ULPGC Universidad de Las Palmas de Gran Canaria | Instituto Universitario de Microelectrónica Aplicada IUMA

Máster Universitario en

Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Diseño de un sistema para la obtención en tiempo real de una referencia de blanco ante condiciones lumínicas variables y su validación en un dron con capacidad para adquirir imágenes hiperespectrales

Estudiante: Daniel Santana Vega

Tutor/es: Dr. Sebastián López Suárez, Dr. Raúl Guerra Hernández

Las Palmas de Gran Canaria. Septiembre 2021

IUMA

t +34 928 451 150
+34 928 451 086
f +34 928 451 083

e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es

Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 1 – 20 julio 2020

ULPGC Universidad de Las Palmas de Gran Canaria | Instituto Universitario de Microelectrónica Aplicada IUMA

Máster Universitario en

Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Diseño de un sistema para la obtención en tiempo real de una referencia de blanco ante condiciones lumínicas variables y su validación en un dron con capacidad para adquirir imágenes hiperespectrales

HOJA DE FIRMAS

Alumno/a:	Daniel Santana Vega	Fdo.:
Tutor/a:	Dr. Sebastián López Suárez	Fdo.:
Tutor/a:	Dr. Raúl Guerra Hernández	Fdo.:

Fecha: Septiembre 2021

IUMA
t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 1 – 20 julio 2020

ULPGC Universidad de Las Palmas de Gran Canaria | Instituto Universitario de Microelectrónica Aplicada IUMA

Máster Universitario en

Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Diseño de un sistema para la obtención en tiempo real de una referencia de blanco ante condiciones lumínicas variables y su validación en un dron con capacidad para adquirir imágenes hiperespectrales

HOJA DE EVALUACIÓN

Calificación:		
Presidente	Dr. José F. López Feliciano	Fdo.:
Secretaria	Dra. Ernestina Martel Jordán	Fdo.:
Vocal	Dr. Aurelio Vega Martínez	Fdo.:

Fecha: Septiembre 2021

IUMA
t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 1 – 20 julio 2020

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

LISTA DE ACRÓNIMOS

Acrónimos	
ENABLE-S3	European Initiative to Enable Validation for Highly Automated Safe and Secure Systems
IUMA	Instituto Universitario de Microelectrónica Aplicada
PLATINO	Plataforma HW/SW distribuida para el procesamiento inteligente de información sensorial heterogénea en aplicaciones de supervisión de grandes espacios naturales
APOGEO	Agricultura de Precisión para la Mejora de la Producción Vitícola en la Macaronesia
TIC	Tecnologías de la Información y la Comunicación
TFM	Trabajo de Fin de Master
VNIR	Visible and Near-Infrared
SWIR	Short-Wave Infrared
RGB	RED-GREEN-BLUE
FPS	Frames Per Second
FOV	Field of View
.csv	Comma-separated values
.json	JavaScript Object Notation

.raw	Imágenes en bruto o “crudo”
WR	White Reference
DR	Dark Reference
IDE	Entorno de Desarrollo Integrado
RMSE	Root-Mean-Square Error
SNR	Signal-to-Noise Ratio
EMR	Electromagnetic radiation
FIFO	First In – First Out

MEMORIA

DESCRIPTIVA

Capítulo 1. Introducción

1.1 Antecedentes

El objetivo principal de este Trabajo de Fin de Master es trabajar con imágenes hiperespectrales y sensores que miden la intensidad de luz y las condiciones ambientes respectivamente, todo esto en un entorno exterior realizando trabajos de campo donde estas variables juegan un papel muy importante. Este documento se centrará en el desarrollo de las capturas: estudio de las imágenes hiperespectrales, toma de datos, desarrollo de código e integrar todos sistemas para hacer la prueba de concepto en un entorno exterior. Si los resultados son satisfactorios, todo el sistema desarrollado será estudiado para ser implementado en trabajos con drones.

La Unión Europea aprobó en 2016 un proyecto de investigación dotado con más de 34 millones de euros, en el cual participa el Instituto Universitario de Microelectrónica Aplicada (IUMA) de la Universidad de Las Palmas de Gran Canaria. ENABLE-S3 (por su nombre en inglés “European Initiative to Enable Validation for Highly Automated Safe and Secure Systems”). Este proyecto, ya finalizado, estaba compuesto por un total de 74 socios de 15 países distintos, de los cuales 8 participantes son españoles. ENABLE-S3 tuvo un carácter claramente industrial, y su objetivo prioritario era el desarrollo de sistemas altamente automatizados, autónomos y seguros. En los últimos años del proyecto se desarrollaron distintas iniciativas destinadas a la creación de coches inteligentes sin conductor, que permitan disminuir el número de accidentes y aumentar el grado de confort y la seguridad de sus ocupantes y de los peatones en general.

Otros dominios industriales objeto de ENABLE-S3 eran: el sector aeroespacial, el ferroviario, el sector hospitalario, el marítimo y el de la agricultura. La principal aportación del IUMA en este proyecto era la de dotar de tecnología hiperespectral a varias de estas aplicaciones, mejorando de esta forma su funcionalidad y prestaciones. [9]

Actualmente, se está trabajando en el proyecto PLATINO (Plataforma HW/SW distribuida para el procesamiento inteligente de información sensorial heterogénea en aplicaciones de supervisión de grandes espacios naturales), que está financiado con más de 747.000 euros por el Ministerio de Ciencia, Innovación y Universidades a través de la convocatoria de 2017 del Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, en el marco del Plan Estatal de Investigación Científica y Técnica y de Innovación. Este proyecto propone un conjunto de soluciones para avanzar en el diseño de plataformas distribuidas HW/SW para el procesamiento inteligente de información sensorial heterogénea. La viabilidad de las propuestas se trata de demostrar un escenario propio de la denominada agricultura inteligente, en la cual se estudia la introducción de redes multisensoriales. Estas redes utilizan gran variedad de sensores para saber en tiempo real el estado de un sistema concreto. De entre todos los posibles sensores, la aportación del IUMA es la de sensores hiperespectrales a bordo de drones. Esto supone un extraordinario avance en términos de producción económica y de calidad, además en la seguridad de los alimentos. Para ello, PLATINO se marca como objetivo realizar avances en cinco aspectos claves para la utilización y gestión de estas redes de sensores:

1- Optimizar la adquisición de datos capturados mediante sensores hiperespectrales y fotónicos de última generación.
2- Diseñar algoritmos eficientes para el procesado de información sensorial compleja.
3- Crear una infraestructura de red segura para el transporte y procesamiento distribuido de datos capturados por sensores heterogéneos.
4- Desarrollar tecnologías eficientes y flexibles de procesamiento de datos de acuerdo a los requisitos de los diferentes escenarios.
5- Implementar sistemas expertos e interfaces de usuario de realidad mixta para la toma de decisiones en sistemas heterogéneos multi-sensoriales.

La viabilidad de las soluciones propuestas se demostrará en un escenario propio de la denominada agricultura inteligente, en el cual la introducción de este tipo de redes multisensoriales supone un extraordinario avance en términos de producción económica y de calidad y seguridad de los alimentos.

El IUMA también participa en el proyecto APOGEO. El objetivo general de este proyecto es mejorar la competitividad e innovación en las empresas vitícolas de las islas de la Macaronesia mediante la investigación, el desarrollo y la transferencia tecnológica basada en la “agricultura de precisión” o agricultura inteligente, formando a los actuales y futuros profesionales del sector primario en el uso de estas tecnologías. Para ello, APOGEO se marca como objetivo realizar avances en tres aspectos claves:

1- Desarrollo y transferencia tecnológica de un sistema comercial de bajo coste basado en drones, sensores espectrales avanzados y aplicaciones móviles para la monitorización en tiempo real del estado de salud de los viñedos de la Macaronesia, generando información relevante para el viticultor.

2- Detección temprana en los viñedos de stress abiótico (falta de nutrientes, salinidad, contaminación) y bióticos (plagas) junto con acciones correctoras, desde aporte de nutrientes a nuevos fitosanitarios selectivos.
3- Creación de un plan formativo y de capacitación agraria basado en cursos y seminarios dirigidos a los actuales y futuros viticultores para difundir, introducir y gestionar nuevas tecnologías en el proceso de cultivo de los viñedos, asegurando la continuidad de los resultados de este proyecto.

Como se puede ver, el IUMA está aplicando la tecnología hiperespectral en muchos sectores: el espacio, la agricultura de precisión (o agricultura inteligente), la medicina, etc. Este Trabajo de Fin de Máster es una aportación al desarrollo destinado a la agricultura de precisión [6] [7], pudiendo extrapolarse a cualquier otro ámbito de aplicación en el que se requiera procesamiento de imágenes hiperespectrales y especialmente para aplicaciones de campo.

La agricultura de precisión o agricultura inteligente es la inclusión de las Tecnologías de la Información y la Comunicación (TIC) en las actividades agrícolas con la finalidad de mejorar la gestión de los campos de cultivo y maximizar la producción. [10]. Existen tres posibles aplicaciones:

1. Gestión de la información: Adquisición, almacenamiento y procesado de datos.

2. Herramientas para toma de decisiones (HAD): gestión de variables espaciales.

3. Automatización de las tareas agrícolas: mediante el uso de robótica para facilitar las actividades del agricultor.

Esta área de investigación se basa en aprovechar el potencial tecnológico actual para disminuir el esfuerzo del agricultor y aumentar la productividad en los campos agrícolas. Además de las connotaciones tecnológicas derivadas de este tipo de proyecto, hay otras de tipo socio-económico y medio ambiental. En cuanto al primer factor, el socio-económico, hay que indicar que según el Banco Mundial en los últimos 50 años se ha perdido casi la mitad de la tierra cultivable por persona (en el caso de España, hemos pasado de 0,53 hectáreas por persona en 1961 a 0,26 hectáreas en 2014) [8], a lo que se une el aumento de población que en el año 2030 pasará a ser de 8.500 millones de personas. Indudablemente esto afectará en gran medida a toda aquella población que tenga una gran dependencia del sector agrícola, y todo lo que sea mejorar los recursos y aumentar la productividad de sus tierras beneficiará a su economía y a su bienestar. Por otro lado, el uso de recursos hídricos para abastecer a los campos de cultivo, así como el de pesticidas para evitar la llegada de plagas o eliminarlas, hace necesaria una estrategia que permita optimizar su utilización, disminuyendo el impacto negativo que pueda tener sobre el medio ambiente y la salud. Los riesgos que los pesticidas tienen para la salud (sobre todo en niños, adolescentes y mujeres embarazadas), hacen que debamos afrontar este problema de forma inteligente para evitar consecuencias fatales como pueden ser el provocar cáncer o acarrear consecuencias para los sistemas reproductivo, inmunitario o nervioso. [9]

La tecnología hiperespectral surgió hace ya varias décadas en aplicaciones de satélites de observación de la tierra. Las imágenes hiperespectrales muestran una alta resolución espectral, lo que permite recopilar información de bandas espectrales muy estrechas y continuas. Dado que todos los materiales reflejan

(reflectacia), absorben (absortancia) o emiten energía electromagnética en longitudes de onda, esta característica de las imágenes hiperespectrales permite reconstruir el espectro de radiación de cada píxel de la imagen y en consecuencia, para identificar diferentes materiales sobre la base de su forma espectral. Este es el mismo principio que se usa en el campo de la astronomía, ya que se puede saber de qué está compuesto un planeta o estrella que está a millones de años luz según la firma espectral que refleja o emite. Cada tipo de átomo tiene un único conjunto de niveles de energía y por ende cada elemento tiene una huella única espectral.

Esta propiedad hiperespectral hace que los datos de las imágenes hiperespectrales sean beneficiosos para muchas aplicaciones más locales como estado de vegetación y seguimiento de recursos hídricos, detección no invasiva de la calidad de los alimentos, geología y diagnóstico de cánceres múltiples, entre otras. Las aplicaciones de este campo pueden ser infinitas. [17]

Una escena hiperespectral se basa en obtener una imagen, de una misma zona espacial, en distintas longitudes de onda del espectro electromagnético. El espectro electromagnético es la distribución energética del conjunto de las ondas electromagnéticas. Referido a un objeto cualquiera, se denomina espectro electromagnético a la radiación electromagnética que emite (espectro de emisión) la que absorbe (espectro de absorción) o refleja (reflectancia) una sustancia. [30]

En una imagen convencional se capta solo una pequeña fracción del espectro: la correspondiente al rango del visible con longitudes de onda que van desde los 400 a 700 nm. En una imagen hiperespectral este rango se ve ampliado

a otras longitudes de onda, como puede ser el infrarrojo o el ultravioleta. El IUMA dispone de cámaras las cuales funcionan con luz visible y la región del infrarrojo cercano, revelando las escenas con más detalle que el ojo humano o una cámara RGB de imagen convencional. Actualmente el estudio se centra en cámaras que usan tecnología hiperespectral de barrido (PUSHBROOM) como las cámara Specim FX10 y Specim FX17 descritas más adelante en este documento. Al ser ambas cámaras del tipo PUSHBROOM de tecnología de barrido, las imágenes constan de una dimensión espacial, fijada a la dimensión Y y la información de todos los espectros en la dimensión X. [11] Para capturar una escena completa, el barrido de la imagen debe de ser a una velocidad de desplazamiento por la escena determinada que depende de múltiples parámetros como: la distancia, el FOV, el número de pixeles y del frame rate. Teniendo en cuenta estos factores se evitan saltos o solapamientos de los datos capturados.

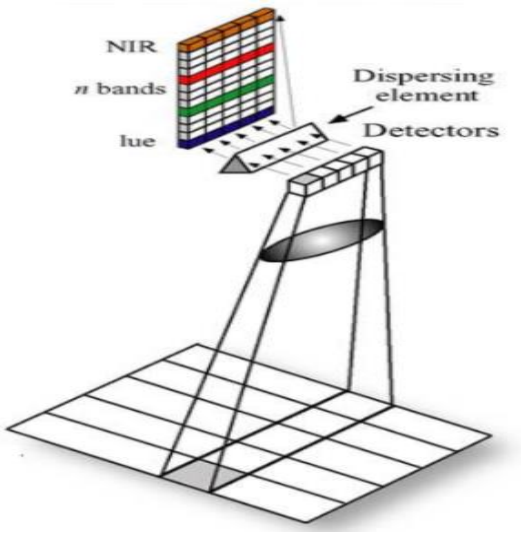

Ilustración 1: Funcionamiento cámara hiperespectral tipo PUSHBROOM

El barrido completo de una escena da como resultado un cubo hiperespectral en el cual radica en el concepto de firma espectral mencionada anteriormente. La firma espectral es la forma específica en que cada material interactúa con las distintas longitudes de onda. Entiéndase la firma espectral como una huella dactilar de un material concreto que en el caso de la agricultura de precisión permite conocer el estado del suelo, la biomasa o similares. En otros campos de aplicación como la medicina, cada tejido del cuerpo humano refleja o absorbe diferentes longitudes de onda.

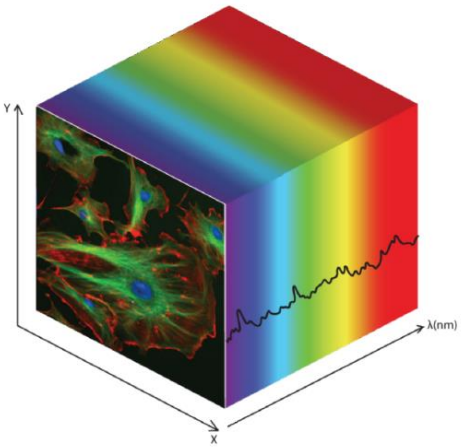

Ilustración 2: Concepto de cubo hiperespectral

A partir de estos datos se puede obtener información físico-química del objeto que se está observando, y este es el principal beneficio que nos otorga esta tecnología, no solo la de detectar objetos sino la de identificar su composición o estado. En la siguiente imagen se pueden observar algunos ejemplos de firmas espectrales de diferentes tipos de vegetación donde se puede destacar los diferentes niveles de reflectancia por cada longitud de onda y por cada tipo de

vegetación. En este caso, el Roble (rojo) tiene una mayor reflectancia entre las longitudes de onda de 700 – 1300 nm.

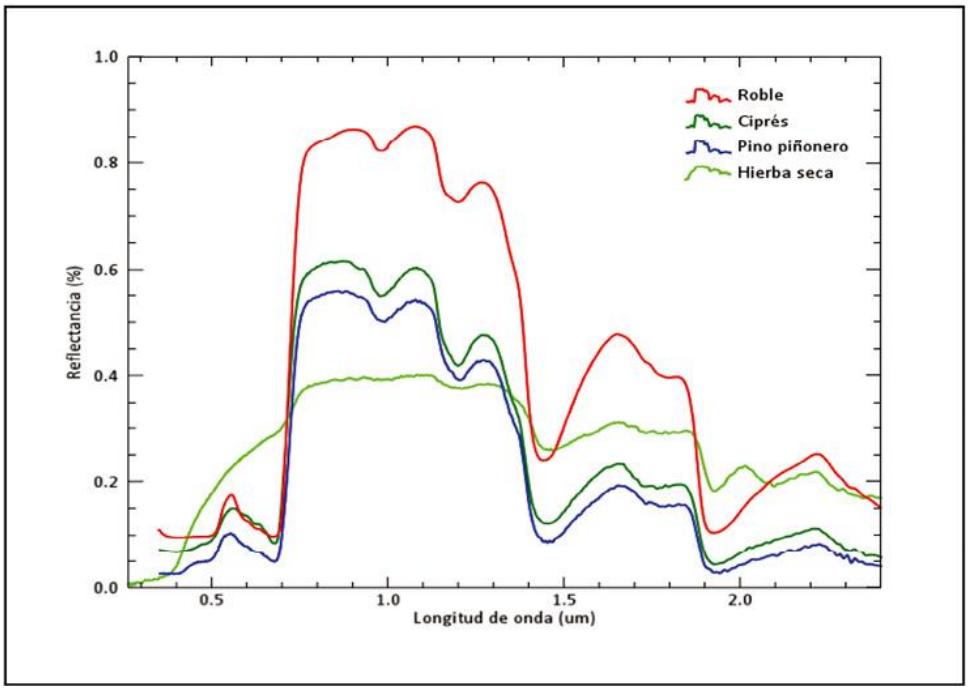

Ilustración 3: Firmas espectrales de vegetación

Uno de los mayores retos a los que nos enfrentamos a la hora de trabajar con esta tecnología es que tratamos con imágenes mucho más complejas, y por lo tanto requieren de un procesamiento también más complejo. [9] Si bien en la actualidad varias empresas afrontan este problema haciendo uso de una red de sensores distribuida por los campos agrícolas, la opción que se siguió el proyecto ENABLE-S3 y en la actualidad los proyectos PLATINO Y APOGEO es la de disponer de un único sensor hiperespectral embarcado en un dron que realice vuelos

frecuentes a lo largo de la zona de cultivo, esto simplifica la red a un único sensor con mayores capacidades de obtener información además de poder adaptarse a cualquier tipo de campo. Para realizar la captura de las imágenes hiperspectrales, el IUMA dispone en sus laboratorios de varias cámaras:

- La FX10 de Specim dispone de 224 bandas que van desde los 400 nm hasta los 1000 nm (VNIR). Tiene un peso de 1.4 Kg. Este tipo de cámaras suelen utilizar sensores con tecnología CMOS.
- La FX17 de Specim funciona en el rango de longitudes de onda desde 900 a 1700 nanómetros (SWIR). Tiene un peso similar a la FX10. Este tipo de cámaras utiliza sensores con tecnología InGaAs.

Conjuntamente con estas cámaras, el IUMA dispone de un dron de altas prestaciones, el Matrice 600 de la empresa Dji, capaz de transportar hasta un máximo de 6 Kg durante un tiempo máximo de 20 minutos. Al tratarse de un dron de altas prestaciones tiene un precio elevado y uno de los objetivos a desarrollar por este trabajo y los trabajos futuros es conseguir un sistema de bajo coste con las mismas prestaciones capaz de realizar los trabajos de campo. Los aspectos más técnicos de los dispositivos se pondrán de manifiesto en este documento más adelante. Ambas infraestructuras se muestran en la siguiente ilustración. [9]

Ilustración 4: Cámara hiperespectral Specim FX10 y DJI Matrice 600

A la hora de afrontar las tareas que conlleva la agricultura de precisión y como parte de este Trabajo de Fin de Master, todos los conceptos y estudios desarrollados pueden ser aplicados a cualquier ámbito que requiere el uso de imágenes hiperespectrales. En una primera fase se busca delimitar una zona en la que se quiera hacer el análisis, luego se prepara todo el sistema para realizar capturas de la intensidad de luz y las condiciones ambientales. Una vez posicionado el equipo, se realizan capturas durante un periodo de tiempo prolongado para crear un dataset de la zona o área designada. Con este dataset se realizan cálculos de regresión para poder llegar a determinar qué nivel de reflectancia es el óptimo según las condiciones ambientales e intensidad de luz en tiempo real. Posteriormente, integrar todo el sistema en un dron y realizar misiones capturando imágenes hiperespectrales calibradas, las cuales una vez procesadas permitan extraer información relativa de las distintas características

de la cosecha (madurez, enfermedades, humedad, etc.) y transmitir esa información en tiempo real al agricultor y/o a la maquinaria agrícola en lo que se refiere a la agricultura de precisión.

1.2 Problemas a solucionar

Las imágenes captadas por una cámara hiperespectral tienen datos "en bruto" (.raw). Este es un formato de archivo digital de imágenes que contiene la totalidad de los datos de la imagen tal y como ha sido captada por el sensor digital de la cámara. [27]

Para poder extraer información útil de ellas, ya sea para detección de objetivos, clasificación, etc., lo normal es convertir los datos a reflectancia o absortancia. La reflectividad o reflectancia es la fracción de radiación incidente reflejada por una superficie u objeto y la absortancia es exactamente lo opuesto, la fracción de radiación absorbida por una superficie u objeto. Este Trabajo de Fin de Master se centra en la medida de la reflectancia para obtener las firmas espectrales. Para poder obtener la reflectancia de los objetos con las cámaras hiperespectrales, estas deben de ser calibradas correctamente.

En un entorno de laboratorio, donde las condiciones ambientales pueden ser controladas, se captura una referencia de blanco y de negro al comienzo y al final del proceso manteniendo la luz constante todo el tiempo. Una vez configuradas las condiciones de captura de la cámara (tiempo de exposición,

intensidad de la luz, etc.) se realiza un barrido sobre un material calibrado para reflejar un alto porcentaje de luz incidente o también llamado referencia de blanco.

Para ello, las cámaras se colocan apuntando a esta referencia de blanco. Como comentado anteriormente, es un material calibrado que puede reflejar casi el 100% de la luz o diferentes porcentajes según el tipo de calibración porcentual que se desea realizar. En este Trabajo de Fin de Master se trabaja con la referencia de blanco de la marca Zenith Polymer que es capaz de reflejar casi el 100% de la luz incidente entre las longitudes de onda de 400 a 1750 nm como se observa en la siguiente ilustración siendo la línea de color Rojo la que tiene disponible el IUMA es un sus laboratorios. [15]

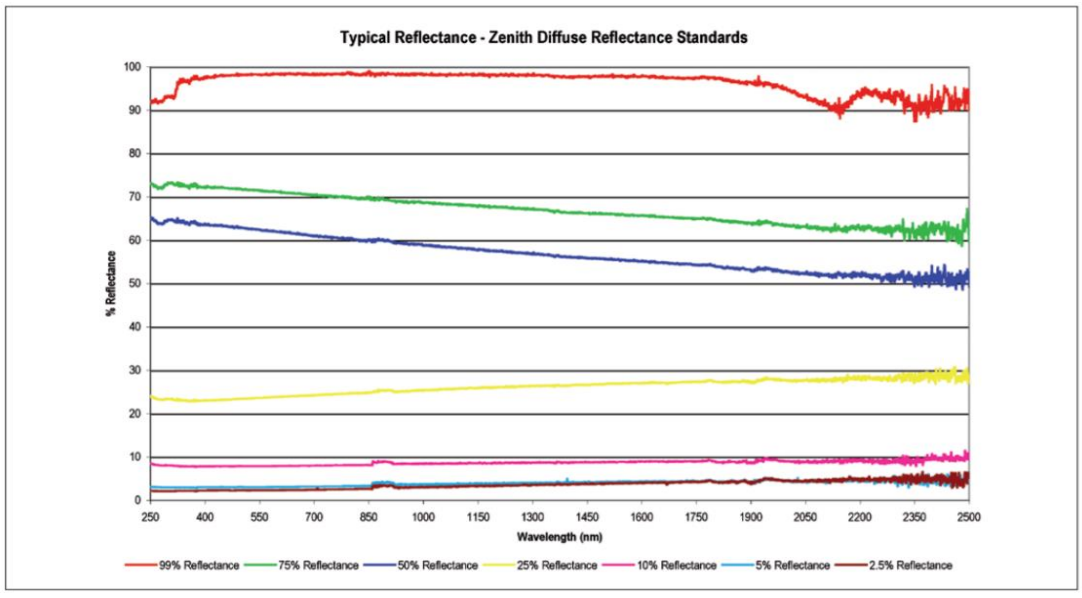

Ilustración 5: Reflectancia para distintos tipos de referencia de blanco

Una vez que la cámara se haya colocado apuntando a la referencia de blanco, se obtiene una captura de la misma, a este proceso se le conoce como referencia blanco, (WR en sus siglas en ingles). Esta captura contiene los valores máximos que el sensor es capaz de medir para cada píxel y para cada banda según la configuración adoptada (tiempo de exposición, intensidad de la luz, etc.), el resultado es una firma espectral de casi el 100 % de la luz reflejada en entre 400 y 1900 nm según la línea roja de la ilustración anterior. En el caso de las cámaras disponibles en el IUMA, las longitudes de onda capturaras son entre 400 – 1000 nm para la cámara Specim FX10 y de entre 900 y 1700 nm para la cámara Specim FX17. Cada cámara tiene una respuesta espectral diferente ante una misma referencia de blanco o un objeto concreto. En la Ilustración 5 la captura de la referencia de blanco se ha hecho con cámaras muy específicas para determinar con exactitud el porcentaje reflejado. A continuación se muestra la respuesta espectral de las cámaras disponibles en el IUMA ante la referencia de blanco donde se puede observar que tienen diferentes respuestas dependiendo de la longitud de onda.

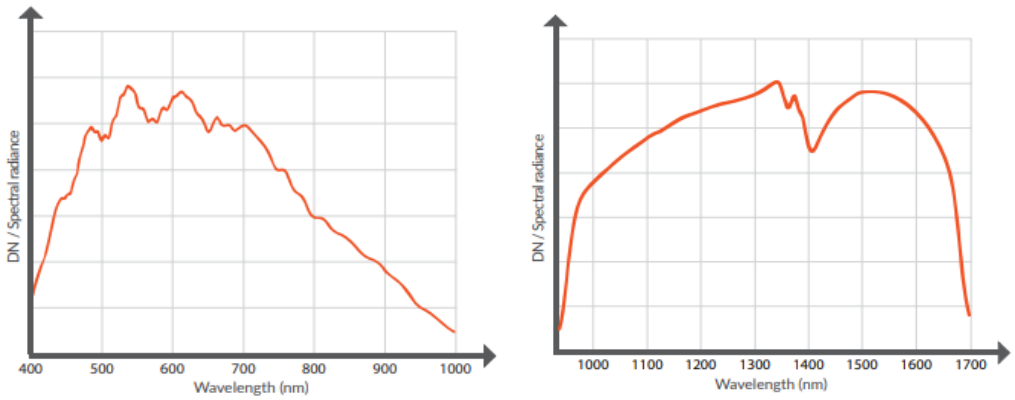

Ilustración 6: de izquierda a derecha: respuesta espectral Specim FX10 y respuesta espectral Specim FX17

Tras este proceso de captura de la reflectancia de la referencia de blanco, la fuente de luz se apaga o bien se tapa el objetivo de la cámara, y se recoge una nueva captura. Esto se suele denominar referencia de negro, (DR en sus siglas en ingles), y contiene los valores mínimos que el sistema puede proporcionar para cada píxel y banda. Idealmente, los valores de DR deberían estar muy cerca de cero, sin embargo, se pueden obtener valores más altos normalmente debido al ruido intrínseco del sensor [16].

Una vez que han tomado estas dos medidas de la cámara hiperespectral, las escenas capturadas posteriormente, como por ejemplo en un trabajo de campo sobre un terreno agrícola, se corrigen aplicando la siguiente formula: [17]

$$reflectancia = \frac{raw - darkRef}{whiteRef - darkRef}$$

Ecuación 1: Formula reflectancia

Cada cámara hiperespectral se compone de sensores de luz y elementos ópticos como los objetivos y defractores que enfocan o separan la luz en diferentes longitudes de onda. Estos componentes de cada cámara difieren entre los distintos tipos de modelos y marcas que existen. Además, cada cámara puede tener tipos de memoria, buffers y procesos de memoria diferentes. Todas estas pequeñas diferencias entre estos dispositivos hacen que los datos en bruto ".raw" puedan diferir en una captura de un objeto o área determinada entre los distintos tipos de cámaras disponibles en el mercado. Eso quiere decir que al aplicar la

fórmula anterior, se obtiene un valor de reflectancia porcentual y escalado con el que se consigue un sistema de referencia global y se aísla las condiciones de medida a la hora de realizar la captura con diferentes cámaras o diferentes condiciones ambientales [17].

En conclusión, el valor obtenido en la fórmula de la Ecuación 1 es el valor real de reflectancia de cada objeto o superficie capturado. Si se aplica el proceso de calibración descrito con diferentes cámaras o condiciones ambientales, el valor debe ser el mismo o muy cercano.

En el caso de los drones o en cualquier trabajo de campo, las condiciones ambientales cambian constantemente haciendo que la referencia de blanco varíe en función del clima, la franja horaria o incluso la absorción atmosférica. El sol es la principal fuente de energía electromagnética o EMR que recibe la tierra y bombardea constantemente con estas EMR pero antes de alcanzar la superficie terrestre tiene que pasar a través de la atmosfera. La atmosfera nos protege de las radiaciones de alta energía como son los Rayos-X o los Rayos Gamma. La radiación que pasa a través de la atmosfera, interactúa con las moléculas y partículas que la conforman. En la atmosfera las EMR son reflejadas o absorbidas y una porción pasa a través de ella hasta llegar a la superficie.

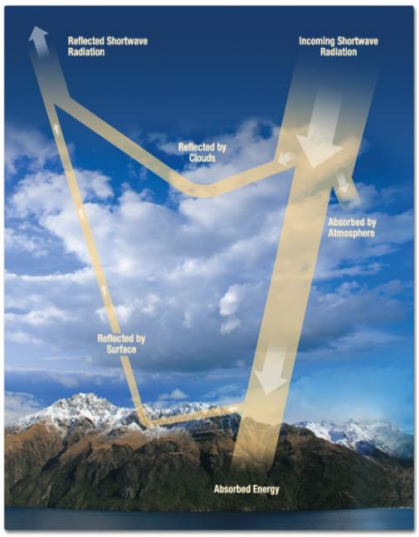

Ilustración 7: Ejemplo de cómo interactúa la radiación solar con la atmosfera

Una porción de la radiación es absorbida por los gases presentes en la atmosfera. Estos gases absorben ciertas longitudes de onda, por lo tanto, en ciertas partes del espectro visible se absorbe muy poca energía, sin embargo en la parte del espectro que corresponde al ultravioleta, se absorbe casi toda la energía entrante. Estas porciones del espectro que son absorbidas por los gases atmosféricos se conocen como bandas de absorción. Los principales gases involucrados en la absorción atmosférica son:

- Vapor de agua (H2O): absorbe mucha radiación en el rango 5500 nm – 7000 nm y por encima de los 2700 nm. Hay que tener en cuenta que el vapor de agua no es constante en el tiempo ni en el espacio que ocupa. Esto significa que la absorción por este gas depende de la localización y la época del año.
- Dióxido de carbono (CO2): Absorbe principalmente el infrarrojo térmico del espectro.

