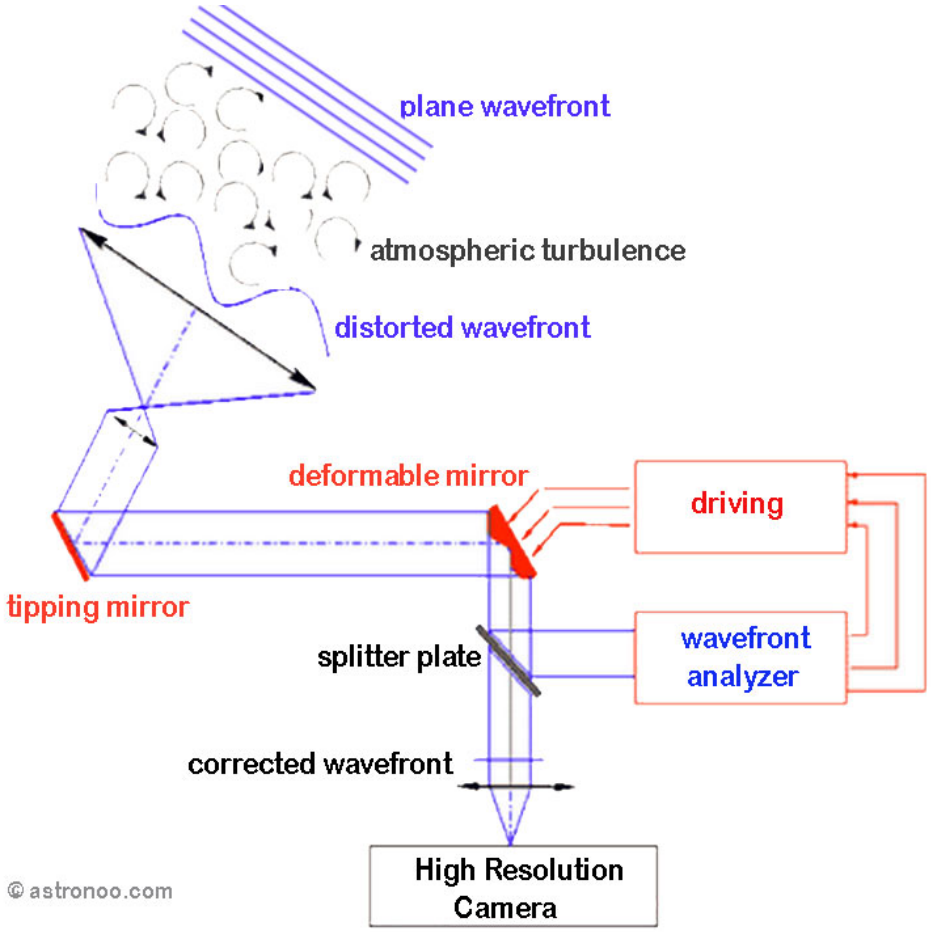


Figura 1.2: Instrumento de un Sistema de Óptica Adaptativa [15]

El método de reconstrucción utilizado para el frente de onda es el método de Shack-Hartmann [28], según el cual el frente de onda es muestreado por un *array* de microlentes, es decir, se divide la pupila del telescopio (la luz que recibe) en subaperturas.

Si el frente de ondas incidente es plano, cada una de las microlentes mandará la luz incidente a su foco, en cambio, si el haz incidente no es plano, es decir, entra con un ángulo en cada una de las subaperturas, ya no formarán imagen en su foco, y el foco de luz aparecerá desplazado de su posición ideal, siendo este desplazamiento proporcional al ángulo de incidencia.

Hoy en día, la construcción de grandes telescopios requiere un desarrollo muy costoso, dejando atrás la atención a un gran número de telescopios más pequeños, pero siendo aún así cruciales para el descubrimiento y la observación de las estrellas. Por ello, el IAC ha emprendido el proyecto de un instrumento ALIOLI *(Adaptive and Lucky Imaging Optics Lightweight Instrument)*, para aplicar y estudiar estas técnicas en telescopios de clase intermedia (entre 1,5 y 2,5 m).

1.1. Motivación y justificación del proyecto

El campo de la astronomía es de vital importancia debido a que un gran número de fenómenos astronómicos influyen en la vida cotidiana aunque a menudo pasan de forma desapercibida. Por ello, es necesario comprender las nociones básicas de astronomía referentes a las estrellas, el sistema solar y los movimientos de la tierra.

El proyecto del IAC se encarga de visualizar imágenes capturadas durante una noche de observación y almacenar imágenes de ciencia procesadas con este sistema óptico, tanto en tiempo real como de forma estática, para llevar un control de las capturas que se están realizando, comprobando así el buen funcionamiento de un sistema de óptica adaptativa.

Este trabajo trata de generar una interfaz de usuario que aporte la máxima información de un modo claro y ordenado buscando así la simplicidad en su manejo.

1.2. Antecedentes y estado actual del tema

Anteriormente se ha realizado un proyecto similar en esta empresa que englobaba muchas más funcionalidades a parte de este Trabajo de Fin de Grado (TFG). Por ello, se tienen versiones de desarrollo previas en C++ [32] junto a simulaciones en MATLAB [6], pero es un programa monolítico y poco versátil, por lo que ahora se pretende unificarlas e implementarlas sobre un *software* modular basado en python con módulos específicos en C++ para los procesos que requieren tiempo real a la hora de cerrar un lazo entre sensado y corrección del frente de ondas. En el caso de este trabajo se ha creado uno de los módulos basado en python.

En el proyecto anterior, se llegó a implementar la generación de algunas gráficas comunes a esta aplicación, pero en mi caso el código era inservible debido a que estaba todo realizado desde cero y en C++, lenguaje que no se usará en ningún momento durante el desarrollo.

Debido a esto todos los objetivos del proyecto, tanto la interfaz gráfica de la aplicación, como la introducción de datos y la generación de gráficas, se han cumplido desde cero sin reutilizar ningún código implementado anteriormente.

1.3. Objetivos

El objetivo principal del proyecto es implementar el código para la representación de gráficos e imágenes del proceso de control y procesado de los parámetros que permiten verificar el funcionamiento del sistema de óptica adaptativa para obtener el cierre del lazo de actuación. Para poder llevarlo a cabo se ha dividido en diferentes tareas:

1. Estudiar el proyecto y lo que se quiere llevar a cabo. Esto es debido a que al ser un TFG propuesto por una empresa hay que dedicar tiempo al entendimiento del problema y el material que viene dado. Además de conseguir entender lo que el cliente quiere con las prestaciones solicitadas.
2. Graficado.
    - Mostrar la imagen capturada por la cámara del sensor del frente de ondas [36].
    - Mostrar la posición de los actuadores del espejo deformable.
    - Mostrar las desviaciones de los centroides mediante un diagrama de fases.
    - Mostrar las dos imágenes anteriores superpuestas para poder comparar la geometría del frente de ondas ideal con el real.
    - Mostrar las regiones de influencia válidas en donde caen los centroides.
    - Mostrar el histograma de los coeficientes de Zernike.
    - Mostrar el frente de ondas reconstruido en 2D.
    - Mostrar el frente de ondas reconstruido en 3D.
3. Introducción de datos estáticos mediante la interfaz gráfica.
4. Diseñar y crear una librería con interfaz gráfica en la que se introduzcan los datos tanto de forma estática como en tiempo real y muestre dichas gráficas.
5. Introducción de datos en tiempo real.

1.4. Fases de desarrollo

El desarrollo del presente proyecto se ha agrupado en diferentes fases. La primera se dedica a entender el funcionamiento de un sistema de óptica adaptativa para establecer un diseño inicial de la aplicación y de las funcionalidades requeridas.

La siguiente fase es en la que se desarrollan los prototipos de la aplicación y se implementan las diferentes funcionalidades para graficar los datos obtenidos tanto de forma estática como a tiempo real.

Por último, se realizará la fase de prueba de la aplicación en condiciones reales en los sistemas del IAC para poder arreglar los posibles errores y añadir o modificar funcionalidades solicitadas por la empresa.

1.5. Estructura de la memoria

Este documento se encuentra estructurado en ocho capítulos. El primero contiene la introducción al trabajo en el que se explica la motivación y la justificación del proyecto, se describen los antedecentes y el estado actual del mismo, se establecen los objetivos y las fases de desarrollo que se han llevado a cabo para finalizar dicho proyecto.

