UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Máster

Diseño de un escáner low cost basado en sensor RGB.

Máster Universitario en Sensores para Aplicaciones Industriales

Autor: Jorge Lorente Benítez

Tutor: Rafael Masot Peris

entregado el: 22. 02. 2018

Declaración de trabajo propio

Declaro haber desarrollado el presente trabajo de forma autónoma y dando a conocer en el mismo de forma completa y exacta todos los medios de ayuda empleados para su elaboración, así como haber señalado y citado de forma adecuada todo aquello que proceda de trabajos de otras fuentes.

_____

Yátova, 13.02.2018, Jorge Lorente Benítez

Abstract

Nowadays, the analysis of chemical samples is of vital importance in a large number of industrial sectors. In this sense, it is necessary to have adequate equipment. Existing solutions are very expensive and not very versatile, given that the devices only accept test kits from its manufacturer.

Therefore, in the present master's thesis, we aim to develop a low-cost, versatile and user-friendly device that allows a quick and accurate analysis of different chemical samples in which a color change must be characterized.

For this purpose, the solution to the problem to be solved will be based on a low cost color sensor.

Once the prototype has been developed, its behavior must be characterized and its suitability for a real application evaluated. The results will reveal that the device manufactured is viable for the analysis of various chemical samples and that, after a series of improvements, it could pose a real alternative to current commercial systems.

Resumen

Hoy en día, el análisis de muestras químicas es de vital importancia en un gran número de sectores industriales. En este sentido, es preciso contar con equipos adecuados. Las soluciones existentes son muy caras y poco versátiles, dado que los dispositivos sólo admiten los kits de pruebas de su fabricante.

Por ello, en el presente trabajo fin de máster se pretende desarrollar un dispositivo de bajo coste, versátil y de fácil manejo que permita un análisis rápido y preciso de diversas muestras químicas en las que se deba caracterizar un cambio de color. Con tal fin, se basará la solución al problema a tratar en un sensor de color de bajo coste.
Una vez desarrollado el prototipo, se habrá de caracterizar su comportamiento y evaluar su adecuación para una aplicación real.

Los resultados revelarán que el dispositivo fabricado es viable para el análisis de muestras y que, tras una serie de mejoras, podría suponer una alternativa real a los sistemas comerciales actuales.

Resum

Avui dia, l'anàlisi de mostres químiques és de vital importància en un gran nombre de sectors industrials. En aquest sentit, cal comptar amb equips adequats. Les solucions existents són molt cares i poc versàtils, atès que el dispositiu només permet kits de proves del seu fabricant.

Per açò, en el present treball de final de màster es pretén desenvolupar un dispositiu de baix cost, versàtil i de fàcil maneig que permeta una anàlisi ràpida i precisa de diverses mostres químiques en les quals s'haja de caracteritzar un canvi de color.

Amb tal fi, es basarà la solució al problema a tractar en un sensor de color de baix cost. Una vegada fabricat el prototip, s'haurà de caracteritzar el seu comportament i avaluar la seua adequació per a una aplicació real.

Els resultats revelaran que el dispositiu fabricat és viable per a l'anàlisi de diverses mostres químiques i que, després d'una sèrie de millores, podria suposar una alternativa real als sistemes comercials actuals.

Índice general

Índice de tablas

Índice de ilustraciones

1 Objetivos

El objetivo principal del presente trabajo de final de máster es la producción y validación de un sistema electrónico, mecánico e informático para el análisis de muestras químicas utilizando tiras reactivas.

Se pretende fabricar un dispositivo de bajo coste, versátil y de sencilla utilización que cuente con prestaciones lo más parecidas posible a las que proporcionan las alternativas comerciales.

Para ello, habrá que cumplir los siguientes objetivos:

- La revisión de los dispositivos comerciales existentes para el análisis de muestras mediante la caracterización de la variación de color.
- La elección de un concepto general para el desarrollo de un prototipo basándose en lo visto en el punto anterior.
- La documentación para la obtención de los conocimientos físicos, electrónicos, químicos e informáticos necesarios para la elaboración del prototipo.
- La caracterización de los dispositivos electrónicos, tales como sensor y motor, para conseguir la máxima fiabilidad en las medidas.
- El diseño conceptual de un prototipo mecánico, su fabricación y la integración con los elementos electrónicos.
- El diseño de un software y de una interfaz gráfica de usuario para manejar el prototipo.
- La caracterización del sistema mediante la lectura de muestras preparadas.
- La aplicación del dispositivo en una aplicación real y la evaluación de los resultados mediante la comparación de los obtenidos con un sistema comercial.
- La proposición de futuras líneas de desarrollo y de mejora para el perfeccionamiento del dispositivo.

1.1 Motivación

En los últimos años se han comenzado a implementar un gran número de sensores para la monitorización de múltiples variables y en sustitución de sistemas más complejos. El éxito de estos dispositivos radica en su bajo coste y en la posibilidad de conseguir gran cantidad de información mediante el tratamiento adecuado de los datos obtenidos.

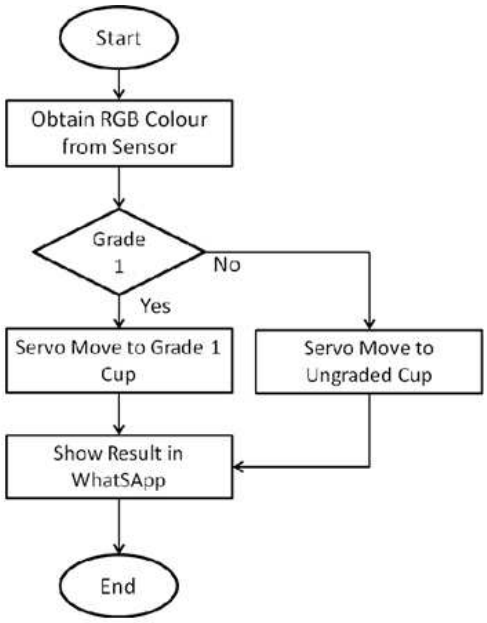
Ejemplos de ello son los teléfonos móviles, las pulseras para la monitorización de constantes vitales, los coches o la industria de cualquier tipo. Los sensores encuentran aplicación en prácticamente todos los ámbitos de la vida actual.

En este sentido, en el Departamento de Química de la Universidad Politécnica de Valencia (UPV) se desea desarrollar un dispositivo versátil y de bajo coste para el análisis de muestras químicas basándose en un sensor de color. El motivo es que el reflectómetro del que se dispone sólo permite la lectura de tiras reactivas de la empresa fabricante, lo que limita bastante la investigación de reacciones químicas para la caracterización de concentraciones a partir de un cambio de color.

Hasta ahora, las tiras reactivas de elaboración propia se leían con un escáner de imágenes. Seguidamente, se extraían los datos de cada zona reactiva mediante un software de edición de imágenes tipo Photoshop. Dichos datos se colocaban en una hoja Excel para correlacionarlos con la concentración de analito. Si se tiene en cuenta que cada tira podía contener más de 10 zonas reactivas, no es difícil apreciar que el análisis y la obtención de los resultados son lentos y costosos. Por ello, la motivación principal de este trabajo es el desarrollo de un dispositivo versátil y sencillo que agilice el proceso de análisis de tiras reactivas de cualquier procedencia.

Para ello se cuenta con el sensor de color TCS34725 de Adafruit, un dispositivo económico y de amplia aplicación en la investigación y en la industria.

Por ejemplo, en el artículo “Automated machine for sorting Sarawak pepper berries”, de A.H.Fauzi et al de 2015, se usa este sensor para la separación automática de granos de pimienta de alta calidad en sustitución de un proceso semiautomático lento y costoso y con un error de solamente el 20 % [1]. En la siguiente imagen se muestra la pimienta que se analizó, el diagrama de bloques del sistema desarrollado y una tabla con los resultados de dos sensores de color.



Trial/ Colour Sensor	1	2	3	4	5	6	7	8	9	10	Success (%)
TCS3200			✓		✓	✓		✓		✓	50
TCS34725	✓	✓	✓	✓		✓		✓	✓	✓	80

Ilustración 1: Granos de pimienta para analizar, diagrama de bloques lógico de la solución propuesta y tabla con los resultados para dos sensores de color según [1].

En otro artículo se integra el TCS34725 en una red de cámaras de bajas prestaciones para la monitorización del medio ambiente. A partir de la información limitada que se obtiene, mediante un modelo matemático se consigue información de alta calidad y definición [2].

Otra aplicación muy interesante de este sensor es el análisis de la calidad de productos alimenticios. En el artículo “Investigating the possibilities of document cameras for quality assessment of foodstuffs by measuring colors”, de Stanka Baycheva et al. (2016). En él, se comparan las lecturas de alimentos recogidas con una cámara web y un sensor TCS34725 y se demuestra que el índice de correlación de ambas medidas es superior a 0.9. Además, se confirma que los resultados obtenidos permiten determinar el estado de productos alimenticios [3].