- Ozono (O3): Absorbe los rayos ultravioleta de alta energía. Es muy importante dado que nos protege de los daños de esta radiación y principal causa de cáncer de piel.

Para solucionar el problema del trabajo de campo, se plantea configurar un sistema que permita medir simultáneamente las condiciones ambientales tales como: la intensidad de luz, la temperatura y/o la humedad; y capturar una referencia de blanco con esas condiciones ambientales. Llevando a cabo estas mediciones se genera gran una base de datos que relaciona la intensidad de luz y las condiciones ambientales con una referencia de blanco concreta en tiempo real.

Utilizando sistemas de optimización (regresión) se pretende determinar cuál es la mejor referencia de blanco para las condiciones ambientales que existen en el momento de realizar el vuelo. Aplicando esta posible solución, se obtienen varias ventajas:

- Precisión a la hora de obtener información gracias a la tecnología hiperespectral.
- Comodidad y adaptabilidad. El poder tener un valor de reflectancia en tiempo real no obliga a aterrizar el dron para volver a tomar un valor de reflectancia en caso de que las condiciones ambientales cambien durante el vuelo. Así las imágenes pueden ser calibradas en tiempo real pasando los valores capturados por la cámara hiperespectral que están en bruto ".raw" directamente a reflectancia.

Todo este proceso está ejecutado por una placa Jetson Xavier NX. Esta actúa como controlador del dron DJI Matrice 600 y tiene capacidad para controlar las cámaras hyperespectales gracias a su sistema operativo integrado. Los sensores intensidad de luz, humedad y temperatura serán controlados con una

placa Arduino Nano que transmite la información vía puerto serie a la placa Jetson Xavier NX.

En total, los dispositivos a estudiar para ser implementados en el dron serían:

- Cámara hiperespectral FX10 o FX17
- Sensor de luz BH1750
- Sensores de temperatura y humedad DHT11
- Cámara RGB ELP USB
- Arduino Nano
- Jetson Xavier NX
- Espectroradiometro STS-VIS. Este es integrado en la prueba de concepto a modo de estudio y comparación con los otros sensores. No está prevista su utilización en drones por su alto coste.

Otro problema a solucionar es el peso total de todos los elementos en un dron. Hasta para un dron de altas capacidades como el DJI Matrice 600 puede llegar a ser un problema. El objetivo es poder trasladar este estudio a drones de gamas más bajas para poder recortar los costes del sistema completo.

1.3 Objetivos principales

En este Trabajo de Fin de Master, de ahora en adelante también denominado como TFM, los objetivos operativos/parciales que se pretenden realizar son los siguientes:

1. Configurar un sistema de captura que permita medir simultáneamente las condiciones ambientales y capturar una referencia de blanco con la cámara hiperespectral SPECIM FX10 en esas condiciones. Todo este trabajo también será adaptable a la cámara SPECIM FX17. Para ello, se estudiarán los siguiente métodos:
    a. Midiendo únicamente la intensidad de luz con el sensor BH1750 y también midiendo la intensidad de luz en unos pocos rangos de longitudes de onda específicos. Para esta segunda parte, se usará la cámara ELP RGB que mide la intensidad de luz en el rango del espectro correspondiente al rojo (650 nm), verde (550 nm) y azul (470 nm). Estos tres colores son la composición del color en términos de la intensidad de los colores primarios de la luz, es decir, es posible representar un color mediante la mezcla por adición de cualquiera de ellos. Sumando todo lo anterior, se dispone de 4 medidas de intensidad de luz útiles para el estudio.
    b. A modo de estudio y comparación, utilizando el espectrómetro pancromático STS-VIS Spectrometers que mide la intensidad de luz aproximadamente igual en longitudes de onda entre 350 y 800 nm con un número de bandas entre 300 y 75. Se configura para obtener resultados para las longitudes de onda que correspondan al RGB. Se hacen capturas de las condiciones ambientales para hacer la comparación con los otros dos sensores.
    c. Medir las condiciones ambientales como humedad relativa y temperatura con el sensor DHT11.

2. Desarrollar una base de datos realizando capturas con el sistema descrito en los subapartados a, b y c para crear un conjunto de datos con los que hacer pruebas. La finalidad de esta base de datos es poder relacionar una referencia de blanco con unas condiciones ambientales concretas y una configuración específica de ganancia y tiempo de

exposición del sensor ELP RGB y la cámara hiperespectral. Idealmente, se tratará de conseguir condiciones ambientales diferentes para generar una base de datos completa que abarque el mayor número situaciones con condiciones ambientales diferentes, exceptuando los días en el clima sea extremadamente duro como por ejemplo con lluvia o viento fuerte. Para desarrollar la base de datos se utilizaran ficheros .json y .csv:

a. Archivo .json: este tipo de archivos almacena objetos y estructuras de datos simples en formato JSON. Este es un formato estándar de intercambio de datos ligeros, están basados en texto, son legibles por humanos y pueden editarse usando un editor de texto.
b. Archivo .csv: Los archivos CSV sirven para manejar una gran cantidad de datos en formato tabla, sin que ello conlleve sobrecoste computacional alguno. Facilita la limpieza de los datos y es compatible con la plataforma utilizada para el estudio de regresión.

3. Una vez obtenida la base de datos, hacer un estudio de regresión para determinar de forma automática y en tiempo real el valor de referencia de blanco optima según la condiciones ambientales medidas al instante por los sensores, además de saber la configuración más adecuada para los sensores y que estos no puedan saturar o perder datos configurando el tiempo de exposición, ganancia, intensidad de la luz, etc. para poder desarrollar un modelo que permita la calibración en tiempo real.

1.4 Estructura del documento

El trabajo que se expone en este documento se encuentra desglosado en seis capítulos como se explica a continuación.

En el presente capítulo, el primero de ellos, se introducen los antecedentes y motivaciones que han llevado a la realización del trabajo y, además, se desarrollan los objetivos del mismo.

El segundo capítulo trata sobre los conocimientos necesarios para el desarrollo del trabajo. Aquí se introducen los recursos hardware y software utilizados.

En el tercer capítulo se exponen las soluciones adoptadas para la toma de datos de intensidad de luz, condiciones ambientales y reflectancia. Se explica la estructura del software desarrollado, detallando el funcionamiento de cada uno de sus módulos, haciendo especial hincapié en la regresión lineal que nos dará el valor óptimo de referencia de blanco acorde a la luz que haya ese día.

En el cuarto capítulo se recogen los resultados obtenidos y las conclusiones, donde se comprobará el grado de consecución de los objetivos planteados y se proponen algunas líneas de trabajo futuras.

En el quinto capítulo se recogen todas las referencias bibliográficas utilizadas en este TFM.

Finalmente, en los anexos están descritos los códigos de programación de las aplicaciones desarrolladas.

Capítulo 2. Conocimientos previos y herramientas empleadas

2.1 Conocimientos previos

En este apartado se describe cada dispositivo por separado señalando los aspectos más importantes que se han estudiado en este TFM. En total, se integran siete dispositivos que funcionan de manera coordinada para poder capturar y clasificar las medidas de intensidad de luz, temperatura y humedad junto con un valor de reflectancia.

2.2 Herramientas hardware

En este apartado se detallan los dispositivos utilizados e integrados en la realización de este TFM.

2.2.1 Cámara hiperespectral Specim FX10 y FX17

Para la toma de imágenes hiperespectrales se dispone de dos cámaras de la marca Specim, concretamente los modelos FX10 y FX17. La Specim FX10 puede obtener imágenes en el rango de longitudes de onda desde 400 a 1000 nanómetros (VNIR) [11] , mientras que la Specim FX17 funciona en el rango de longitudes de onda desde 900 a 1700 nanómetros (SWIR) [12] según la siguiente ilustración y la Tabla 1.

Ilustración 8: de izquierda a derecha: FX10 de Specim y FX17 de Specim

Cámara	Rango espectral	Resolución espacial	FPS	Bandas	Interfaz
Specim FX10	400-1000 nm	1024 pixels	330	224	GigE
Specim FX17	900-1700 nm	640 pixels	527	224	GigE

Tabla 1: Especificaciones cámaras Specim FX10 y FX17

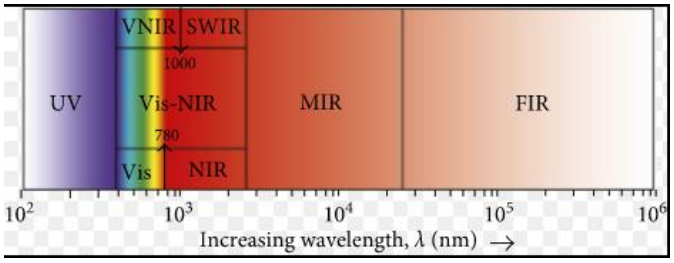

Ilustración 9: Evaluación del espectro

Estas cámaras funcionan con luz visible y la región del infrarrojo cercano, revelando detalles en superficies y objetos con más detalle que el ojo humano o una cámara RGB. Puede detectar manzanas magulladas o plantas enfermas antes de que el daño sea visible, en otros campos de aplicación como la medicina se utilizan este tipo de cámaras para detectar tejidos como tumores o en el ámbito espacial con cámaras situadas en satélites. Estas cámaras disponen de tecnología

hiperespectral del barrido (PUSHBROOM). Al ser del tipo PUSHBROOM las imágenes constan de una dimensión espacial, fijada a la dimensión "Y" y la información de todos los espectros en la dimensión "X". [11] Para capturar una escena completa, el barrido de la imagen debe ser a una velocidad determinada que depende de múltiples parámetros como: la distancia, el FOV, el número de pixeles y del frame rate. Teniendo en cuenta estos factores se evitan saltos o solapamientos de los datos capturados. La cámara captura la intensidad de luz reflejada por los objetos que están en la escena devolviendo una firma espectral de la misma.

La cámara no capta de manera homogénea todas las longitudes de onda para las que está diseñada. Esta es menos sensible a la intensidad de luz en los extremos entre 400 – 500 nm y 900 – 1000 nm. Además hay que tener en cuenta la absorción de la atmósfera explicada en apartados anteriores. En la siguiente ilustración se compara la respuesta espectral de la referencia de blanco en el datasheet de la cámara Specim FX10 capturada en un entorno de laboratorio con la intensidad de luz y las condiciones ambientales controladas, por otro lado, la respuesta espectral de la misma cámara en un trabajo de campo en el exterior. Se puede observar un pico de absorción entre 750 -780 nm debido a la absorción atmosférica. Hay que tener esto en cuenta a la hora de capturar los datos y descartar posibles valores incorrectos.

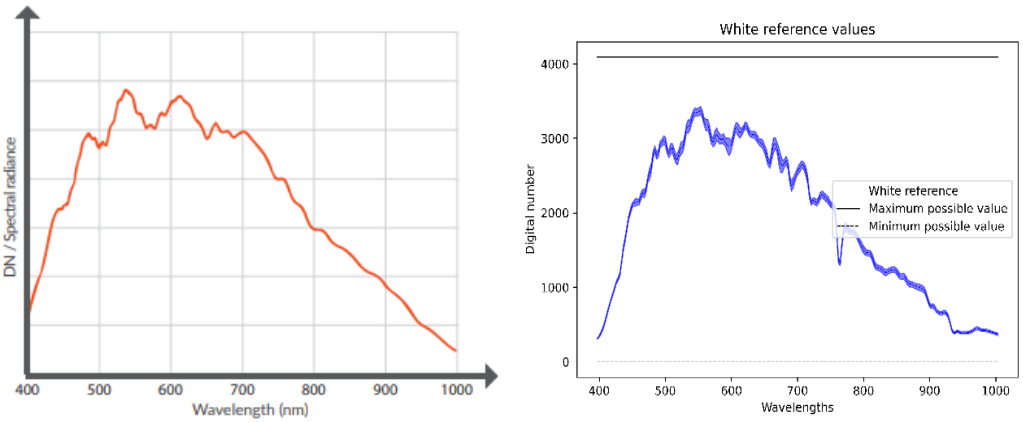

Ilustración 10: De izquierda a derecha: Respuesta espectral del datasheet de cámara Specim FX10 y respuesta espectral de la cámara Specim FX10 en un trabajo de campo en el exterior

La cámara dispone de un buffer de memoria de 12 bit para convertir los datos de intensidad de luz a valores digitales, es decir, cuando se digitaliza la intensidad de luz el valor máximo que puede medir es 2^12 = 4095 por cada longitud de onda. Esta parte se estudia más adelante en este documento.

Estas cámaras disponen de una interfaz GigE Vision. La interfaz GigE Vision es un estándar para cámaras industriales de alto rendimiento. Ofrece un marco para la transmisión de vídeo a alta velocidad y los datos de control relacionados por redes Ethernet [26]. Este protocolo tiene cuatro elementos principales:

- Protocolo de control (GVCP): se ejecuta en el protocolo UDP. El estándar define cómo controlar y configurar dispositivos.
- Protocolo GigE Vision Stream (GVSP): se ejecuta en el protocolo UDP. Abarca la definición de tipos de datos y las formas en que se pueden transferir imágenes a través de GigE.

- Mecanismo de descubrimiento de dispositivos GigE: proporciona mecanismos para obtener direcciones IP.
- XML basado en un esquema definido por el estándar GenICam de la Asociación Europea de Visión Artificial que permite el acceso a los controles de la cámara y flujos de imágenes.

Esta interfaz es muy útil para la configuración de la cámara y la rápida transferencia de datos. El tiempo de exposición y la ganancia son las dos características que se deben tener en cuenta a la hora de realizar capturas. Para ello, se debe configurar la cámara antes de cada captura. Para poder controlar la interfaz GigE vision se dispone de la librería de código abierto Aravis-wrapper. Esta librería se describe más adelante en este TFM.

El setup físico se ha preparado para las dos cámaras. Para este TFM se trabaja con la cámara Specim FX10 pero todos los aspectos desarrollados para esta cámara pueden ser trasladados a la cámara Specim FX17.

2.2.2 Jetson Xavier NX

Esta placa es el mini PC que controla todo el sistema. Además de estar integrada en el dron Dji Matrice 600 y controlar todos los aspectos relacionados con el vuelo del dron, es compatible con el sistema operativo de código abierto Ubuntu Linux. Esto la hace configurable para poder ser adaptada a este TFM. Además, dispone de una GPU *NVIDIA Volta* con 384 núcleos *NVIDIA CUDA* y 48 *Tensor Cores* con la que poder realizar tareas de cómputo avanzado tales como:

mezclado o reducción de datos; y paralelismos idóneos para el tratamiento de imágenes hiperespectrales con un reducido consumo de energía.

Esta placa almacena los datos recogidos y se encarga de realizar la regresión que seleccione la reflectancia de blanco idónea según las condiciones ambientales al momento de realizar una captura. En trabajos futuros, esta regresión se debe realizar en tiempo real al momento de llevar a cabo una misión de vuelo con el dron. Sobre esta placa se desarrollan los códigos principales en lenguaje de alto nivel Python y actúa como controlador principal de los siguientes dispositivos:

- Cámara hyperespectral vía Ethernet Gige vision.
- Aduino Nano vía USB como puerto serie bajo el protocolo RS-232
- Cámara RGB ELP vía USB
- Espectrómetro STS-VIS vía USB

Ilustración 11: Jetson Xavier NX

Más adelanté en este TFM, se describen las conexiones con cada uno de los dispositivos dependientes de la placa Jetson Xavier NX, los modos de comunicación entre ellos y los códigos en leguaje de programación de alto nivel Python desarrollados para cada uno de los sistemas integrados.

2.2.3 Zenith Polymer

Este material altamente reflectante esta hecho a base de PTFE. Es resistente al calor, la humedad y la exposición a altos niveles de radiación, lo que lo hace ideal para su uso como estándares de reflectancia. Existen diferentes valores de reflectancia del 99%, 80%, 60%, 50%, 25%, 10%, 5% y 2,5% como se mostró en la Ilustración 4 y mantiene la reflectancia constante con longitudes de onda que van desde los 250nm hasta los 2450nm con una tolerancia en los valores de reflectancia de ±3%. [15]

Se coloca esta referencia de blanco en el suelo con la cámara hiperespectral Specim FX10 o Specim FX17 apuntando directamente hacia ella para obtener el valor de reflectancia. Más adelante en este TFM se muestra el proceso de obtención del valor reflectancia y los datos recogidos. También existen en el mercado referencias de negro y referencias de gris, aunque también tienen valores cuantificados de reflectancia y absorbancia, estas no son tan idoneas para este estudio puesto que buscamos el máximo posible de reflectancia. Esto solo se consigue con una buena referencia de blanco.

Ilustración 12: A la derecha: Referencia de blanco Zenith Polymer

2.2.4 Arduino Nano

Arduino Nano, es una versión reducida de Arduino UNO. Esencialmente la placa Arduino UNO es una placa electrónica basada en el chip de Atmel ATmega328. Arduino Nano minimiza la demanda de energía que consume y también hace que no se necesite tanto espacio para alojar la paca, por lo que es ideal para proyectos donde el tamaño y el peso sean importante. Las especificaciones están descritas en la siguiente tabla.

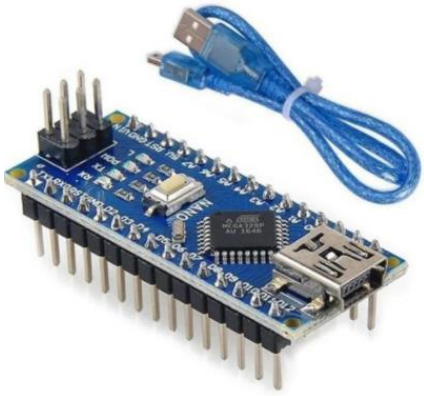

Ilustración 13: Arduino Nano

Entrada y Salida de datos	Pines entrada/salida	Velocidad del reloj
Puerto Serie RS-232	14	16 MHz

Tabla 2: Especificaciones Arduino Nano

Se utiliza esta pequeña placa para controlar el sensor de luz BH1750 y el sensor de humedad y temperatura DHT11 usando el lenguaje de programación de alto nivel C++ modificado. El Arduino Nano actúa como esclavo de la placa Jetson Xavier NX y solo proporciona datos al controlador principal cuando este se los solicita vía USB usando el protocolo de puerto serie RS-232.

Más adelanté en este TFM, se describen las conexiones con cada uno de los dispositivos dependientes de la placa Arduino Nano, los modos de comunicación entre ellos y los códigos en leguaje de programación de alto nivel C++ modificado desarrollados para cada uno de los sistemas integrados.

2.2.5 Sensor de luz BH1750

EL módulo BH1750 es un sensor de luz, que a diferencia del LDR es digital y nos entrega valores de medición en Lux (lumen /m²) que es una unidad derivada del Sistema Internacional de Unidades para la iluminancia o nivel de iluminación. Las especificaciones son las siguientes.

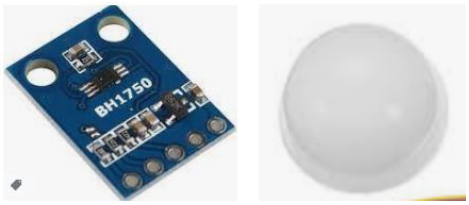

Ilustración 14: Sensor de luz BH1750 y cubierta

Sensor	Rango espectral	Resolución
BH1750	400-700 nm	1-65535 lux

Tabla 3: Especificaciones sensor BH1750

Tiene alta precisión y una resolución de entre 1 – 65535 lx. Se puede configurar con tres modos de trabajo descritos en la siguiente tabla.

Modo	Resolución	Tiempo de medición
High resolution Mode2	0.5 lx	120 ms
High resolution Mode	1 lx	120 ms
Llow resolution Mode	4 lx	16 ms

Tabla 4: Modos de configuración BH1750

De estas tres configuraciones se subdividen en dos, en modo CONTINUOUS y ONE_TIME, el primer modo realiza mediciones constantemente, el segundo modo no realiza medidas de forma constante. Cuando está configurada en modo ONE_TIME, el modulo se apaga después de realizar la medida y para volver a leer es necesario volver a configurar.

Para este TFM se ha optado por el modo de configuración *High resolution Mode CONTINUOUS MODE* la cual nos ofrece una buena resolución y el tiempo de medición suficiente para captar la luz en un instante determinado. Este sensor está controlado por el dispositivo Arduino Nano utilizando sus entradas digitales. Los pines de este sensor son los siguientes:

- VCC: Alimentación: 2.4 v – 5 v
- GND: Conexión de tierra
- SCL (Serial Clock Line): usada para proporcionar los pulsos de reloj del bus I2C
- SDA (Serial Data Address): usada para transferir los datos por el bus I2C
- ADDR: Direccion del dispositivo. Usado para seleccionar si hay más de un dispositivo en la misma entrada.

Más adelante en este TFM, se indicarán las conexiones y el modo de programación de este dispositivo. Además se ha colocado un difusor para evitar los reflejos directos que puedan perturbar o saturar la medida como se detalla más adelante en este TFM.

2.2.6 Cámara RGB ELP USB CAMERA

Esta cámara RGB captura los tres rangos de colores del espectro visible: rojo (650 nm), verde (550 nm) y azul (470 nm); los cuales complementan al sensor de luz BH1750 con tres medidas adicionales en tres longitudes de onda distintas. Estos tres colores son la composición del color en términos de la intensidad de los colores primarios de la luz, es decir, es posible representar un color mediante la mezcla por adición de cualquiera de ellos.

El brillo, el contraste, la saturación de color, la definición, el formato del archivo, la ganancia y el tiempo de exposición; son los parámetros ajustables de la cámara, sin embargo, las caracterizas más importantes para el desarrollo de este TFM son la ganancia y el tiempo de exposición dado que esta cámara va estar con su objetivo enfocando al cielo para poder capturar toda la luz incidente. Estos dos parámetros se deben de tener en cuenta y configurar antes de cada captura para obtener las longitudes de onda que corresponden al RGB sin que lleguen a saturar. Como se verá más adelante en este TFM, es necesario colocar un difusor de luz delante del objetivo para tratar de evitar problemas de saturación. Este dispositivo es compatible con Linux y está controlado por la placa Jetson Xavier NX vía bus USB.

Ilustración 15: Cámara RGB ELP USB CAMERA

2.2.7 Sensor de temperatura y humedad DHT11

Este sensor aporta las medidas de temperatura y humedad para medir las condiciones ambientales. Son dos medidas útiles que puede aportar mucha información a la hora de hacer la calibración de cámara hiperespectral dado que representan las condiciones ambientales al momento de realizar la captura. Estos pueden variar en función de la hora del día o la época del año.

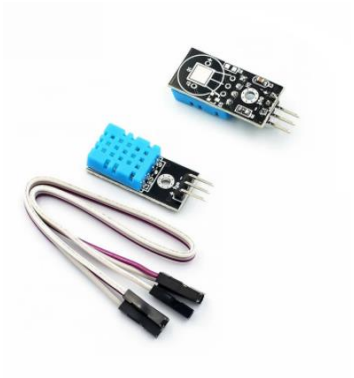

Ilustración 16: Sensor DHT11

Rango de medida	Precisión humedad	Precisión temperatura	Resolución
20 - 90%RH **0 – 50 ºC**	±5%RH	±2ºC	1

Tabla 5: Especificaciones DHT11

El fabricante recomienda hacer una medición de temperatura y humedad cada 10 segundos, esto ralentiza el proceso global de toma de datos a la espera de que cada iteración del programa tenga que esperar por este sensor. Para evitar este problema, se utilizan tres de estos sensores para crear redundancia y evitar posibles fallos o falsas medidas y acelerar el proceso. Junto con el sensor de luz

BH1750, se pueden obtener las condiciones climáticas necesarias para la base de datos. Este dispositivo es controlado por el Arduino Nano a través de sus salida/entrada digitales. Los pines son los siguientes:

- VCC: Alimentación: 2.4 v – 5 v
- GND: Conexión de tierra
- DATA: Salida de datos

2.2.8 Espectroradiometro STS-VIS Spectrometers

Este dispositivo capaz de medir longitudes de onda que van desde los 350 nm hasta los 800 nm se puede utilizar en aplicaciones que van desde mediciones de absorbancia de baja concentración hasta caracterización con láser de alta intensidad. Alguno de los ejemplos de aplicación se encuentra en los microscopios médicos, siendo capaz de obtener valores hiperespectrales en las células que se están analizando. En este TFM a modo de estudio y comparación, se configura este dispositivo para que solo capte las tres longitudes de onda que corresponden a los colores rojo (650 nm), verde (550 nm) y azul (470 nm). Esto nos permite comparar los resultados obtenidos por la cámara RGB con los este espectroradiomentro, que como se verá más adelante en este documento, será de utilidad para adquirir resultados que se puedan usar en un modelo de regresión.

Ilustración 17: Espectroradiometro STS-VIS

Rango espectral	Resolución óptica	Tiempo de integración
350nm – 800nm	1.5nm	10us – 10 s
	12.0nm	
	3.0nm	
	6.0nm	

Tabla 6: Especificaciones STS-VIS

Este dispositivo es compatible con Linux y está controlado por la placa Jetson Xavier NX vía bus USB. Más adelante en este TFM se explica por qué es necesario la implementación de este dispositivo.

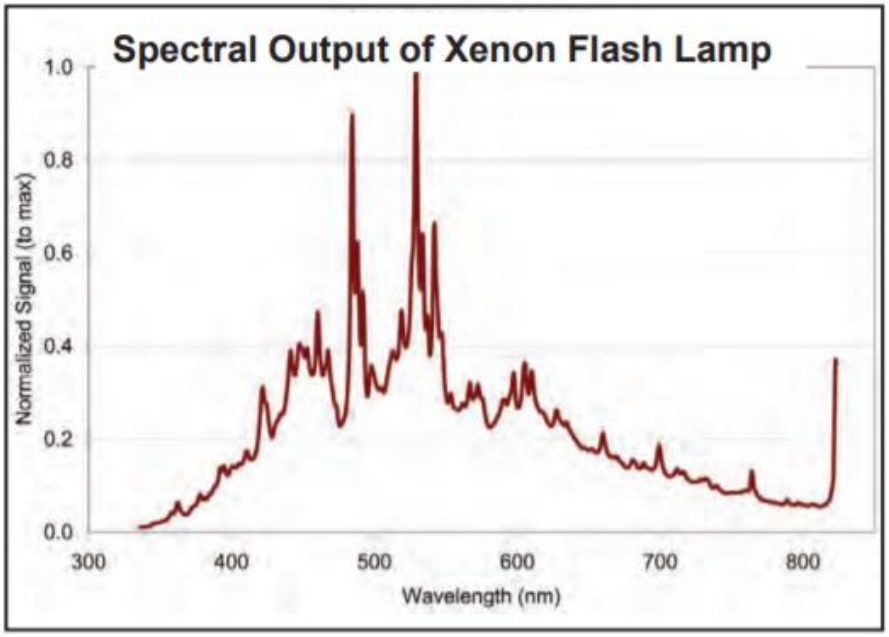

Ilustración 18: Respuesta espectral STS-VIS

2.3 Herramientas software

En este apartado se detallan las herramientas software utilizadas para el desarrollo de este TFM.

2.3.1 Arduino IDE

El software Arduino IDE (entorno de desarrollo integrado o IDE) es una aplicación multiplataforma (Windows, macOS, Linux) de código abierto que Se utiliza para escribir y cargar programas en placas compatibles con Arduino, pero también, con la ayuda de núcleos de terceros, se puede usar con placas de desarrollo de otros proveedores.

El código fuente para el IDE se publica bajo la Licencia Pública General de GNU, versión 2. El IDE de Arduino admite los lenguajes C y C ++ modificado utilizando reglas especiales de estructuración de códigos. El IDE de Arduino suministra una biblioteca de software del proyecto Wiring, que proporciona muchos procedimientos comunes de E/S. El código escrito por el usuario solo requiere dos funciones básicas, para iniciar el boceto y el ciclo principal del programa, que se compilan y vinculan con un apéndice de programa main() en un ciclo con el GNU toolchain, que también se incluye. El IDE de Arduino emplea el programa avrdude para convertir el código ejecutable en un archivo de texto en codificación hexadecimal que se carga en la placa Arduino mediante un programa de carga en el firmware de la placa. [28]

Esta herramienta software se ha utilizado para desarrollar el código que controla los sensores BH1750 y DHT11 además de gestionar el envío de la información vía puerto serie a la placa Jetson Xavier NX.

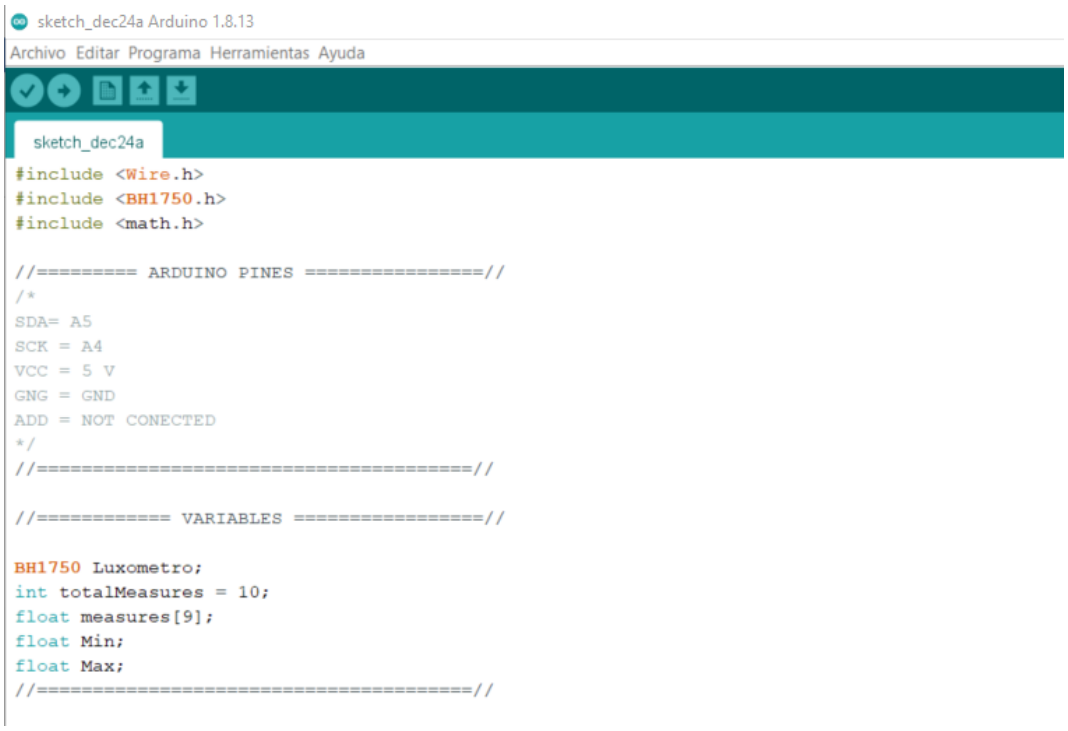

Ilustración 19: Arduino IDE

2.3.2 Pycharm

IDE de Python para el desarrollo de código de alto nivel. Proporciona análisis de código, depuración gráfica, integración con VCS / DVCS y soporte para el desarrollo web con Django. Una de sus principales ventajas es la gestión de los interpretes del código de alto nivel Python además de la asistencia y análisis de codificación, con finalización de código, sintaxis y resaltado de errores, integración delinter y arreglos rápidos. [29]

Se pueden integrar librerías para la fácil gestión de las funciones de las cámaras tales como v4l2 o aravis. Estas librerías se describen más adelante.

En esta herramienta software se desarrolla el cuerpo para crear una base de datos en datasets. Además se encarga de gestionar el sistema de archivos .json y .csv con los datos recogidos de las capturas. Este proceso se explicará más adelante en este documento.