El segundo capítulo describe las herramientas y tecnologías que se han utilizado para realizar el trabajo propuesto. Estas herramientas van desde el entorno de desarrollo hasta el control de versiones que se ha utilizado.

El tercer capítulo presenta el proceso que se ha realizado para poder comprender el proyecto en profundidad e implementar todas las funcionalidades requeridas por el cliente sin que se genere ningún problema de comunicación.

El cuarto y quinto capítulo pretenden dar una descripción detallada de cómo se ha desarrollado la aplicación tanto a nivel teórico como las partes más relevantes de la implementación y estructura del código.

El sexto y séptimo capítulo son muy importantes para poder concluir que se han cumplido los requisitos más relevantes, entre ellos, la introducción de datos a tiempo real desde otra aplicación. También se comenta de la visión del futuro de este proyecto y de posibles mejoras.

El último capítulo trata del presupuesto del proyecto.

Capítulo 2

Herramientas y tecnologías de desarrollo

En el capítulo anterior se ha introducido la justificación del proyecto, los antecedentes y el estado actual del tema, los objetivos y las fases de desarrollo para llevarlo a cabo.

Este capítulo se centra en describir las herramientas y tecnologías que se han utilizado para realizar este trabajo.

Se ha utilizado Visual Studio Code [11] como entorno de desarrollo de la aplicación. Además, para la interfaz gráfica se ha utilizado QtCreator junto con Python 3.8 [9] para desarrollar las funcionalidades necesarias. Por último, se ha utilizado el control de versiones de Git [5].

- **Visual Studio Code**

  Es un editor de código fuente potente que se ejecuta en el escritorio,lel de código abierto y gratuito que ha sido desarrollado por Microsoft.

- **QtCreator 4.1.14**

  Qt Creator es un IDE [1] multiplataforma programado para el desarrollo de aplicaciones con Interfaces Gráficas de Usuario (GUI) con las bibliotecas Qt. Los sistemas operativos que soporta son Linux, MacOs y Windows.

- **Python 3.8**

  Python es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos de alto nivel eficientes y un simple pero efectivo sistema de programación orientado a objetos.

  Junto lo anterior, este lenguaje cuenta con una librería de generación de gráficos llamada matplotlib [7], facilitando así el objetivo de este proyecto.

- **Git**

  Git es un *software* de control de versiones eficiente, confiable y compatible con el mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

- **Github**

  GitHub es una forja para alojar proyectos utilizando el sistema de control de versiones Git.

Capítulo 3

Análisis de requisitos y de la aplicación

En este capítulo se va a explicar el análisis inicial de los requerimientos de la aplicación que se va a desarrollar para así poder reflejar de forma detallada las necesidades de los usuarios en este proyecto de la mejor manera posible. Esto se ha logrado principalmente a partir de múltiples reuniones y estrategias de planificación y organización.

La primera reunión fue para conocer al cliente y tener una primera toma de contacto con la idea general que se quería desarrollar sin entrar mucho en detalle. Debido a que este TFG se ha realizado en un campo de la ciencia, la óptica [12], en el que no se tenía conocimiento previo se explicó a un nivel muy básico el funcionamiento del sistema de óptica que utilizan y su instrumentación.

Las siguientes reuniones se hicieron con el fin de realizar una descomposición funcional, identificando así los distintos componentes que englobarán este proyecto y las relaciones entre ellos. De esta forma, podremos realizar un sistema con funcionalidades independientes que pueden reusarse y reemplazarse fácilmente a lo largo del tiempo. En este caso, se solicitó crear una aplicación que generase determinadas gráficas a partir de sus respectivos parámetros introducidos tanto de forma estática mediante la interfaz como a tiempo real enviados por una aplicación secundaria. Para apoyar este proceso se definieron todas las funcionalidades en forma detallada de la aplicación de manera desglosada y en una lista para tener una organización y una buena visualización de como se relacionan entre ellas.

Una vez creado el plan del proyecto se procedió a crear un diseño previo de la interfaz gráfica que cumpla con todo lo solicitado y que a su vez sea simple, minimalista e intuitiva. En este caso se dibujó a mano la distribución de la aplicación decidiendo agrupar las funcionalidades que generan las gráficas a partir de datos introducidos mediante la interfaz en una pestaña y en otra diferente el grupo que realiza la toma de datos y la visualización de gráficas a tiempo real.

Dentro de la componente de datos estáticos se dividió la ventana en secciones según el tipo de parámetro que genere la imagen y en cada una se ellas de añadió un botón, por funcionalidad individual, cuyo nombre representa la acción a realizar que cuente con información extra, sólo si fuera necesario para no romper la simplicidad. Los botones se basan en cargar datos, mostrar la gráfica en una pestaña emergente y en guardar la imagen.

Por el otro lado, se encuentra la pantalla para la toma de datos a tiempo real en la que solo se muestran los botones que generan su respectivas gráficas y otra sección, dentro de la misma interfaz de usuario, que muestra la gráfica manteniendo un diseño sencillo.

Terminado todo este proceso inicial, se empezó a programar y crear la aplicación implementando el código necesario para realizar las tareas descritas con anterioridad y algunas que surjan durante el proceso que no se hayan tenido en cuenta. Intercaladamente, se realizan reuniones de seguimiento para aclarar dudas y enseñar la forma que va tomando el proyecto para modificar o incluso añadir nuevas funcionalidades solicitadas por el cliente con el fin de conseguir la mayor satisfacción de éste durante y después del desarrollo de la aplicación.

Por último, se realizó un proceso de prueba de uso por parte de ellos en situaciones que simulan casos reales para comprobar el buen funcionamiento y cerrar definitivamente el desarrollo proyecto.

Capítulo 4

Desarrollo

En este apartado se va a explicar el desarrollo todas de las funcionalidades de la aplicación escritas en python. Primero, se van a explicar las dos formas de introducción de datos. Luego se comentarán los distintos gráficos e imágenes del proceso de control que permiten verificar el funcionamiento de un sistema de óptica adaptativa para obtener el cierre del lazo de actuación y por último se explicará la interfaz gráfica y su funcionamiento.

4.1. Introducción de datos

Para representar dichas gráficas se necesitan ciertos parámetros calculados a partir de las imágenes capturadas por el telescopio mediante el proceso de Sistema de Óptica Adaptativa. Dichos datos usados son los siguientes.

- **Imagen de la cámara sensado**

  Imagen tomada de la cantidad de luz que entra por cada microlente del telescopio.

- **Coordenada X**

  Coordenadas en el eje X de los centroides de cada una de las microlentes leídos por el WFS *(Wavefront Sensor)* para la imagen de referencia.

- **Coordenada Y**

  Coordenadas en el eje Y de los centroides de cada una de las microlentes leídos por el WFS para la imagen de referencia.

- **Dx**

  Diferencia entre el centroide medido por el WFS durante una iteración respecto al medido para la imagen de referencia en el eje X.

- **Dy**

  Diferencia entre el centroide medido por el WFS durante una iteración respecto al medido para la imagen de referencia en el eje Y.

- **Coeficientes de Zernike**

  Los coeficientes de Zernike son un conjunto de polinomios que forman una base ortogonal, de forma que cualquier frente de onda se puede descomponer en una

suma de estos polinomios con determinados coeficientes. En este caso se utilizan para reconstruir el frente de ondas y visualizarlo.

- **Radio para mostrar la región de influencia de un centroide**

  La proyección sobre el detector del tamaño de cada microlente.

4.1.1. Introducción de datos estáticos

La primera forma de introducir los datos en la aplicación ha sido de forma estática, es decir, el usuario carga los datos a representar desde la interfaz gráfica.

El módulo desarrollado permite importar la imagen de la cámara de sensado compatible con los formatos FITS *(Flexible Image Transport System)* [3], PNG o JPG. Por otro lado, el resto de datos se introducen mediante ficheros normales tipo TXT los cuales cada línea representa la posición equivalente con el vector que se va a almacenar dicha información en el código.

Para trabajar en python con el formato FITS se ha utilizado la librería astropy.io.fits [4], la cual ofrece un acceso a ficheros con este formato. Una vez cargado el archivo en python, usamos la función *open* dada en esta librería obteniendo la información de la imagen en la primera posición del vector devuelto. El resto de datos dados para este caso son irrelevantes.