Finalmente, en el artículo “Illumination adaptation with rapid-response color sensors”, de Xinchi Zhang et al, se emplea el TCS34725 como elemento clave en una alternativa de bajo coste para la medida y el ajuste inteligente de los niveles de iluminación en interiores.

Se hace hincapié en que, al contrario que en la mayoría de los sistemas comerciales actuales, el sensor utilizado no puede tomar imágenes, respetando así la privacidad del usuario [4].

En conclusión, se puede observar que el TCS34725 es un dispositivo de bajo coste y muy versátil, puesto que encuentra aplicación en diversos sectores. Por ello, en el presente trabajo se tratará de utilizar este sensor de color para el análisis de muestras químicas.

1.2 Estructura del trabajo

El presente trabajo se estructurará de la siguiente manera.

- Capítulo 1: Se presentarán los objetivos generales, la motivación del trabajo y su estructura.
- Capítulo 2: Aquí se explicará el estado actual de la técnica en el análisis óptico de muestras químicas y se elegirá una opción como punto de partida.
- Capítulo 3: Este capítulo estará dedicado a la presentación del marco teórico físico, electrónico, químico e informático necesario para una completa compresión de este trabajo.
- Capítulo 4: Aquí se caracterizarán el sensor y el motor necesarios para el prototipo. Además, se explicará el proceso que se siguió para diseñar y construir el prototipo y el software para su manejo.
- Capítulo 5: Estará dedicado a la caracterización del prototipo mediante el análisis de pruebas preparadas para ello.
- Capítulo 6: En este capítulo se evaluarán los resultados del prototipo en una aplicación real, concretamente, la determinación de la concentración de nitratos en agua mediante la lectura de tiras reactivas.
- Capítulo 7: Estará destinado a mostrar el coste detallado del prototipo.
- Capítulo 8: Se resumirán los resultados obtenidos y se explicarán posibles mejoras para futuros desarrollos del prototipo.

2 Estado actual de la técnica

En este capítulo se presentará brevemente el estado actual de la técnica para el análisis de muestras químicas en las que se da un cambio de color. En ese contexto se hablará de la colorimetría y la fotometría, ciencias en las que están basados los productos más comunes de este sector. En el primer caso se prestará especial atención a los tests de tiras reactivas y en el segundo a los equipos más comunes: el espectrofotómetro, el reflectómetro, el escáner y los sensores RGB. Finalmente, se concluirá con una breve descripción de la solución adoptada.

2.1 Colorimetría

La colorimetría es la ciencia que estudia la medida y la caracterización de los colores y su percepción [5]. En química es de vital importancia para la estimación de la concentración de un gran número de analitos. Con tal fin existen un gran número de tiras reactivas que permiten una rápida obtención de resultados [6].

En general, se define una tira reactiva como un trozo de papel u otro material que cuenta con una zona reactiva que contiene una sustancia o combinación de sustancias que, en presencia de un analito cambia de color [5, 7].

El cambio de color permite la detección cualitativa y, en ocasiones, cuantitativa del analito [7].

Para la estimación de la concentración se ha de contar con un patrón de colores de referencia [7, 8].

En el mercado se encuentra un gran número de tests que contienen tiras reactivas con un patrón de color para la evaluación de los resultados. Éste es el caso para algunas tiras para el análisis de orina, tiras para la determinación de glucosa y algunos tests de nitratos, entre muchos otros. El problema de este método es que la percepción del color es subjetiva, por lo que los resultados también lo serán. Si se desea que éstos sean objetivos, es conveniente emplear sistemas fotométricos [8]. El precio de las tiras reactivas depende del de los reactivos que contengan, pero suele encontrarse entre 1 € y 50 €, dependiendo de la aplicación objetivo [9].

2.2 Fotometría

En la fotometría se mide la intensidad de color de forma objetiva. Para ello, es de ayuda saber que el color se da cuando un objeto absorbe y absorbe determinadas longitudes de onda. Así, en la fotometría se hace uso del espectro visible para determinar la concentración en disoluciones coloreadas [8, 10].

Con tal fin, se basan los resultados en la señal recibida de fotoreceptores, es decir, dispositivos que producen una señal eléctrica cuando son expuestos a la luz tras haber incidido ésta en la muestra. Según la radiación electromagnética que la muestra ha reflejado, absorbido o transmitido y su longitud de onda es posible determinar la concentración del analito [6, 7, 10]. Por ello, los elementos básicos de estos equipos son fotorresistencias, fotodiodos, fotomultiplicadores o detectores CCD [10].

A continuación, se expondrán brevemente algunos de los equipos comerciales fotométricos más frecuentes: el espectrofotómetro, el reflectómetro, el escáner y el sensor RGB.

2.2.1 Espectrofotómetro

El espectrofotómetro (ilustración 2) es un instrumento que mide la cantidad de luz que absorbe una muestra y la usa para calcular la concentración de analito [10, 11].

Para ello, se proyecta un haz de luz monocromática sobre la muestra, es decir, de una única longitud de onda en función del analito a detectar [11].

Consta de las siguientes partes, una fuente de luz, un monocromador, una cubeta en la que se coloca la muestra y uno o varios fotodetectores para medir la cantidad de luz que ha logrado atravesarla [11]. Dependiendo de la calidad y la capacidad, el precio de este tipo de dispositivos se encuentra aproximadamente entre 1000 € y 12000 € [9].

Ilustración 2: Espectrofotómetro de [11].

2.2.2 Reflectómetro

Un reflectómetro (ilustración 3) es un instrumento de medición que proyecta un haz de luz sobre una muestra y mide la cantidad, intensidad y longitud de onda de la luz reflejada. A partir de éstos parámetros se determina la concentación de analito en la muestra [10, 12].

Consta de una fuente de iluminación (es frecuente el empleo de LEDs), un espacio para la deposición y sujeción de la muestra y un fotoreceptor para la valoración de los resultados [12]. Este tipo de dispositivos es frecuente para el análisis colorimétrico de tiras reactivas [8]. Sin embargo, presentan la desventaja de que suelen estar limitados a los tests de la empresa fabricante del equipo [8]. Dependiendo de su calidad y la capacidad, el precio de un reflectómetro se encuentra aproximadamente entre 1000 € y 5000 € [9].

Ilustración 3: Reflectómetro según [12].

2.2.3 Escáner de imágenes

Un escáner de imágenes (ilustración 4) es un instrumento que genera un documento digital a partir de uno físico [13]. Su uso es frecuente para copiar o duplicar imágenes, documentos o textos.

Ilustración 4: Escáner de imágenes de [13].

En química, es posible emplear estos dispositivos para el análisis de los resultados de reacciones en las que tenga lugar un cambio de color [8, 13]. Mediante un software para editar imágenes digitales es posible extraer los datos de la muestra y usarlos para la determinación de concentraciones.

Hoy en día, la mayoría de escáneres consisten en una caja dentro de la que se encuentran los elementos para el escaneo. En la parte superior hay un cristal sobre el cual se deposita el documento que se desee escanear. Hay una tapa para aislar los sensores del interior del dispositivo de la luz ambiente mientras se esté trabajando [13].

Destacan dos tipos de escáneres: los basados en un dispositivo CCD y los basados en un dispositivo CIS [14].

Un dispositivo CCD (ilustración 5) es un sensor formado por un array de células fotoeléctricas para el registro de imágenes [15]. En un escáner que lo contenga, la muestra es iluminada por una fuente de luz muy intensa. Mediante un sistema de espejos móviles, se envía la información a través de un sistema de lentes hasta el dispositivo CCD. El movimiento de los espejos permite cubrir toda el área del documento a escanear [14, 15].

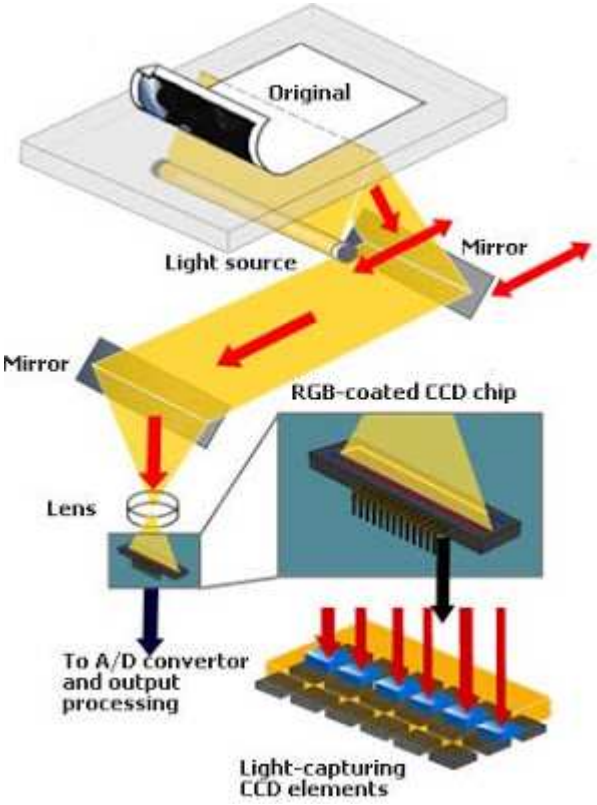


Ilustración 5: Escáner con dispositivo CCD de [14].