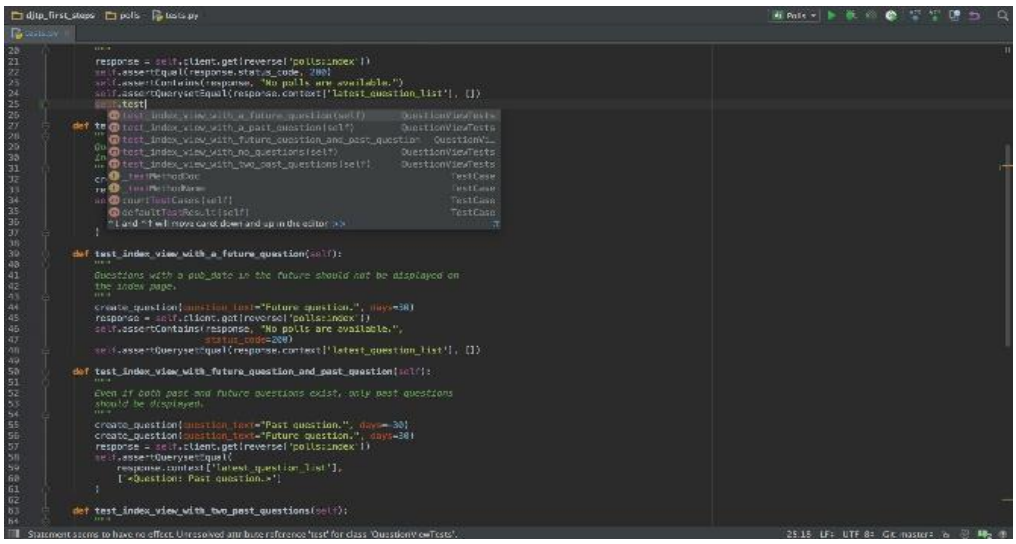

Ilustración 20: PyCharm

2.3.3 Kaggle

Es una subsidiaria de Google LLC. Es una comunidad en línea de científicos de datos y profesionales del aprendizaje automático. Kaggle permite a los usuarios encontrar y publicar conjuntos de datos, explorar y construir modelos en un entorno de ciencia de datos basado en la web. Esta aplicación es útil para desarrollar modelos de regresión lineal y otros modelos de Machine Learning o Deep Learning en condigo de alto nivel Python. [14]

Esta herramienta software se ha utilizado para desarrollar y entrenar el código de los modelos de regresión para obtener la referencia de blanco óptima según las condiciones ambientales usando la librería Scikit-learn. Esta es muy útil para el tratamiento de los datasets en lo que se refiere a limpieza, ordenación y mostrar datos en tablas o gráficas. Además, esta librería incluye los algoritmos de Maching Learning como la regresión logística o la regresión lineal necesarias para determinar la reflectancia adecuada según las condiciones ambientales. Se ampliará esta información más adelante en este documento.

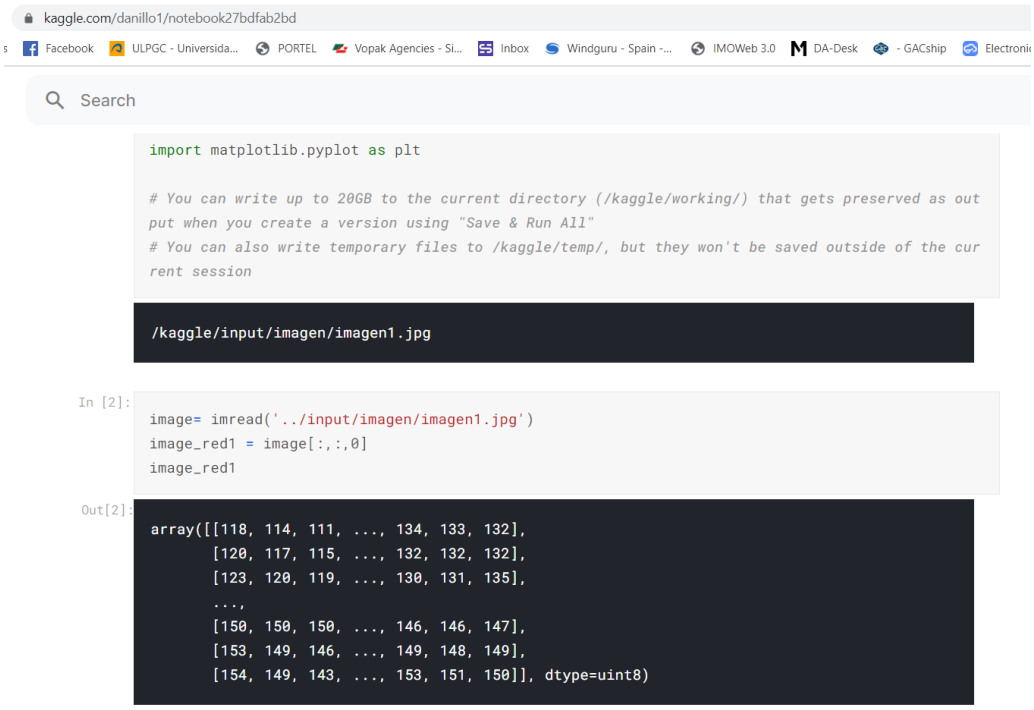

Ilustración 21: Kaggle

2.3.4 Zebra

Esta herramienta desarrollada por el IUMA es un software que permite visualizar los que está capturando la cámara hiperespectral. El funcionamiento principal de este programa se basa en leer el *ring buffer* la *shared memory* de sistema operativo LINUX para poder visualizar las imágenes. Este programa no está diseñado específicamente para las cámaras de Specim FX10 y FX17, es decir, cualquier cámara que escriba en este *ring buffer* puede utilizar este software para la visualización de las imágenes. Leyendo y escribiendo el ring buffer, las sucesivas imágenes se van recogiendo en un espacio de memoria RAM previamente reservado y que puede ser accedido desde diferentes programas. En este caso, accede tanto la cámara cuando almacena la información, como el código que la controla y el software de visualización Zebra. El hecho de que se almacenen los frames capturados de esta manera hace que no haya necesidad de que se guarden como archivos en memoria física no volátil lo que, a su vez, no permitiría la visualización instantánea de lo que está capturando la cámara en todo momento.

Entrando en detalles, un ring buffer es una estructura de tamaño fijo que depende del número de frames que va a almacenar, del número de píxeles totales de cada frame y del número de bits por pixel. Este tipo de memoria sigue un comportamiento FIFO, o lo que es lo mismo, el primero que entra al ring buffer es el primero en salir. Así, suponiendo un ring buffer de 10 elementos, se irán almacenando los frames en espacios de memorias sucesivos. Cuando el ring buffer se llene, se comenzarán a sobrescribir de nuevo comenzando por el más antiguo o primero en escribirse.

- Ring buffer vacío:

- Se comienzan a almacenar los frames:

Frame 1									

Frame 1	**Frame 2**								

Frame 1	**Frame 2**	**Frame 3**							

- **…**

Frame 1	**Frame 2**	**Frame 3**	**…**	**Frame 9**	**Frame 10**

- Una vez se llena el ring buffer, al seguir el comportamiento de una FIFO, el primer frame que fue escrito será el primero que se borre y así sucesivamente:

Frame 11	**Frame 2**	**Frame 3**	**…**	**Frame 9**	**Frame 10**

Frame 11	**Frame 12**	**Frame 3**	**…**	**Frame 9**	**Frame 10**

Esta capacidad de procesamiento le proporciona una gran flexibilidad al Zebra ya que no se ha desarrollado para un tipo de cámara concreto, sino que es independiente de la cámara a utilizar. Los únicos requisitos que tiene el Zebra para su utilización son:

- Ha de poderse controlar el que la cámara pueda almacenar las imágenes en una shared memory.
- Se ha de generar un archivo de configuración xml que contenga la información necesaria para que el Zebra pueda saber el formato y las características de los frames para componer las imágenes, así como el identificador de la shared memory utilizada. Este archivo es generado cuando se crea el objeto cámara para la Specim y guarda la configuración de las imágenes hiperespectrales capturadas y de la shared memory utilizada. De esta manera, cuando se abre el Zebra y se pulsa el botón 'Configuration' se te abre un navegador que apunta a la localización del archivo de configuración creado. Esto permite al Zebra leer los parámetros del xml y, por lo tanto, saber en qué dirección se localiza la shared memory en donde se almacenan las imágenes, número de píxeles de las imágenes, número de bits por pixel, endianess, etc. Por otro lado en ese archivo se inicializan tres tipos de shared memories:
    - shmIdentifier_CameraIsStreaming: En este elemento de la shared memory se almacena un booleano que señala si la cámara está realizando Streaming o no. Esto es necesario para que el Zebra permita la visualización de las imágenes capturadas por la cámara.

    - shmIdentifier_LastImageBufferIndex: Esta porción de shared memory indica el último índice del ring buffer en el que se almacenó un frame. Esto es necesario para que el Zebra posicione el puntero de lectura de la memoria en la posición correcta teniendo en cuenta el tamaño de cada frame (byteInRingBuffer = nRows*nCols*nBytesPerPixel*LastImageBufferIndex).

    - shmIdentifier_ImageBuffer: Esta memoria compartida contiene el ring buffer que almacena los frames según se van capturando.

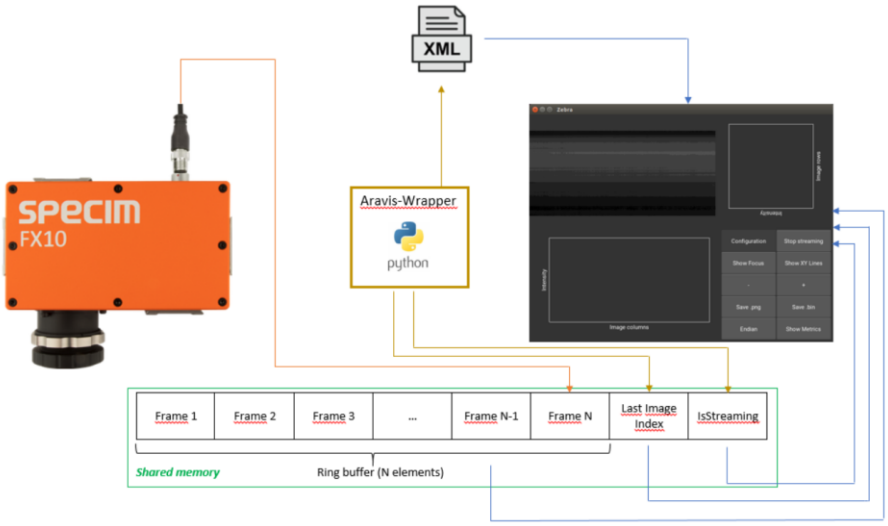

Ilustración 22: Esquemático de funcionamiento del Zebra

Capítulo 3. Desarrollo de la aplicación

3.1 Estructura general del sistema implementando la cámara FX10, el sensor de luz BH1750, los tres sensores de humedad/temperatura, la cámara RGB ELP USB, espectroradiometro STS-VIS, el Arduino Nano y la Jetson Xavier NX

El trabajo de campo plantea ciertos retos que deben de ser afrontados, no solo el hecho de no ser un entorno controlado como el de un laboratorio adaptado para la toma de imágenes hiperespectrales supone el máximo problema. Dada la cantidad de elementos necesarios para realizar capturas de datos, se hace complicado trasladar todo el equipo cada día a realizar pruebas en el exterior. Existe un gran riesgo de rotura, pérdida o robo de cualquiera de los equipos. Hay que incluir también que parte de sistema se quede en el laboratorio por error humano o descuido. Además, hay que tener en cuenta el factor del clima ya que no todos los días son óptimos para realizar las pruebas en exterior, la lluvia o el excesivo viento imposibilitan o dificultan la toma de muestras. Otro factor a tener en cuenta es la energía, en un trabajo de campo habitualmente no existe una toma de corriente de la red eléctrica que pueda proporcionar electricidad al sistema.

Para facilitar la toma de datos en el trabajo de campo se ha dispuesto una configuración que permite transportar todo el sistema al exterior de una sola vez sin tener que estar montando y desmontando cada uno de los sistemas que integra en cada momento que se haga una prueba. Además, se ha dispuesto de una batería LiPo para solventar el problema de la energía. La siguiente ilustración muestra de manera preliminar el sistema completo.

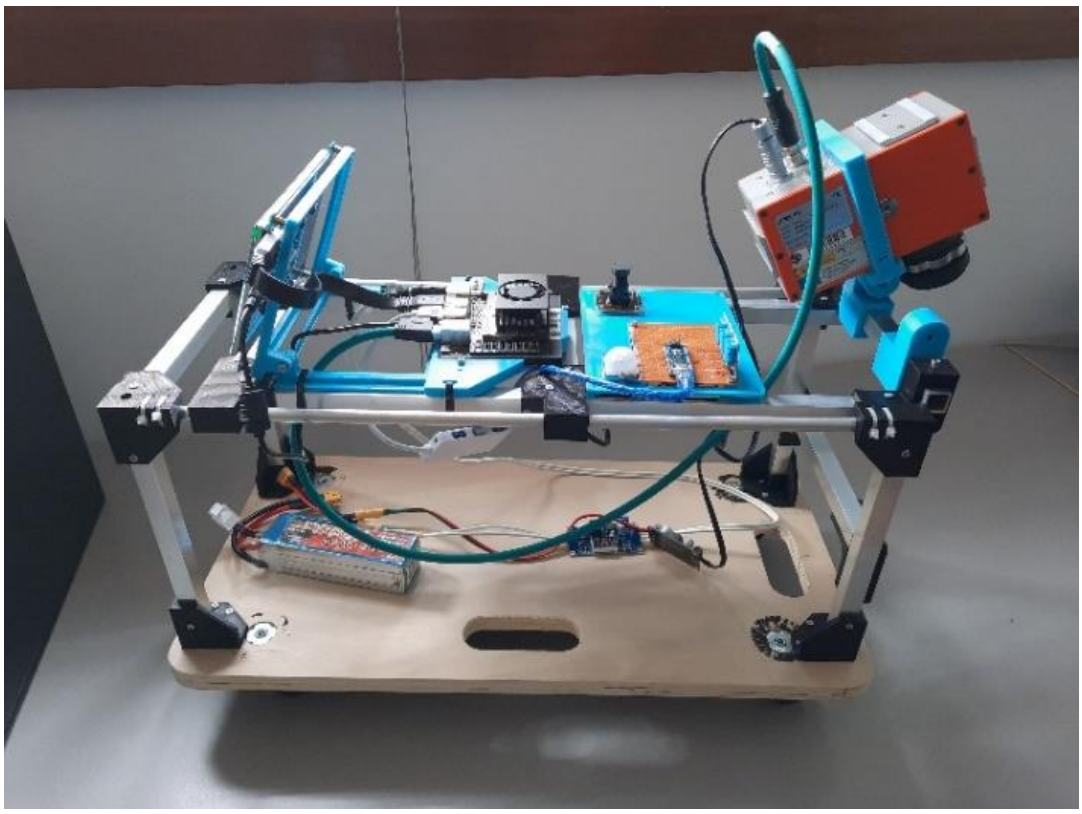

Ilustración 23: Equipo completo. De izquierda a derecha: Pantalla táctil, Jetson Xavier NX, Cámara ELP RGB, Sensor BH1750, Arduino Nano, Sensores DHT11 y Cámara SPECIM FX10. En la parte inferior y de izquierda a derecha: Bacteria de litio y convertidores de tensión BOOST/BUCK

Toda la estructura se ha desarrollado para el trabajo en el exterior y se han fabricado piezas de plástico para colocar cada elemento en su lugar.

Cada día el sistema se coloca en el exterior, en este caso, se posiciona en el exterior del edificio central del Parque Científico Tecnológico de la ULPGC para

realizar capturas de las condiciones ambientales en esa zona siempre teniendo en cuenta la previsión del tiempo y cómo puede afectar en los resultados que el día este soleado, nublado, lluvioso, etc. En un futuro, la zona o área donde realizar las pruebas podría ser un campo de cultivo o una zona que precise la toma de imágenes hiperespectrales. Siempre que se cambia de zona, hay que realizar estas pruebas para desarrollar una base de datos de cada área dado que no en todos los lugares tienen las mismas condiciones ambientales, factores como la humedad, los días soleados al año o las precipitaciones anuales entre otras determinan las condiciones ambientales del lugar. El resultado de esta técnica es un dataset de una zona determinada. En futuros estudios se podría plantear la opción de ampliar estas zonas por toda la orografía de la isla e incluso poder llegar a obtener un dataset lo suficientemente grande como para poder generalizarlo a la isla de Gran Canaria. Para poder llegar a este punto, la cantidad de datos tiene que ser lo suficientemente grande como para poder generar un modelo que converja y determine qué referencia de blanco es la adecuada en cada situación.

Es importante, realizar las pruebas siempre en las mismas franjas horarias y que estas sean a diferentes horas del día para poder obtener unos resultados que se puedan correlacionar y sean representativos de la zona a estudio. Una opción de toma de muestras podría ser la siguiente:

- Entre las 07:00 hrs y 08:00 hrs: Se capturan las condiciones ambientales a primera hora de la mañana. mínima intensidad de luz al amanecer.
- Entre las 12:00 hrs y 13:00 hrs: Horas centrales del día. Máxima intensidad de luz.
- Entre las 18:00 hrs y 19:00 hrs: Final del día. Mínima intensidad luz al anochecer.

En este TFM, la mayoría de las pruebas se han realizado en las horas centrales del día cuando el sol esta su zenit, de esta forma la intensidad de luz recibida en los sensores es máxima. Debido a la falta de tiempo en la realización, se han tomado unas capturas de prueba del sistema completo para después diseñar un modelo de regresión preliminar.

Por otro lado, la referencia de blanco Zenith Polymer debe colocarse en una posición que no haga sombra para que pueda reflejar toda luz incidente. En la siguiente ilustración se muestra un ejemplo de toma de datos.

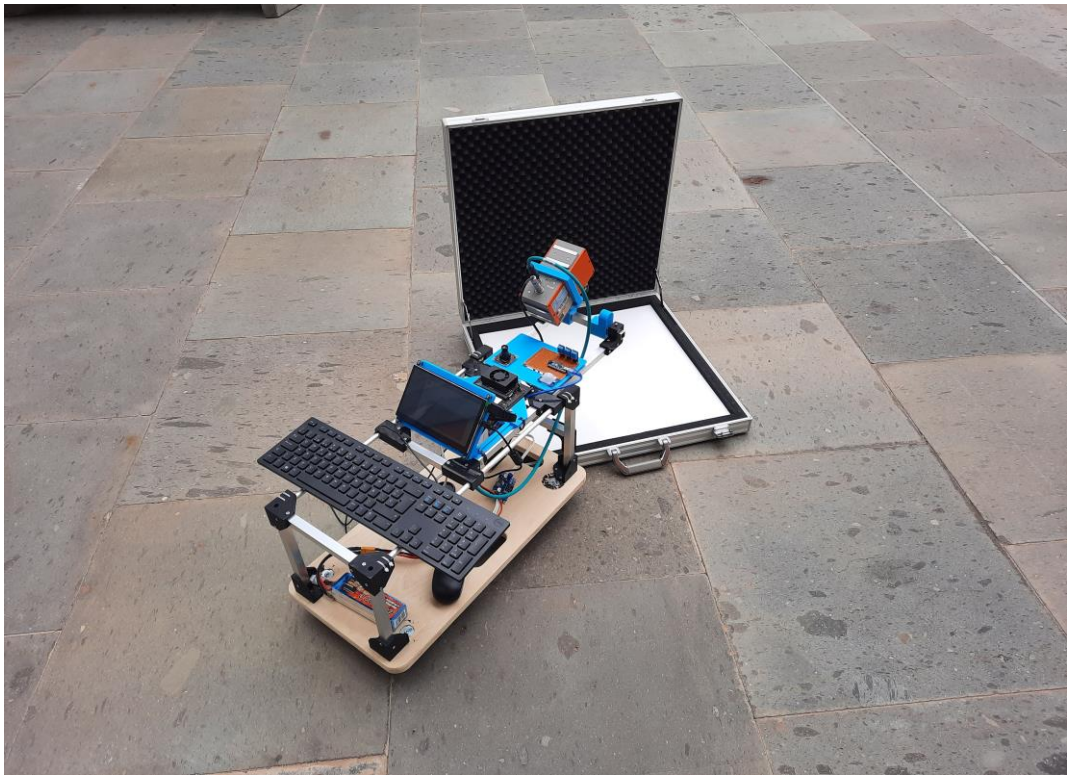

Ilustración 24: Equipo completo, toma de datos

Para poder empezar a capturar datos hay que asegurarse de que la cámara hiperespectral está apuntando directamente a la referencia de blanco y que no esté capturando el suelo, una sombra o el canto del maletín que transporta la referencia en sí misma. Para ello y utilizando el programa Zebra descrito en apartados anteriores, se centra en la pantalla la imagen de la referencia de blanco para captar toda la luz reflejada. En la siguiente imagen se puede ver que en la ventana de visualización del Zebra, la referencia obtenida es uniforme para todos los píxeles como se puede observar en la siguiente ilustración dentro del cuadro rojo. Esto indica que la cámara está apuntando al interior de la referencia de blanco en su totalidad.

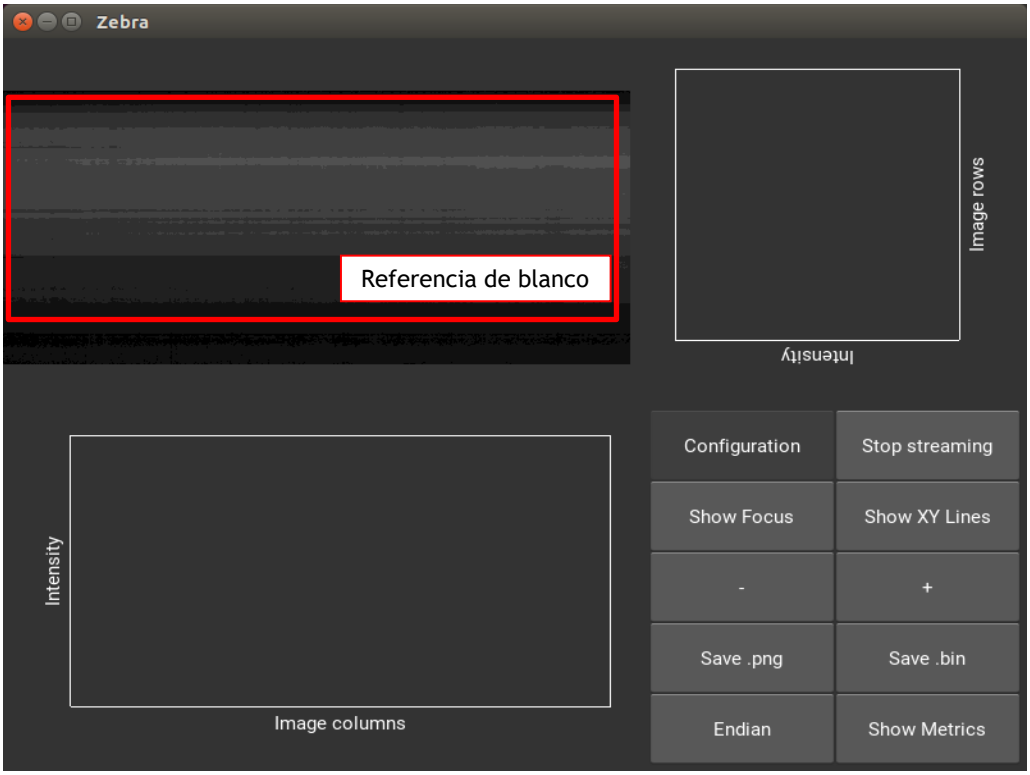

Ilustración 25: Imagen del programa Zebra apuntando a la referencia de blanco de manera correcta

Sin embargo, en la siguiente imagen se puede observar una discontinuidad en el centro de la ventana de visualización. Esto es debido a que se apuntó a una esquina de la referencia de blancos, es decir, la mitad de los píxeles apuntan a la referencia y la otra mitad al maletín y al suelo. Si se hubiesen obtenido las capturas de esa manera y sin comprobarlas, la referencia sería errónea y se obtendría un valor de reflectancia inútil para el estudio.

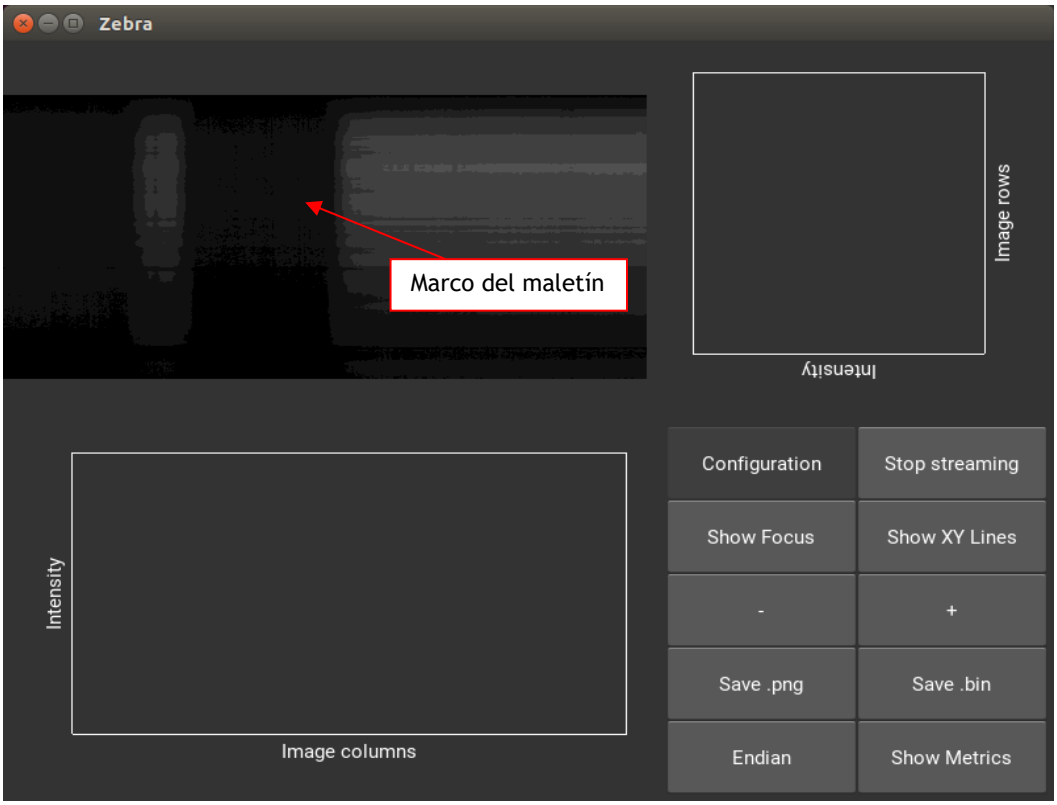

Ilustración 26: Imagen del programa Zebra apuntando a la referencia de blanco de manera incorrecta

Se ha centrado cámara y se ha asegurado de que se está tomando de manera correcta la referencia de blanco, se puede proceder a tomar datos de la intensidad de luz y las condiciones ambientales.

Tras las primeras pruebas de campo, se pudo observar que los datos tanto de la cámara ELP RGB y de cómo el sensor de luz BH1750 están saturados debido la exposición de luz directa. Aun con los tiempos de exposición y ganancia de la cámara ELP RGB al mínimo, con valores de 0 y 1 respectivamente, los datos siguen estando saturados. Para tratar de evitar este problema, se coloca un difusor en ambos dispositivos para refractar la luz y no crear reflejos indeseados. La siguiente ilustración muestra este proceso.

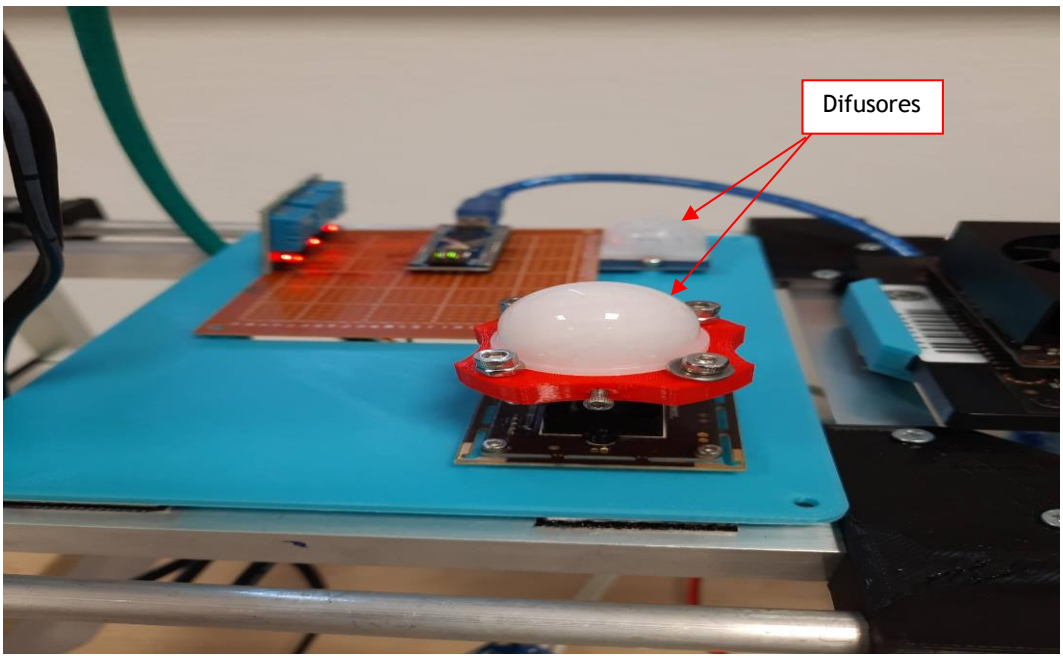

Ilustración 27: Cámara ELP RGB y sensor BH1750 con difusor

Aun teniendo en cuenta esta nuevas adaptaciones en la cámara ELP RGB y el sensor BH1750, los datos siguen estando saturados. Para tratar de solucionar definitivamente el problema, se podría optar por poner en cada una de los dispositivos unos filtros atenuadores calibrados para reducir la cantidad de luz directa. Esta parte del proyecto está en estudio y los filtros no están disponibles para realizar pruebas. Como alternativa, se ha optado por colocar el espectróradiometro STS-VIS Spectrometers para tratar de tomar datos no saturados.

Este dispositivo utiliza tecnología CMOS y puede capturar longitudes de onda de luz entre 350 y 800 nm. Tras hacer las primeras pruebas de campo se comprueba que los datos ya no saturan y pueden ser utilizados en la base datos. Este sensor tiene un coste elevado y no está prevista su utilización en nuestros drones pero ayuda a la compresión de los datos y a tomar muestras que puedan ser útiles para poder hacer la regresión prevista. Más adelante en este TFM se muestran los resultados obtenidos para todos los casos.