4.1.2. Introducción de datos a tiempo real

En este caso los datos son introducidos a tiempo real, es decir, mientras se van capturando las imágenes durante el proceso del lazo de actuación se van enviando a la aplicación para que vaya mostrando de forma continua la distinta información que le llega.

La obtención y el procesado de datos tiene que ser en tiempo muy pequeño debido a que el frente de ondas cambia cada treinta milisegundos por lo que habría que recalcular todo, pero en el caso del graficado se puede tener un ligero retraso en su visualización debido a que no es tan relevante su rapidez.

Paso de mensajes

Esta aplicación recibirá los datos necesarios para generar las gráficas mediante la técnica de paso de mensajes, en la cual la comunicación se hace mediante operaciones explícitas y de recepción. Ésta es una alternativa bastante buena y eficiente comparada al modelo de memoria compartida debido a la complejidad de su implementación en este proyecto.

Otras de las ventajas de este modelo es que es válido para cualquier arquitectura de computadores, tanto sistemas distribuidos como en arquitecturas paralelas sin memoria compartida o en sistemas de memoria compartida. Tampoco existiría problema universal del acceso en exclusión mutua a datos compartidos.

Dentro del código, esto se ha realizado con la librería de mensajes ZeroMQ [37], la cual posibilita crear complejos sistemas de comunicación basado en *sockets* [34] que permiten

comunicar procesos entre máquinas usando un sistema de colas con poco esfuerzo, siendo a su vez muy rápido.

En nuestro caso, la aplicación será el cliente que se conecte al servidor. Primero, se crea la instancia del contexto y a partir de este el *socket* se conecta al servidor especificando su dirección IP[20] y un puerto.

El tipo de transporte utilizado es el TCP *(Transmission Control Protocol)*[19] permitiendo así que dos *hosts* se conecten e intercambien flujos de datos. Este protocolo garantiza que los paquetes se entreguen de forma confiable y sin errores.

También se ha usado el patrón de diseño Pub/Sub [17] que proporciona mensajería asíncrona, confiable y comunicación de varios a varios entre aplicaciones. Las aplicaciones del publicador pueden enviar mensajes en un tema y las otras aplicaciones pueden suscribirse a este tema para recibir los mensajes.

En el siguiente fragmento de código se puede observar como se realiza todo lo explicado anteriormente de forma muy sencilla.

```
context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.connect("tcp://XXX.XXX.XXX.XXX:PPPP")
socket.subscribe("")
```

Serialización

Una vez configurado para compartir los datos entre diferentes lenguajes, hay que globalizar los tipos de datos para poder ser usados en ambos lados sin problemas de compatibilidad. Para ello se han serializado, siendo esto el proceso de codificación de un objeto en un medio de almacenamiento con el fin de transmitirlo a través de una conexión de red.

En este caso se ha utilizado *Protocol Buffers (Protobuf)* [21] el cual permite a las aplicaciones almacenar e intercambiar datos estructurados fácilmente, incluso si los programas están en lenguajes diferentes.

Para utilizarlo definimos nuestro propio formato de datos en un archivo de formato proto, el cual es el archivo de configuración estándar de este formato de serialización. Dentro de él, se describen todas las estructuras de datos que se deseen implementar. Por cada estructura que se quiera serializar solo se tiene que añadir un mensaje, especificar los nombres y tipos de cada campo de este mensaje y añadir el modificador o modificadores que se quiera.

El archivo proto de esta aplicación tiene la siguiente estructura.

```
syntax = "proto3";

package SerializableData;

message Image {
    uint32 height = 1;
    uint32 width = 2;
    bytes data = 3;
}
```

```
message Delta {
    sint32 X = 1;
    sint32 Y = 2;
}

message Centroid {
    uint32 X = 1;
    uint32 Y = 2;
}

message Data {
    Image Imagen = 1;
    repeated Centroid Centroids = 2;
    repeated Delta Deltas = 3;
    repeated double Zernikes = 4;
    repeated uint32 Radius = 5;
}
```

Como se puede observar en lo anterior, en primer lugar se especifica la versión correspondiente de Protobuf, en este caso, proto3, seguida de la descripción del paquete de *software* cuyo datos se desean estructurar.

A continuación, figuran los mensajes de Protobuf, que pueden estar compuestos por un número ilimitado de campos. Entre ellos, están los datos típicos, como bool, int32, float, etc. Además, a los campos que son vectores se les ha añadido el modificador *repeated*, el cual permite repetir los datos un número indefinido de veces.

Una vez definidas las estructuras de datos, hay que generar las clases necesarias para la lectura y escritura de estos mensajes. Para ello, se compila el archivo de configuración con el compilador Protoc teniendo que especificar el directorio fuente, el destino y la ruta del archivo. Como en este caso, hay que generar clases python, se utiliza siguiente el comando.

```
protoc -I=[DIR_FUENTE] --python_out=[DIR_DESTINO] [RUTA/FICHERO/a.proto]
```

Por último, dentro de nuestro código, incluimos la clase generada obteniendo así los datos enviados.

```
# Clase generada en python con Protoc con
# la estructura de datos comun al servidor
import message_pb2

# Mientras siga leyendo datos
while not self.stop_reading:
  self.rcv_data = False

  # Obtenemos el mensaje enviado
  # por el servidor si lo hay
  message = socket.recv()

  # Decodificamos el mensaje obtenido
  # a nuestra estructura de datos
  self.rt_data = message_pb2.Data().FromString(message)

  self.rcv_data = True
```

Simulación

Para comprobar que la introducción de datos a tiempo real funciona correctamente se ha simulado todo el proceso mediante un programa creado en python que genera datos simulando las condiciones reales y se han realizado los pasos anteriores pero configurándolo como el servidor.

```
context = zmq.Context()
socket = context.socket(zmq.PUB)
socket.bind("tcp://*:PPPP")
```

Podemos observar que en el caso del servidor el *socket* se crea como publicador.

Por último, se generan los datos simulados y se almacenan en la clase generada con Protobuf y se envían.

```
message = message_pb2.Data()
message.Imagen.data = img.tobytes()
message.Imagen.height = img.height
message.Imagen.width = img.width
message.Zernikes.extend(W)
message.Radius.extend([1, 2, 3, 4, 5])

for i in range(len(star_refx)):
    c = message_pb2.Centroid()
    c.X = star_refx[i]
    c.Y = star_refy[i]
    message.Centroids.append(c)

dx = slopes[:(len(star_refx))]
dy = slopes[(len(star_refx)):]
for i in range(len(dx)):
    d = message_pb2.Delta()
    d.X = dx[i]
    d.Y = dy[i]
    message.Deltas.append(d)

socket.send(message.SerializeToString())
```

4.2. Graficado

Se han generado un total de ocho gráficas distintas, cada una con una representación y significado diferente, cuyo conjunto cumple con el objetivo de visualizar el funcionamiento de un sistema de Óptica Adaptativa.

4.2.1. Imagen cámara de sensado

La gráfica de la imagen de sensado es la luz incidente sobre el vector de microlentes.

El objetivo de visualizar esta gráfica es comprobar que la luz está llegando correctamente a la cámara de sensado. Cualquier desalineamiento del sistema óptico provocaría un viñeteo sobre el sensor de frente de onda y la reconstrucción no se llevaría a cabo de manera satisfactoria.