Un dispositivo CIS (ilustración 6) es un array lineal de fotodetectores. En un escáner que lo integre se encuentra cubierto por un sistema de lentes y flanqueado por LEDs rojos, verdes y azules para la iluminación [16].

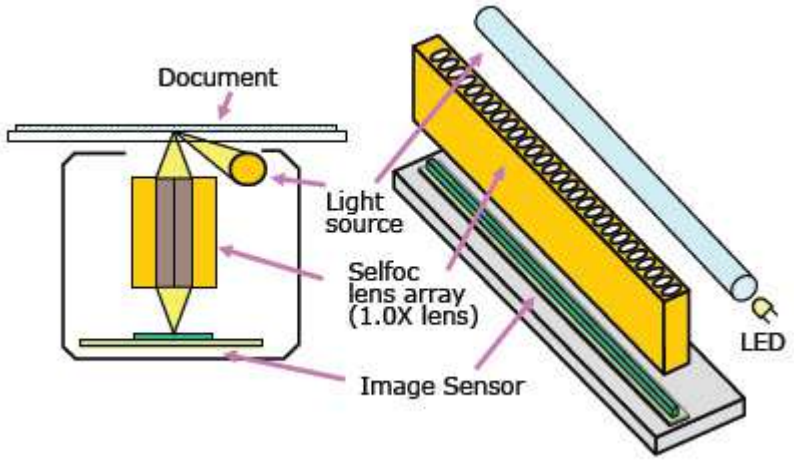


Ilustración 6: Principio de funcionamiento de un escáner basado en un dispositivo CIS según [16].

En función de las prestaciones que contenga, el precio de un escáner de imágenes puede ser de 40 € en algunos casos y en otro incluso de 500 € [9].

2.2.4 Sensor RGB

Un sensor RGB en un dispositivo pequeño y sencillo que proporciona información sobre las coordenadas RGB (del inglés red, green y blue) de la muestra analizada, es decir, de la intensidad del rojo, del verde y del azul. Encuentra aplicación en el control de la intensidad de la iluminación, la medida de la temperatura de color de la luz ambiente, el análisis de gases y fluidos y la verificación y análisis del control de productos. Por ello, se encuentra en productos como televisiones, móviles, tablets, ordenadores, impresoras. También es frecuente su uso en el sector médico y de automatización industrial [17].

Normalmente consiste en un array de fotodiodos con uno o varios LEDs para la iluminación de la muestra [17]. A continuación, se muestran varios modelos de sensores RGB.

Ilustración 7: Tres modelos distintos de sensores RGB y sus partes más importantes, adaptado de [18].

La medición se puede llevar a cabo por reflexión, es decir, se mide la luz reflejada por la muestra, o por absorbancia, es decir, se caracteriza la luz que ha logrado pasar a través de la muestra antes de llegar al array de fotodiodos (iustración 8).

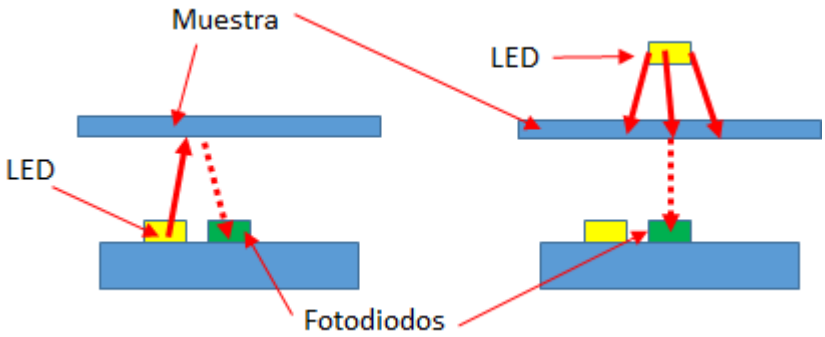


Ilustración 8: Medida de la luz reflejada (izquierda) y por transmisión (derecha).

El precio de este tipo de dispositivos se sitúa de 1 € a 20 € en función de la resolución o del fabricante [9, 18].

2.4 Solución adoptada

Tras tener en cuenta las opciones existentes en el mercado y teniendo en cuenta que se desea desarrollar un sistema de bajo coste, se elige el sensor RGB para el desarrollo del prototipo. Teniendo en cuenta que las muestras que se pretende analizar con el prototipo a desarrollar son principalmente tiras reactivas, habrá que diseñar un dispositivo que integre el sensor.

Aunque el procedimiento se explicará con detalle en el capítulo 4, se puede adelantar que se tomará como referencia la estructura externa del escáner de imágenes. Así, se diseñará una caja con un cristal en la cara superior para la colocación de las muestras. El producto final será una especie de escáner unidimensional para la lectura lineal de muestras. Aparte del sensor RGB TCS34725, se requerirá un motor y su driver para el “escáner”, finales de carrera para definir el inicio y el final del recorrido, una placa Arduino para controlar las componentes electrónicas y un ordenador con un software que permita guardar y visualizar los datos de las lecturas.

En el diagrama de bloques lógico que se encuentra a continuación se pueden observar las componentes anteriormente mencionadas con el tipo de comunicación que requieren.

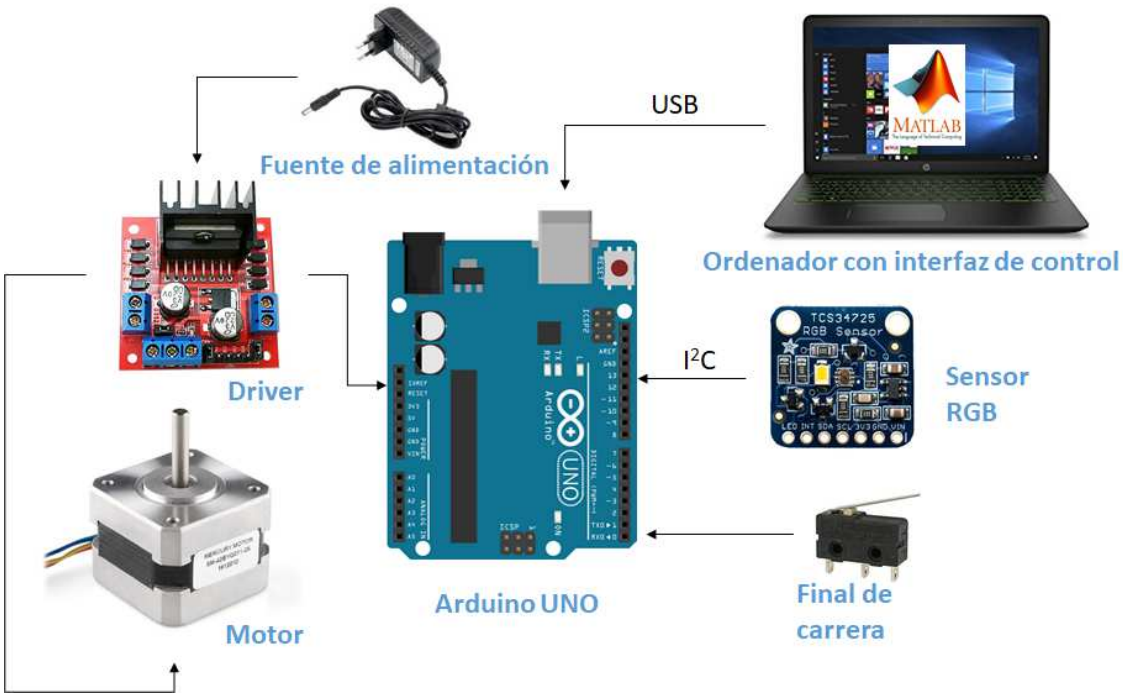


Ilustración 9: Diagrama de bloques lógico del sistema desarrollado.

3 Marco teórico

Después de tratar brevemente el estado actual de la técnica en el mundo de los escáneres y los sensores de color cabe dedicar un capítulo a los conceptos básicos relevantes para este trabajo.

En primer lugar, se explicaran los conceptos básicos relacionados con la óptica, tales como el color, su formación y distintos espacios de representación. A continuación, se presentarán los aspectos de la química relevantes para este trabajo y sus posibles aplicaciones en este campo. Seguidamente, se procederá a explicar el funcionamiento de los sistemas electrónicos empleados para el desarrollo del prototipo. Finalmente, se explicará brevemente el software empleado en este trabajo.