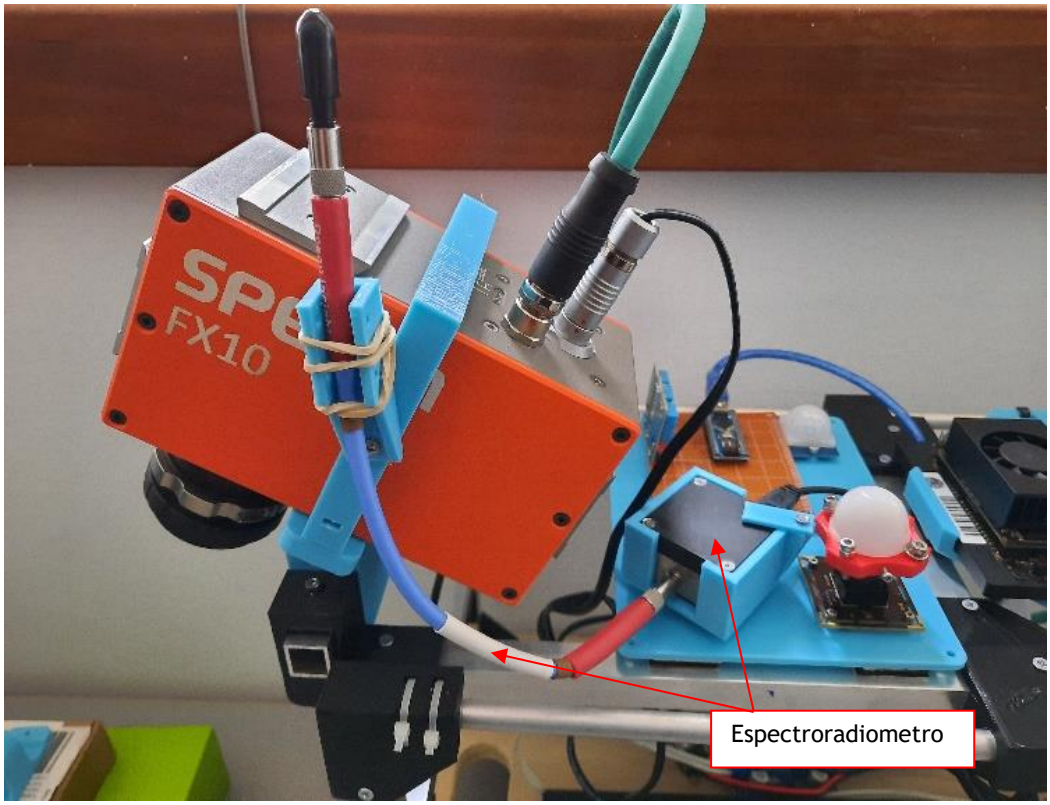

Ilustración 28: Specim FX10, STS-VIS, ELP RGB, BH1750 y DHT11

Por otro lado, algunos de los datos recogidos por la cámara Specim FX10 también aparecen saturados con ciertos niveles de ganancia y tiempos de exposición. Más adelante, se explica el proceso para limpiar los datos y utilizar solo aquellos que sean útiles para la regresión.

Finalmente al sistema se le incorpora una batería que alimenta todos los equipos. Esto elimina la necesidad de estar conectado a la red eléctrica haciendo el sistema independiente y trasladable a cualquier trabajo de campo. Como varios de los sistemas tienen voltajes de funcionamiento diferentes también se

incorporan convertidores de corriente continua (BOOST/BUCK) para aumentar/reducir la tensión. Más adelante en este TFM se muestran los esquemas de conexiones.

3.1.1 Esquema de conexiones

En este apartado se detalla el esquema de las conexiones para cada uno de los dispositivos descritos en este TFM. Estas conexiones deben de ser las mismas una vez que el equipo se monte en un dron de manera definitiva siempre que no existan cambios o adiciones al sistema.

- Jetson Xavier NX
  - USB1 Entrada/Salida: Puerto Serie RS-232 del Arduino Nano + Alimentación Arduino Nano
  - USB2 Entrada/Salida: Cámara ELP RGB + Alimentación de la cámara ELP RGB
  - USB3: Entrada/Salida: Cámara STS-VIS + Alimentación del sensor STS -VIS
  - Ethernet Entrada/Salida: Conexión GigE visión de la cámara FX10
- Arduino Nano
  - A4: BH1750 - SDA
  - A5: BH1750 – SCL
  - 5V: BH1750 - VCC
  - GND: BH1750 – GND
  - D4: DHT11 - DATA
  - D3: DHT11 - DATA
  - D2: DHT11 – DATA
  - 5V: DHT11 – VCC
  - GND: DHT11 – GND
  - USB1 Entrada/Salida: Puerto Serie RS-232 de la placa Jetson Xavier + Alimentación Arduino Nano

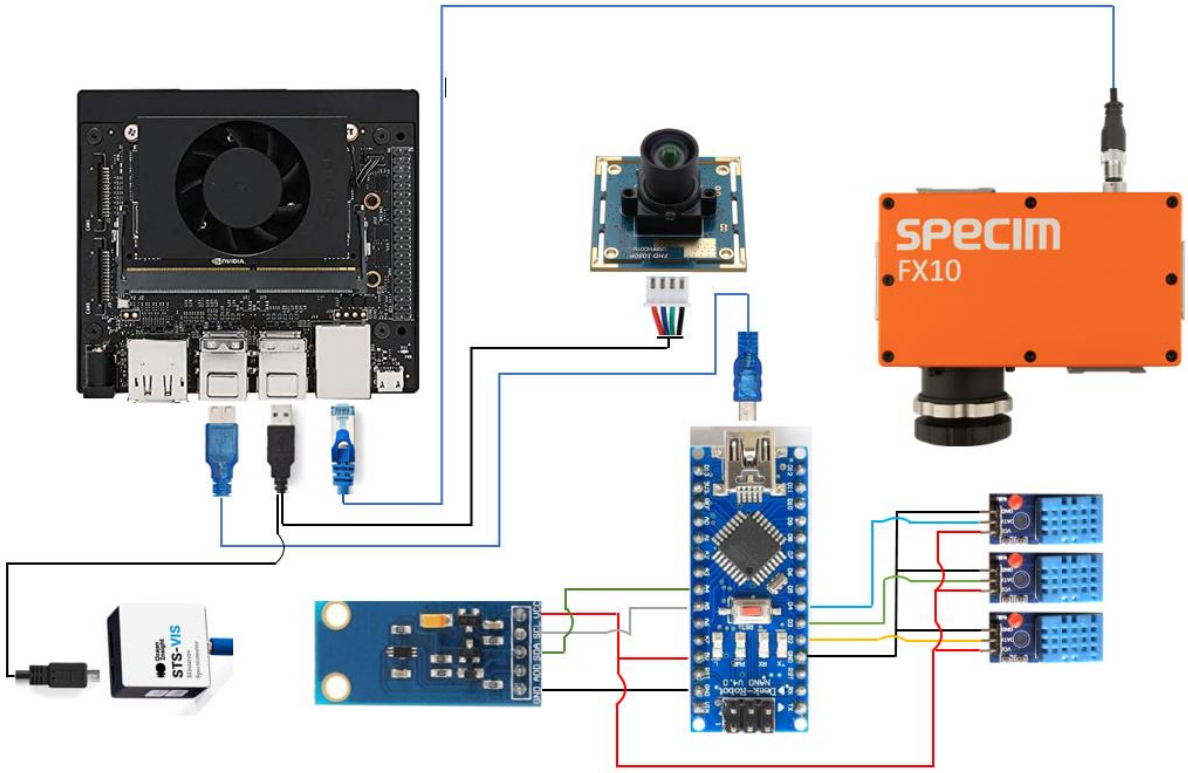

Ilustración 29: Esquema de conexiones

La conexión de la batería para alimentar al sistema completo se encuentra el siguiente esquema.

- Batería LiPo ZOP Power 11.1V 2200MAH 3S 25C
- Conector paralelo XT60
- Fischer Connector DBPLU1031Z012|130G: Alimentación de la cámara Specim Fx10
- Barril connector: Conector de alimentación para la Jetson Xavier NX
- HW-128 HC V0.1: Convertidor BOOST para alimentar la Jetson Xavier NX.
    - Voltaje de entrada: 4.5 V - 32 V

  - Voltaje de salida: 5 V - 55 V / Set a 19.5 V para alimentar a la Jetson Xavier Nx
  - Corriente de entrada: 4 A máximo
  - Eficiencia máxima: 94 %
- DK-SD15W40V-ADJ (Version: 2.0): Convertidor BUCK para alimentar la cámara Specim FX10
  - Voltaje de entrada: 4 V - 40 V
  - Voltaje de salida: 1.23 V – 37 V / Set a 12 V para alimentar a la cámara Specim FX10
  - Corriente de salida: 3 A máximo
  - Eficiencia máxima: 92 %

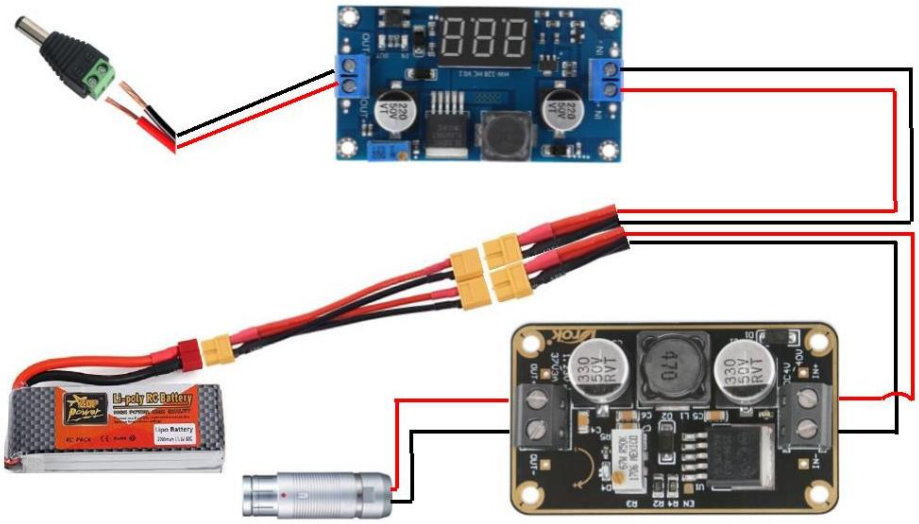

Ilustración 30: Esquema alimentación

Finalmente en la siguiente imagen se muestra el sistema completo con todos los dispositivos conectados.

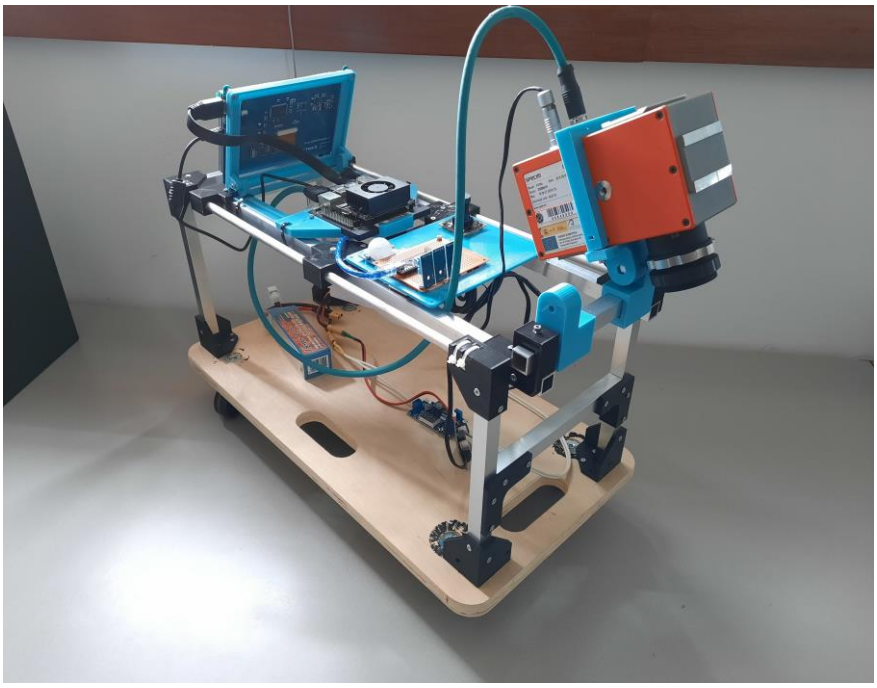

Ilustración 31: Esquipo completo. Parte inferior: bateria LiPo y convertidores BOOST/BUCK

3.2 Estructura general de las aplicaciones desarrolladas

Las aplicaciones se han desarrollado por separado para poder hacer modificaciones individuales sin tener que cambiar un programa completo. Esto le da al conjunto de programas solidez y a la vez lo hace escalable, es decir, se pueden ir añadiendo pequeñas modificaciones o nuevas aplicaciones según sean las necesidades. En este TFM, se han desarrollado cuatro aplicaciones: "Capturing", "JsonProgramme", "mainSerialCommunication" y "Linear Regresion". Además se han utilizado los siguientes programas ya desarrollados por el IUMA para la correcta implementación de los equipos:

- Zebra: Programa utilizado para la visualización de la cámara hiperespectral.
- DisplayWhiteRef: Pequeño programa desarrollado en lenguaje de programación de alto nivel Python que permite visualizar las gráficas de los datos hiperespectrales. Devuelve una gráfica con la firma espectral del objeto capturado, en el caso de este TFM, devuelve la firma espectral de la referencia de blanco Zenith Polymer.

El programa principal encargado de gestionar todo el sistema es "Capturing". Este llama a cada bloque por separado para chequear cada dispositivo, configurarlo, recopilar los datos y guardarlos en los ficheros. Se varían en un bucle las siguientes variables de configuración para capturar valores de los sensores con diferentes configuraciones.

- Camara Specim FX10:
  - Gain: 0.1 y 1.
  - ExposureTime: 1000 – 10000 microsegundos. Saltos de 1000 microsegundos

- Cámara ELP RGB: No se varían estos datos de configuración debido a que los datos que se recogen a valores mínimos están ya saturados.

  - Gain: 0. Sin ganancia.
  - ExposureTime: 1 microsegundo.
- Espectroradiometro STS-VIS:
  - IntegrationTime: 10, 200010, 400010, 600010 y 800010 microsegundos.

Los bloques individuales de los programas se estructuran de la siguiente forma:

- ELP_Camera: contiene las funciones para la configuración y guardado de datos recibidos por la cámara ELP RGB.
- AutomaticLuminanceCapturer: contiene las funciones para la configuración y toma de datos de los sensores BH1750 y DHT11 desde el Arduino Nano.
- AutomaticStreamingController: contiene las funciones de configuración y toma de datos de la cámara hiperespectral usando la librería Aravis-wrapper, esta librería se describe más adelante en este TFM. Este programa es válido tanto para la cámara Specim FX10 como para la Specim FX17.
- Spectroradiometer: contiene las funciones de configuración y toma de datos del espectroradiómetro STS-VIS.

Estos bloques han sido desarrollados por el IUMA a lo largo de los diferentes proyectos mencionados en este TFM (ENABLE S-3, APOGEO y PLATINO).

El completar un día de capturas, da como resultado un sistema de ficheros con los datos de los sensores BH1750 y DHT11, la cámara ELP RGB, la cámara hiperespectral Specim FX10 y el espectrómetro STS-VIS guardados de la siguiente forma:

- elpCamera.json y elpCamera.csv: contiene los datos recogidos por la cámara ELP RGB.
- GeneralSensors.json y GeneralSensors.csv: contiene los datos recogidos por los sensores BH1750 y DHT11.
- spectroradiometerInformation.json y spectroradiometerInformation.csv: contiene los datos recogidos por el espectroradiometro STS-VIS.
- hyperspectralImages_white.bin: Contiene los datos de reflectancia de la cámara Specim FX10 con un total de 229 376 datos (1024 pixeles x 224 bandas).
- hyperspectralImages_white.hdr: Contiene los datos de configuración e información de la cámara Specim FX10 tales como: nombre de la cámara, unidades de ancho de banda, numero de bandas, etc.

El programa "mainSerialCommunication" se desarrolla para el control del dispositivo Arduino Nano. Está escrito en el lenguaje de alto nivel C++ modificado y al ser la comunicación vía puerto serie no existe complicación a la hora de enviar y recibir datos entre la placa Jetson Xavier NX y el Arduino Nano. Carga las librerías de Arduino BH1750.h y DHT.h, configura el modo de trabajo de los sensores y obtiene los valores para ser enviados por el puerto serie usando el protocolo RS-232 a través del bus USB. Otra parte de este programa está escrito en el lenguaje de alto nivel Python. Esta parte se implementa bajo la placa Jetson Xavier NX y desarrolla las clases y funciones para la creación de un objeto para utilizar la cámara ELP RGB de manera sencilla.

El otro programa llamado "JsonProgramme", se encarga de recopilar la información de los ficheros creados por el programa "Capturing" y de crear un único dataset con toda la información de estos ficheros. Como se ha adelantado en apartados anteriores, muchos de los datos aparecen saturados debido a que

los sensores están apuntando directamente al cielo. Hay que añadir a este problema que algunos de los datos recogidos por la cámara Specim FX10 aparecen saturados con ciertos niveles de ganancia y de tiempos exposición. Para poder utilizar los datos recogidos hay que limpiar los ficheros .json originales y realizar un dataset con valores útiles para poder hacer un modelo de regresión. Tras realizar un estudio de los datos recogidos se ha podido filtrar los datos de la referencia de blanco y guardar todas aquéllas que se obtuvieron con Gain = 1 y ExposureTime = 3000 microsegundos para la cámara Specim FX10 y un IntegrationTime = 200010 microsegundos para el espectroradiómetro STS-VIS. Todos los datos tomados con diferentes configuraciones están saturados y no pueden ser incluidos en el dataset final. Hay que tener en cuenta que la calibración de la cámara hiperespectral hay que hacerla para cada uno de los 1024 píxeles disponibles en la cámara por separado, esto podría valer para una prueba de concepto. No obstante, en lugar de coger la referencia de blanco de solo un píxel (por ejemplo del pixel 512), se ha calculado la media de los 1024 píxeles, quedando pues el pixel medio con las 224 bandas que captura la Specim FX10. Sin embargo, de esas 224 bandas se han eliminado las primeras 20 bandas y las últimas 44 bandas, debido a su bajo Signal-To-Noise Ratio quedando finalmente 160 bandas disponibles. Por último, de las 160 bandas restantes se les hizo un smooth y un down-sampling, es decir, se calculó la media de cada 5 bandas (smooth) quedando solo una de cada 5 (down sampling). En resumen, el programa filtra todos los datos originales obteniendo finalmente en la salida un .json con una línea por cada captura que contiene una lista con los 32 elementos que resultan de hacer la media de cada 5 bandas obteniendo un total de 160/5=32. De esta manera se obtienen menos datos de salida que estimar al hacer la regresión.

Una vez terminada la secuencia de recopilación y filtrado de datos se obtiene un fichero .json con los datos finales. El programa convierte el formato .json a un formato .csv. Este formato es muy útil para los datasets y permite cargarlo en la plataforma Kaggle para poder realizar los estudios de regresión.

En resumen, el programa recoge todos los datos creados por el programa "Capturing" y crea la siguiente distribución de datasets en formato .csv:

1- elpCamera.csv + GeneralSensors.csv + hyperspectralImages_white.bin + hyperspectralImages_white.hdr: el .csv resultante incluye el R, G, B obtenido a partir de la cámara ELP RGB con un Gain = 0 y un ExposureTime = 1 para cada muestra recogida con Gain = 1 y ExposureTime = 3000 de la cámara hiperespectral Specim FX10. Además de los datos de humedad y temperatura recogidos por el sensor DHT11 y el valor de luminancia recogido con el sensor BH1750.
2- spectroradiometerInfo.csv + GeneralSensors.csv + hyperspectralImages_white.bin + hyperspectralImages_white.hdr: el .csv resultante incluye el R, G, B y brillo obtenido a partir del espectroradiómetro con un IntegrationTime = 20010 para cada muestra recogida con Gain = 1 y ExposureTime = 3000 de la cámara hiperespectral Specim FX10. Además de los datos de humedad y temperatura recogidos por el sensor DHT11. En este caso no se integra el valor de luminancia del sensor BH1750 dado que ya dispone del valor Brigthness recogido por el espectroradiómetro STS-VIS.

Por último, se ha diseñado el programa que realiza la regresión lineal. Este utiliza los datasets finales en formato .csv con los datos ya filtrados para determinar qué referencia de blanco es la adecuada en cada caso y en tiempo real según las condiciones ambientales. Para ello se utilizan los valores estadísticos de RMSE y SME que nos indicaran como de bueno es el modelo de regresión y si

puede ser utilizado para nuestro propósito. Esta parte se describe más adelante en este documento.

3.2.1 Librerías

En este apartado de describen las librerías utilizadas en el desarrollo del este TFM. Se describen también las funciones, métodos o clases más utilizadas. Todos estos métodos y clases serán descritos en los siguientes apartados conforme aparecen en el proceso de captura.

3.2.1.1 BH1750.h

Librería de Arduino que controla el sensor de luz BH1750. Es simple y robusta a la hora gestionar los comandos [21]. Las clases y los métodos utilizados son:

- Class BH1750
- bool begin(Mode mode = CONTINUOUS_HIGH_RES_MODE, byte addr = 0x23, TwoWire* i2c = nullpt)
- bool configure(Mode mode)
- float readLightLevel(bool maxWait = false);

3.2.1.2 DHT.h

Librería de Arduino que controla los tres sensores de temperatura y humedad DHT11. Es simple y robusta a la hora gestionar los comandos [20]. Las clases y los métodos utilizados son:

- Class DHT
- void begin(uint8_t usec = 55)
- float readTemperature(bool S = false, bool force = false)
- float readHumidity(bool force = false)

3.2.1.3 math.h

Librería de Arduino que incluye una gran cantidad de funciones matemáticas útiles para manipular números de coma flotante.

3.2.1.4 PIL

Librería que permite la edición de imágenes directamente desde Python. Soporta una variedad de formatos, incluidos los más utilizados como GIF, JPEG y PNG. Una gran parte del código está escrito en C, por cuestiones de rendimiento [19].

3.2.1.5 V4L2

La librería V4L2 es un controlador que proporciona uno o varios canales de video. Se utiliza para direccionar la salida de una aplicación de video a aplicaciones típicas de tratamiento de video (reproductores, codificadores, PVR) que son compatibles con dispositivos v4l2 [22]. Esta librería permite configurar todos los parámetros de la cámara ELP RGB:

- V4L2_CID_PIXEL_FORMAT: “MPEG” / (string): “MPEG” o “YUYV” value = “MPEG”
- V4L2_CID_EXPOSURE_AUTO_PRIORITY: 10094851 / (bool) : default=0 value=0
- V4L2_CID_EXPOSURE_AUTO": 10094849 / (menu): min=0 max=3 default=3 value=3
- V4L2_CID_EXPOSURE_ABSOLUTE": 10094850/ (int): min=1 max=5000 step=1 default=157 value=157 flags=inactive
- V4L2_CID_BRIGHTNESS": 9963776 / (int): min=-64 max=64 step=1 default=0 value=0
- V4L2_CID_CONTRAST": 9963777 / (int): min=0 max=64 step=1 default=32 value=32
- V4L2_CID_SATURATION": 9963778 / (int) : min=0 max=128 step=1 default=64 value=64
- V4L2_CID_HUE": 9963779/ (int) : min=-40 max=40 step=1 default=0 value=0
- V4L2_CID_AUTO_WHITE_BALANCE": 9963788 (bool) : default=1 value=1
- V4L2_CID_GAMMA": 9963792 / (int): min=72 max=500 step=1 default=100 value=100
- V4L2_CID_GAIN": 9963795 / (int): min=0 max=100 step=1 default=0 value=0
- V4L2_CID_POWER_LINE_FREQUENCY": 9963800 / (menu) : min=0 max=2 default=1 value=1
- V4L2_CID_WHITE_BALANCE_TEMPERATURE": 9963802/ (int): min=2800 max=6500 step=1 default=4600 value=4600 flags=inactive
- V4L2_CID_SHARPNESS": 9963803 / (int) : min=0 max=6 step=1 default=3 value=3
- V4L2_CID_BACKLIGHT_COMPENSATION": 9963804 7 (int) : min=0 max=2 step=1 default=1 value=1

3.2.1.6 Scikit-learn

Proporciona una gama de algoritmos de aprendizaje supervisados y no supervisados en Python además tratamiento de datos en datasets. [25] Las clases y los métodos utilizados son:

- LinearRegression (): Función para aplicar regresión lineal. Esta es una técnica de modelado estadístico que se emplea para describir una variable de respuesta continua como una función de una o varias variables predictoras.
- fit(): método para entrenar el modelo de regresión lineal.
- predict(): método para hacer predicciones con nuevos valores.
- mean_squared_error(): Error cuadrático medio, si el modelo está bien entrenado el RMSE estará cercano a 0.

3.2.1.6.1 NumPy

Librería de Python que pertenece a Scikit-learn especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos. Los objetos arrays permiten representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación [23].

3.3.1.6.2 Pandas

Librería de Python que pertenece a Scikit-learn especializada en el manejo y análisis de estructuras de datos. Es muy útil para el leer y escribir archivos en formato CSV en cual se guardan los datos de las capturas [24].

3.2.1.7 Aravis-wrapper

La librería Aravis está basada en glib / gobject para la adquisición de video usando cámaras Genicam. Actualmente implementa los protocolos gigabit Ethernet (GigE vision) y USB3 utilizados por las cámaras industriales. También proporciona un simulador de cámara ethernet básico y un visor de video simple. El IUMA ha desarrollado un wrapper que permite el entendimiento entre los lenguajes de programación Python, que es el lenguaje de programación de alto

nivel de todas las aplicaciones desarrolladas en este TFM, y el lenguaje C, que es el lenguaje de programación de alto nivel del controlador de las cámaras hiperespectrales. Las funciones, métodos, clases, etc que se crean en Python y esta librería interpreta y enlaza con los elementos del código en C. Las clases y métodos más importantes son:

- JumboPacketsUnset(Exception): controla las ip de la interfaz GigE vision de las cámaras hiperespectrales.
- UnexistenGivenIPAddress(Exception): Comprueba si la cámara hiperespectral está conectada.
- Camera(Specim): Crea el objeto para controlar la cámara hiperespectral.

3.2.1.8 Matplot

Librería para generar gráficas a partir de datos contenidos en listas, vectores, en el lenguaje de programación Python y en su extensión matemática NumPy. Algunos de sus métodos principales son:

- Plot(): representa las graficas.
- Subplot():crea una figura y una cuadrícula de subtramas con una sola llamada.
- Fill(): rellena la gráfica con posibilidad de poner colores.
- Xlabel(): Define el nombre del eje x.
- Ylabel():Define el nombre del eje y.

3.3 Funciones principales

En este apartado se describen las funciones más importantes de los programas desarrollados.

3.3.1 Funciones principales programa Capturing

Como se ha descrito en apartados anteriores, este programa se puede considerar como el principal puesto que se encarga de controlar todos los dispositivos para obtener los datos de cada uno de ellos y guardarlos en el sistema de ficheros. Además permite al sistema conocer que cámara hiperespectral se está utilizando para toma de datos la Specim FX10 o la Specim FX17. Para poder llegar a este punto, se implementan en el programa los siguientes métodos y clases importantes:

- class Camera(0, nCols, nRows, pixelFormat): Crea un objeto de la clase Camera que controla la cámara ELP RGB. Directamente en la definición del objeto, se puede configurar el dispositivo, el número de columnas y filas que tiene la captura y el formato de la imagen RGB.
- class serial.Serial(port, baudRate, timeout=5): crea un objeto de la clase Serial que controla todo lo relativo al puerto serie. Es muy útil para el envío de pequeños mensajes entre la placa Jetson Nano NX y el Arduino Nano.
- def parsingHeaderFile(stringToFind, limitString, headerFilePath): Esta pequeña funcion sirve para analizar el archivo .hdr y comprueba que la imagen hiperespectral está correctamente guardada en el sistema de ficheros.
- CameraClass.Camera.get_available_devices(): Obtiene las cámaras hiperespectrales conectadas disponibles. Esta función pertenece a la libreria Aravis-wrapper.
- CameraClass.start_camera_device(connectedDevices[0], SUDO_PASSWORD): inicia la cámara hiperespectral conectada

seleccionada por la función anterior. Esta función también pertenece a la libreria Aravis-wrapper.

- camera.start_streaming(): inicia la captura de frames hiperespectrales.
- camera.stop_streaming(): termina la captura de frames hiperespectrales cuando se han conseguido los deseados.

El código completo de este programa se describe en los anexos de este documento.

3.3.2 Funciones principales programa JsonProgramme

Este programa es en sí mismo una única función que gestiona los ficheros .json creados y guardados por el programa "Capturing". Con los datos obtenidos de la capturas, este programa desarrolla una única base de datos en formato .json y .csv que facilita el manejo de los datos en la plataforma Kaggle. Este programa se describe en el Capítulo 6 de este TFM. Para poder llegar a este punto, se implementan en el programa los siguientes métodos importantes:

- Bucle FOR dailyDirectory: Este bucle recorre los días de capturas y los selecciona para leer los valores de los sensores.
- Bucle FOR directory: Este bucle recorre las diferentes capturas hechas en un día.
- multibandread(selectedHyperspectralImageFilePath, dimArray, dataType, interleaf, endianness): guarda en un array los 1024 x 224 valores de la referencia de blanco capturada.
- numpy.mean(WR, axis=1): hace la media de todos los pixeles quedando 224 valores.
- pd.read_json (outputFilePath): Lee el fichero .json con los datos guardados.

El código completo de este programa se describe en los anexos de este documento.

3.3.3 Funciones principales programa mainSerialCommunication

Este programa controla los sensores BH1750 y DHT11 implementados en la placa Arduino Nano para obtener la intensidad de luz, la temperatura y la humedad respectivamente. Se ha desarrollado sobre la herramienta software Arduino IDE. Para poder llegar a este punto, se implementan en el programa los siguientes métodos importantes:

- float getLuminanceValue(): Obtiene el valor de intensidad de luz y devuelve la media aritmética de 20 muestras tomadas.
- float getTemperature(): Obtiene el valor de la temperatura. Al haber tres sensores DHT11 para crear redundancia, se hace la medida aritmética de cada sensor por separado y luego se hace la media de las medias de los tres los sensores.
- float getHumidity(): Obtiene el valor de la humedad. Al haber tres sensores DHT11 para crear redundancia, se hace la medida aritmética de cada sensor por separado y luego se hace la media de las medias de los tres los sensores.

El código completo de este programa se describe en los anexos de este documento.

3.3.4 Diagramas de flujo

En este apartado se describen los diagramas de flujo de los tres programas descritos en el apartado 3.3 desarrollados en este TFM. En estos diagramas se muestra el proceso de captura de datos, obtener el nivel de reflectancia y la creación de la base de datos en formato .csv.

3.3.4.1 Diagrama de flujo Capturing

El siguiente diagrama de flujo muestra el proceso automático para realizar las capturas de referencia de blanco con la cámara hiperespectral Specim FX10 y los sensores BH1750, DHT11 y la cámara ELP RGB. Los datos recopilados se guardan en fichero siguiendo una estructura de datos definida, esta parte de jerarquía de ficheros se explicara más adelante en este TFM. En este apartado se explica cada proceso, bucle y definición de variables. Además se detallan las funciones que se ejecutan en cada uno de los procesos.