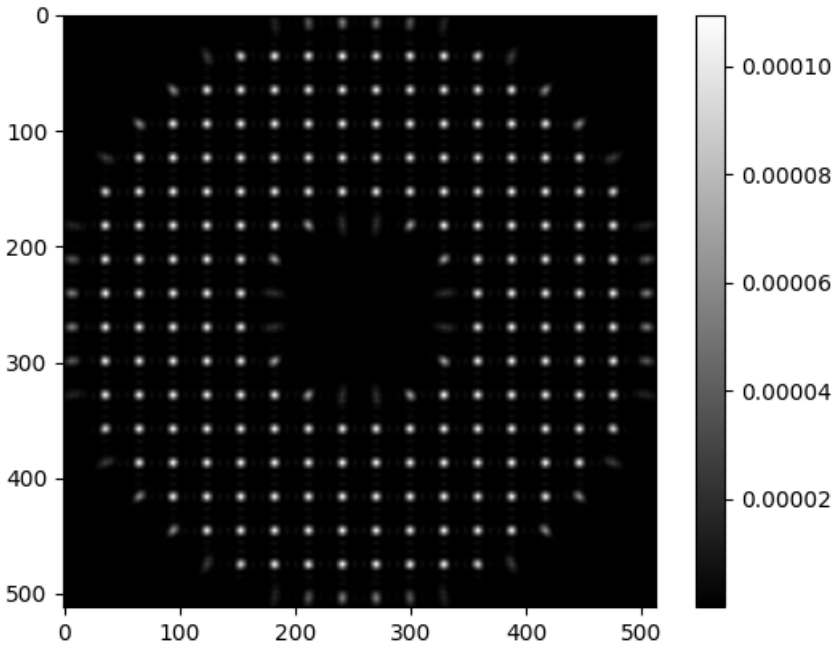


Figura 4.1: Ejemplo de una imagen capturada por la cámara se sensado

4.2.2. Desviaciones de los centroides mediante un diagrama de fases

Esta gráfica representa las desviaciones de los centroides de las microlentes con respecto a la imagen de referencia. Con ella podemos observar el movimiento de los actuadores del espejo deformable.

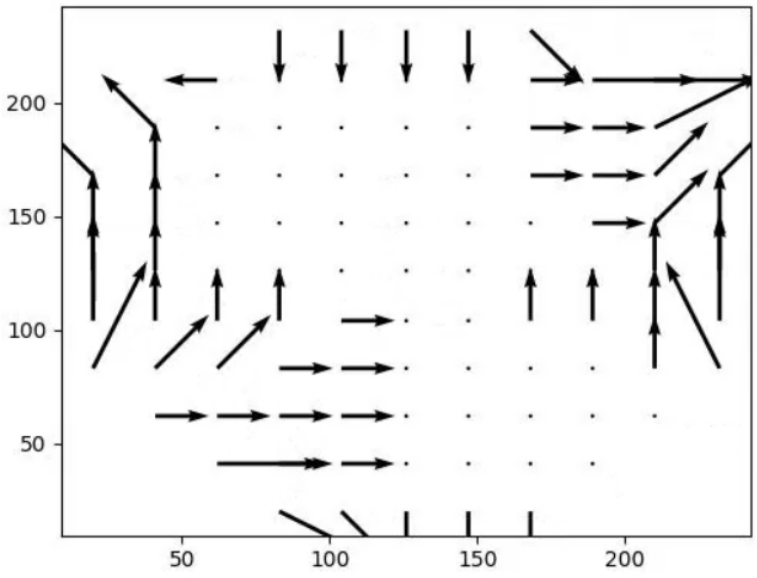


Figura 4.2: Ejemplo de las desviaciones de los centroides

4.2.3. Superposición de las dos gráficas anteriores

Realizando la superposición de las imagen de la cámara de sensado junto al diagrama de fases de las desviaciones de los centroides de las microlentes permite al usuario conocer qué efecto tienen los cambios producidos a lo largo del camino óptico sobre la imagen a procesar.

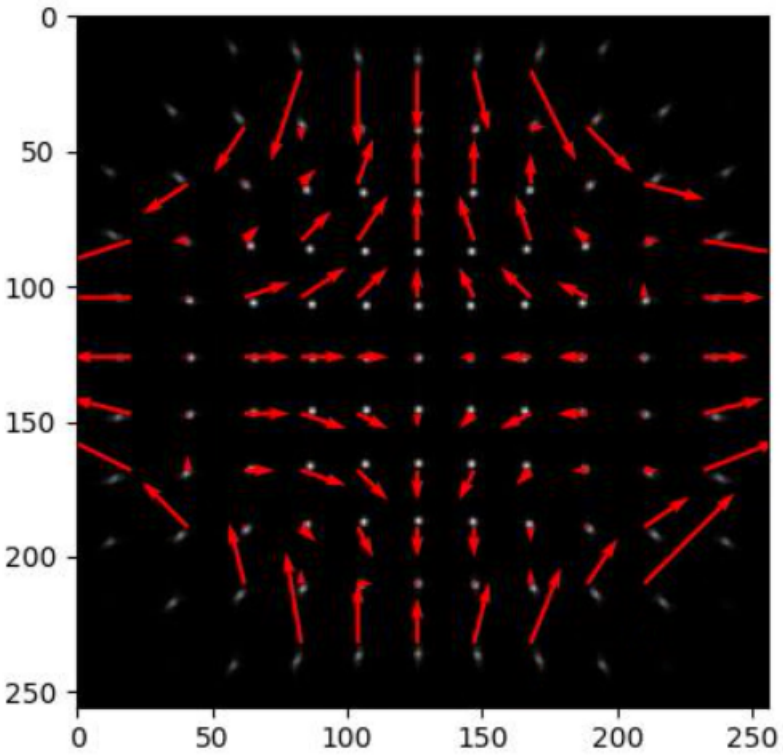


Figura 4.3: Ejemplo de la superposición de la imagen de sensado junto a las desviaciones

4.2.4. Regiones válidas en donde caen los centroides

Esta gráfica visualiza una máscara que se le añade a la imagen de sensado siendo esta la zona de influencia de cada microlente en donde los valores de los centroides pueden ser válidos.

Con ella podremos saber si hemos perdido el control sobre el deformable, ya que los *spots* de algunas microlentes se saldrían de su zona de influencia, y por tanto las medidas para la reconstrucción estarían falseadas.

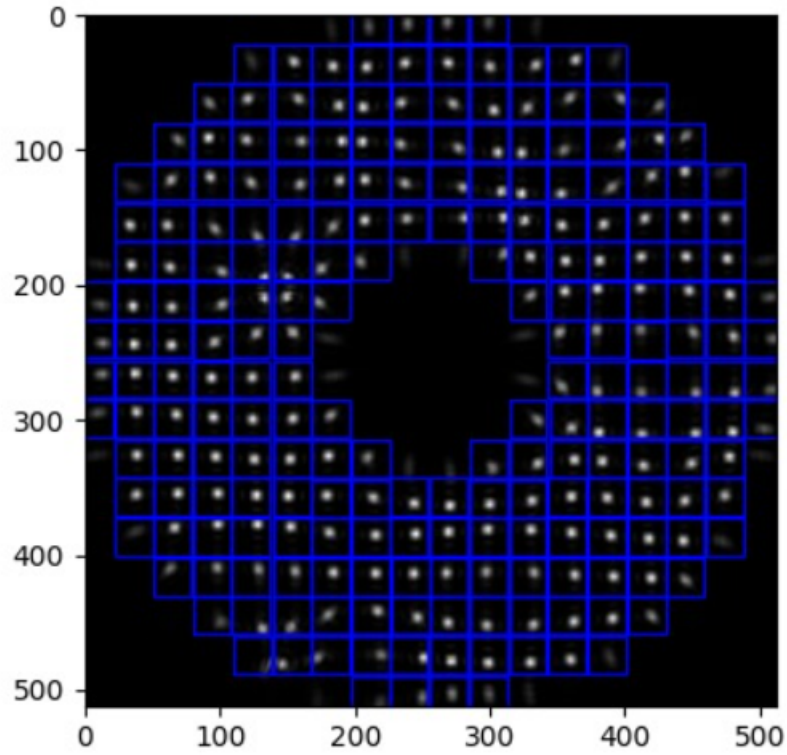


Figura 4.4: Ejemplo de la región de influencia de los centroides

4.2.5. Histograma de los coeficientes de Zernike

Visualizamos el peso de los coeficientes de Zernike mediante un histograma de barras para ver como varían a lo largo del tiempo. La utilidad de esta gráfica cobrará importancia cuando el lazo de control se establezca, ya que comprobaremos como la lectura de estos valores tenderá a cero.

4.2.6. Frente de Ondas reconstruido en 2D y 3D

Para reconstruir el Frente de Ondas se realiza con los polinomios de Zernike con las siguientes operaciones matemáticas.