3.1 Marco teórico físico

En la obtención de imágenes a partir de un objeto físico es de vital importancia el significado del color, su formación, su relación con la percepción humana y los modelos matemáticos que lo describen [19]. Tales conceptos se desarrollan en este apartado.

3.1.1 Definición de color

El color es una característica de la percepción visual derivada de la estimulación de los conos del ojo del ser humano por radiación electromagnética dentro del espectro visible. Antes de pasar al siguiente apartado, es de ayuda profundizar en los conceptos propuestos en esta definición [19].

En primer lugar, los conos son unas células fotosensibles situadas en la retina del ojo. La excitación de los mismos es transmitida por medio de impulsos nerviosos a través del nervio óptico hasta el cerebro, que interpreta dichos impulsos dando lugar a la formación de una imagen. En la ilustración 10 se puede observar la anatomía del ojo humano y la ubicación de los conos dentro del mismo. Cabe añadir que existen tres tipos de conos: los que detectan la luz amarilla (o de longitud de onda larga, llamados conos L), los que detectan la luz verde (o de longitud de onda media, los conos M) y los que detectan la luz

violeta (o de longitud de onda corta, los conos S). En la misma zona se encuentran también los bastones, células fotosensibles que permiten la visión en blanco y negro en entornos de baja iluminación [19] que en la siguiente imagen están representados en un color azul oscuro. En la misma imagen, los conos L se muestran de color rojo, los conos M de color verde y los conos S de color azul claro.

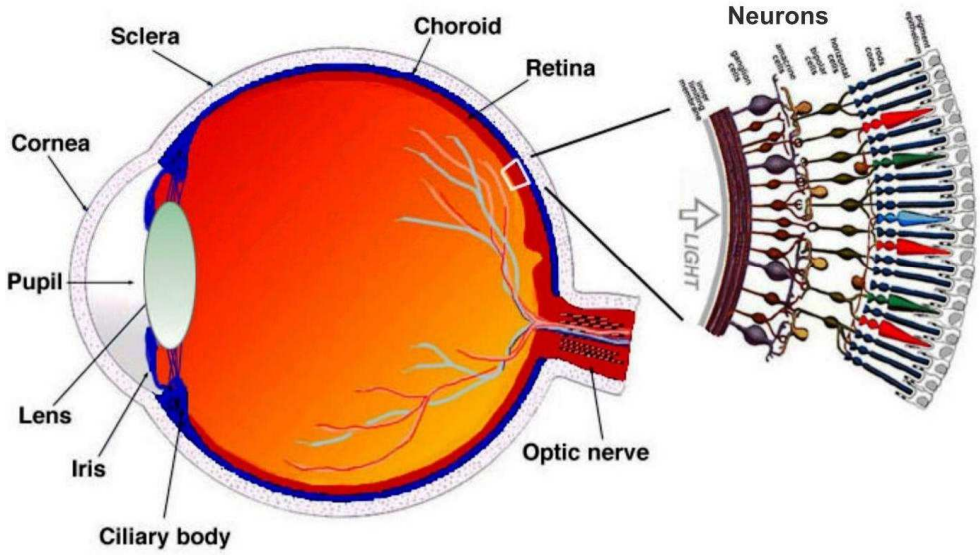


Ilustración 10: Anatomía del ojo humano con detalle de la retina, adaptada de [19].

En la siguiente imagen se puede observar la intensidad de la respuesta de los conos del ser humano a distintas longitudes de onda [20]. Nótese que la respuesta de los conos M y L es muy parecida.

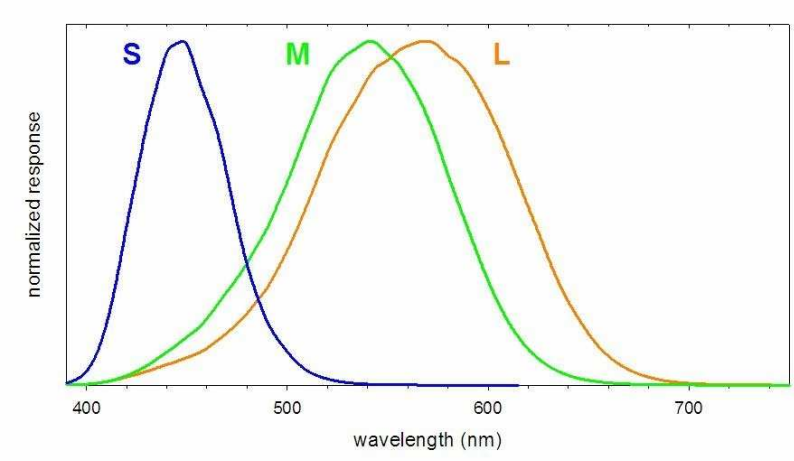


Ilustración 11: Intensidad de la respuesta de los conos S, M y L en función de la longitud de onda según [20].

En segundo lugar, la radiación electromagnética se refiere a las ondas (o a sus cuantos, los fotones) de un campo electromagnético que se propaga por el espacio transportando energía. Ejemplos de ondas de energía electromagnética son las ondas de radio, la luz visible, la luz ultravioleta o los rayos X [21]. La ilustración 12 muestra la representación de una onda electromagnética a partir de su campo eléctrico, su campo magnético y su dirección. Las tres magnitudes son perpendiculares entre sí.

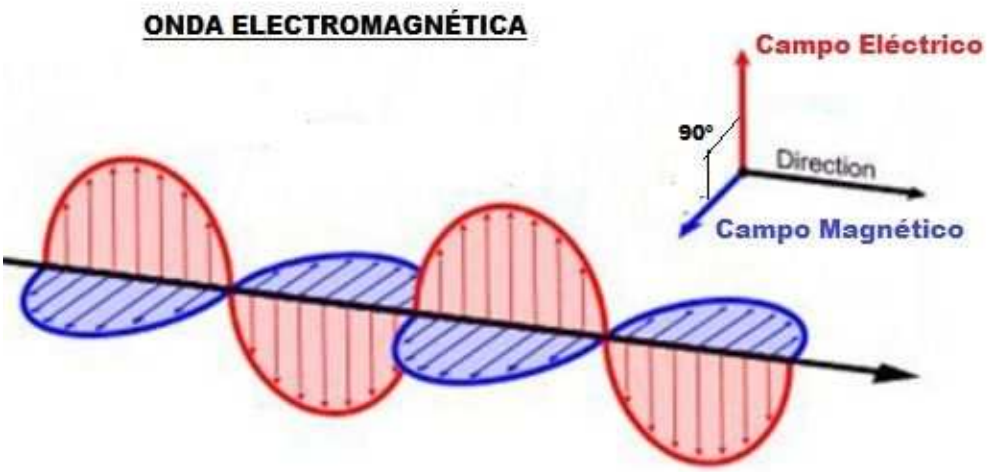


Ilustración 12: Onda electromagnética adaptada de [21].

La radiación electromagnética se caracteriza por su longitud de onda λ [nm] o su frecuencia f [Hz] y su intensidad I [cd]. Si la longitud de onda de una fuente luminosa se encuentra entre 390 nm y 700 nm puede ser percibida por el ojo humano como color y forma parte del espectro visible. Dentro de este espectro, la longitud de onda de 390 nm corresponde al color violeta y la longitud de onda de 700 nm corresponde al color rojo [21]. En la ilustración 13 se muestran los colores del espectro visible, que comportan una ínfima parte de la radiación electromagnética que existe.

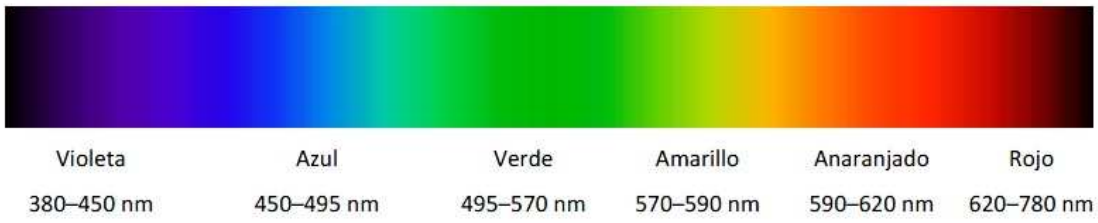


Ilustración 13: Colores del espectro visible con sus correspondientes longitudes de onda según [22].

Las fuentes de luz emiten a distintas longitudes de onda relacionadas con la naturaleza física de la fuente. Se denomina espectro de emisión a la radiación emitida por una sustancia. Este espectro es característico de cada sustancia, puesto que se da cuando electrones excitados vuelven a su estado original, liberando el excedente de energía en forma de radiación electromagnética en una o varias longitudes de onda propias de cada material [22, 23]. Por ejemplo, al ser excitado, el sodio emite en la longitud de onda correspondiente al amarillo (ilustración 14).

Ilustración 14: Espectro de emisión del sodio según [23].