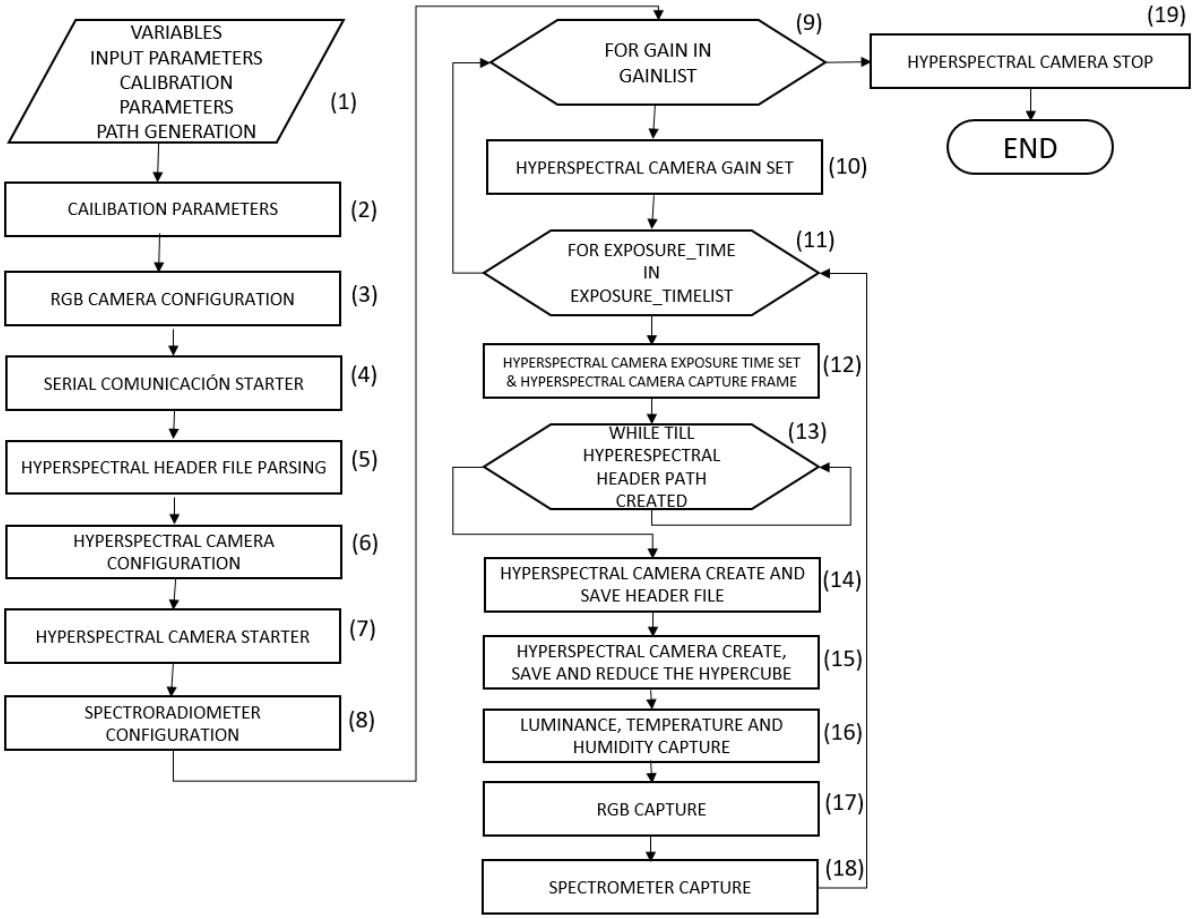

Ilustración 32: Diagrama de flujo de la aplicación Capturing

1- Este proceso engloba varios apartados:
   a. Se definen las variables necesarias para las rutas de los archivos de configuración de las cámaras y los sensores o de los archivos con la salida de los resultados finales.
   b. Se definen las variables necesarias para las cámaras tanto las Specim FX10, STS-VIS y la cámara ELP RGB. Ganancia Specim FX10 = 0.1 - 15.999, Exposure_Time Specim FX10 =1000 - 10000, Ganancia ELP RGB = 0 - 0, Exposure_Time RGB =1 -1. La ganancia y el tiempo de exposición de la cámara ELP RGB permanece constante al mínimo valor posible debido a

que a los valores están saturados para estos, aumentarlos solo saturaría aún más los datos recogidos. Esto se desarrolla más adelante en este TFM.

2- Proceso para dar valores a los parámetros de calibración de la Ganancia y el Exposure_time de la cámara Specim FX10 y la cámara ELP RGB.
3- Proceso que configura la cámara ELP RGB usando los archivos de configuración. Uso del bloque ELP_Camera utilizando su clase principal Camera(0, nCols, nRows, pixelFormat).
4- Se configura y se abre el puerto serie para la comunicación con el Arduino Nano.
5- Función para analizar el fichero .hdr de la cámara Specim FX10.
6- Proceso que crea los ficheros de configuración de la cámara y donde se guardan todos los datos.
7- Comienza a realizarse la captura hiperespectral.
8- Configura e inicia el espectroradiometro STS-VIS.
9- Primer buque “for” que recorre la lista de ganancias configuradas en el proceso (1).
10- Se selecciona el valor de la ganancia.
11- Segundo buque “for” que recorre la lista de tiempos de exposición configurados en el proceso (1).
12- Se selecciona el valor del tiempo de exposición, se crea el directorio final para guardar el valor de reflectancia y captura un frame.
13- Bucle “while” que espera y crea el fichero .hdr con la configuración de la cámara a la hora de se capturar un frame.
14- Se guarda el fichero .hdr.
15- Se calcula la referencia de blanco promedio.
16- Se crean los ficheros para los sensores y la cámara ELP RGB. Se capturan los valores de intensidad de luz, temperatura y humedad de los sensores BH1750 y DHT11.
    a. Formato recibido en la placa Jetson Xavier NX: luminance = 20135.0 temp = 19.67; humid = 65.33
    b. Formato guardado en el archivo GeneralSensors.json: {"luminance": 20135.0, "temperature": 19.67, "humidity": 65.33}

17- Se capturan los valores de intensidad de luz para los colores rojo (650 nm), verde (550 nm) y azul (470 nm) obtenidos de la camara ELP RGB.
    a. Formato guardado en el archivo elpCamera.json: {"0": {"1": {"R": 249.26, "G": 255.0, "B": 253.0}
18- Se capturan los valores de intensidad de luz para los colores los colores rojo (650 nm), verde (550 nm) y azul (470 nm) obtenidos del espectroradiometro STS-VIS.
19- Una vez terminado el proceso (9), se para el proceso de capturas y el programa termina.

3.3.3.2 Diagrama de flujo JsonProgramme

El siguiente diagrama de flujo muestra el proceso automático que recoge todos los datos de los ficheros creados por el programa "*Capturing*" y crea un solo dataset con todos los números recopilados. Como se ha adelantado en apartados anteriores, el resultado final de este dataset es reducir el número de estos datos eliminando los posibles resultados saturados y otros datos que no puedan intervenir en la regresión lineal, como por ejemplo, el modelo de la cámara, la fecha o la ganancia. Previo a ejecutar este programa se han analizado los datos para seleccionar la configuración óptima de la captura de datos.

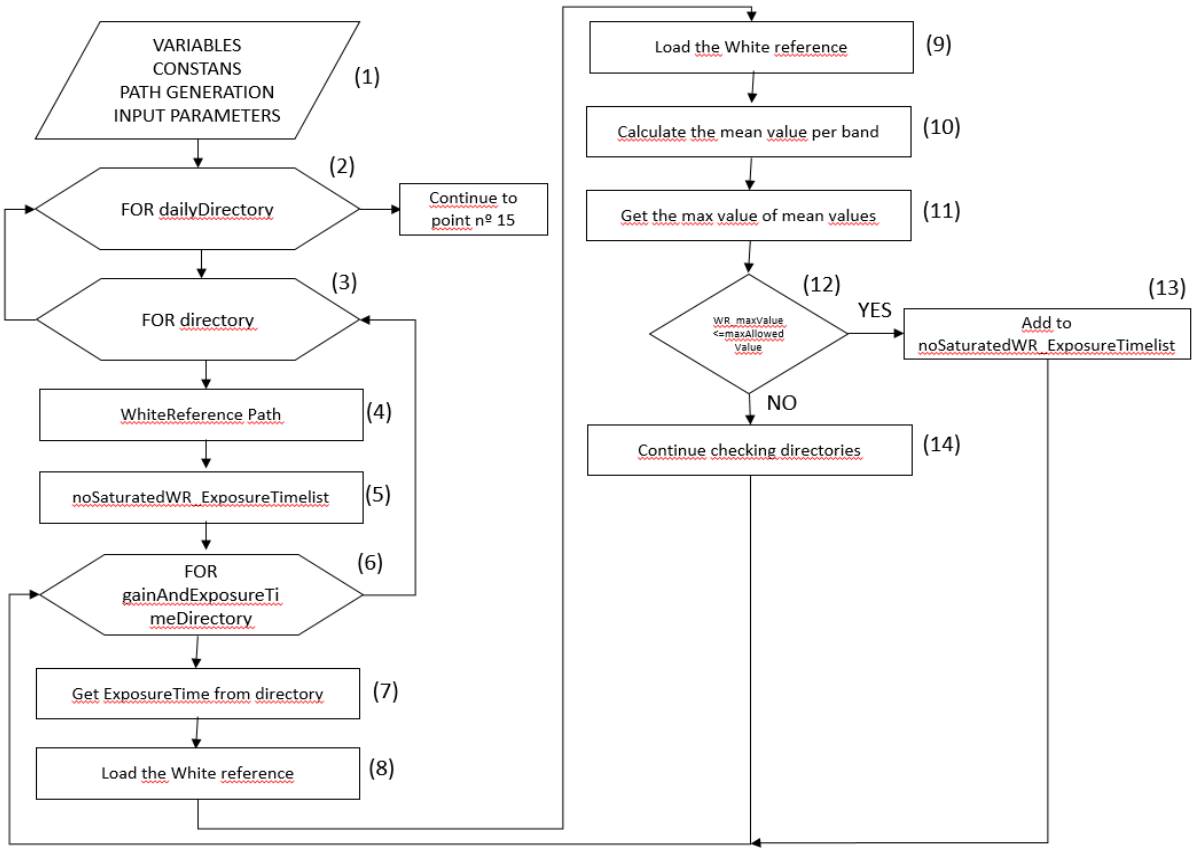

Ilustración 33: Diagrama de flujo de la aplicación JsonProgramme - Parte 1

1- Este proceso engloba varios apartados:
    a. Se definen las variables necesarias para las rutas de los archivos con los resultados de las cámaras y los sensores y la cámara hiperespectral usada
    b. Se definen las variables necesarias para las cámaras tanto la Specim FX10 como la FX17: Bandas, rango dinámico, número de pixeles, etc.
2- Bucle FOR que recorre los ficheros de los días de capturas.
3- Bucle FOR que recorre todas las capturas tomadas en un día.
4- Selecciona la referencia de blanco de la captura.
5- Crea una lista de datos de referencias de blanco que no estén saturados según cada tiempo exposición.
6- Bucle FOR que recorre cada carpeta con el Gain y el Exposure_Time guardados.
7- Selecciona el directorio del Exposure_Time.
8- Carga la referencia de blanco con el Exposure_Time seleccionado.
9- Carga la referencia de blanco con el Exposure_Time seleccionado.
10- Calcula el valor medio por cada banda. 224 resultados en total.
11- Obtiene el valor máximo de las medias.
12- Condicional IF, si el valor medio máximo es menor o igual que el valor máximo permitido se añade a la lista del punto nº5, si no, sigue buscando.
13- Añade la lista del punto nº5.
14- Continúa buscando.

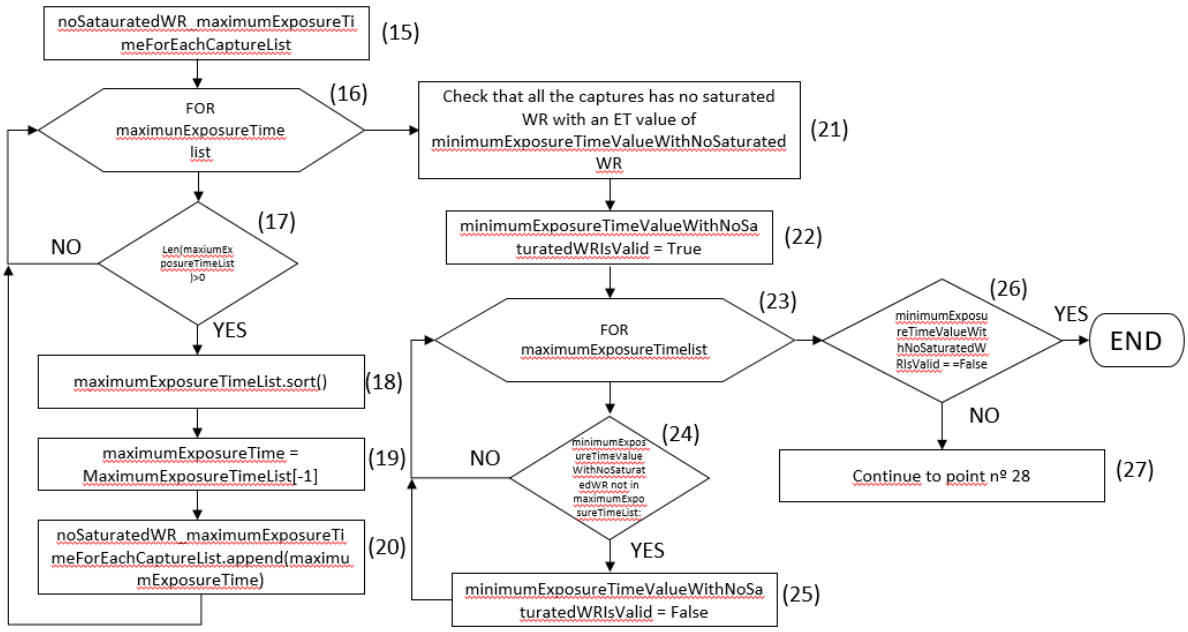

Ilustración 34: Diagrama de flujo de la aplicación JsonProgramme - Parte 2

15- Lista de Exposure_Time máximo para referencias de blanco no saturadas para en cada captura.
16- Bucle FOR que recorre la lista del punto nº5.
17- Condicional IF, si el valor dentro de la lista es mayor que 0 continua, si no, sigue buscando.
18- Ordena la lista de menor a mayor.
19- Obtiene el valor máximo de Exposure_Time.
20- Añade a la lista creada en el punto nº15.
21- A partir de este punto se empieza a chequear la lista del punto nº15 para comprobar que valor máximo de Exposure_Time existe para cada referencia de blanco sin saturar.
22- Variable booleana para comprobar si existe un valor máximo de Exposure_Time para cada referencia de blanco sin saturar.
23- Bucle FOR para comprobar la lista del punto nº15.
24- Condicional IF, si el valor mínimo está en la lista continua, si no, la variable booleana del punto nº22 se iguala a FALSE.

25- minimumExposureTimeValueWithNoSaturatedWRIsValid = False.
26- Condicional IF, si la variable del punto nº22 es igual a TRUE el programa continua, si no, el programa acaba con el siguiente error: 'ERROR: EXPOSURE TIME SELECTED IS NOT VALID'
27- El programa continua en el punto nº28.

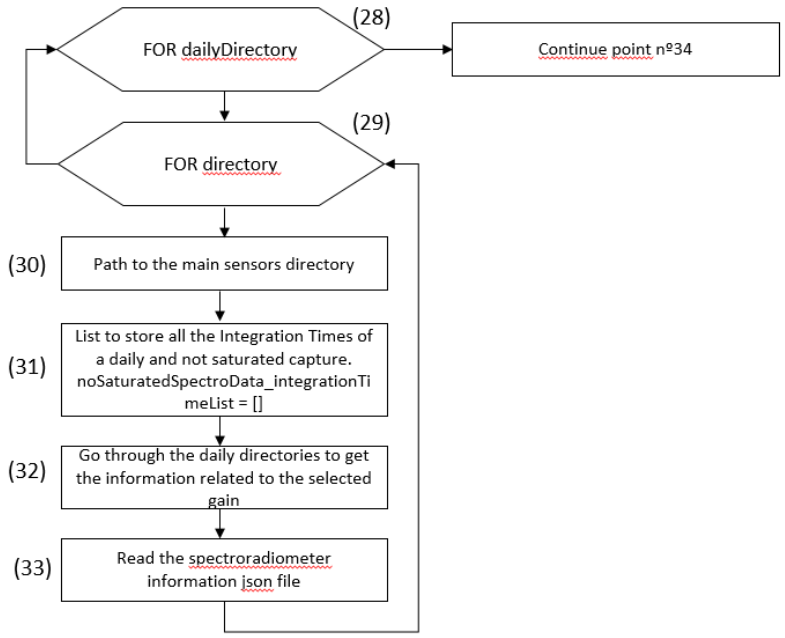

Ilustración 35: Diagrama de flujo de la aplicación JsonProgramme - Parte 3

28- Bucle FOR que recorre los ficheros de los días de capturas.
29- Bucle FOR que recorre todas las capturas tomadas en un día.
30- Obtiene la dirección del fichero de los sensores.
31- Se crea la lista para almacenar los valores de Integration_Time del espectroradiómetro. noSaturatedSpectroData_integrationTimeList = []
32- Se obtiene la información de los ficheros según la ganancia seleccionada en el bucle del punto nº29.
33- Lee los datos obtenidos del espectroradiómetro.

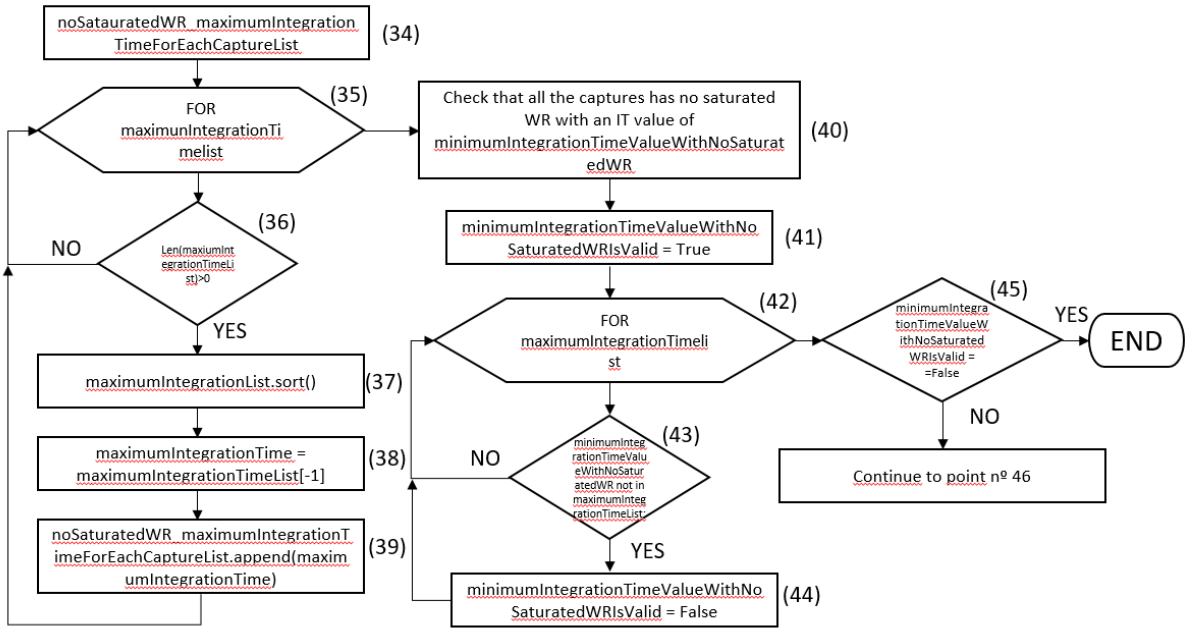

Ilustración 36: Diagrama de flujo de la aplicación JsonProgramme - Parte 4

34- Lista de Integration_Time máximo para referencias de blanco no saturadas para en cada captura.
35- Bucle FOR que recorre la lista del punto nº35.
36- Condicional IF, si el valor dentro de la lista es mayor que 0 continua, si no, sigue buscando.
37- Ordena la lista de menor a mayor.
38- Obtiene el valor máximo de Integration_Time.
39- Añade a la lista creada en el punto nº35.
40- A partir de este punto se empieza a chequear la lista del punto nº15 para comprobar que valor mínimo de Integration_Time existe para cada referencia de blanco sin saturar.
41- Variable booleana para comprobar si existe un valor máximo de Integration_Time para cada referencia de blanco sin saturar.
42- Bucle FOR para comprobar la lista del punto nº15.
43- Condicional IF, si el valor máximo está en la lista continua, si no, la variable booleana del punto nº41 se iguala a FALSE.
44- minimumIntegrationTimeValueWithNoSaturatedWRIsValid = False.

45- Condicional IF, si la variable del punto nº41 es igual a TRUE el programa continua, si no, el programa acaba con el siguiente error: 'ERROR: INTEGRATION TIME SELECTED IS NOT VALID'

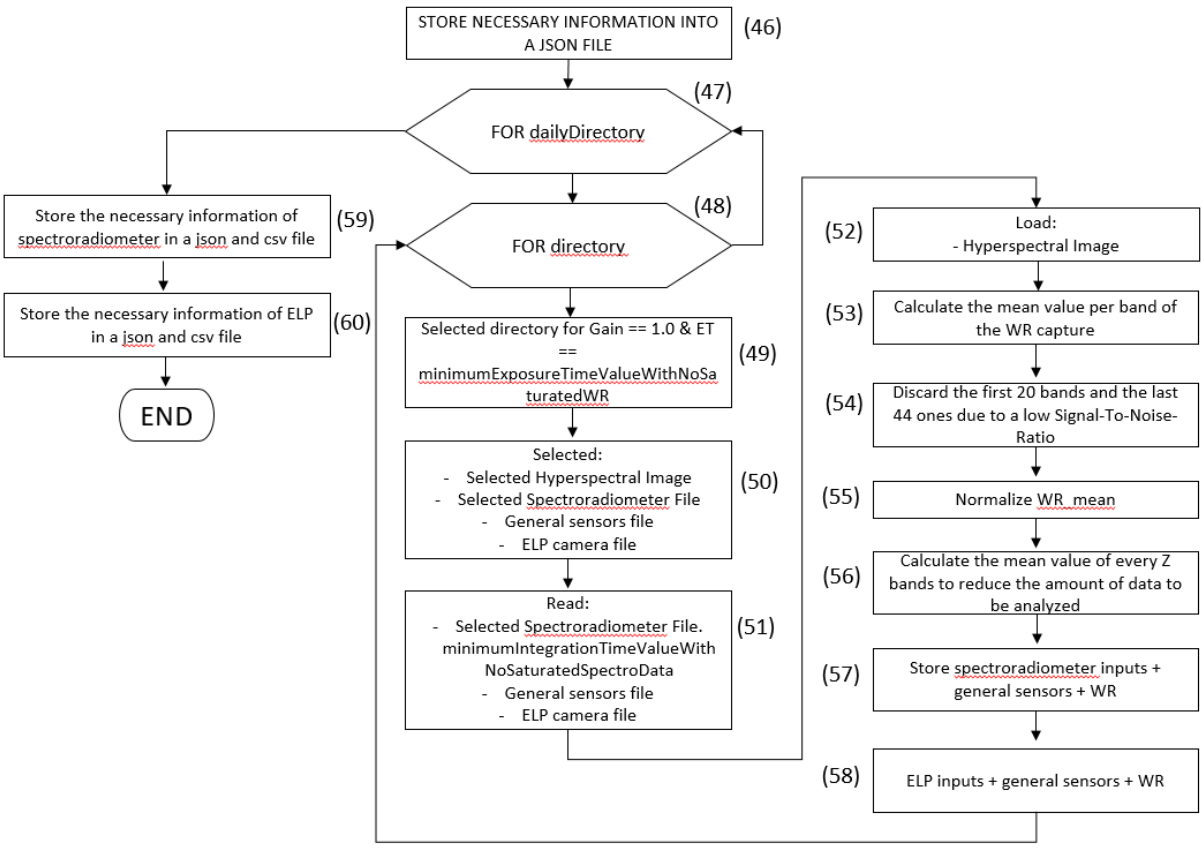

Ilustración 37: Diagrama de flujo de la aplicación JsonProgramme – Parte 5

46- A partir de este punto se inicia el proceso de recopilación y limpieza de datos para guardarlos en un único dataset.
47- Bucle FOR que recorre los ficheros de los días de capturas.
48- Bucle FOR que recorre todas las capturas tomadas en un día.
49- Se selecciona el directorio correspondiente al Exposure Time mínimo valido.
50- Selecciona:
    a. Fichero de la imagen hiperespectral.

   b. Fichero de valores del espectroradiómetro STS-VIS.
   c. Fichero de valores de los sensores BH1750 y DHT11.
   d. Fichero de valores de la cámara ELP RGB.

51- Lee:
   a. Fichero de valores de espectroradiometro STS-VIS. Tiene en cuenta los valores mínimos de la lista minimumIntegrationTimeValueWithNoSaturatedSpectroData.
   b. Fichero de valores de los sensores BH1750 y DHT11.
   c. Fichero de valores de la cámara ELP RGB.
52- Carga los valores de la imagen hiperespectral.
53- Calcula la media por cada banda. 224 valores
54- Descarta los valores de los extremos que no pueden ser utilizados debido al bajo ratio de señal-ruido. Las 20 primeras y las 44 últimas bandas. 160 bandas en total.
55- Normaliza los valores entre 0 – 1.
56- Calcula la media de cada 5 bandas para reducir el número de datos. 32 bandas resultantes.
57- Se juntan en una lista los valores: Espectroradiometro STS-VIS + sensores BH1750 y DHT11 + 32 valores de referencia de blanco.
58- Se juntan en una lista los valores: Cámara ELP RGB + sensores BH1750 y DHT11 + 32 valores de referencia de blanco.
59- Guarda la información del punto nº58 en un fichero .json.
60- Guarda la información del punto nº59 en un fichero .csv.

3.3.3.3 Diagrama de flujo MainSerialCommunication

El siguiente diagrama de flujo muestra el proceso automático para obtener los valores de humedad y temperatura con los sensores BH1750 y DHT11. Una vez obtenidos, se envían vía puerto serie a la placa Jetson Xavier NX con el siguiente formato.

- luminance = 20135.0
- temp = 19.67; humid = 65.33

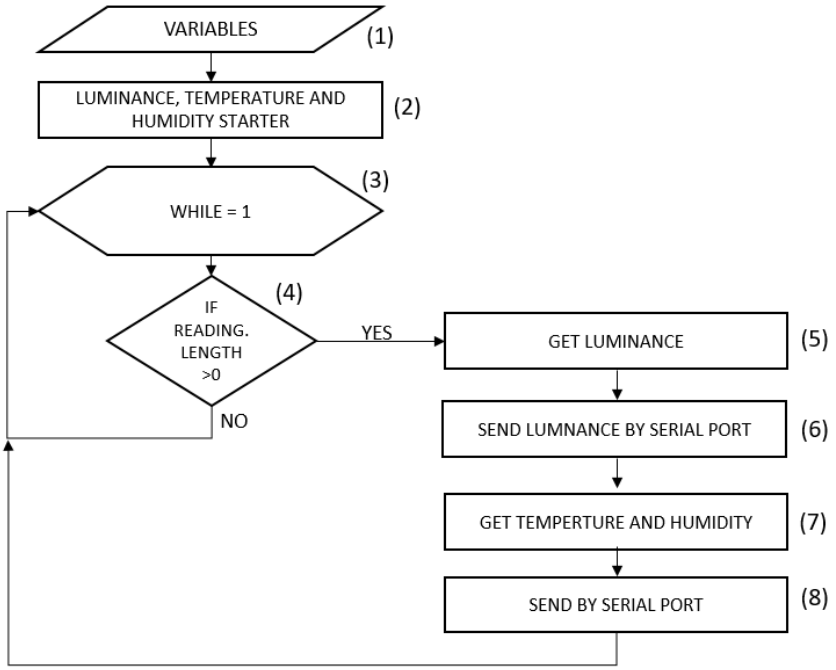

Ilustración 38: Diagrama de flujo de la aplicación mainSerialCommunicacion

1- Se definen las variables para la configuración de los sensores BH1750 y DHT11.
2- Se inician ambos sensores.
3- Bucle infinito “while” que chequea constantemente el puerto serie.
4- Si se ha escrito en el puerto serie, continua al proceso (5). Si no, sigue chequeando el puerto serie.

5- Se obtiene el valor de intensidad de luz.
6- Se envía por puerto serie. luminance = 20135.0
7- Se obtiene el valor de humedad y temperatura.
8- Se envía por puerto serie. temp = 19.67; humid = 65.33

3.4 Estructura general de los datos

La estructura de los datos está organizada por carpetas de acuerdo con el siguiente diagrama jerárquico.

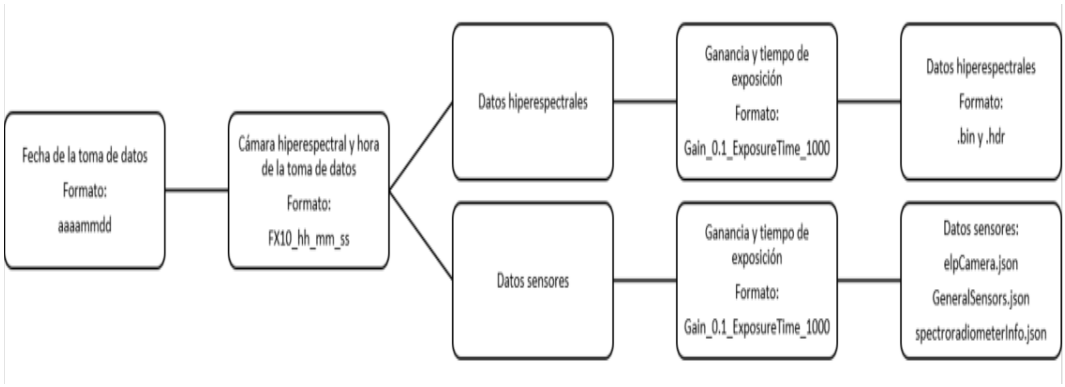

Ilustración 39: Jerarquía de ficheros

Los datos recogidos de los sensores son almacenados en un archivo .json que posteriormente se convierten en un archivo .csv para el tratamiento con la plataforma Kaggle y así poder hacer la regresión junto al valor de reflectancia capturado por la cámara hiperespectral. Como se explicó anteriormente, debido a que las capturas con la cámara ELP RBG y el sensor BH1750 están saturaras a falta de hacer las pruebas de campo con los filtros atenuadores, se opta por utilizar el espectroradiometro STS-VIS a modo de estudio. Es por ello que los archivos se han divido en dos grupos de datasets:

1- calibrationInformation_ELP: elpCamera. Json + GeneralSensors.json + elpCamera. csv + GeneralSensors.csv + hyperspectralImages_white.bin + hyperspectralImages_white.hdr
2- calibrationInformation_Spectroradiometer: spectroradiometerInfo.json + GeneralSensors.json + spectroradiometerInfo.csv + GeneralSensors.csv + hyperspectralImages_white.bin + hyperspectralImages_white.hdr

Los datos que se deben escoger son aquellos que puedan tener influencia directa sobre el resultado al final del muestreo, es por ello por lo que lo datos han sido previamente filtrados. Para este TFM y utilizando el conjunto de datos que pertenece al grupo 1 se ha trabajado con los siguientes:

- Input_R_Normalized_ELP: Valor normalizado que corresponde al color rojo de la cámara ELP RGB. 0 – 1 - float64
- Input_G_Normalized_ELP: Valor normalizado que corresponde al color verde de la cámara ELP RGB. 0 - 1 - float64
- Input_B_Normalized_ELP: Valor normalizado que corresponde al color azul de la cámara ELP RGB. 0 – 1 - float64
- Input_Luminance: Valor normalizado de luminancia del sensor BH1750. - float64
- Input_Humidity: Valor de humedad del sensor DHT11. 20 – 90 % - float64
- Input_Temperature: Valor de temperatura del sensor DHT11. 0 - 50 ºC - float64
- Output_WR_Normalized_#: lista de 32 valores normalizados de la cámara hiperespectral Specim FX10. - float64

[26]:
```
df_results_ELP
```

[26]:

	Input_R_Normalized_ELP	Input_G_Normalized_ELP	Input_B_Normalized_ELP	Input_Luminance	Input_Humidity	Input_Temperature	Output_WR_Normalized_0	Output_WR_Normalized_1	Output_WR_Normalized_2
0	0.753779	0.763664	0.759621	0.149722	70.00	18.00	0.104906	0.109139	0.119967
1	0.968627	1.000000	0.992157	1.000000	67.00	21.33	0.429763	0.465115	0.548062
2	0.971373	0.998903	0.997835	0.366613	74.00	18.33	0.164157	0.173745	0.197405
3	0.972549	1.000000	1.000000	0.763882	56.00	24.33	0.296709	0.319080	0.372666
4	0.968627	1.000000	0.992157	1.000000	55.67	23.67	0.483019	0.521628	0.613569
5	0.971437	0.998130	0.997753	1.000000	63.00	23.67	0.301129	0.326105	0.383208

Ilustración 40: Ejemplo de estructura de los datos del dataset del primer grupo

Utilizando el conjunto de datos que pertenece al grupo 2 se ha trabajado con los siguientes:

- Input_R_Normalized: Valor normalizado que corresponde al color rojo del espectroradiometro STS-VIS. 0 – 1 - float64
- Input_G_Normalized: Valor normalizado que corresponde al color verde del espectroradiometro STS-VIS. 0 – 1 - float64
- Input_B_Normalized: Valor normalizado que corresponde al color azul del espectroradiometro STS-VIS. 0 – 1 - float64
- Input_Brigthness_Normalized: Valor normalizado del brillo del espectroradiometro STS-VIS. 0 - 1 - float64
- Input_Humidity: Valor de humedad del sensor DHT11. 20 – 90 % - float64
- Input_Temperature: Valor de temperatura del sensor DHT11. 0 - 50 ºC - float64
- Output_WR_Normalized_#: lista de 32 valores normalizados de la cámara hiperespectral Specim FX10. - float64

```
df_results_spectroradiometer
```

[11]:

	Input_R_Normalized	Input_G_Normalized	Input_B_Normalized	Input_Brigthness_Normalized	Input_Humidity	Input_Temperature	Output_WR_Normalized_0	Output_WR_Normalized_1	Output_WR_Normalized_2
0	0.167838	0.190686	0.157265	0.163885	70.00	18.00	0.104906	0.109139	0.119967
1	0.514515	0.678889	0.389170	0.492909	67.00	21.33	0.429763	0.465115	0.548062
2	0.245712	0.301400	0.211460	0.236543	74.00	18.33	0.164157	0.173745	0.197405
3	0.340148	0.440993	0.277023	0.327957	56.00	24.33	0.296709	0.319080	0.372666
4	0.519645	0.684271	0.395496	0.496853	55.67	23.67	0.483019	0.521628	0.613569
5	0.404490	0.520299	0.309072	0.384138	63.00	23.67	0.301129	0.326105	0.383208

Ilustración 41: Ejemplo de estructura de los datos segundo grupo

Utilizando el programa DisplayWhiteRef podemos visualizar un ejemplo de la firma espectral con un valor de ganancia y tiempo de exposición determinados. En el eje X se encuentran los valores de longitud de onda y en el eje Y se haya los valores de intensidad de luz convertidos a valores digitales. La cámara dispone de un buffer de memoria de 12 bit para convertir los datos de intensidad de luz a valores digitales, es decir, cuando se digitaliza la intensidad de luz el valor máximo que puede medir es 2^12 = 4095 por cada longitud de onda.