En general, los polinomios de Zernike están definidos como:

$$Z_n^m(\rho,\Theta) = \begin{cases} N_n^m R_n^{|m|}(\rho)\cos m\Theta & ;for \quad m \geq 0 \\ -N_n^m R_n^{|m|}(\rho)\sin m\Theta & ;for \quad m < 0 \end{cases}$$

[33]

donde N_n^m es el factor normalización descrito más en detalle abajo y $R_n^{|m|}(\rho)$ es dado por:

$$R_n^{|m|}(\rho) = \sum_{s=0}^{(n-|m|)/2} \frac{(-1)^s (n-s)!}{s![0{,}5(n+|m|)-s]!0{,}5(n-|m|)-s]!}\rho^{n-2s} \quad [33]$$

La normalización es dada por:

$$N_n^m = \sqrt{\frac{2(n+1)}{1+\delta_{m0}}} \text{ [33]}$$

Una vez se crean las funciones necesarias para calcular los polinomios, es decir, Z_n^m, se comienza con la reconstrucción del frente de ondas mediante el método modal. Primero generamos una matriz entre 1 y -1, cuyo tamaño puede variar, la cual contendrá las coordenadas de referencia.

Con los componentes X y con los componentes Y de dicha matriz se calcula Θ y ρ que serán las coordenadas polares.

Luego se genera una máscara lógica que marca la región donde está la pupila (una circunferencia). Ésta será una matriz cuadrada con las dimensiones de X, con ceros en los puntos en los que R_n^m es mayor a uno y unos en los que es menor o igual.

Lo siguiente es generar los polinomios de Zernike con los que se quiere trabajar, el número depende de la configuración del módulo de sensado y por lo tanto del tamaño del vector de pesos que se tiene en un inicio. Por lo que se itera tantas veces el tamaño del vector de pesos, calculando en cada iteración el valor de m y n a partir del número de iteración en el que esté. Después se calcula el polinomio con las fórmulas comentadas anteriormente, le aplicamos la máscara y almacenamos el polinomio en un vector.

Una vez calculado todos los polinomios de Zernike con los que se está trabajando, se realiza la reconstrucción multiplicando cada polinomio por el peso dado y obteniendo así la expresión que define la superficie.

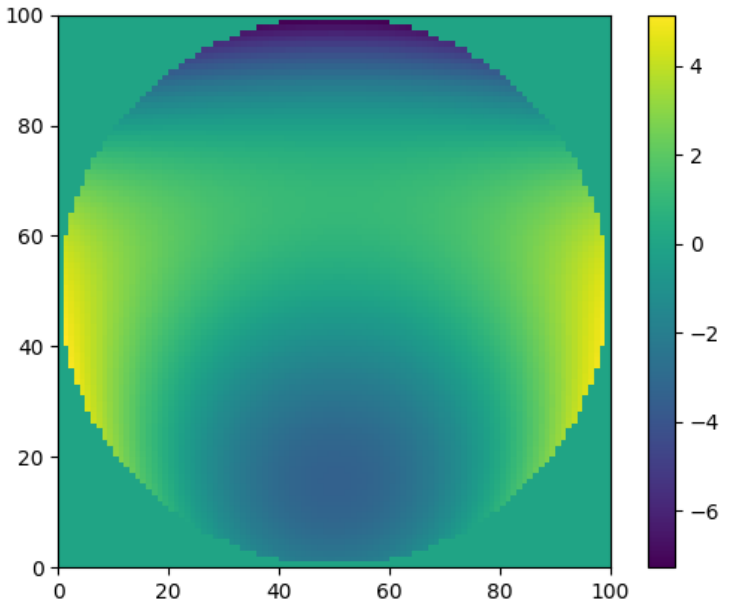


Figura 4.5: Ejemplo del Frente de Ondas reconstruido en 2D

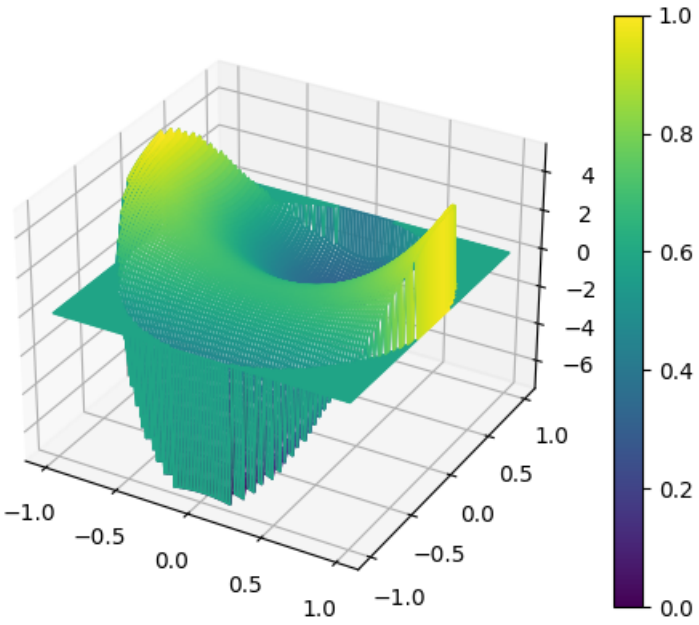


Figura 4.6: Ejemplo del Frente de Ondas reconstruido en 3D

4.2.7. Gráfica Spot

En esta gráfica lo que se representa son los valores de los Dx frente a los Dy para cada una de las microlentes, dándonos información de la dispersión de estos valores. Esta representación tiene su correspondencia con la imagen que se ve en la cámara de ciencia.

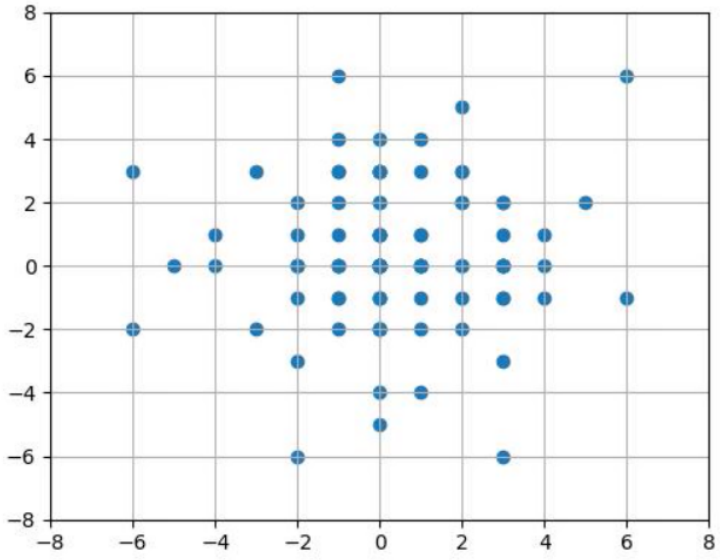


Figura 4.7: Ejemplo de la gráfica spot

4.3. Interfaz Gráfica

El diseño gráfico se ha desarrollado con QtCreator, la cual ofrece una guía visual, sencilla y cómoda para crear la interfaz de la aplicación.

Se ha dividido la aplicación en dos pestañas para separar las diferentes maneras de introducir los datos. La primera tiene como título *Static* y como su nombre indica es la que se encarga de mostrar los datos introducidos por el usuario a través de un archivo con un determinado formato dependiendo de la gráfica que se quiera visualizar, surgiendo una ventana emergente para poder tener varias abiertas a la vez.

Se ha buscado un diseño claro, minimalista e intuitivo para conseguir el mayor entendimiento del usuario y que sea fácil de usar, incluyendo personas que la utilicen por primera vez.