Para llegar al estado de excitación, una sustancia debe absorber energía. El espectro de absorción muestra las longitudes de onda requeridas por un determinado material para llevar al estado de excitación a los electrones correspondientes. Por ejemplo, el espectro de la fuente de radiación que se emplee para excitar sodio mostrará huecos en las longitudes de onda correspodientes al amarillo [22, 23].

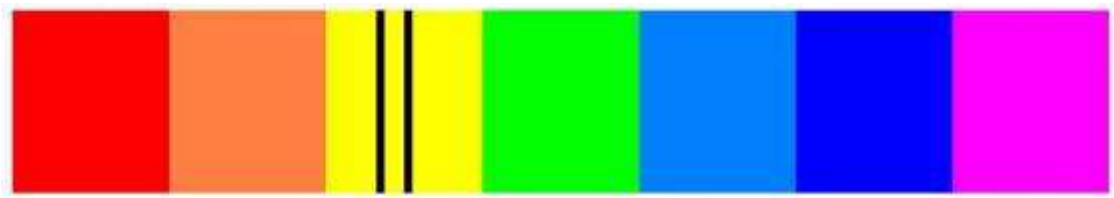


Ilustración 15: Espectro de absorción del sodio según [23].

Así, los distintos tipos de color se asocian con las distintas longitudes de onda reflejadas y absorbidas por un objeto dentro del espectro visible. Dicha reflexión está relacionada con las propiedades físicas del objeto, que se reflejan en su espectro de emisión y de absorción y, por lo tanto, en el color con el que será percibido por el ser humano [21, 22, 23].

Por lo tanto, un objeto será percibido como blanco si refleja todas las longitudes de onda del espectro visible. Por contra, un objeto será percibido como negro si absorbe todas las longitudes de onda del espectro visible [22]. En la ilustración 16 se puede observar cómo se da el color de distintos materiales al reflejar éstos una longitud de onda determinada.

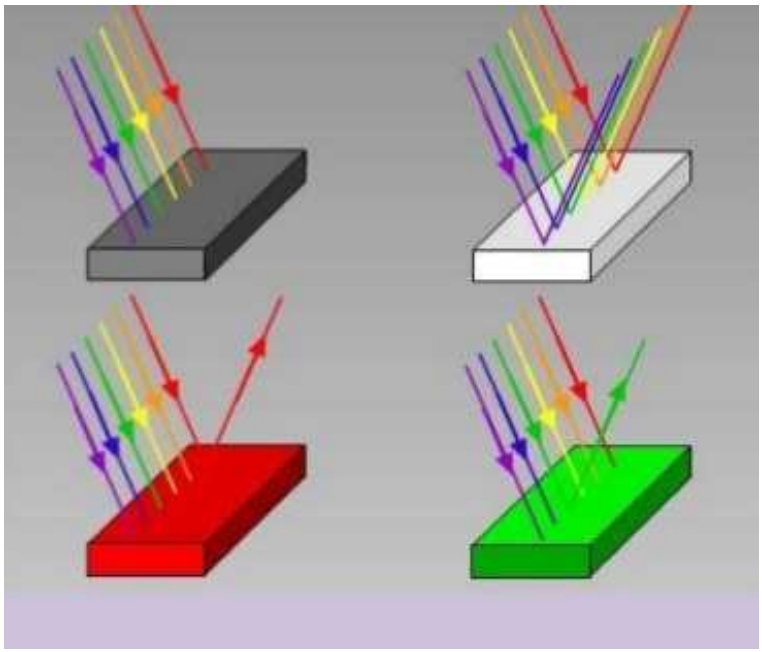


Ilustración 16: Esquema del origen de la percepción de los colores a partir de luz blanca, adaptada de [23].

3.1.2 Espacios de color

Un concepto de vital importancia para el presente trabajo es el de espacio o modelo de color. En un modelo de color los colores se representan como combinaciones de números. Según el modelo cada color está descrito por tres o cuatro valores. El objetivo principal de los espacios de color es normalizar la especificación de colores en sus aplicaciones. Existe un gran número de espacios de color, de los cuales se procederá a continuación a explicar algunos, centrando la atención en los más relevantes para el presente trabajo [24]. El siguiente diagrama muestra un resumen de los espacios de color, resaltando los que se explicarán con más detalle.

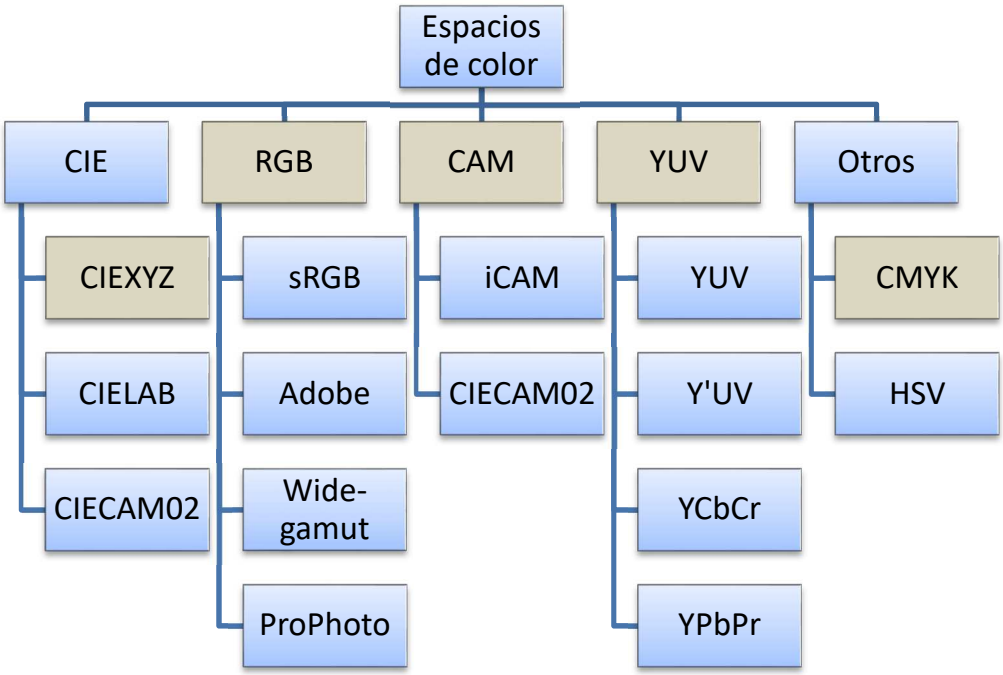


Ilustración 17: Diagrama de algunos de los principales espacios de color según [24].

CIEXYZ

La CIE (Comission Internationale d'Éclairage) desarrolló en 1931 un sistema para establecer una relación entre la sensación humana de color y las causas físicas del mismo. Para su elaboración, un determinado número de personas debía lograr imitar un color propuesto mediante la regulación de la luminosidad de tres focos de luz de diferentes colores: uno rojo, otro verde y el último azul. Se partía de la teoría de que con tres fuentes de luz adecuadas se podía reproducir cualquier impresión lumínica [25].

Con el promedio de los resultados obtenidos por múltiples personas se obtuvo un espacio tridimensional que contiene todos los colores perceptibles por un "observador estándar" [25].

Así, todos los colores de este espacio se pueden expresar por medio de un triestímulo, que engloba los valores X, Y y Z. Estos parámetros están relacionados con las respuestas de los conos S, L y M del ser humano ante estímulos de color. Así, Z es muy similar a la respuesta de los conos S (tono azul-violeta), mientras que el valor de X está relacionado con la respuesta de los conos L y M (correspondientes al color amarillo y verde, respectivamente). El parámetro Y expresa la luminosidad [25].

Con la finalidad de expresar este espacio de forma más compresible, se desarrolló una figura bidimensional a partir de este modelo en la que la coordinada z se puede calcular a partir de las otras dos mediante la ecuación x+y+z=1 [25, 26].

Dicha figura (ilustración 18) tiene forma de herradura dentro de la que se encuentran todos los colores visibles por el ser humano. En los bordes de la figura se encuentran los llamados colores puros o espectrales, que se dan a partir de una fuente de iluminación de una única longitud de onda Así, la gráfica se encuentra delimitada por la línea espectral y la línea del púrpura. En el interior de la figura se encuentran los colores resultantes de una variación de la luminosidad [25, 26].

El punto de origen es el correspondiente al color blanco (punto W en la imagen), cuyas coordinadas son x=y=z=1/3. Dependiendo de la iluminación, el punto de origen puede cambiar de ubicación dentro de la figura [26].

Dentro de la gráfica es también de importancia la curva del cuerpo negro, al lo largo de la cual se representan los colores según de la temperatura de un foco ideal en grados Kelvin. Un cuerpo negro es un objeto ideal (y, por lo tanto, teórico) que absorbe toda la radiación que incide sobre él [21].