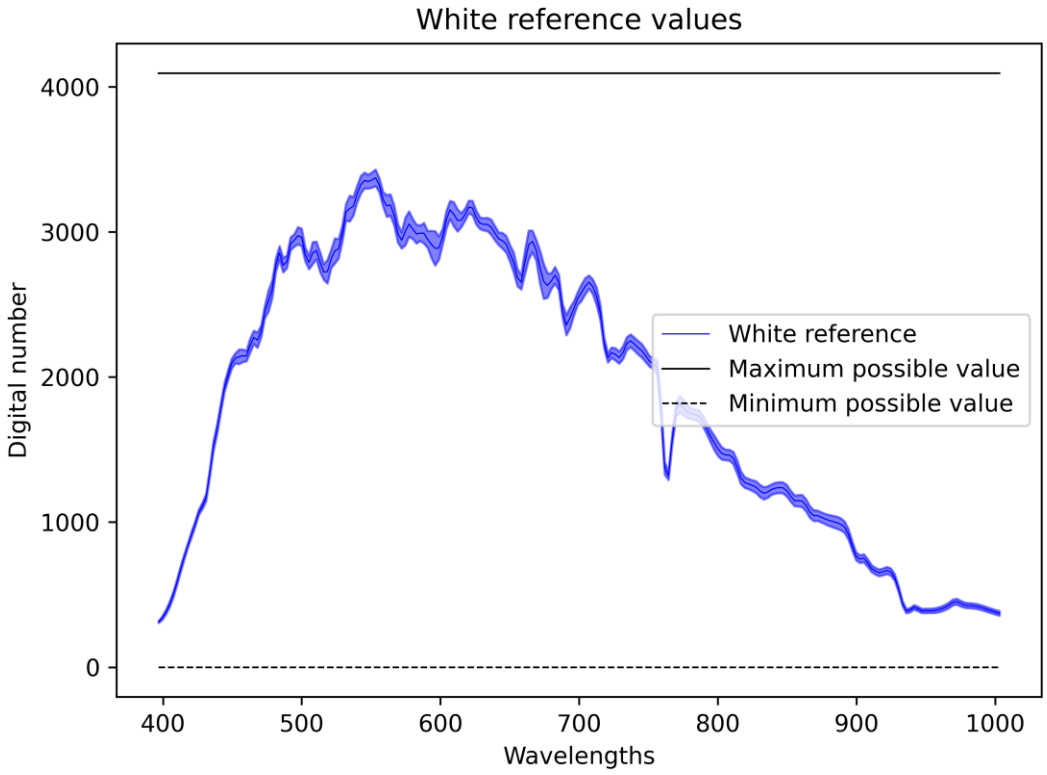

Ilustración 42: Firma espectral de la referencia de blanco capturada

La cantidad de datos obtenidos tiene que ser lo suficientemente grande como para poder entrenar un modelo de regresión y pueda converger, las ilustraciones anteriores solo se muestran las cinco primeras filas de los datasets a modo de ejemplo.

3.5 Regresión lineal

Tras un profundo estudio sobre las diferentes técnicas de aprendizaje supervisado o Maching Learning se ha optado por desarrollar un modelo de regresión lineal. La regresión lineal es una técnica de modelado estadístico que se emplea para describir una variable de respuesta continua como una función de una o varias variables predictoras. Puede ayudar a comprender y predecir el comportamiento de sistemas complejos o a analizar datos experimentales. Con los resultados obtenidos en las pruebas de campo se trata de predecir qué valor de reflectancia es el adecuado según las condiciones ambientales en tiempo real.

Al cargar en la plataforma Kaggle los datos del .csv resultante del programa "JsonProgramme" se puede observar que los datos son valores continuos no categóricos de tipo float64. Los datos ya vienen filtrados del programa anterior y no hay que hacer trabajos de limpieza o relleno de datos incorrectos o incompletos.

Una vez cargados los datos se crean los datasets de train y test para entrenar y probar un modelo de regresión lineal. Se crean varios grupos de datos

para probar con que variables se obtienen mejores resultados, es decir, se separa el dataset de entrenamiento en los siguientes grupos de variables predictoras:

1. Input_Brigthness_Normalized + Input_R_Normalized + Input_G_Normalized + Input_B_Normalized + Input_Humidity + Input_Temperature
2. Input_Brigthness_Normalized
3. Input_R_Normalized + Input_G_Normalized + Input_B_Normalized
4. Input_Humidity + Input_Temperature

Cuando el modelo está correctamente entrenado, se procede a hacer predicciones y se calcula el RMSE y SNR. El RMSE mide la cantidad de error que hay entre dos conjuntos de datos, en otras palabras, compara un valor predicho y un valor observado o conocido. Este valor puede cuantificar cuán diferente es un conjunto de valores. Cuanto más pequeño es un valor RMSE, más cercanos son los valores predichos y observados. El RMSE se calcula mediante la siguiente formula.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(P_i - O_i)^2}{n}}$$

Ecuación 2: Formula RMSE

Por otro lado, la señal detectada contiene una señal real y una señal de fondo o ruido de fondo. En términos generales para detectar el objetivo a partir de una identificación de señal de fondo, se requiere un índice suficientemente bueno de intensidad de la señal real en función de la señal de fondo. A esto se le denomina relación entre señal-ruido (SNR). En general, optimizar la señal real o minimizar las señales de fondo son formas comunes de mejorar el SNR. En el caso de las imágenes hiperespectrales cada píxel además de estar constituido por esta combinación de espectros de los materiales, también presenta un componente de error que hay que considerar, ya que puede llegar a tener un gran peso en relación los valores de la imagen. Tal es el caso de las imágenes que presentan bandas con bajo SNR, es decir, imágenes con mucho ruido en relación a la información que contienen, que hacen estas bandas inservibles en muchos casos. En este componente de error cabría englobar toda la información presente en la imagen que no puede ser explicada como combinación de los propios materiales, y que constituye normalmente lo que se denomina ruido proveniente de interferencias atmosféricas. Cuanto mayor sea este valor, mejor es la calidad de la imagen. El SNR se calcula mediante la siguiente formula.

$$SNR = 20 \text{ x } \log 10 \frac{\sum x_i^2}{\sum (x_i - x_{ref})^2}$$

Ecuación 3: Formula SNR

Capítulo 4. Resultados obtenidos y conclusiones

4.1 Resultados obtenidos con el sistema completo

En este apartado, se analizan los resultados obtenidos del sistema hardware completo y de los programas desarrollados. El sistema hardware construido para realizar las pruebas de campo se ajusta perfectamente a las necesidades de este estudio.

En total se han realizado un total de trece días de capturas que han dado lugar a dos datasets con la siguiente distribución sin filtrar los datos y sin incluir los valores hiperespectrales.

- elpCamera. csv + GeneralSensors.csv: 550 filas x 20 columnas = 11 000 valores

	date	currentTime	hyperspectralCamera	hyperspectralGain	hyperspectralExposureTime	luminance	temperature	humidity	elpGain	elpExposureTime	meanR	meanG	meanB	offset	percentageLo
0	20210106	10:21:03	Fx10	0.1	1000	0.160250	18.0	68.0	0	1	0.763375	0.785829	0.783685	15	
1	20210106	10:21:03	Fx10	0.1	10000	0.151034	18.0	69.0	0	1	0.754649	0.766372	0.762222	15	
2	20210106	10:21:03	Fx10	0.1	2000	0.160555	19.0	67.0	0	1	0.763946	0.782571	0.778578	15	
3	20210106	10:21:03	Fx10	0.1	3000	0.158709	18.0	68.0	0	1	0.760898	0.775288	0.771620	15	
4	20210106	10:21:03	Fx10	0.1	4000	0.155566	18.0	68.0	0	1	0.760684	0.773633	0.769543	15	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
525	20213006	18:21:40	Fx10	1.0	5000	0.115648	17.0	84.0	0	1	0.660628	0.676080	0.666592	15	
526	20213006	18:21:40	Fx10	1.0	6000	0.116716	17.0	84.0	0	1	0.669330	0.685268	0.675921	15	
527	20213006	18:21:40	Fx10	1.0	7000	0.121767	17.0	84.0	0	1	0.687961	0.700597	0.692503	15	
528	20213006	18:21:40	Fx10	1.0	8000	0.124346	17.0	84.0	0	1	0.684533	0.699500	0.691071	15	
529	20213006	18:21:40	Fx10	1.0	9000	0.120912	17.0	84.0	0	1	0.668992	0.685507	0.676616	15	

530 rows × 20 columns

Ilustración 43: Resultados del dataset elpCamera. csv + GeneralSensors.csv

Tras analizar estos datos recogidos por la cámara ELP RGB y los sensores BH1750 y DHT11, se ha podido determinar lo siguiente:

- Los datos recogidos por la cámara ELP RGB y el sensor BH1750 alcanzan valores iguales o cercanos a uno (ya normalizados) lo que indica que son valores saturados, hay que evitar este tipo de datos puesto que no aportan una información correcta de las condiciones ambientales. Para tratar de solucionar este problema, se ha probado a poner un difusor cubriendo ambos sensores pero no se ha conseguido corregir este problema.
- Muchos de los datos tienen los mismos valores a diferentes ganancias y tiempos de exposición. Esto indica que se puede optimizar que métodos de configuración son los adecuados para realizar el experimento sin sobre dimensionar el dataset con valores que puedan perturbar la posterior regresión lineal al estar repetidos.
- Los valores de la fecha, el tipo de cámara utilizada, la hora, etc. Son valores que están incluidos en el dataset pero no serán utilizados a la hora de hacer la regresión lineal al no poderse correlacionar con los valores hiperespectrales.

- spectroradiometerInfo.csv + GeneralSensors.csv: 1 850 filas x 13 columnas = 24 050 valores

	date	currentTime	hyperspectralCamera	hyperspectralGain	hyperspectralExposureTime	luminance	temperature	humidity	spectroradiometer_integrationTime	spectroradiometer_R	spectroradiometer_G
0	20210106	10:21:03	Fx10	0.1	1000	0.160250	18.0	68.0	10	0.121084	0.121762
1	20210106	10:21:03	Fx10	0.1	1000	0.160250	18.0	68.0	200010	0.172074	0.196110
2	20210106	10:21:03	Fx10	0.1	1000	0.160250	18.0	68.0	400010	0.223451	0.271421
3	20210106	10:21:03	Fx10	0.1	1000	0.160250	18.0	68.0	600010	0.275248	0.347425
4	20210106	10:21:03	Fx10	0.1	1000	0.160250	18.0	68.0	800010	0.327178	0.423541
...	...	...	...	...	...	...	...	...	...	...	...
1745	20213006	18:21:40	Fx10	1.0	9000	0.120912	17.0	84.0	10	0.120892	0.121565
1746	20213006	18:21:40	Fx10	1.0	9000	0.120912	17.0	84.0	200010	0.159429	0.178403
1747	20213006	18:21:40	Fx10	1.0	9000	0.120912	17.0	84.0	400010	0.198192	0.235763
1748	20213006	18:21:40	Fx10	1.0	9000	0.120912	17.0	84.0	600010	0.236884	0.293105
1749	20213006	18:21:40	Fx10	1.0	9000	0.120912	17.0	84.0	800010	0.275639	0.350586

1750 rows × 13 columns

Ilustración 44: Resultados del dataset spectroradiometerInfo.csv + GeneralSensors.csv

Tras analizar estos datos recogidos por el espectroradiómetro y los sensores BH1750 y DHT11, se ha podido determinar lo siguiente:

- Los valores capturados por el espectroradiómetro se ajustan perfectamente al dataset y no saturan en ninguna de las longitudes de onda seleccionadas. Además, aporta la medida del brillo (Brigthness) que puede ser utilizada como medida de cantidad de luz en sustitución del sensor BH1750.
- Muchos de los datos tienen los mismos valores a diferentes ganancias y tiempos de exposición. Esto indica que se puede optimizar que métodos de configuración son los adecuados para realizar el experimento sin sobre dimensionar el dataset con valores que puedan perturbar la posterior regresión lineal al estar repetidos.
- Los valores de la fecha, el tipo de cámara utilizada, la hora, etc. Son valores que están incluidos en el dataset pero no serán utilizados a la hora de hacer la regresión lineal al no poderse correlacionar con los valores hiperespectrales.

Tras el primer análisis realizado, se puede determinar que los valores obtenidos por la cámara ELP RGB no son útiles para esta prueba de concepto y se utilizan los datos capturados por el espectroradiometro para realizar la regresión lineal.

Los datos recogidos por el programa “Capturing” se ejecutan en el programa “JsonProgramme” para unificar en un único dataset los datos del espectroradiómetro STS-VIS y los datos hiperespectrales de la cámara Specim FX10 y limpiar los posibles datos con configuraciones erróneas que dan lugar a resultados saturados y repetidos. Los datasets finales resultantes que se pueden utilizar para hacer la regresión lineal son los siguientes:

- spectroradiometerInfo.csv + GeneralSensors.csv + hyperspectralImages_white.bin : 19 filas x 38 columnas = 772 valores

```
df_results_spectroradiometer
```

[3]:

	Input_R_Normalized	Input_G_Normalized	Input_B_Normalized	Input_Brigthness_Normalized	Input_Humidity	Input_Temperature	Output_WR_Normalized_0	Output_WR_Normalized_1	Output_WR_Normalized_2	Output_WR_Normalized_3	...	Output_WR_Normalized_22	Output_WR_Normal
0	0.167838	0.190686	0.157265	0.163885	0.7000	0.3600	0.104906	0.109139	0.119967	0.125555	...	0.101942	0
1	0.514515	0.678889	0.389170	0.492909	0.6700	0.4266	0.429763	0.465115	0.548062	0.591974	...	0.428262	0
2	0.528319	0.698303	0.396771	0.505717	0.4333	0.6734	0.426140	0.460399	0.542309	0.585632	...	0.417962	0
3	0.475866	0.627839	0.363621	0.456629	0.5933	0.5066	0.368969	0.398488	0.468523	0.505725	...	0.362805	0
4	0.245712	0.301400	0.211460	0.236543	0.7400	0.3666	0.164157	0.173745	0.197405	0.209954	...	0.161813	0
5	0.340148	0.440993	0.277023	0.327957	0.5600	0.4866	0.296709	0.319080	0.372666	0.400856	...	0.290239	0
6	0.519645	0.684271	0.395496	0.496853	0.5567	0.4734	0.483019	0.521628	0.613569	0.661246	...	0.475082	0
7	0.404490	0.520299	0.309072	0.384138	0.6300	0.4734	0.301129	0.326105	0.383208	0.415014	...	0.307116	0
8	0.168681	0.185373	0.150174	0.161034	0.7767	0.3600	0.092012	0.096853	0.106817	0.113483	...	0.103791	0
9	0.478183	0.629099	0.369269	0.458479	0.6733	0.4466	0.408911	0.441602	0.519275	0.559783	...	0.401150	0
10	0.536878	0.710630	0.408754	0.514699	0.6500	0.4534	0.198702	0.210360	0.240727	0.255655	...	0.178098	0
11	0.206298	0.243340	0.181108	0.197953	0.7033	0.4000	0.137773	0.145673	0.164408	0.174946	...	0.140481	0
12	0.217599	0.266998	0.199224	0.212833	0.7100	0.3400	0.150007	0.157814	0.177651	0.187519	...	0.135619	0
13	0.133617	0.140466	0.133911	0.133151	0.8500	0.3200	0.065274	0.066063	0.068285	0.069427	...	0.062236	0
14	0.213171	0.257720	0.193129	0.207495	0.6867	0.3266	0.145048	0.152428	0.171327	0.180794	...	0.137986	0
15	0.453069	0.605921	0.366248	0.440262	0.6233	0.3734	0.541567	0.585355	0.687694	0.740298	...	0.529452	0
16	0.464594	0.600096	0.354021	0.440564	0.4167	0.4934	0.383101	0.413004	0.484667	0.522685	...	0.368625	0
17	0.208050	0.257150	0.194880	0.205241	0.5333	0.4600	0.151666	0.158599	0.178118	0.187216	...	0.134766	0
18	0.161220	0.180738	0.152487	0.157730	0.8400	0.3400	0.096445	0.100113	0.109303	0.114078	...	0.093863	0

19 rows × 38 columns

Ilustración 45: Resultados filtrados y normalizados spectroradiometerInfo.csv + GeneralSensors.csv + hyperspectralImages_white.bin

Como se puede observar, el dataset final se ha visto reducido y tras pasarlo por el programa “JsonProgramme” todos los valores son correctos para realizar un estudio de regresión lineal. Estos valores se pueden llegar a correlacionar entre si dado que están normalizados entre 0-1 y como se puede ver en la siguiente gráfica, los valores son proporcionales en todos los sensores utilizados.

La línea azul ancha denominada en la leyenda como “White Reference” representa el valor de reflectancia en todas las longitudes de onda medidas, además esta incluye la desviación típica siendo el valor central la media de todos los pixeles.

La línea de puntos suspensivos azul denominada “Spectroradiometer Brightness” es el valor normalizado del brillo recogido por el espectroradiometro STS-VIS.

La línea violeta denenominada “Spectroradiometer RGB” representa los valores de los espectros rojo (650 nm), verde (550 nm) y azul (470 nm). recogidos por el esproradiometro STS-VIS. Estos tres valores están representados por los siguientes tres puntos:

- El punto inicial de izquierda a derecha. Azul (470 nm)
- La arista o punto de inflexión. Verde (550 nm)
- El punto final de izquierda a derecha. Rojo (650 nm)

La línea de puntos suspensivos verdes denominada “Temperature (50ºC)” representa el valor de la temperatura. El máximo valor posible a medir son 50 ºC.

La línea de puntos roja denominada “Relative humidity” representa la humedad relativa.

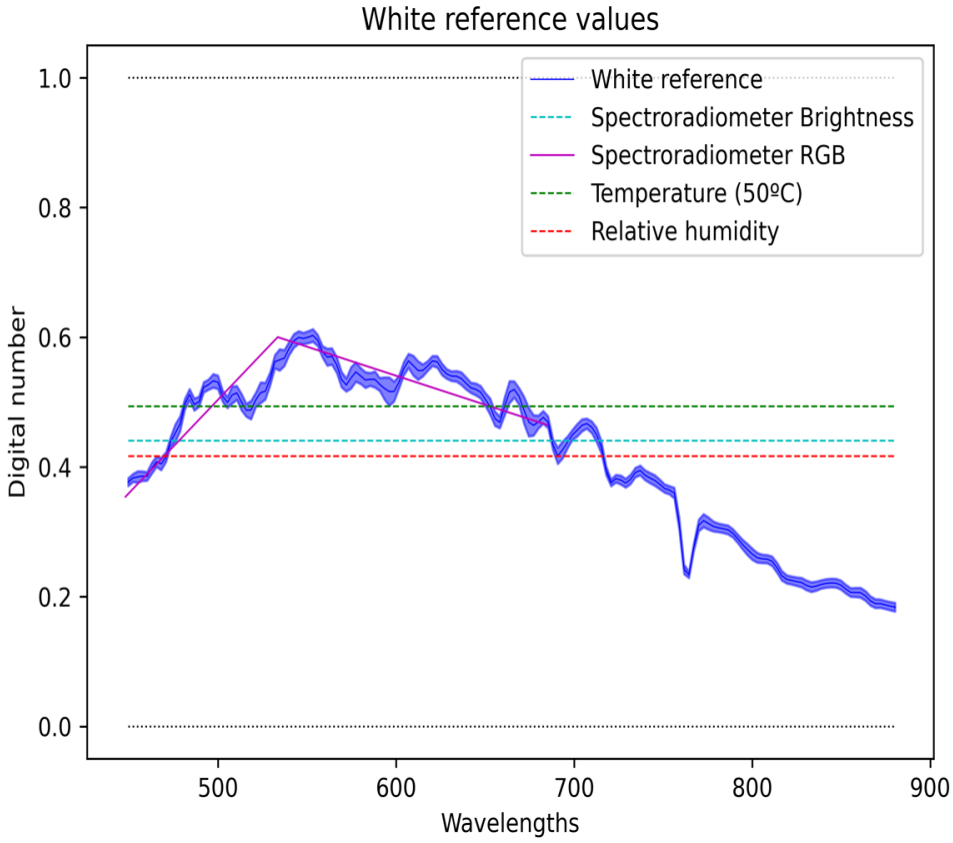

Ilustración 46: Grafica con los valores normalizados

Por otro lado, en las siguientes graficas se puede comparar un día de captura normal y otro día de captura con calima en la atmosfera, este es un fenómeno usual en la isla y conviene tener en cuenta los resultados.

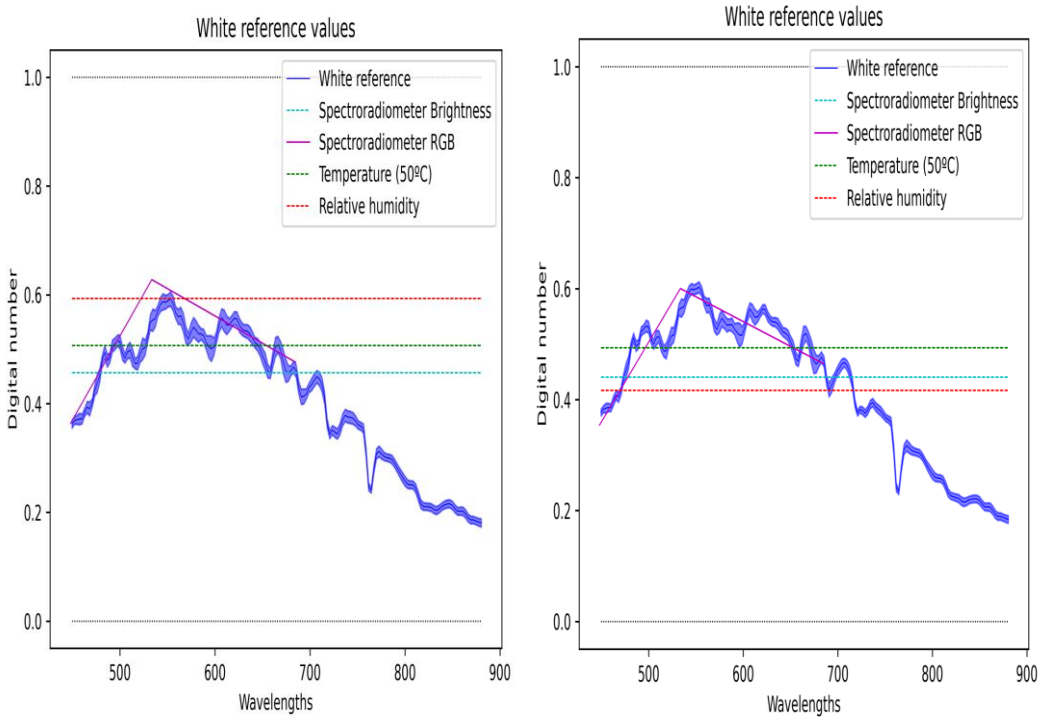

Ilustración 47: de izquierda a derecha: día normal de captura y día con calima en la atmosfera

En la imagen anterior, se muestra dos días de capturas a la misma hora pero con dos meses de diferencia entre ellas, en la gráfica de la derecha, el día es una mañana normal con condiciones ambientales óptimas para la toma de datos y la gráfica de la izquierda es un día de intensa calima en la isla de Gran Canaria. Se puede observar que los valores de humedad, brillo y temperatura aumentan y la intensidad de los valores hiperespectrales disminuyen aunque se mantiene la misma curva de reflectancia, es decir, el aumento de la humedad y la temperatura no afecta la reflectancia pero la forma de la curva es equivalente, por lo que incluso en circunstancias tan adversas, con más datos, potencialmente el modelo de regresión debería de funcionar también.

Esta grafica demuestra, la importancia de hacer las capturas con diferentes condiciones ambientes incluyendo las potencialmente influyentes que puedan perturbar la curva de reflectancia.

Siguiendo con este análisis, en las siguientes gráficas se muestran dos capturas realizadas el mismo día a diferentes horas. La primera a las 17:30 hrs GMT+1 y la segunda a las 20:00 hrs GMT+1.

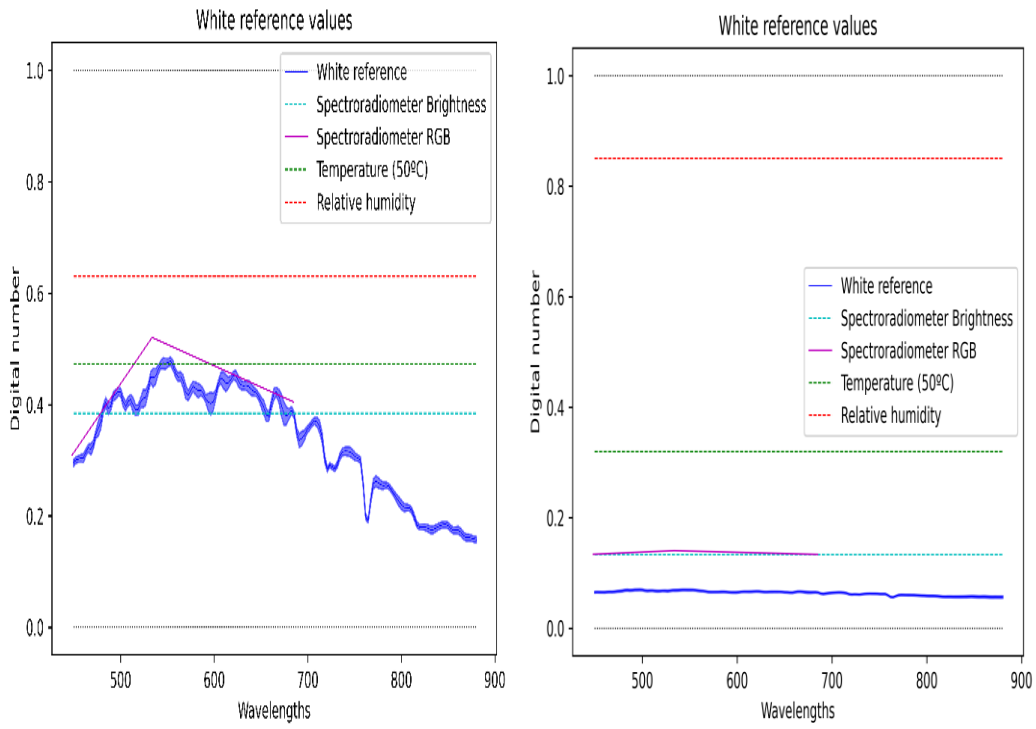

Ilustración 48: de izquierda a derecha: Grafica captura 17:30 hrs GMT+1 y grafica captura 20:00 hrs GMT+1

Se puede observar que la intensidad de la luz decae considerablemente casi al anochecer de un día normal. Esta grafica demuestra, la importancia de hacer las capturas a diferentes horas del día incluyendo las potencialmente influyentes que puedan perturbar la curva de reflectancia.

4.2 Resultados obtenidos de la regresión lineal

Con los datos obtenidos por el sistema y tras haberlos filtrados mediante el programa "JsonProgramme", se procede a hacer la prueba de concepto de la regresión lineal. En total se ha hecho la regresión con un dataset de 19 filas x 38 columnas dando un total de 772 valores. Tras definir el modelo de regresión lineal (LR) se han hecho unas predicciones y se ha obtenido los siguientes resultados.

- El RMSE de test LR_All_spectroradiometer es: 0.132
- El RMSE de test LR_Brigthness_spectroradiometer es: 0.129
- El RMSE de test LR_RGB_spectroradiometer es: 0.138
- El RMSE de test LR_Humidity_Temperature_spectroradiometer es: 0.124
- El SNR de test LR_All_spectroradiometer es: 21.357
- El SNR de test LR_Brigthness_spectroradiometer es: 18.349
- El SNR de test LR_RGB_spectroradiometer es: 18.081
- El SNR de test LR_Temperature_spectroradiometer es: 25.922

Con respecto al RMSE, el valor obtenido por el test LR_All_spectroradiometer es mayor que el obtenido por LR_Brigthness_spectroradiometer, esto puede deberse a la falta de datos para que el modelo pueda converger con todos los valores de incluidos en esta prueba. También se puede observar que el valor del test LR_Humidity_Temperature_spectroradiometer es el óptimo para este modelo, al

igual que el caso anterior, habría que tomar más datos para poder determinar si las medidas de temperatura y humedad pueden influir en la toma de imágenes hiperespectrales. En general, todos los resultados del RMSE están muy cercanos a cero. Esto significa que los valores predichos no difieren mucho de los valores entrenados.

Por otro lado analizando los valores de SNR, se puede observar que el valor de los test LR_All_spectroradiometer y LR_Temperature_spectroradiometer son los más altos y por lo tanto, la relación señal/ruido es bastante buena. Potencialmente, el test LR_All_spectroradiometer puede mejorar obteniendo más datos con días de capturas.

En base a esto, todo parece indicar que los datos obtenidos son satisfactorios. Aumentando los elementos se puede llegar a desarrollar una base de datos que pueda converger realizando estudios de regresión. También se puede observar que algunos datos no llegan a influir en la regresión y por lo tanto no se pueden correlacionar, en estos casos habría que estudiar eliminarlos de futuras sesiones de capturas dado que parecen no ser relevantes.