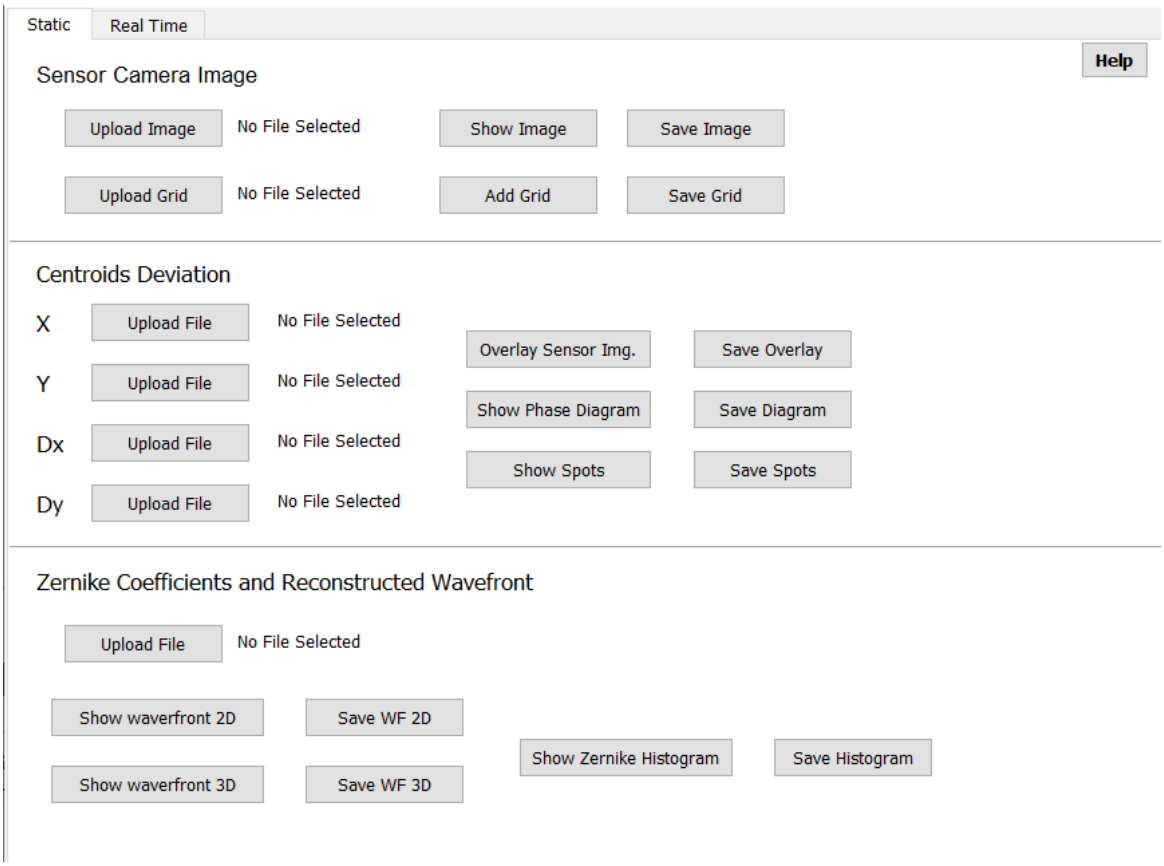


Figura 4.8: Ventana del módulo de generación de gráficas con datos introducidos por el usuario

En la imagen anterior, podemos observar que la pestaña se ha dividido en tres zonas con el fin de agrupar las funcionalidades según el dato que se le introduce. En el primer apartado se puede subir la imagen de sensado y el vector de distancias por cada microlente desde el centroide hasta el límite de la región de influencia. Con ello, al pulsar los botones respectivos a cada gráfica se muestran la figura 4.1 o la figura 4.4.

En la segunda división se introducen las posiciones de los centroides de cada microlente junto a sus desviaciones. Se puede seleccionar la visualización de las gráficas de la figura 4.3, 4.2 y la 4.7.

La última sección carga el vector de polinomios de Zernike y muestra las gráficas de

las figuras 4.5, 4.6 o el histograma que visualiza sus pesos.

Cada vez que se quiera cargar un fichero se abre el explorador de archivos para elegir el documento deseado de forma cómoda. Una vez abierto, se muestra su nombre en la interfaz al lado de su respectivo botón.

Además, existe la posibilidad de almacenar la imagen de forma local usando los botones de guardado de las correspondientes gráficas.

También se presentan otras funcionalidades para facilitar su uso, como un botón de ayuda en ambas pestañas para resolver posibles dudas de su uso. A parte, se controla el caso en el que no se tienen los datos necesarios para generar o guardar una gráfica mostrando una ventana emergente avisando de qué datos son necesarios para realizar la operación de que se desea.

Por otro lado, se encuentra la ventana con el módulo de visualización de gráficas a tiempo real, llamada *Real Time*. Solo contiene los diferentes botones de las gráficas a mostrar, ya que los datos se reciben desde otra aplicación de la forma en la que se ha explicado en el apartado 4.1.2, y una sección en donde se va a mostrar la imagen incrustada en la aplicación, en el caso de que no se reciban datos se mostrará una gráfica vacía.

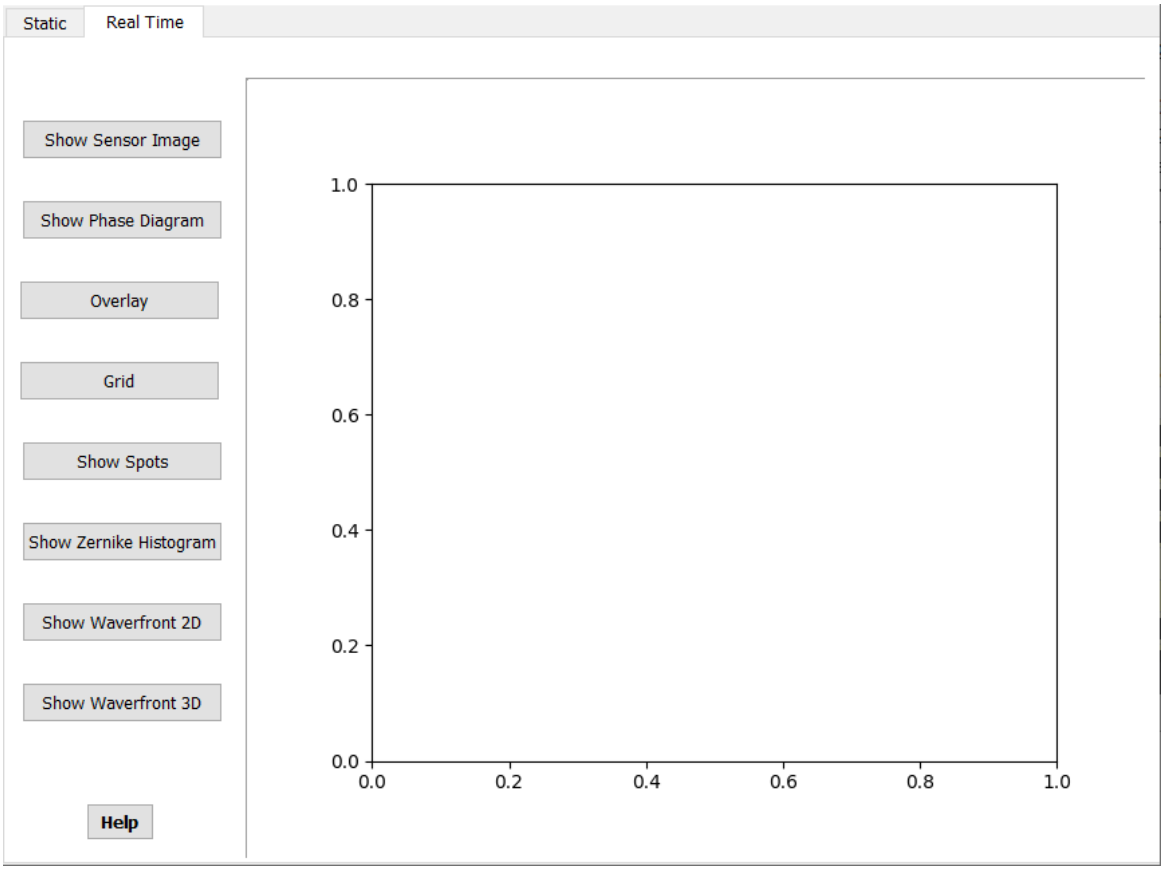


Figura 4.9: Ventana del módulo de generación de gráficas con datos introducidos a tiempo real

Capítulo 5

Estructura del proyecto

En el capítulo anterior se explicó el desarrollo de la aplicación a nivel teórico, por ello este capítulo se centra en indagar en la forma en la que se han unido las distintas herramientas utilizadas y de como se ha estructurado el proyecto y el código.

Todos los ficheros que componen el proyecto entero se encuentran englobados en una misma carpeta. Además del código que implementa las funcionalidades de la aplicación, se encuentran otros generados por QtCreator.

- **form.ui**

  Contiene todos los componentes de la interfaz, desde sus nombres, su tamaño hasta donde están posicionados. Al utilizar QtCreator este documento se modifica automáticamente usando su editor gráfico para crear el diseño con mayor facilidad.

- **main.pyproject**

  Contiene todos los ficheros que pertenecen al proyecto para que al abrirlo con el IDE (*Integrated Development Environment*) de Qt se carguen para poder modificarlos desde ahí.

- **main.pyproject.user**

  Contiene la configuración del proyecto en el IDE.