En la ilustración 18 se muestra la tabla de color normalizada elaborada a partir del espacio de color de la CIE. En dicha imagen, se puede observar la línea espectral con sus correspondientes longitudes de onda y la curva de temperatura de color de un cuerpo negro.

A pesar de que fue desarrollado en 1931, por lo que presenta fallos de precisión debido al método de obtención de los datos y la menor fiabilidad de los equipos de medida en comparación con los actuales, este modelo sigue siendo utilizado hoy en día, si bien se le suelen aplicar algunas modificaciones matemáticas [25, 26].

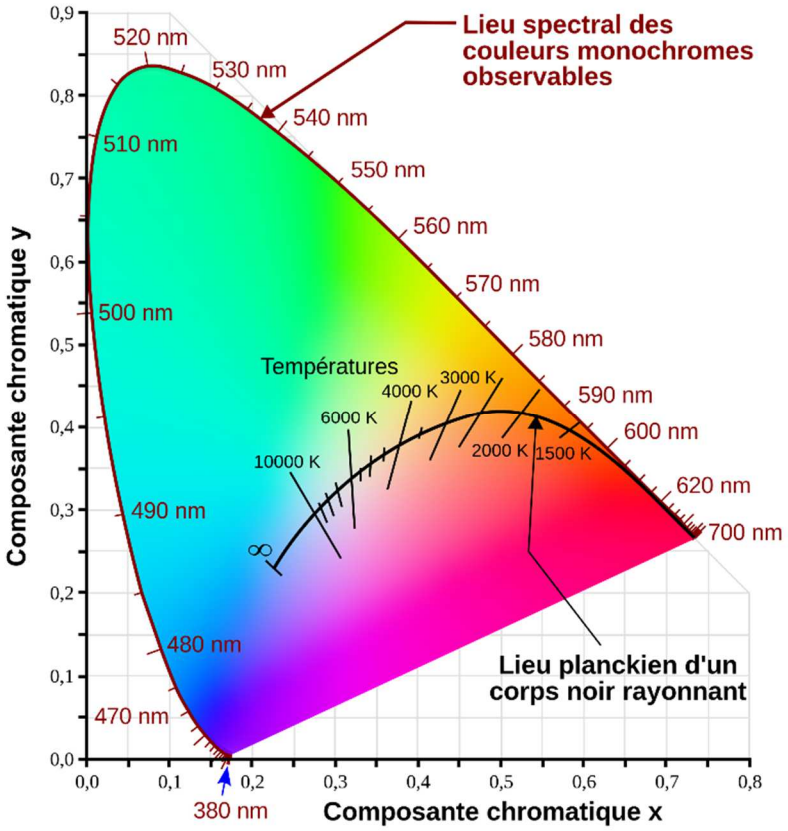


Ilustración 18: Tabla de color normalizada de la CIE, adaptada de [27].

CMYK

CMYK (del inglés cyan, magenta, yellow y black o key) es un modelo de color sustractivo que consta de cuatro de colores (ilustración 19). Está relacionado con el proceso de impresión, es decir, la deposición de tinta sobre un sustrato.

Se define este espacio de color como sustractivo porque trabaja añadiendo colores a un fondo blanco o, dicho de otra forma, restando brillo al fondo [28].

Así, el blanco es el color natural del papel, mientras que el resto de colores se consiguen a partir de una combinación de cian, magenta y amarillo. En teoría, el color negro debería obtenerse a partir de la combinación de estos tres colores, pero en la práctica se usa tinta negra para este fin.

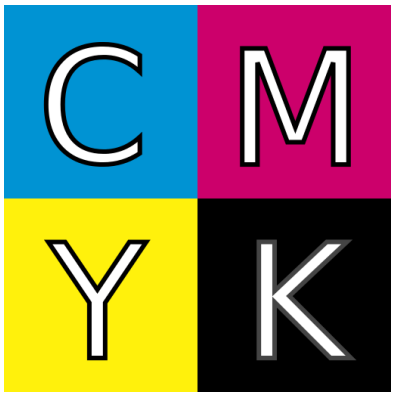


Ilustración 19: Colores básicos del sistema CMYK según [28].

Esto es debido a que el color obtenido a partir de la combinación de cian, amarillo y magenta no es exactamente negro, sino más bien una especie de gris oscuro, como se muestra en la ilustración 20 [28].

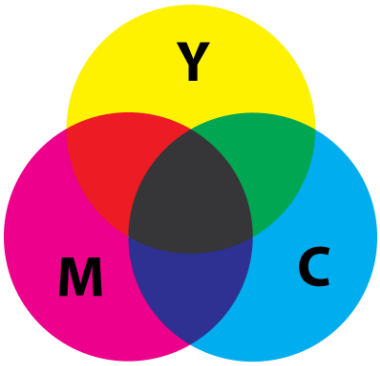


Ilustración 20: Color resultante de la adición de los tres “colores primarios“ del espacio CMYK según [28].

Además, el gasto de tinta que supondría imprimir en negro a partir de la combinación de varios colores sería muy grande. Otro punto a tener en cuenta es que el papel sobre el que se suele imprimir se mojaría en exceso a causa de una concentración de tinta demasiado elevada. Por ello, se introdujo el negro como cuarto color del modelo CMYK [28].
Este espacio de color tiene gran aplicación en el mundo de la imprenta. Presenta la desventaja de que la gama de color representable a través de sus colores básicos no es muy grande [28]. En la ilustración 21 se pueden observar los colores que engloba el sistema CMYK dentro de los colores visibles y en comparación con el sistema RGB, que se explicará en el siguiente apartado.

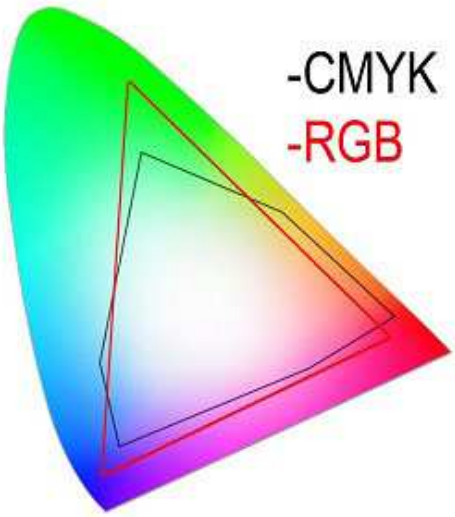


Ilustración 21: CMYK y RGB dentro de la tabla de colores normalizada de la CIE [27].

Con este sistema aparecen problemas con los colores claros y muy saturados, en los que el patrón del toner se hace visible. Esto se puede solucionar añadiendo tinta de más colores. Es muy frecuente el llamado sistema CcMmYyK, que añade cartuchos de tinta de color cian claro, magenta claro y amarillo claro a los habituales del sistema CMYK [28].

Modelo RGB

Se trata de un modelo de color aditivo en el que luz roja, verde y azul (del inglés red, green y blue) se combinan de diferentes formas para producir una amplia gama de colores [28, 29]. A continuación se muestran los tres colores básicos del espacio RGB.
La elección de estos tres colores como base del modelo se debe principalmente a la fisiología del ojo humano. Como se ha visto en el apartado 3.1.1, el ojo consta de tres tipos de células fotosensibles que reaccionan a determinadas longitudes de onda, correspondientes al amarillo (λ = 570 nm), el verde (λ = 540 nm) y el violeta (λ = 440 nm) [20, 29].
La diferencia en la intensidad de la señal recibida por cada uno de los tipos de células permite al cerebro identificar el color de lo que se está observando [23, 29].
Dado que el amarillo contiene un porcentaje elevado de verde y el violeta se encuentra en el umbral visible, para el sistema RGB se toman el rojo, el verde y el azul como colores básicos.

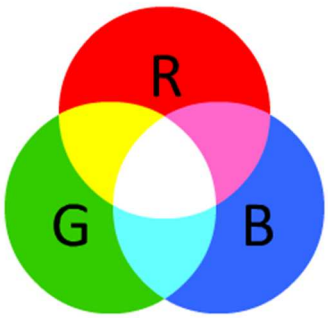


Ilustración 22: Colores básicos del sistema RGB según [29].

Dentro de los colores que percibe el ser humano, el espacio RGB se puede representar como un triángulo cuyos vértices se corresponden a las longitudes de onda de sus colores básicos. Los colores contenidos en el modelo se encuentran en el interior de dicho triángulo [29]. sistema Con este sistema no se pueden representar todos los colores que ve el ser humano (ilustración 23).