4.3 Conclusiones y trabajos futuros

Tras las repetidas pruebas de campo se pueden sacar varias conclusiones:

- Se realizan capturas satisfactorias de manera cómoda y eficiente gracias al sistema desarrollado para el trabajo de campo.
- La solución de utilizar una batería es muy efectiva para poder desplazar todo el sistema sin necesidad de estar conectado a la red eléctrica.
- Los resultados obtenidos son bastante satisfactorios además de aportar mucha información en una sola sesión de toma de muestras.
- Las sesiones de capturas deben aumentar para obtener más resultados. Por falta de tiempo y malas condiciones climatológicas solo se han podido hacer treces sesiones de capturas. En futuros estudios, los datos deben ser mayores para poder obtener más conclusiones.
- Los datos de la cámara ELP RGB y el sensor BH1750 están saturados debido a que la cámara y el sensor apuntan directamente al cielo. Aun poniendo difusores de luz, siguen habiendo valores muy próximos a 1. Una posible solución es añadir al objetivo un filtro atenuador para reducir la cantidad luz incidente. Por falta de tiempo y disponibilidad de los filtros no se ha podido realizar esta prueba.
- Los resultados obtenidos en la regresión lineal son satisfactorios y validan esta prueba de concepto. Los valores RMSE y SNR son alentadores y con más datos se puede llegar a sacar una conclusión de que valores tomados por los sensores son los que más se pueden correlacionar con la referencia de blanco y así poder desarrollar un modelo de calibración en tiempo real.
- Se ha determinado la importancia de obtener la mayor variabilidad en las condiciones ambientales y las franjas horarias en los días de capturas.
- Todo parece indicar que los datos obtenidos son satisfactorios. Aumentando los elementos se puede llegar a desarrollar una base de datos que pueda converger realizando estudios de regresión.
- También se puede observar que algunos datos no llegan a influir en la regresión y por lo tanto no se pueden correlacionar, en estos casos habría

que estudiar eliminarlos de futuras sesiones de capturas dado que parecen no ser relevantes.

Capítulo 5. Referencias

[1] “Introduction to hyperspectral imaging”, en www.microimages.com/documentation/Tutorials,

[Última visita Noviembre 2020]

[2] H. Fabelo, S. Ortega, S. Kabwama, G. M Callico, D. Bulters, A. Szolna, J. F. Pineiro, R. Sarmiento, “HELICOID Project: a new use of hyperspectral imaging foir brain cáncer detection in real-time during neurosurgical operations”, SPIE Commercial+ Scientific Sensing and Imaging, Baltimore, EEUU, 17-21

[Última visita Noviembre 2020]

[3] S Ortega, H Fabelo, R Camacho, ML Plaza, GM Callico, R Lazcano, D Madroñal, R Salvador, E Juárez, R Sarmiento, “Detection of human brain cáncer in pathological slides using hyperspectral images”, Neuro-oncology, vol. 19

[Última visita el Noviembre 2020]

[4] L. Santos, L. Berrojo, J. Moreno, J. F. López, R. Sarmiento, “Multispectral and hyperspectral lossless compressor for space applications (HyLoC): a low complexity FPGA implementation of the CCSDS 123 standard”, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 9, num. 2, enero 2015

[Última visita el Diciembre 2020]

[5] A. García, L. Santos, S. López, G. M. Callicó, J. F. López, R. Sarmiento, "Efficient lossy compression implementations of hyperspectral images: tolos, hardware platforms and comparisons", SPIE Sensing Technology + Applications, Baltimorer, EEUU, 5-9 mayo 2014

[Última visita el Diciembre 2020]

[6] M. Teke, H.S. Deveci, O. Haliloglu, S.Z. Gurbuz, U. Sakarya, "A short survey of hyperspectral remote sensing applications in agriculture", 6th International Conference on Recent Advances in Space Technologies (RAST), Estambul, Turquía, 12-14 junio 2013

[Última visita el Diciembre 2020]

[7] R.N. Sahoo, S.S. Ray, K.R. Manjunath, "Hyperspectral remote sensing of agriculture", Current Science, vol. 108, num. 5, marzo 2015

[Última visita el Diciembre 2020]

[8] J. Pedraza, "La revolución que nos dará de comer (y cuidará el planeta)", Periódico El País, 21 abril, 2017

[Última visita el Diciembre 2020]

[9] Carlos Herrera Falcón, José Francisco López Feliciano, Pablo Sebastián Horstrand Andaluz "Programación y optimización del vuelo de un dron orientado a la agricultura inteligente"

[Última visita el Diciembre 2020]

[10] "What is Smart Farming? - Smart-AKIS." [Online]. Available: https://www.smartakis.com/index.php/network/what-is-smart-farming/.

[Última visita el Diciembre 2020]

[11] https://www.specim.fi/downloads/Specim-FX10-Technical-Datasheet-01.pdf

[Última visita el Diciembre 2020]

[12] https://www.specim.fi/wp-content/uploads/2020/03/Specim-FX17-Technical-Datasheet-02.pdf – From Specim website datasheets.

[Última visita el Diciembre 2020]

[13] https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf

[Última visita el Diciembre 2020]

[14] https://en.wikipedia.org/wiki/Kaggle

[Última visita el Diciembre 2020]

[15] https://sphereoptics.de/en/product/zenith-polymer-reflectance-standards/

[Última visita Febrero 2021]

[16] "Hyperspectral Push-Broom Microscope Development and Characterization" SAMUEL ORTEGA, RAÚL GUERRA , MARÍA DÍAZ , HIMAR FABELO , SEBASTIÁN LÓPEZ , (Senior Member, IEEE), GUSTAVO M. CALLICÓ , (Member, IEEE), AND ROBERTO SARMIENTO

[Última visita Febrero 2021]

[17] PABLO HORSTRAND , RAÚL GUERRA , AYTHAMI RODRÍGUEZ, MARÍA DÍAZ , SEBASTIÁN LÓPEZ , (Senior, Member, IEEE), AND JOSÉ FCO. LÓPEZ "A UAV Platform Based on a Hyperspectral Sensor for Image Capturing and On-Board Processing"

[Última visita Febrero 2021]

[18] https://www.dji.com/es/matrice600

[Última visita Abril 2021]

[19] https://recursospython.com/guias-y-manuales/instalar-pil-pillow-efectos/

[Última visita Abril 2021]

[20] https://www.arduinolibraries.info/libraries/dht-sensor-library

[Última visita Abril 2021]

[21] https://www.arduinolibraries.info/libraries/bh1750

[Última visita Abril 2021]

[22] https://sourceforge.net/projects/v4l2vd/

[Última visita Abril 2021]

[23] https://aprendeconalf.es/docencia/python/manual/numpy/

[Última visita Abril 2021]

[24] https://aprendeconalf.es/docencia/python/manual/pandas/

[Última visita Abril 2021]

[25] https://aprendeia.com/libreria-scikit-learn-de-python/

[Última visita Abril 2021]

[26] https://en.wikipedia.org/wiki/GigE_Vision

[Última visita Abril 2021]

[27] https://es.wikipedia.org/wiki/Raw_(formato)

[Última visita Abril 2021]

[28] https://es.wikipedia.org/wiki/Arduino_IDE

[Última visita Abril 2021]

[29] https://www.ecured.cu/Pycharm

[Última visita Abril 2021]

[30] https://es.wikipedia.org/wiki/Espectro_electromagn%C3%A9tico

[Última visita Abril 2021]

ANEXOS A LA MEMORIA

Anexos a la memoria

Anexo 1: Aplicación principal de capturas automáticas "Capturing"

Capturing.py
Aplicación principal de capturas automáticas

```
import __init__
import os
import time
import serial
import syslog
from datetime import datetime
from pathlib import Path
import math
import json

from PIL import Image
import numpy as np

from SpecimSoftware.aravisWrapperPy import CameraClass
from hypercube.hypercube import *
from utils import *
from arduinoSensors import *
from CameraELP import *
from SpectroradiometerUtils import *
```

```
# ----- CONSTANT VARIABLES ----- #
CALIBRATION_DIRECTORY_PATH = '/home/jetsonnx1/Documents/'
PATH_TO_THE_PYTHON_INTERPRETER_INSIDE_VIRTUAL_ENVIRONMENT =
'/home/jetsonnx1/Documents/CalibrationProcess/AutomaticStreamingController/venv/Automati
cStreamingVenv/bin'
SUDO_PASSWORD = 'ubuntu'
HYPERSPECTRAL_IMAGES_OUTPUT_PATH =
'/home/jetsonnx1/Documents/CalibrationProcess/AutomaticCapturingController/Output'
HYPERSPECTRAL_IMAGES_OUTPUT_PATH_IN_MEDIA = '/media/7614-8CA8/jetsonTest'
PATH_TO_THE_DIRECTORY_WHERE_SAVE_THE_NECESSARY_IMAGES_TEMPORAL =
'/home/jetsonnx1/Documents/CalibrationProcess/ELP_Camera/elp/Output/tmp'
PATH_TO_THE_JSON_CONFIGURATION_FILE =
'/home/jetsonnx1/Documents/CalibrationProcess/ELP_Camera/elp/config/ELPInitialConfig.json'



# ----- INPUT PARAMETERS ----- #
NUMBER_OF_FRAMES_TO_CAPTURE_FOR_CALIBRATION = 100


#  HYPERSPECTRAL CAMERA
#     GAIN CONSTANTS
minimumGain = 0.1 # MinValue: 0.1
maximumGain = 1 # Max value15.999
gainIncrement = 0.9
```

```
#    EXPOSURE TIME CONSTANTS
minimumExposureTime = 1000 # MinValue: 10
maximumExposureTime = 10000 # MAxValue: 419000
exposureTimeIncrement = 1000

#  ELP CAMERA
#    GAIN CONSTANTS

ELP_MinimumGain = 0 # Minimum gain = 0 (OriginalValueSet = 0)
ELP_MaximumGain = 0 # 100 # Maximum gain = 100 (OriginalValueSet = 100)
ELP_GainIncrement = 10 # Minimum step = 1 (OriginalValueSet = 10)
#    EXPOSURE TIME CONSTANTS
ELP_MinimumExposureTime = 1 # Minimum_exposure = 1 (OriginalValueSet = 1)
ELP_MaximumExposureTime = 1 # 10 # Maximum_exposure = 5000 (OriginalValueSet = 500)
ELP_ExposureTimeIncrement = 2 # Minimum exposure step = 1 (OriginalValueSet = 100)
#    OFFSET TO STORE THE PERCENTAGE OF PIXELS THAT ARE UPPER THAN (100-OFFSET)% AND
LOWER THAN (0+OFFSET)%
ELP_Offset = 15

#  SPECTRORADIOMETER
numberOfSpectroradiometerIntegrationTimeSamples = 5 # 10 # Number of different integration
time values to be used
spectroradiometerIntegrationTimeStepSizeInMicroseconds = 200000 # 100000 # 0.1 seconds -
Increment between integration values
customMinimumIntegrationTime = 10 # 0.1 seconds or None if the minimum integration time
limit value is desired
numberOfSamplesToCalculateTheMeanSpectralValue = 5 # 10 # NUmber of meassurements to
take in order to calculate the mean value of them
blueSpectralRange = [420, 476] # nm
greenSpectralRange = [497, 570 ] # nm
```

```
redSpectralRange = [620, 750] # nm

# ADC - 14 bits => Max. value = 16384
spectrometerMinValue = 0
spectrometerMaxValue = 2**14-1

# ----- END INPUT PARAMETERS ----- #









# ----- CALIBRATION PARAMETERS ----- #
#  HYPERSPECTRAL CAMERA
#    Generating the list of the Gains and Exposure times to
gainList = []
gain = minimumGain
while gain <= maximumGain:
  gainList.append(gain)
  gain = gain + gainIncrement

if gain < maximumGain + gainIncrement:
  gainList.append(maximumGain)

exposureTimeList = []
exposureTime = minimumExposureTime
while exposureTime <= maximumExposureTime:
  exposureTimeList.append(exposureTime)
  exposureTime = exposureTime + exposureTimeIncrement
```

```
if exposureTime < maximumExposureTime + exposureTimeIncrement:
  exposureTimeList.append(maximumExposureTime)

#  ELP CAMERA
#    Generating the list of the Gains and Exposure times to
elpGainList = []
gain = ELP_MinimumGain
while gain <= ELP_MaximumGain:
  elpGainList.append(gain)
  gain = gain + ELP_GainIncrement

if gain < ELP_MaximumGain + ELP_GainIncrement:
  elpGainList.append(ELP_MaximumGain)

elpExposureTimeList = []
exposureTime = ELP_MinimumExposureTime
while exposureTime <= ELP_MaximumExposureTime:
  elpExposureTimeList.append(exposureTime)
  exposureTime = exposureTime + ELP_ExposureTimeIncrement

if exposureTime < ELP_MaximumExposureTime + ELP_ExposureTimeIncrement:
  elpExposureTimeList.append(ELP_MaximumExposureTime)




# ----- PATHS GENERATION ----- #

# Generate the path to store the hyperspectral captured images
```

```
if os.path.exists(HYPERSPECTRAL_IMAGES_OUTPUT_PATH_IN_MEDIA):
  outputPath = HYPERSPECTRAL_IMAGES_OUTPUT_PATH_IN_MEDIA
else:
  outputPath = HYPERSPECTRAL_IMAGES_OUTPUT_PATH

if not os.path.isdir(outputPath):
  os.mkdir(outputPath)
# Generate the directory to store the calibration information using a timestamp
#  Getting the current date and time to generate the container hyperspectral calibration
information
now = datetime.now() # current date and time
date = now.strftime("%Y%d%m")
#  Forming the directory path using the date and time information
calibrationDirectoryPath = os.path.join(outputPath, date)
if not os.path.isdir(calibrationDirectoryPath):
  os.mkdir(calibrationDirectoryPath)
# Generate the path to store the temporal RGB images
if not os.path.isdir(PATH_TO_THE_DIRECTORY_WHERE_SAVE_THE_NECESSARY_IMAGES):
  os.mkdir(PATH_TO_THE_DIRECTORY_WHERE_SAVE_THE_NECESSARY_IMAGES)
if not
os.path.isdir(PATH_TO_THE_DIRECTORY_WHERE_SAVE_THE_NECESSARY_IMAGES_TEMPORAL):
  os.mkdir(PATH_TO_THE_DIRECTORY_WHERE_SAVE_THE_NECESSARY_IMAGES_TEMPORAL)




# ----- ELP CAMERA CONFIGURATION - GENERAL VARIABLES FOR THE ELP CAMERA ----- #
#  Configuration variables
pixelFormat = None
nRows = None
```

```
nCols = None

#  Read the json configuration file
jsonInformation = getInformationFromJsonFile(PATH_TO_THE_JSON_CONFIGURATION_FILE)
#  Set the values of the configuration variables
pixelFormat = jsonInformation["pixelFormat"]
nRows = jsonInformation["nRows"]
nCols = jsonInformation["nCols"]

# ----- ELP CAMERA INSTANCE ----- #
currentElpCamera = Camera(0, nCols, nRows, pixelFormat)
# Setting the exposure time to manual
currentElpCamera.setPropertyFromId('exposureAuto', 1)



# ----- SERIAL COMMUNICATION WITH ARDUINO ----- #
#  The following line is for serial over GPIO
port = '/dev/ttyUSB0'
baudRate = '9600'

os.system('sudo chmod a+rw ' + port)

serialCommunication = serial.Serial(port, baudRate, timeout=5)
time.sleep(3) # wait for Arduino



# Function for parsing the header file
def parsingHeaderFile(stringToFind, limitString, headerFilePath):
  # Parsing the header file to know when the hyperspectral image is correctly saved
  filePointer = open(headerFilePath)
```

```
  headerInformation = filePointer.read()
  # Obtaining the nBands information
  auxiliarHeaderInformation = headerInformation
  firstIndex = auxiliarHeaderInformation.find(stringToFind)
  auxiliarHeaderInformation = auxiliarHeaderInformation[firstIndex + len(stringToFind):]
  secondIndex = auxiliarHeaderInformation.find(limitString)
  auxiliarHeaderInformation = auxiliarHeaderInformation[:secondIndex + 1]
  try:
    value = int(auxiliarHeaderInformation)
  except:
    value = auxiliarHeaderInformation.strip()
  return value




# ----- SPECIM SOFTWARE ----- #
# Getting the connected camera
connectedDevices = CameraClass.Camera.get_available_devices()

if len(connectedDevices) == 1:
  camera = CameraClass.start_camera_device(connectedDevices[0], SUDO_PASSWORD)
else:
  print('\n\n No supported hyperspectral camera is connected \n\n')
  exit()

# Forming the directories for the connected hyperspectral cameras
hyperspectralImagesDirectoryForCalibration = ''
for device in connectedDevices:
  lowerStringDevice = device.lower()
```

```
  now = datetime.now() # current date and time
  currentTime = now.strftime("%H_%M_%S")
  if 'fx10' in lowerStringDevice:
    #  Create a directoy to store the Specim Fx10 hyperspectral information
    fx10CalibrationDirectoryPath = os.path.join(calibrationDirectoryPath, 'Fx10_' + currentTime)
    if not os.path.isdir(fx10CalibrationDirectoryPath):
      os.mkdir(fx10CalibrationDirectoryPath)
    hyperspectralImagesDirectoryForCalibration = fx10CalibrationDirectoryPath
  elif 'fx17' in lowerStringDevice:
    #  Create a directoy to store the Specim Fx10 hyperspectral information
    fx17CalibrationDirectoryPath = os.path.join(calibrationDirectoryPath, 'Fx17_' + currentTime)
    if not os.path.isdir(fx17CalibrationDirectoryPath):
      os.mkdir(fx17CalibrationDirectoryPath)
    hyperspectralImagesDirectoryForCalibration = fx17CalibrationDirectoryPath

# Start streaming with the hyperspectral camera in order to capture frames
camera.start_streaming()



# ----- SPECTRORADIOMETER ----- #
# Initialize spectrometer if it is connected
spectrometer = None
try:
  # Get a list of available spectrometer devices
  spectroradiometerDevices = list_devices()

  # If there is any spectrometer connected, open connection with the first one
  spectrometer = Spectrometer(spectroradiometerDevices[0])
except:
  pass
```

```
# ----- Capturing calibration information ----- #
for gain in gainList:
  # Setting the gain to the connected camera configuration
  camera.set_gain(gain)
  # Forming the directory path to store the final hyperspectral images taking into account both the gain and the exposure time information
  for exposureTime in exposureTimeList:
    # Setting the gain to the connected camera configuration
    camera.set_exposure_time(exposureTime)
    # Forming the directory path to store the final hyperspectral images taking into account both the gain and the exposure time information
    HyperspectralCameraDirectory = os.path.join(hyperspectralImagesDirectoryForCalibration, 'HyperspectralCamera')
    if not os.path.isdir(HyperspectralCameraDirectory):
      os.mkdir(HyperspectralCameraDirectory)
    hyperspectralImagesDirectoryForCalibration_ET = os.path.join(HyperspectralCameraDirectory, 'Gain_' + str(gain) + '_ExposureTime_' + str(exposureTime))
    if not os.path.isdir(hyperspectralImagesDirectoryForCalibration_ET):
      os.mkdir(hyperspectralImagesDirectoryForCalibration_ET)
    hyperspectralImagesForCalibrationFilePath = os.path.join(hyperspectralImagesDirectoryForCalibration_ET, 'hyperspectralImages_white')
    # Capturing the specified number of frames and storing them in the corresponding directory
    camera.capture_frames(NUMBER_OF_FRAMES_TO_CAPTURE_FOR_CALIBRATION, hyperspectralImagesForCalibrationFilePath)
    #  Evaluating if the saving file process finished
```

```
#    Waiting until the header file is created
headerNotCreated = True
while headerNotCreated:
  if os.path.exists(hyperspectralImagesForCalibrationFilePath + '.hdr'):
    headerNotCreated = False
  time.sleep(1)
#    Parsing nBands from header file
stringToFind = 'bands = '
limitString = '\n'
headerFilePath = hyperspectralImagesForCalibrationFilePath + '.hdr'
nBands = parsingHeaderFile(stringToFind, limitString, headerFilePath)
#    Parsing nPixels from header file
stringToFind = 'samples = '
limitString = '\n'
headerFilePath = hyperspectralImagesForCalibrationFilePath + '.hdr'
nPixels = parsingHeaderFile(stringToFind, limitString, headerFilePath)
#    Parsing data type from header file
stringToFind = 'data type = '
limitString = '\n'
headerFilePath = hyperspectralImagesForCalibrationFilePath + '.hdr'
dataType = parsingHeaderFile(stringToFind, limitString, headerFilePath)
#    Parsing the image format (bil, bip) from header file
stringToFind = 'interleave = '
limitString = '\n'
headerFilePath = hyperspectralImagesForCalibrationFilePath + '.hdr'
interleave = parsingHeaderFile(stringToFind, limitString, headerFilePath)
#    Parsing the endianness from header file
stringToFind = 'byte order = '
limitString = '\n'
headerFilePath = hyperspectralImagesForCalibrationFilePath + '.hdr'
```

```
    endianness = parsingHeaderFile(stringToFind, limitString, headerFilePath)
    #    Calculating the size of the hyperspectral image (This is not necessary when waiting for the header file, but is a double check)
    if dataType == 12:
      nBytes = 2
    elif dataType == 16:
      nBytes = 2
    elif dataType == 8:
      nBytes = 1

    hyperspectralImageSizeInBytes = nBands * nPixels * nBytes * NUMBER_OF_FRAMES_TO_CAPTURE_FOR_CALIBRATION
    #    Checking if the hyperspectral file is completelly saved
    keepWaiting = True
    while keepWaiting:
      temporalFileSize = Path(hyperspectralImagesForCalibrationFilePath + '.bin').stat().st_size
      if temporalFileSize >= hyperspectralImageSizeInBytes:
        keepWaiting = False
      else:
        time.sleep(1)

    # Once the hyperspectral cube for the white reference is captured, the cube is reduced calculating the mean of the NUMBER_OF_FRAMES_TO_CAPTURE_FOR_CALIBRATION obtained
    hyperImage = HyperImage(hyperspectralImagesDirectoryForCalibration_ET)


    # Capture information with the Daniel equipment
    parametersList = []
    valuesList = []
```

```
    #  Forming the directory path to store the spectrometer data taking into account both the gain and the exposure time information
    hyperspectralImagesDirectoryForCalibration_Sensors = os.path.join(hyperspectralImagesDirectoryForCalibration, 'Sensors')
    if not os.path.isdir(hyperspectralImagesDirectoryForCalibration_Sensors):
      os.mkdir(hyperspectralImagesDirectoryForCalibration_Sensors)
    hyperspectralImagesDirectoryForCalibration_Sensors = os.path.join(hyperspectralImagesDirectoryForCalibration_Sensors, 'Gain_' + str(gain) + '_ExposureTime_' + str(exposureTime))
    if not os.path.isdir(hyperspectralImagesDirectoryForCalibration_Sensors):
      os.mkdir(hyperspectralImagesDirectoryForCalibration_Sensors)

    #  Asking for the current luminance value
    parametersList.append('luminance')
    orderInfomation = 'luminance'
    communicationOutput = serialCommunicationWithArduino(serialCommunication, orderInfomation)
    time.sleep(1)

    resultOk, variableValue, errMsg = variableParser(communicationOutput)
    if resultOk:
      # valuesList.append(variableValue)
      maximumLuminanceValue = 54612.49
      minimumLuminanceValue = 0
      normalizeValue = (variableValue - minimumLuminanceValue)/(maximumLuminanceValue - minimumLuminanceValue)
      valuesList.append(normalizeValue)
    else:
      valuesList.append('')
      print('ERROR: ', errMsg)
```

```
    # Capture information with the Humidity and Temperature sensors
    #  Asking for the current temperature and humidity values
    orderInfomation = 'temperatureAndHumidity'
    communicationOutput = serialCommunicationWithArduino(serialCommunication,
orderInfomation)
    time.sleep(1)
    temperatureStringToBeParsed = communicationOutput[:communicationOutput.index(';')]
    # Get the temperature
    parametersList.append('temperature')
    resultOk, variableValue, errMsg = variableParser(temperatureStringToBeParsed)
    if resultOk:
      valuesList.append(variableValue)
    else:
      valuesList.append('')
      print('ERROR: ', errMsg)
    # Get the humidity value
    parametersList.append('humidity')
    humidityStringToBeParsed = communicationOutput[communicationOutput.index(';')+1:]
    resultOk, variableValue, errMsg = variableParser(humidityStringToBeParsed)
    if resultOk:
      valuesList.append(variableValue)
    else:
      valuesList.append('')
      print('ERROR: ', errMsg)

    filepath = os.path.join(hyperspectralImagesDirectoryForCalibration_Sensors,
'GeneralSensors.json')
    saveJsonFile(filepath, parametersList, valuesList)
```

```
    # Capture information with the RGB camera
    hyperspectralImagesDirectoryForCalibration_Sensors =
os.path.join(hyperspectralImagesDirectoryForCalibration, 'Sensors')
    if not os.path.isdir(hyperspectralImagesDirectoryForCalibration_Sensors):
      os.mkdir(hyperspectralImagesDirectoryForCalibration_Sensors)
    hyperspectralImagesDirectoryForCalibration_Sensors =
os.path.join(hyperspectralImagesDirectoryForCalibration_Sensors, 'Gain_' + str(gain) +
'_ExposureTime_' + str(exposureTime))
    if not os.path.isdir(hyperspectralImagesDirectoryForCalibration_Sensors):
      os.mkdir(hyperspectralImagesDirectoryForCalibration_Sensors)



    elpJsonFilePath = os.path.join(hyperspectralImagesDirectoryForCalibration_Sensors,
'elpCamera.json')
    capturingFramesForEachParameterValue(currentElpCamera, elpGainList,
elpExposureTimeList, elpJsonFilePath, ELP_Offset)

    # Capture information with the Radiospectrometer
    #  Run the function to obtain all the spectroradiometer information if it is connected
    if spectrometer is not None:
      spectroradiometerJsonFilePath =
os.path.join(hyperspectralImagesDirectoryForCalibration_Sensors,
'spectroradiometerInformation.json')
      captureInformationWithConnectedSpectroradiometer(spectrometer,
customMinimumIntegrationTime, numberOfSpectroradiometerIntegrationTimeSamples,
numberOfSamplesToCalculateTheMeanSpectralValue,
spectroradiometerIntegrationTimeStepSizeInMicroseconds, spectrometerMaxValue,
spectrometerMinValue, blueSpectralRange, greenSpectralRange, redSpectralRange,
spectroradiometerJsonFilePath)
```

```
# Stop streaming after capturing the desired number of frames
camera.stop_streaming()
```

Anexo 2: Aplicación para crear el dataset completo "JsonProgramme"