Para crear las funcionalidades de la aplicación, se ha dividido el proyecto en diferentes clases escritas en python para luego acoplarlas de tal forma que sea fácil cualquier modificación que se quiera hacer en un futuro.

- **MplCanvas** [24]

  Para poder utilizar la librería de generación de gráficas de python dentro de la aplicación se ha creado una clase con el fin de configurar un *canvas* que hereda de FigureCanvasQtAgg [2] y según lo que se le especifique se configura para crear una figura 2D o 3D.

  Además, contiene un método que almacena una gráfica especificando la ruta en donde se quiera guardar.

  En el caso de que se quiera añadir funcionalidades que engloban alguna configuración o funcionalidad relacionadas con el *canvas* o con la librería matplotlib simplemente habría que extender esta clase.

- **graphics.py**

  Este fichero contiene todas las funciones que generan las gráficas usando la clase explicada en el punto anterior. Dependiendo de lo que se quiera generar se le envía a la función con ninguno, uno o más parámetros junto a la opción de si se va a mostrar mediante una ventana emergente, en el caso de que se introduzcan los datos mediante el usuario, o si se quiere empotrar en la interfaz directamente.

  Todas las funciones para crear la figura siguen el mismo patrón, primero se crea una instancia de MplCanvas. Luego, mediante los atributos de ella se crea la gráfica llamando al respectivo método de matplotlib que se quiera usar para generar un determinado tipo de gráfica. Lo siguiente que se realiza, de forma opcional, es configurar la figura dependiendo de lo que se quiera, por ejemplo, añadir una barra de la paleta de colores de la gráfica, añadir cuadrículas o incluso limitar los ejes dentro de un rango entre otras muchas opciones. Por último, si se ha elegido la opción de mostrar de forma emergente, se llama a la función *show* que se encarga de generarla y mostrarla directamente, o si se quiere empotrar simplemente se devuelve la clase MplCanvas con la figura creada.

  Además, hay otras funciones que ayudan a realizar cálculos necesarios para generar las gráficas, por ejemplo para reconstruir el frente de ondas explicado en el sección 4.2.6.

- **StaticTab**

  Esta clase se encarga de conectar todos los botones de la pestaña de introducción de datos estáticos de la aplicación con los respectivos métodos que implementan dichas funcionalidades.

  Primero, se realiza la conexión de los botones para cargar los datos con sus respectivos métodos, luego se conectan los botones que muestran las gráficas y por último se conectan los botones que guardan las imágenes.

  Luego se implementan los métodos de la clase. Primero se encuentra el que almacena una imagen dada por parámetro para guardarla en local. Para ello se abre el explorador de archivos para seleccionar la carpeta y elegir el nombre que se desea y se llama al método de la clase MplCanvas comentado anteriormente para almacenarla.

  Después, se crea el método que declara un vector a partir de la lectura de un fichero separando cada valor por líneas y lo devuelve.

  A continuación, se encuentran botones que se encargan de cargar los datos, los cuales también siguen un patrón muy parecido. Primero, se abre el explorador de archivos para seleccionar qué imagen o archivo se quiere abrir y se almacena su ruta. Se comprueba si se ha llegado a seleccionar un fichero y en el caso afirmativo, se escribe su nombre en la interfaz y se recogen sus datos. En el caso de la imagen de sensado se comprueba si es formato FITS para coger los datos de la forma explicada en el apartado 4.1.1 o en el caso de que sea JPG o PNG se abre con la librería opencv [8]. En el caso de que sea un fichero con un vector se llama a la función nombrada anteriormente para leer su cotenido enviando la ruta del fichero.

Finalmente, si no ha ocurrido nada fuera de lo normal, se almacena ese dato en un atributo de la clase.

Por último, se encuentran los botones que muestran las gráficas en una pantalla emergente y que siguen el mismo patrón entre ellos. Primero de todo se comprueba si existen los datos necesarios para generar la gráfica, si esto no ocurre se abre una ventana con un mensaje que especifica los datos necesarios a introducir. En el caso contrario, se llama a la función correspondiente dentro del fichero *graphics.py*, que se explica en el apartado 5, y se muestra.

En este caso, resulta muy sencillo añadir, modificar o eliminar tipos de datos a introducir o gráficas a mostrar, ya que siguen el mismo patrón y es muy fácil de replicar. Además, está todo implementado de tal forma que cada funcionalidad sea lo más individual y simple posible para que no haya conflictos a la hora de realizar alguna de las acciones mencionadas.

- **RealTimeTab**

  Esta clase implementa todas las funcionalidades necesarias que se encuentran en la pestaña de *Real Time*. Como se ha mencionado anteriormente, esta pantalla tiene dos secciones, los botones para seleccionar la gráfica a mostrar y el *canvas* empotrado en la aplicación para pintarlas.

  Cada botón se conecta a su respectiva función, los cuales si son presionados cambian una variable la cual contiene el tipo de gráfica que se quiera mostrar. Para ello se han definido las siguientes constantes:

  ```
  SENSOR_CAMERA_IMAGE = 0
  CENTROIDS_DEVIATION = 1
  ZERNIKE_COEFF_HIST = 2
  WAVEFRONT_2D = 3
  WAVEFRONT_3D = 4
  OVERLAY = 5
  GRID = 6
  SPOTS = 7
  ```

  Por otro lado, se encuentra el método que se conectará al servidor e irá leyendo continuamente los datos cuando éste los envie. Luego se decodifican como ha sido explicado en el apartado 4.1.2 y se almacenan en un atributo de la clase para que pueda ser accedido por todos los métodos.

  La sección donde se pintarán las gráficas es un objeto del tipo QWidget [29] para poder mostrar las imagénes e ir cambiándolas continuamente. Para ello, se han implementado tres métodos auxiliares. El primero, limpia el *canvas* de las imágenes que se han mostrado anteriormente ya que la librería combina todas las gráficas abiertas que no se han cerrado con anterioridad.

  Luego se ha creado otra función que introduce un *delay* en el refresco de los datos de la gráfica, la cual se encarga de actualizar el *canvas* a la nueva imagen pausándola un momento. Por último se cierra la imagen para no consumir memoria almacenando todas las gráficas generadas. Sin esta función todas las imágenes generadas se abrirían al mismo tiempo cuando se haya terminado de recibir datos por temas internos de como está realizada la librería. Existe una función ya hecha dentro de

ella, pero el problema es que la forma en la que está realizada siempre mostrará una gráfica emergente al final y eso no es lo que queremos. Por lo que se ha realizado a mano.

El siguiente método se encarga de pintar la gráfica en el lienzo. Primero, crea un atributo de la clase del tipo FigureCanvas [23] a partir de la figura que se va a mostrar en ese momento. Luego, se añade al *layout* del objeto QWidget mencionado anteriormente la figura creada que se vaya a mostrar en ese momento.

Por último, se encuentran las funciones que generan las gráficas siguiendo un patrón muy sencillo entre ellas. Primero, se limpia el *canvas* por si hubiera alguna imagen anterior, luego dependiendo de lo que se quiera mostrar, se usan los datos necesarios para esa gráfica obtenidos de la estructura que se ha deserializado previamente y se llama a la función correspondiente del fichero explicado en el apartado 5, esta vez, indicando que no se quiere mostrar de forma emergente y devolviendo así solo figura generada. Finalmente, se llama al método que pinta la figura en el lienzo y se pausa por un periodo de tiempo antes de cerrarla para pasar a la siguiente.

Igual que se ha mencionado anteriormente, poder añadir una gráfica nueva, modificarla o eliminarla es muy sencillo simplemente siguiendo el mismo patrón que se hace para crear el resto.

- **MainWindow**

  Esta clase engloba toda la aplicación y la ejecuta. Primero, crea una instancia de las clases mencionadas anteriormente implementando así las funcionalidades de ambas ventanas desde la ventana principal.

  Luego, para evitar que la GUI *(Graphic User Interface)* se congele debido a estar ejecutando en el hilo principal un bucle infinito en el que está constantemente esperando a recibir datos y bloqueando así el resto de procesos, se envía a un hilo secundario el método de recibir datos a tiempo real mediante paso de mensajes de la ventana que implementa el módulo de tiempo real y lo ejecuta para estar siempre funcionando de fondo sin que dé ningún problema. De esta forma se puede alternar meter datos estáticos y a tiempo real sin inconvenientes.