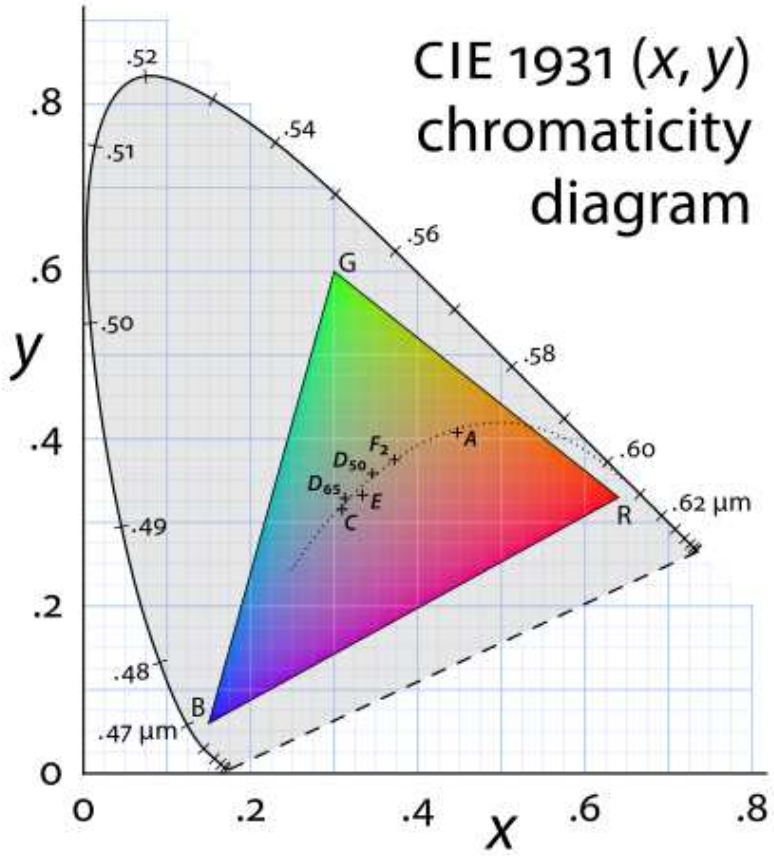


Ilustración 23: Espacio RGB dentro del diagrama de cromaticidad de la CIE según [27] y [29].

La elección de los vértices del triángulo determina la gama de colores que abarcará. Según su localización, existen distintos sistemas RGB, como por ejemplo, el sistema sRGB, que contiene el 69.4 % del espectro visible, el Adobe RGB, que contiene el 86.2 %, el Adobe Wide Gamut, que contiene el 99.1%, el ProPhoto, que contiene el 100% del espectro visible (ilustración 24), y muchos otros. La elección del sistema vendrá determinada por la aplicación objetivo. Así, la mayoría de pantallas electrónicas de color trabajan con el modelo sRGB, mientras que para aplicaciones de índole fotográfica y artística es de interés el espacio Adobe RGB [28, 29].

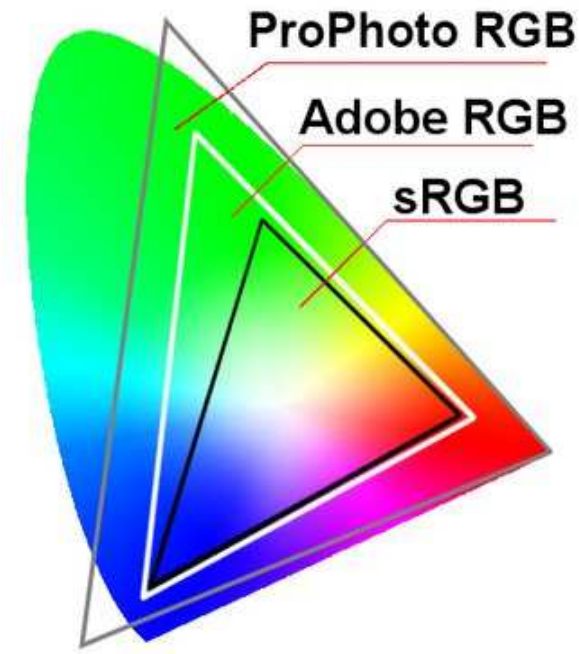


Ilustración 24: Comparación de la gama de color del sistema sRGB, Adobe RGB y ProPhoto RGB adaptado de [29].

Dentro de cualquier sistema RGB, los colores se definen por su contenido de rojo, verde y azul. Si el contenido de los tres colores básicos es nulo, el resultado es el negro, mientras que si es máximo se obtiene el blanco. Dado que el blanco se obtiene de la suma del rojo, el verde y el azul, los sistemas RGB se denominan aditivos [29].

Existen varias formas de representar la componente roja, verde y azul de los colores del sistema RGB de forma numérica [29]:

- En escala de 0 a 1, 1 es el valor máximo y 0 el mínimo. Los valores intermedios se dan en decimales.

- En porcentaje.
- En números de 8 bits (0-255), el cero es el valor mínimo y el 255 el máximo. Es frecuente en pantallas electrónicas y se puede codificar en hexadecimal.
- En números de 16 bits (0-65535), el cero es el valor mínimo. Se aplica en equipos digitales de alta definición.

Dado que todos los colores se expresan como la combinación de la componente roja, verde y azul dentro de los espacios RGB, si se asume que el ordenamiento de las componentes entre los valores máximos es lineal, se puede representar cualquier sistema RGB como un espacio de coordenadas cartesiano en el que la cantidad de rojo equivale a la coordenada “x”, la cantidad de verde a la coordinada “y” y la cantidad de azul a la coordinada “z”. Así, se forma un cubo de colores en cuyo punto de origen se encuentra el color negro y en cuyo punto (1, 1, 1) se encuentra el color blanco [29].

Cualquier color dentro de este espacio se representa por medio de las coordenadas tridimensionales (r, g, b) (ilustración 25) [29].

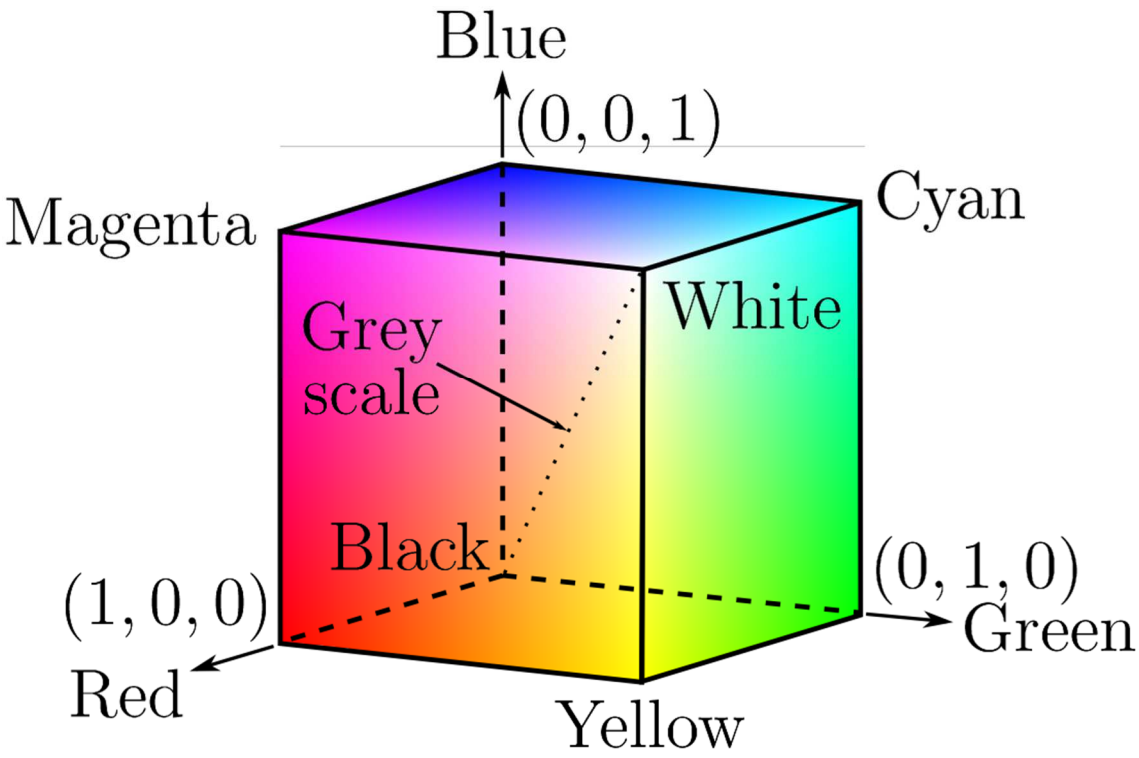


Ilustración 25: Cubo RGB según [29].

Una ventaja de este sistema de coordenadas es que las diferencias entre colores se convierten en distancias, lo que supone una ventaja en el procesamiento informático y matemático de colores [28, 29].

Para concluir, la aplicación de los espacios de color RGB se halla sobre todo en el sensado y la representación de imágenes en equipos electrónicos tales como televisiones, teléfonos móviles, escáners y cámaras de fotos digitales, entre otros [27].