jsonProgramme.py
Aplicación para crear el dataset completo

```
import os, json
import numpy
import pandas as pd

from DataIO import *

# Getting the current directory path
currentWorkingDirectory = os.getcwd()

# ----- CONSTANTS ----- #
CURRENT_HYPERSPECTRAL_CAMERA = 'FX10'
PERCENTAGE_TO_CONSIDER_THE_WHITE_REFERENCE_AS_SATURATED = 95 #
maxAllowedVAlue=2**(dynamicRangeInBits-
1)*PERCENTAGE_TO_CONSIDER_THE_WHITE_REFERENCE_AS_SATURATED/100
OUTPUT_DIRECTORY_PATH = 'Output'
CALIBRATION_DATA_DIRECTORY_PATH = currentWorkingDirectory +
'/WhiteReferences'


# ----- INPUTS ----- #
#  Spectroradiometer
maxSpectroradiometerNormalizedValue = 0.90 # A value higher than
this one is considered saturated
#  Hyperspectral cameras
if CURRENT_HYPERSPECTRAL_CAMERA == 'FX10':
   wavelength =
```

```
numpy.array([396.86,399.49,402.12,404.75,407.38,410.01,412.64,415.
27,417.90,420.54,423.17,425.81,428.44,431.08,433.72,436.36,439.00,
441.64,444.28,446.92,449.56,452.21,454.85,457.50,460.14,462.79,465
.44,468.09,470.74,473.39,476.04,478.69,481.34,484.00,486.65,489.31
,491.96,494.62,497.28,499.94,502.60,505.26,507.92,510.58,513.24,51
5.90,518.57,521.23,523.90,526.57,529.23,531.90,534.57,537.24,539.9
1,542.59,545.26,547.93,550.61,553.28,555.96,558.63,561.31,563.99,5
66.67,569.35,572.03,574.71,577.39,580.08,582.76,585.45,588.13,590.
82,593.51,596.19,598.88,601.57,604.26,606.95,609.65,612.34,615.03,
617.73,620.42,623.12,625.82,628.52,631.22,633.92,636.62,639.32,642
.02,644.72,647.43,650.13,652.84,655.54,658.25,660.96,663.67,666.38
,669.09,671.80,674.51,677.22,679.94,682.65,685.37,688.08,690.80,69
3.52,696.24,698.96,701.68,704.40,707.12,709.84,712.57,715.29,718.0
2,720.74,723.47,726.20,728.93,731.66,734.39,737.12,739.85,742.58,7
45.32,748.05,750.79,753.52,756.26,759.00,761.74,764.47,767.22,769.
96,772.70,775.44,778.18,780.93,783.67,786.42,789.17,791.91,794.66,
797.41,800.16,802.91,805.66,808.42,811.17,813.92,816.68,819.44,822
.19,824.95,827.71,830.47,833.23,835.99,838.75,841.51,844.28,847.04
,849.80,852.57,855.34,858.10,860.87,863.64,866.41,869.18,871.95,87
4.73,877.50,880.27,883.05,885.82,888.60,891.38,894.16,896.93,899.7
1,902.49,905.28,908.06,910.84,913.63,916.41,919.20,921.98,924.77,9
27.56,930.35,933.14,935.93,938.72,941.51,944.30,947.10,949.89,952.
69,955.48,958.28,961.08,963.88,966.68,969.48,972.28,975.08,977.88,
980.69,983.49,986.30,989.10,991.91,994.72,997.53,1000.33,1003.15])
    dynamicRangeInBits = 12
    nBands = 224
    nPixels = 1024
    interleaf = 'bil'
    endianness = 'ieee-le'
    dimArray = [nBands, nPixels, 1] # [nr, nc, nb]
    dataType = 'uint16'
    maxAllowedVAlue = math.ceil((2**dynamicRangeInBits-
1)*PERCENTAGE_TO_CONSIDER_THE_WHITE_REFERENCE_AS_SATURATED/100)
```

```
elif CURRENT_HYPERSPECTRAL_CAMERA == 'FX17':
    pass
else:
    print('ERROR: SPECIFIED HYPERSPECTRAL CAMERA IS NOT SUPPORTED')
    exit()


# ----- WHITE REFERENCES (WR) ----- #

# The following code is done in order to find out the minimum
exposure time that satisfies 2 conditions:
#  1 - WR is not saturated
#  2 - It is present in all the captured data

# List that contains the maximum value of Exposure Time for each
capture in which it is not saturated the hyperspectral image
noSaturatedWR_maxExposureTimesList = []

# Analyze all the hyperspectral images to know the target exposure
time
for dailyDirectory in os.listdir(CALIBRATION_DATA_DIRECTORY_PATH):
    if dailyDirectory != '.DS_Store':
        # Go through all the daily captures to get the desired
hyperspectral information
        for directory in
os.listdir(os.path.join(CALIBRATION_DATA_DIRECTORY_PATH,
dailyDirectory)):
            if directory != '.DS_Store':
                whiteReferenceDirectoryPath =
os.path.join(CALIBRATION_DATA_DIRECTORY_PATH, dailyDirectory,
directory, 'HyperspectralCamera')
                # List to store all the Exposure Times of a daily and
not saturated capture
```

```
            noSaturatedWR_exposureTimeList = []
            # Go through the daily directories to get the
information related to all the gain and exposure time selected
            for gainAndExposureTimeDirectory in
os.listdir(whiteReferenceDirectoryPath):
                if gainAndExposureTimeDirectory != '.DS_Store':
                    # Analyze only the white references for gain==1
                    firstIndex =
gainAndExposureTimeDirectory.index('_')
                    strGain =
gainAndExposureTimeDirectory[firstIndex+1:]
                    secondIndex = strGain.index('_')
                    strGain = str(strGain[:secondIndex])
                    if float(strGain) == 1.0:
                        # Get the Exposure Time used to capture the
current WR
                        index =
gainAndExposureTimeDirectory.index('_')
                        strET =
gainAndExposureTimeDirectory[index+1:]
                        index = strET.index('_')
                        strET = strET[index+1:]
                        index = strET.index('_')
                        strET = strET[index+1:]
                        currentExposureTime = float(strET)
                        # Load the white reference data
                        wrFilePath =
os.path.join(whiteReferenceDirectoryPath,
gainAndExposureTimeDirectory, 'hyperspectralImages_white.bin')
                        WR = multibandread(wrFilePath, dimArray,
dataType, interleaf, endianness)
                        # Calculate the mean value per band of the
loaded WR
```

```
                    WR_mean = numpy.mean(WR, axis=1)
                    # Get the max value of the mean white
reference
                    WR_maxValue = numpy.amax(WR_mean)
                    # Check if the current mean white reference
is saturated or not
                    if WR_maxValue <= maxAllowedVAlue:

noSaturatedWR_exposureTimeList.append(float(currentExposureTime))

noSaturatedWR_maxExposureTimesList.append(noSaturatedWR_exposureTi
meList)

noSaturatedWR_maximumExposureTimeForEachCaptureList = []
for maximumExposureTimeList in noSaturatedWR_maxExposureTimesList:
   if len(maximumExposureTimeList) > 0:
      # Get the maximum Exposure Time with no WR saturation
      maximumExposureTimeList.sort()
      maximumExposureTime = maximumExposureTimeList[-1]
      # Append this value to the general list

noSaturatedWR_maximumExposureTimeForEachCaptureList.append(maximum
ExposureTime)
# Main goal of this portion of code
noSaturatedWR_maximumExposureTimeForEachCaptureList.sort()
minimumExposureTimeValueWithNoSaturatedWR =
int(noSaturatedWR_maximumExposureTimeForEachCaptureList[0])
# Check that all the captures has no saturated WR with an ET value
of minimumExposureTimeValueWithNoSaturatedWR
minimumExposureTimeValueWithNoSaturatedWRIsValid = True
for maximumExposureTimeList in noSaturatedWR_maxExposureTimesList:
   if minimumExposureTimeValueWithNoSaturatedWR not in
maximumExposureTimeList:
```

```
        minimumExposureTimeValueWithNoSaturatedWRIsValid = False
if not minimumExposureTimeValueWithNoSaturatedWRIsValid:
    print('ERROR: EXPOSURE TIME SELECTED IS NOT VALID')
    exit()



# ----- SPECTRORADIOMETER ----- #
# The following code is done in order to find out the Integration
Time that satisfies 2 conditions:
#   1 - Spectroradiometer data are not saturated
#   2 - It is present in all the captured data

# List that contains the maximum value of Exposure Time for each
capture in which it is not saturated the hyperspectral image
noSaturatedSpectroData_maxIntegrationTimesList = []

# Analyze all the spectroradiometer data to know the target
integration time
for dailyDirectory in os.listdir(CALIBRATION_DATA_DIRECTORY_PATH):
    if dailyDirectory != '.DS_Store':
        # Go through all the daily directories to get the desired
information
        for directory in
os.listdir(os.path.join(CALIBRATION_DATA_DIRECTORY_PATH,
dailyDirectory)):
            if directory != '.DS_Store':
                # Path to the main sensors directory
                sensorsInformationDirectoryPath =
os.path.join(CALIBRATION_DATA_DIRECTORY_PATH, dailyDirectory,
directory, 'Sensors')
                # List to store all the Integration Times of a daily
and not saturated capture
```

```
            noSaturatedSpectroData_integrationTimeList = []
            # Go through the daily directories to get the
information related to the selected gain
            gainAndExposureTimeDirectory = 'Gain_' + str(1.0) +
'_ExposureTime_' + str(minimumExposureTimeValueWithNoSaturatedWR)
            currentSensorsInformationDirectoryPath =
os.path.join(sensorsInformationDirectoryPath,
gainAndExposureTimeDirectory)
            spectroradiometerInformationFilePath =
os.path.join(currentSensorsInformationDirectoryPath,
'spectroradiometerInformation.json')
            # Read the spectroradiometer information json file
            filePointer =
open(spectroradiometerInformationFilePath, 'r')
            jsonInformation = json.loads(filePointer.read())
            filePointer.close()
            for spectroradiometerCapture in jsonInformation:
               if
spectroradiometerCapture["spectrometer_simulatedMean_R_Normalized"
] <= maxSpectroradiometerNormalizedValue \
                and
spectroradiometerCapture["spectrometer_simulatedMean_G_Normalized"
] <= maxSpectroradiometerNormalizedValue \
                and
spectroradiometerCapture["spectrometer_simulatedMean_B_Normalized"
] <= maxSpectroradiometerNormalizedValue \
                and
spectroradiometerCapture["spectrometer_normalizedBrightness"] <=
maxSpectroradiometerNormalizedValue:

noSaturatedSpectroData_integrationTimeList.append(spectroradiomete
rCapture["spectrometer_integrationTime"])
            if len(noSaturatedSpectroData_integrationTimeList) >
```

```
0:

noSaturatedSpectroData_maxIntegrationTimesList.append(noSaturatedS
pectroData_integrationTimeList)

noSaturatedSpectroData_maximumIntegrationTimeForEachCaptureList =
[]
for maximumIntegrationTimeList in
noSaturatedSpectroData_maxIntegrationTimesList:
   if len(maximumIntegrationTimeList) > 0:
      # Get the maximum integration time with no spectroradiometer
saturated data
      maximumIntegrationTimeList.sort()
      maximumIntegrationTime = maximumIntegrationTimeList[-1]
      # Append this value to the general list

noSaturatedSpectroData_maximumIntegrationTimeForEachCaptureList.ap
pend(maximumIntegrationTime)
# Get the minimum IT of the previous list that satisfied both
conditions:
noSaturatedSpectroData_maximumIntegrationTimeForEachCaptureList.so
rt()
minimumIntegrationTimeValueWithNoSaturatedSpectroData =
noSaturatedSpectroData_maximumIntegrationTimeForEachCaptureList[0]
# Check that all the captures has no saturated spectroradiometer
data with an integration time value of
minimumIntegrationTimeValueWithNoSaturatedSpectroData
minimumIntegrationTimeValueWithNoSaturatedSpectroDataIsValid =
True
for IT_List in noSaturatedSpectroData_maxIntegrationTimesList:
   if minimumIntegrationTimeValueWithNoSaturatedSpectroData not in
IT_List:
      minimumIntegrationTimeValueWithNoSaturatedSpectroDataIsValid
```

```
= False

if not
minimumIntegrationTimeValueWithNoSaturatedSpectroDataIsValid:
    exit()

#  ----- STORE NECESSARY INFORMATION INTO A JSON FILE ----- #
jsonInformationList = []
jsonInformationListELP = []

for dailyDirectory in os.listdir(CALIBRATION_DATA_DIRECTORY_PATH):
    if dailyDirectory != '.DS_Store':
        # Go through all the directories to get the desired
information
        for directory in
os.listdir(os.path.join(CALIBRATION_DATA_DIRECTORY_PATH,
dailyDirectory)):
            if directory != '.DS_Store':
                customJsonInformationDict = {}
                customJsonInformationDictELP = {}
                whiteReferenceDirectoryPath =
os.path.join(CALIBRATION_DATA_DIRECTORY_PATH, dailyDirectory,
directory, 'HyperspectralCamera')
                spectroradiometerInformationDirectoryPath  =
os.path.join(CALIBRATION_DATA_DIRECTORY_PATH, dailyDirectory,
directory, 'Sensors')
                # Selected directory for Gain == 1.0 & ET ==
minimumExposureTimeValueWithNoSaturatedWR
                selectedDirectoryPath = 'Gain_1.0_ExposureTime_' +
str(minimumExposureTimeValueWithNoSaturatedWR)
                # Selected Hyperspectral Image
                selectedHyperspectralImageFilePath =
os.path.join(whiteReferenceDirectoryPath, selectedDirectoryPath,
```

```
'hyperspectralImages_white.bin')
            # Selected Spectroradiometer File
            selectedSpectroradiometerInformationFilePath =
os.path.join(spectroradiometerInformationDirectoryPath,
selectedDirectoryPath, 'spectroradiometerInformation.json')
            # General sensors file
            selectedGeneralSensorsInformationFilePath =
os.path.join(spectroradiometerInformationDirectoryPath,
selectedDirectoryPath, 'GeneralSensors.json')
            # ELP camera file
            selectedELPCameraInformationFilePath =
os.path.join(spectroradiometerInformationDirectoryPath,
selectedDirectoryPath, 'elpCamera.json')
            # Read the spectroradiometer json information
            filePointer =
open(selectedSpectroradiometerInformationFilePath, 'r')
            jsonInformation = json.loads(filePointer.read())
            filePointer.close()
            for spectroradiometerCapture in jsonInformation:
                if
spectroradiometerCapture["spectrometer_integrationTime"] ==
minimumIntegrationTimeValueWithNoSaturatedSpectroData:
                    currentSpectrometerIntegrationTime =
spectroradiometerCapture["spectrometer_integrationTime"]
                    currentSpectrometerNormalized_R =
spectroradiometerCapture["spectrometer_simulatedMean_R_Normalized"
]
                    currentSpectrometerNormalized_G =
spectroradiometerCapture["spectrometer_simulatedMean_G_Normalized"
]
                    currentSpectrometerNormalized_B =
spectroradiometerCapture["spectrometer_simulatedMean_B_Normalized"
]
```

```
                currentSpectrometerBrightness =
spectroradiometerCapture["spectrometer_normalizedBrightness"]
            # Read the general sensors json information
            filePointer =
open(selectedGeneralSensorsInformationFilePath, 'r')
            jsonInformation = json.loads(filePointer.read())
            filePointer.close()

            # Read the ELP CAMERA json information
            filePointerELP =
open(selectedELPCameraInformationFilePath, 'r')
            jsonInformationELP = json.loads(filePointerELP.read())
            filePointerELP.close()

            currentSensorsHumidity = jsonInformation["humidity"]
            currentSensorsTemperature =
jsonInformation["temperature"]
            currentSensorsLuminance = jsonInformation["luminance"]

            currentELPCameraRed =
jsonInformationELP[0]['0']['1']['R']
            currentELPCameraGreen =
jsonInformationELP[0]['0']['1']["G"]
            currentELPCameraBlue =
jsonInformationELP[0]['0']['1']["B"]


            # Load the WR of all the captures for the following
case: Gain == 1.0 & ET ==
minimumExposureTimeValueWithNoSaturatedWR
            WR = multibandread(selectedHyperspectralImageFilePath,
dimArray, dataType, interleaf, endianness)
            # Calculate the mean value per band of the WR capture
```

```
            WR_mean = numpy.mean(WR, axis=1)
            # Discard the first 20 bands and the last 44 ones due
to a low Signal-To-Noise-Ratio
            initialIndexes = range(0,20)
            finalIndexes = range(224-44, 224)
            WR_mean = numpy.delete(WR_mean, finalIndexes)
            WR_mean = numpy.delete(WR_mean, initialIndexes)
            # Normalize WR_mean
            Normalize_WR_mean = WR_mean/(2**dynamicRangeInBits-1)
            # Calculate the mean value of every Z bands to reduce
the amount of data to be analyzed
            index = 0
            SmoothWR = []
            while index < Normalize_WR_mean.shape[0]:
               counter = 0
               accumulator = 0
               while counter < 5:
                  accumulator = accumulator +
Normalize_WR_mean[index]
                  counter = counter + 1
                  index = index + 1
               meanValue = accumulator / counter
               SmoothWR.append(meanValue)
            #Spectroradiometer inputs + general sensors + WR
            customJsonInformationDict["Input_R_Normalized"] =
currentSpectrometerNormalized_R
            customJsonInformationDict["Input_G_Normalized"] =
currentSpectrometerNormalized_G
            customJsonInformationDict["Input_B_Normalized"] =
currentSpectrometerNormalized_B

customJsonInformationDict["Input_Brigthness_Normalized"] =
currentSpectrometerBrightness
```

```
            customJsonInformationDict["Input_Humidity"] =
currentSensorsHumidity/100
            customJsonInformationDict["Input_Temperature"] =
currentSensorsTemperature/50
            i = 0
            for data in SmoothWR:
                title = "Output_WR_Normalized_" + str(i)
                customJsonInformationDict[title] = SmoothWR[i]
                i=i+1

            jsonInformationList.append(customJsonInformationDict)

            # ELP inputs + general sensors + WR
            customJsonInformationDictELP["Input_R_Normalized_ELP"]
= currentELPCameraRed
            customJsonInformationDictELP["Input_G_Normalized_ELP"]
= currentELPCameraGreen
            customJsonInformationDictELP["Input_B_Normalized_ELP"]
= currentELPCameraBlue
            customJsonInformationDictELP["Input_Luminance"] =
currentSensorsLuminance
            customJsonInformationDictELP["Input_Humidity"] =
currentSensorsHumidity/100
            customJsonInformationDictELP["Input_Temperature"] =
currentSensorsTemperature/50
            i = 0
            for data in SmoothWR:
                title = "Output_WR_Normalized_" + str(i)
                customJsonInformationDictELP[title] = SmoothWR[i]
                i=i+1


jsonInformationListELP.append(customJsonInformationDictELP)
```

```
# Store the necessary information of spectroradiometer in a json
file
if not os.path.isdir(OUTPUT_DIRECTORY_PATH):
   os.mkdir(OUTPUT_DIRECTORY_PATH)
outputFilePath = os.path.join(OUTPUT_DIRECTORY_PATH,
'calibrationInformation_Spectroradiometer.json')
filePointer = open(outputFilePath, 'w')
filePointer.write(json.dumps(jsonInformationList))
filePointer.close()

df = pd.read_json (outputFilePath)
outputFilePathCSV = os.path.join(OUTPUT_DIRECTORY_PATH,
'calibrationInformation_Spectroradiometer.csv')
df.to_csv (outputFilePathCSV,index = None)

# Store the necessary information of ELP camera in a json file
if not os.path.isdir(OUTPUT_DIRECTORY_PATH):
   os.mkdir(OUTPUT_DIRECTORY_PATH)
outputFilePath = os.path.join(OUTPUT_DIRECTORY_PATH,
'calibrationInformation_ELP.json')
filePointer = open(outputFilePath, 'w')
filePointer.write(json.dumps(jsonInformationListELP))
filePointer.close()

df = pd.read_json (outputFilePath)
outputFilePathCSV = os.path.join(OUTPUT_DIRECTORY_PATH,
'calibrationInformation_ELP.csv')
df.to_csv (outputFilePathCSV,index = None
```

Anexo 3: Aplicación de regresión lineal

LinearRegresion.ipynb
Aplicación de regresión lineal

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv)
import datetime
import math
from sklearn import linear_model
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score , mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
import matplotlib.pyplot as plt
import seaborn as sb
from scipy import stats
%matplotlib inline

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing
Shift+Enter) will list all files under the input directory

import os
```

```
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
#.
INPUT_DIR = '../input/results-150720211/'
df_results_spectroradiometer = pd.read_csv(INPUT_DIR +
'calibrationInformation_Spectroradiometer.csv')
df_results_spectroradiometer
df_results_spectroradiometer.shape
df_results_spectroradiometer.dtypes
#.
INPUT_DIR = '../input/results-150720211/'
df_results_ELP = pd.read_csv(INPUT_DIR +
'calibrationInformation_ELP.csv')
df_results_ELP
X_spectroradiometer =
np.array(df_results_spectroradiometer.drop(['Input_Brigthness_Norm
alized', 'Input_R_Normalized','Input_G_Normalized',
'Input_B_Normalized','Input_Humidity','Input_Temperature'],1))

y_Brigthness_spectroradiometer =
np.expand_dims(np.array(df_results_spectroradiometer['Input_Brigth
ness_Normalized']), axis=1)
y_Normalized_R_spectroradiometer =
np.expand_dims(np.array(df_results_spectroradiometer['Input_R_Norm
alized']), axis=1)
y_Normalized_G_spectroradiometer =
np.expand_dims(np.array(df_results_spectroradiometer['Input_G_Norm
alized']), axis=1)
y_Normalized_B_spectroradiometer =
np.expand_dims(np.array(df_results_spectroradiometer['Input_B_Norm
alized']), axis=1)
```

```
y_Humidity_spectroradiometer =
np.expand_dims(np.array(df_results_spectroradiometer['Input_Humidi
ty']), axis=1)
y_Temperature_spectroradiometer =
np.expand_dims(np.array(df_results_spectroradiometer['Input_Temper
ature']), axis=1)


y_All_spectroradiometer =
np.concatenate((y_Brigthness_spectroradiometer,y_Normalized_R_spec
troradiometer,y_Normalized_G_spectroradiometer,y_Normalized_B_spec
troradiometer,y_Humidity_spectroradiometer,y_Temperature_spectrora
diometer), axis=1)

y_RGB_spectroradiometer =
np.concatenate((y_Normalized_R_spectroradiometer,y_Normalized_G_sp
ectroradiometer,y_Normalized_B_spectroradiometer), axis=1)

y_Humidity_Temperature_spectroradiometer =
np.concatenate((y_Humidity_spectroradiometer,y_Temperature_spectro
radiometer), axis=1)
X_train_Brigthness_spectroradiometer,
X_test_Brigthness_spectroradiometer,
y_train_Brigthness_spectroradiometer,
y_test_Brigthness_spectroradiometer = train_test_split(
X_spectroradiometer, y_Brigthness_spectroradiometer,
test_size=0.2, random_state=0)

X_train_All_spectroradiometer, X_test_All_spectroradiometer,
y_train_All_spectroradiometer, y_test_All_spectroradiometer =
train_test_split( X_spectroradiometer, y_All_spectroradiometer,
test_size=0.2, random_state=0)
```

```
X_train_RGB_spectroradiometer, X_test_RGB_spectroradiometer,
y_train_RGB_spectroradiometer, y_test_RGB_spectroradiometer =
train_test_split( X_spectroradiometer, y_RGB_spectroradiometer,
test_size=0.2, random_state=0)

X_train_Humidity_Temperature_spectroradiometer,
X_test_Humidity_Temperature_spectroradiometer,
y_train_Humidity_Temperature_spectroradiometer,
y_test_Humidity_Temperature_spectroradiometer = train_test_split(
X_spectroradiometer, y_Humidity_Temperature_spectroradiometer,
test_size=0.2, random_state=0)
LR_All_spectroradiometer = LinearRegression()
LR_All_spectroradiometer.fit(X_train_All_spectroradiometer,
y_train_All_spectroradiometer)
y_pred_All_spectroradiometer =
LR_All_spectroradiometer.predict(X_test_All_spectroradiometer)

LR_Brigthness_spectroradiometer = LinearRegression()
LR_Brigthness_spectroradiometer.fit(X_train_Brigthness_spectroradi
ometer, y_train_Brigthness_spectroradiometer)
y_pred_Brigthness_spectroradiometer =
LR_Brigthness_spectroradiometer.predict(X_test_Brigthness_spectror
adiometer)

LR_RGB_spectroradiometer = LinearRegression()
LR_RGB_spectroradiometer.fit(X_train_RGB_spectroradiometer,
y_train_RGB_spectroradiometer)
y_pred_RGB_spectroradiometer =
LR_RGB_spectroradiometer.predict(X_test_RGB_spectroradiometer)

LR_Humidity_Temperature_spectroradiometer = LinearRegression()
```

```
LR_Humidity_Temperature_spectroradiometer.fit(X_train_Humidity_Tem
perature_spectroradiometer,
y_train_Humidity_Temperature_spectroradiometer)
y_pred_Humidity_Temperature_spectroradiometer =
LR_Humidity_Temperature_spectroradiometer.predict(X_test_Humidity_
Temperature_spectroradiometer)
def SNR (y_pred,y_ref):
    diff = y_pred - y_ref
    diff2 = np.multiply(diff,diff)
    denominator = np.sum(diff2)
    num = np.sum(np.multiply(y_ref, y_ref))
    snr = 20 * math.log10(num/denominator)
    return snr
rmse_LR_All_spectroradiometer = mean_squared_error(y_true  =
y_test_All_spectroradiometer, y_pred =
y_pred_All_spectroradiometer ,squared = False)

rmse_LR_Brigthness_spectroradiometer = mean_squared_error(y_true
= y_test_Brigthness_spectroradiometer, y_pred =
y_pred_Brigthness_spectroradiometer ,squared = False)

rmse_LR_RGB_spectroradiometer = mean_squared_error(y_true  =
y_test_RGB_spectroradiometer, y_pred =
y_pred_RGB_spectroradiometer ,squared = False)

rmse_LR_Humidity_Temperature_spectroradiometer =
mean_squared_error(y_true  =
y_test_Humidity_Temperature_spectroradiometer, y_pred =
y_pred_Humidity_Temperature_spectroradiometer ,squared = False)

print(f"El RMSE de test LR_All_spectroradiometer es:
{rmse_LR_All_spectroradiometer}")
```

```
print(f"El RMSE de test LR_Brigthness_spectroradiometer es:
{rmse_LR_Brigthness_spectroradiometer}")
print(f"El RMSE de test LR_RGB_spectroradiometer es:
{rmse_LR_RGB_spectroradiometer}")
print(f"El RMSE de test LR_Humidity_Temperature_spectroradiometer
es: {rmse_LR_Humidity_Temperature_spectroradiometer}")

print ("El SNR de test LR_All_spectroradiometer es: ",
       SNR(y_pred_All_spectroradiometer,
y_test_All_spectroradiometer))
print ("El SNR de test LR_Brigthness_spectroradiometer es: ",
       SNR(y_pred_Brigthness_spectroradiometer,
y_test_Brigthness_spectroradiometer))
print ("El SNR de test LR_RGB_spectroradiometer es: ",
       SNR(y_pred_RGB_spectroradiometer,
y_test_RGB_spectroradiometer))
print ("El SNR de test LR_Temperature_spectroradiometer es: ",
       SNR(y_pred_Humidity_Temperature_spectroradiometer,
y_test_Humidity_Temperature_spectroradiometer))
```

Anexo 4: Aplicación para controlar los sensores BH1750 y DHT11

mainSerialCommunication.ino
Aplicación para controlar los sensores BH1750 y DHT11

```
#include <Wire.h>
#include <BH1750.h>
#include <math.h>
#include <DHT.h>



// ======== Defining variables for SERIAL COMMUNICATION ======== //
String readString;
String responseValue;



// ======== Defining variables for LUMINANCE sensor ======== //
BH1750 luxometro;
int numberOfSamples = 20;
float luminance;

const byte luxMode = BH1750::CONTINUOUS_HIGH_RES_MODE;
// BH1750_CONTINUOUS_HIGH_RES_MODE
// BH1750_CONTINUOUS_HIGH_RES_MODE_2
// BH1750_CONTINUOUS_LOW_RES_MODE
// BH1750_ONE_TIME_HIGH_RES_MODE
// BH1750_ONE_TIME_HIGH_RES_MODE_2
```

```
// BH1750_ONE_TIME_LOW_RES_MODE



// ======== Defining variables for TEMPERATURE and HUMIDITY sensor ======== //
#define DHTPIN_1 2
#define DHTPIN_2 4
#define DHTPIN_3 6
// Depending on the type of sensor
#define DHTTYPE DHT11

// Inicializamos el sensor DHT11
DHT dht_1(DHTPIN_1, DHTTYPE);
DHT dht_2(DHTPIN_2, DHTTYPE);
DHT dht_3(DHTPIN_3, DHTTYPE);




void setup() {
  // Initializing serial port
  Serial.begin(9600);
  // Initializing sensor
  luxometro.begin(luxMode);
  // Initializing temperature and humidity sensors
  dht_1.begin();
  dht_2.begin();
  dht_3.begin();
}
```

```
float getLuminanceValue() {

 float lux = 0.0;
 float luxAccumulator = 0.0;
 int counter = 0;
 float meanLuxForSamples;
 while(counter < numberOfSamples){
  lux = luxometro.readLightLevel(); // Lectura del BH1750
  //Serial.print(F("\nIluminancia:  "));
  //Serial.print(lux);
  luxAccumulator = luxAccumulator + lux;
  counter = counter + 1;
 }
 meanLuxForSamples = luxAccumulator / numberOfSamples;
 //Serial.print(F("\nMeanIluminancia:  "));
 //Serial.print(meanLuxForSamples);
 return meanLuxForSamples;

}



float getTemperature(){

 float temperatureSensor_1, temperatureSensor_2, temperatureSensor_3,
      meanTemperature_1, meanTemperature_2, meanTemperature_3, meanTemperature,
      accumulator_T1, accumulator_T2, accumulator_T3, totalAccumulator,
      minTemp_1, maxTemp_1, minTemp_2, maxTemp_2, minTemp_3, maxTemp_3;
 int acceptableDifferencePercentage, counter_T1, counter_T2, counter_T3, index,
```

```
  maxNumberOfSamples, totalCounter;

acceptableDifferencePercentage = 10;
maxNumberOfSamples = 10;
index = 0;

accumulator_T1 = 0;
counter_T1 = 0;
accumulator_T2 = 0;
counter_T2 = 0;
accumulator_T3 = 0;
counter_T3 = 0;

while (index <= maxNumberOfSamples){
 // Get the temperature in Celsius (by default)
 temperatureSensor_1 = dht_1.readTemperature();
 temperatureSensor_2 = dht_2.readTemperature();
 temperatureSensor_3 = dht_3.readTemperature();
 // Calculate the range to filter bad data from different sensors
 minTemp_1 = temperatureSensor_1 * (1 - acceptableDifferencePercentage/100);
 maxTemp_1 = temperatureSensor_1 * (1 + acceptableDifferencePercentage/100);
 minTemp_2 = temperatureSensor_2 * (1 - acceptableDifferencePercentage/100);
 maxTemp_2 = temperatureSensor_2 * (1 + acceptableDifferencePercentage/100);
 minTemp_3 = temperatureSensor_3 * (1 - acceptableDifferencePercentage/100);
 maxTemp_3 = temperatureSensor_3 * (1 + acceptableDifferencePercentage/100);

 if ((temperatureSensor_1 > minTemp_2) | (temperatureSensor_1 <= maxTemp_2) &
    (temperatureSensor_1 > minTemp_3) | (temperatureSensor_1 <= maxTemp_3)){
  accumulator_T1 = accumulator_T1 + temperatureSensor_1;
  counter_T1 = counter_T1 + 1;
```

```
  }

  if ((temperatureSensor_2 > minTemp_1) | (temperatureSensor_2 <= maxTemp_1) &
     (temperatureSensor_2 > minTemp_3) | (temperatureSensor_2 <= maxTemp_3)){
    accumulator_T2 = accumulator_T2 + temperatureSensor_2;
    counter_T2 = counter_T2 + 1;
  }

  if ((temperatureSensor_3 > minTemp_1) | (temperatureSensor_3 <= maxTemp_1) &
     (temperatureSensor_3 > minTemp_2) | (temperatureSensor_3 <= maxTemp_2)){
    accumulator_T3 = accumulator_T3 + temperatureSensor_3;
    counter_T3 = counter_T3 + 1;
  }

  index = index + 1;
}

totalCounter = 0;
totalAccumulator = 0;

if (counter_T1 > 0) {
    meanTemperature_1 = accumulator_T1 / counter_T1;
    totalAccumulator = totalAccumulator + meanTemperature_1;
    totalCounter = totalCounter + 1;
    //Serial.print(totalAccumulator);
}
if (counter_T2 > 0) {
    meanTemperature_2 = accumulator_T2 / counter_T2;
    totalAccumulator = totalAccumulator + meanTemperature_2;
    totalCounter = totalCounter + 1;
```

```
    //Serial.print(totalAccumulator);
  }
  if (counter_T3 > 0) {
    meanTemperature_3 = accumulator_T3 / counter_T3;
    totalAccumulator = totalAccumulator + meanTemperature_3;
    totalCounter = totalCounter + 1;
    //Serial.print(totalAccumulator);
  }

  //Serial.print(totalCounter);

  if (totalCounter > 0) {
   meanTemperature = totalAccumulator / totalCounter;
  }
  return meanTemperature;

}


float getHumidity(){

  float humiditySensor_1, humiditySensor_2, humiditySensor_3,
       meanHumidity_1, meanHumidity_2, meanHumidity_3, meanHumidity,
       accumulator_H1, accumulator_H2, accumulator_H3, totalAccumulator,
       minHumid_1, maxHumid_1, minHumid_2, maxHumid_2, minHumid_3, maxHumid_3;
  int acceptableDifferencePercentage, counter_H1, counter_H2, counter_H3, index,
     maxNumberOfSamples, totalCounter;

  acceptableDifferencePercentage = 5;
  maxNumberOfSamples = 10;
```

```
index = 0;

accumulator_H1 = 0;
counter_H1 = 0;
accumulator_H2 = 0;
counter_H2 = 0;
accumulator_H3 = 0;
counter_H3 = 0;

while (index <= maxNumberOfSamples){
 // Get the temperature in Celsius (by default)
 humiditySensor_1 = dht_1.readHumidity();
 humiditySensor_2 = dht_2.readHumidity();
 humiditySensor_3 = dht_3.readHumidity();
 // Calculate the range to filter bad data from different sensors
 minHumid_1 = humiditySensor_1 * (1 - acceptableDifferencePercentage/100);
 maxHumid_1 = humiditySensor_1 * (1 + acceptableDifferencePercentage/100);
 minHumid_2 = humiditySensor_2 * (1 - acceptableDifferencePercentage/100);
 maxHumid_2 = humiditySensor_2 * (1 + acceptableDifferencePercentage/100);
 minHumid_3 = humiditySensor_3 * (1 - acceptableDifferencePercentage/100);
 maxHumid_3 = humiditySensor_3 * (1 + acceptableDifferencePercentage/100);

 if ((humiditySensor_1 > minHumid_2) | (humiditySensor_1 <= maxHumid_2) &
    (humiditySensor_1 > minHumid_3) | (humiditySensor_1 <= maxHumid_3)){
  accumulator_H1 = accumulator_H1 + humiditySensor_1;
  counter_H1 = counter_H1 + 1;
 }
 if ((humiditySensor_2 > minHumid_1) | (humiditySensor_2 <= maxHumid_1) &
    (humiditySensor_2 > minHumid_3) | (humiditySensor_2 <= maxHumid_3)){
  accumulator_H2 = accumulator_H2 + humiditySensor_2;
```

```
    counter_H2 = counter_H2 + 1;
  }
  if ((humiditySensor_3 > minHumid_1) | (humiditySensor_3 <= maxHumid_1) &
     (humiditySensor_3 > minHumid_2) | (humiditySensor_3 <= maxHumid_2)){
    accumulator_H3 = accumulator_H3 + humiditySensor_3;
    counter_H3 = counter_H3 + 1;
  }

  index = index + 1;
}

totalCounter = 0;
totalAccumulator = 0;

if (counter_H1 > 0) {
    meanHumidity_1 = accumulator_H1 / counter_H1;
    totalAccumulator = totalAccumulator + meanHumidity_1;
    totalCounter = totalCounter + 1;
}
if (counter_H2 > 0) {
    meanHumidity_2 = accumulator_H2 / counter_H2;
    totalAccumulator = totalAccumulator + meanHumidity_2;
    totalCounter = totalCounter + 1;
}
if (counter_H3 > 0) {
    meanHumidity_3 = accumulator_H3 / counter_H3;
    totalAccumulator = totalAccumulator + meanHumidity_3;
    totalCounter = totalCounter + 1;
}
```

```
  if (totalCounter > 0) {
    meanHumidity = totalAccumulator / totalCounter;
  }
  return meanHumidity;

}




void loop()
{
  while(!Serial.available()) {}
  // serial read section
  while (Serial.available())
  {
    if (Serial.available() >0)
    {
      readString = Serial.readStringUntil('\n');  //gets one byte from serial buffer
    }
  }

  if (readString.length() >0)
  {

    //Serial.print("Arduino received: ");
    //Serial.println(readString); //see what was received
    if (readString == "luminance"){

      responseValue = "";
```

```
    responseValue = "Luminance = " + String (getLuminanceValue()) + "\n";

    Serial.println(responseValue);
   }

   if (readString == "temperatureAndHumidity"){

    float temperature = getTemperature();
    float humidity = getHumidity();

    responseValue = "";
    responseValue = "temp = " + String (temperature) + "; humid = " + String (humidity) + "\n";
    Serial.println(responseValue);
   }


  }

  delay(500);

}
```