Por último, fuera de cualquier clase se encuentra la función principal que declara la ventana principal con todos sus componentes y ejecuta la interfaz. En ella además, hasta que no se cierra la aplicación, tendrá un bucle dentro del hilo principal el cual, en el caso de se que se hayan recibido datos, llame al método seleccionado implementado en la clase que contiene el módulo de tiempo real para pintar la gráfica en el *canvas* en la ventana de *Real Time* y si es así, se comprueba qué modo se ha seleccionado, es decir, si se ha pulsado un botón para mostrar cierta gráfica. En el caso de que ocurra, se llama al método que pinta la imagen en la aplicación. Si no se ha recibido ningún dato o no se ha seleccionado ningún modo se muestra una gráfica vacía.

Capítulo 6

Conclusiones y líneas futuras

Para concluir, resaltar el gran aprendizaje que ha conllevado la realización de este proyecto, utilizando nuevas herramientas de las que no tenía conocimiento anteriormente. Además, lograr alcanzar las metas establecidas a medida de que iba aprendiendo ha sido bastante satisfactorio y ha generado un crecimiento académico notorio.

Los objetivos generales propuestos inicialmente se han cumplido, desarrollando así una aplicación que genera todas las gráficas solicitadas en un principio, e incluso algunas pedidas posteriormente, introduciendo los datos tanto por el usuario desde la interfaz gráfica como por otra aplicación a tiempo real. Incluso se han añadido más funcionalidades de las esperadas en los objetivos inciales, como por ejemplo, guardar las imágenes de las gráficas.

Sin embargo, hay que tener en cuenta ciertas mejoras que se podrían realizar en un futuro. Estas mejoras serían por ejemplo la capacidad de conectar o cambiar de servidor de forma dinámica mientras se ejecuta la aplicación o funcionalidades extras para facilitar y mejorar la monitorización.

Por lo tanto, al cumplir todos los objetivos establecidos, esta aplicación puede ser útil para el IAC pudiendo utilizarse en situaciones reales durante una noche de observación. Además, cuenta con un diseño claro, sencillo y minimalista, con documentación incluida en la interfaz por si surgieran posibles dudas en el momento e incluye las distintas funcionalidades mencionadas en capítulos anteriores.

Capítulo 7

Summary and Conclusions

To conclude, highlight the great learning that the realization of this project has entailed, using new tools of which I was not previously aware. In addition, achieving the goals established as I was learning has been quite satisfactory and has generated notable academic growth.

The general objectives initially proposed have been met, thus developing an application that generates all the graphics requested at the beginning, and even some orders later, entering the data both by the user from the graphic interface and by another application in real time. Even more functionalities have been added than expected in the initial objectives, such as saving the images of the graphs.

However, there are certain improvements that could be made in the future to be taken into account. These improvements would be, for example, the ability to connect or change the server dynamically while the application is running or extra functionalities to facilitate and improve monitoring.

Therefore, by meeting all the established objectives, this application can be useful for the IAC and can be used in real situations during a night of observation. In addition, it has a clear, simple and minimalist design, with documentation included in the interface in case possible doubts at the time and includes the different functionalities mentioned in previous chapters.

Capítulo 8

Presupuesto

Este capítulo consta de una propuesta del presupuesto que estima el coste del trabajo según el tiempo empleado en el desarrollo de las tareas a lo largo del proyecto.

Este presupuesto se ha generado basado en el equipo utilizado y en la mano de obra ya que la aplicación creada no usa ningún servicio externo y todas las herramientas con las que se ha desarrollado la aplicación son de *software* libre.

8.1. Sección Uno

Tareas	Horas	Presupuesto
Introducción y entendimiento del proyecto	22 horas	7€/hora
Generación de gráficas solicitadas	50 horas	10€/hora
Diseño de la interfaz gráfica	6 horas	15€/hora
Introducción de datos estáticos en la interfaz	3 horas	7€/hora
Generación de gráficas con datos estáticos en la interfaz	9 horas	12€/hora
Funcionalidades extras en el módulo de datos estáticos	5 horas	15€/hora
Introducción de datos en tiempo real	36 horas	18€/hora
Generación de gráficas con datos en tiempo real en la interfaz	16 horas	15€/hora
Total	147 horas	1.836€

Tabla 8.1: Modelo de presupuesto para los tres meses de duración del proyecto

Bibliografía

[1] El concepto de ide. https://www.redhat.com/es/topics/middleware/what-is-ide.

[2] Figurecanvasqtagg. https://matplotlib.org/2.0.0/api/backend_qt5agg_api.html.

[3] Fits. https://es.wikipedia.org/wiki/FITS.

[4] Fits file handling. https://docs.astropy.org/en/stable/io/fits/index.html.

[5] Git. https://git-scm.com/.

[6] Matlab. https://www.mathworks.com/products/matlab.html.

[7] Matplotlib. https://matplotlib.org/stable/index.html.

[8] Opencv-python. https://pypi.org/project/opencv-python/.

[9] Python 3.8. https://www.python.org/downloads/release/python-380/.

[10] Qt creator: Un ide multiplataforma ideal para los desarrolladores qt. https://blog.desdelinux.net/qt-creator-ide-multiplataforma-ideal-desarrolladores-qt/.

[11] Visual studio code. https://code.visualstudio.com/.

[12] Óptica. https://es.wikipedia.org/wiki/%C3%93ptica.

[13] Alioli: presentation and first steps. https://ui.adsabs.harvard.edu/abs/2020SPIE11448E..2ES, 2020.

[14] AstroMía. El telescopio reflector. https://www.astromia.com/historia/historiareflector.htm.

[15] astronoo. http://www.astronoo.com/es/articulos/optica-adaptativa.html.

[16] NASA Ciencia. ¿cómo funcionan los telescopios? https://spaceplace.nasa.gov/telescopes/sp/.

[17] Google Cloud. ¿qué es pub/sub? https://cloud.google.com/pubsub/docs/overview, 2021.

[18] Instituto Astrofísico de Canarias. https://www.iac.es/.

[19] TCP. Protocolo de Control de Transmisión. https://developer.mozilla.org/es/docs/Glossary/TCP, 2021.

[20] Definición de IP. https://definicion.de/ip/.

[21] Protobuf: estructurar el código con Protocol Buffers. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/protocol-buffers/, 2020.

[22] José L. Fernández. Frente de onda. https://www.fisicalab.com/apartado/frente-de-onda.

[23] FigureCanvas. https://matplotlib.org/stable/tutorials/intermediate/artists.html#:~:text=the%20matplotlib.backend_bases.FigureCanvas%20is,to%20paint%20onto%20the%20canvas.

[24] Martin Fitzpatrick. Plotting with matplotlib. https://www.mfitzp.com/tutorials/plotting-matplotlib/.

[25] José Francisco Chicano García. Grandes telescopios Ópticos. page 14.

[26] GitHub. https://github.com/.

[27] NASA. About the hubble space telescope. https://www.nasa.gov/mission_pages/hubble/about, 2021.

[28] Francisco Javier Gantes Nuñez. Análisis y reducción de datos en pruebas de hartmann y shackhartmann. *España*, page 12, 2015.

[29] QWidget. https://doc.qt.io/qt-5/qwidget.html.

[30] María Estela Raffino. Concepto de telescopio. https://concepto.de/telescopio/.

[31] UNAH REVISTA DE LA ESCUELA DE FÍSICA. Polinomios de zernike y su aplicación en oftalmolófica. https://fisica.unah.edu.hn/assets/Revista/Volumen-V-N1/REF-UNAH-51-21.pdf, 2017.

[32] Angel Robledano. Qué es c++: Características y aplicaciones. https://openwebinars.net/blog/que-es-cpp/, 2019.

[33] Jim Schwiegerling. Description of zernike polynomials. page 1.

[34] Socket. https://www.speedcheck.org/es/wiki/socket/.

[35] Wikipedia. Espejo deformable - deformable mirror. https://es.mihalicdictionary.org/wiki/Deformable_mirror.

[36] Wikipedia. Shack–hartmann wavefront sensor. https://en.wikipedia.org/wiki/Shack%E2%80%93Hartmann_wavefront_sensor.

[37] ZeroMQ. https://zeromq.org/.

[38] Óptica Adaptativa. https://www.eso.org/public/spain/teles-instr/technology/adaptive_optics/.