Resulta interesante el hecho de que, aunque se trabaje con el mismo espacio RGB (por ejemplo, el sRGB), los colores variarán de un equipo a otro debido a las variaciones en la manufactura de las fuentes lumínicas de cada uno [27].

CAM (Modelo de apariencia de color)

El CAM (del inglés Color Appearance Model) es un modelo matemático que busca reflejar los aspectos de la visión humana del color, especialmente en las condiciones en las que la apariencia de un color para el humano no concuerda con la medida física del mismo [30].

La percepción del color es un fenómeno subjetivo a pesar de que la distribución del espectro visible es universal. El primer intento de normalizar dicha percepción fue llevado a cabo por la CIE en 1931 [27]. Sin embargo, el modelo resultante (CIE XYZ) requiere condiciones específicas de iluminación y de ubicación de la muestra y el observador para su correcto empleo. Por ello, si hay un cambio notable de las condiciones ambientales, resulta insuficiente [27]. En ese caso, un modelo de percepción de color resulta más adecuado [30].

A diferencia de los modelos anteriormente explicados, el sistema CAM no se basa en un sistema tricromático, sino que describe el color a través de los siguientes parámetros: el matiz, la luminosidad, el brillo, la intensidad, el colorido y la saturación [30]. Debido a que la explicación del CAM no tiene aplicación práctica en este trabajo, sino que busca ofrecer una vista de conjunto de los espacios de color existentes, se procede a continuación únicamente a una breve exposición del significado de cada parámetro.

El matiz se obtiene a partir de la similitud de la percepción del color con los matices denominados rojo, verde, azul y amarillo (ilustración 26) [30].

Ilustración 26: Escala de matices, extraído de [30].

La luminosidad representa la variación en la percepción de un color o un espacio de color debida a un cambio en la intensidad de la iluminación incidente (ilustración 27) [30]. Un color claro muestra una luminosidad elevada, mientras que la luminosidad de un color oscuro es baja.

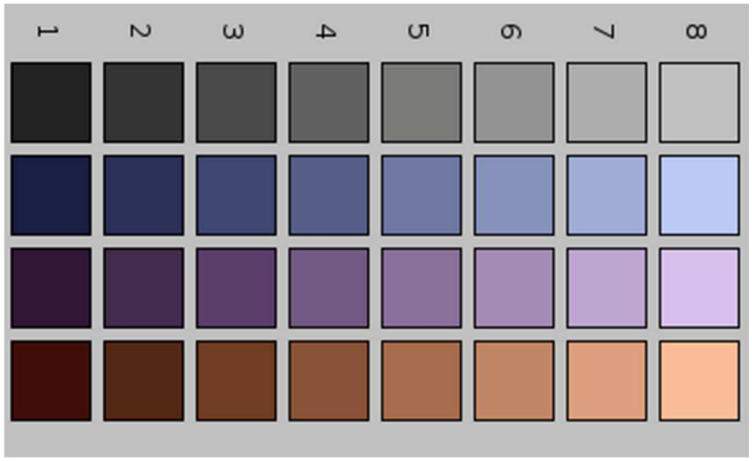


Ilustración 27: Distintos matices con luminosidad creciente (de izquierda a derecha) según [30].

El brillo es un parámetro que cuantifica la cantidad de luz que parece irradiar o reflejar un objeto según su percepción visual [30]. En la ilustración 28 se muestra un tono con brillo creciente de izquierda a derecha.

Ilustración 28: Tono oscuro con brillo creciente (de izquierda a derecha) a partir de [30].

La saturación, el colorido y la pureza son parámetros que reflejan la intensidad de un matiz específico [30]. Por ejemplo, un color con un valor de saturación elevado muestra un color vivo, mientras que un color poco saturado tiende a aparecer grisáceo y apagado. En la ilustración 29 se muestra una gradación de la saturación con el valor máximo a la izquierda y el mínimo a la derecha.

Ilustración 29: Tono con máxima saturación a la izquierda y mínima saturación a la derecha según [30].

Existen varios modelos basados en el CAM, como por ejemplo el RLAB, el LLAB o el Hunt Model. Encuentran aplicación en el procesado y la producción de imágenes de alta calidad y resolución [30].

YUV

El sistema de color YUV fue el resultado del intento de desarrollar una televisión de color a partir de la televisión el blanco y negro [31]. Así, está pensado como un sistema de codificación de color para imágenes y vídeos. Tiene en cuenta la percepción humana del color y permite enmascarar algunos errores de transmisión y compresión de datos [32].

Consta de tres parámetros principales: uno de luminancia y dos de crominancia [31, 32].

La luminancia representa el brillo de una imagen y viene a ser su representación en blanco y negro [31]. Se puede obtener de la suma ponderada de la corrección gamma de las coordenadas RGB de una imagen. La corrección gamma es una operación matemática no lineal que se emplea para codificar y decodificar valores tricromáticos o de luminancia en vídeos o imágenes [32].

La crominancia es una señal empleada en sistemas de video para transmitir la información de color separada de la señal de luminancia [31]. La señal de crominancia consta de las señales U (luminancia azul) y V (luminancia roja) en los sistemas de video compuestos.

La amplitud y la fase de dicha señal se corresponden aproximadamente con el matiz y la saturación del color. Las señales U y V se pueden entender como señales de diferencia de color [31, 32]. En la ilustración 30 se muestra una imagen y su descomposición en Y', U y V.

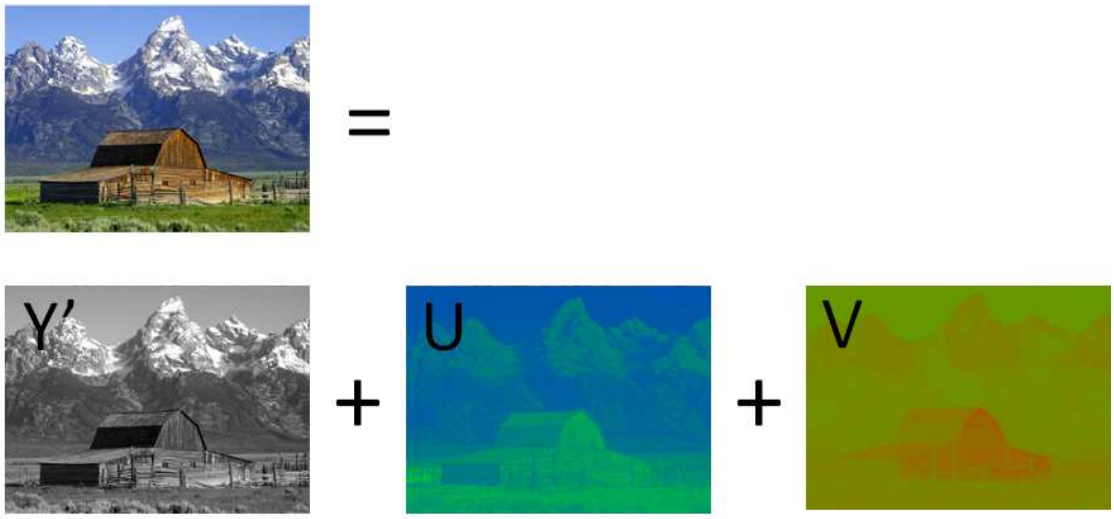


Ilustración 30: Descomposición de una imagen en Y', U y V según [31].

Cuanto mayores son los valores de U y de V mayor es la saturación del píxel en la pantalla [32]. Un píxel es la menor unidad homogénea en color que forma parte de una imagen digital [31].

Para valores de U y V pequeños, no se requiere apenas señal y la saturación del píxel es mucho menor. Si estos parámetros llegan a 0, se obtiene un tono gris en la pantalla [32].

Como se ha mencionado anteriormente, el sistema YUV encuentra aplicación en la transmisión de imágenes y video y, por lo tanto, en la televisión analógica, en la digital y en equipo fotográfico especializado. Una de las principales ventajas de este sistema es que permite el submuestreo de crominancia, es decir, que se transmita la crominancia a una menor frecuencia que la luminancia, aprovechando que la percepción humana del color es inferior a la del brillo [31, 32].

Algunos modelos basados en este sistema son Y'UV, YUV, YCbCr y YPbPr [31].

3.2 Marco teórico químico

Dado que el producto resultante del presente trabajo está enfocado a una aplicación en el campo de la química, resulta de interés hacer hincapié en algunas nociones básicas relacionadas con la misma. En especial, interesa centrarse en las reacciones químicas relacionadas con un cambio de color.

3.2.1 Reacciones químicas

Una reacción química es un proceso termodinámico en el cual una o más sustancias, denominadas reactivos, cambian su estructura molecular y sus enlaces atómicos formando nuevas sustancias, llamadas productos. Las características de los productos dependen de la naturaleza de los reactivos así como de las condiciones bajo las que se da la reacción química y suelen ser distintas entre sí [8].

Existen muchos tipos de reacciones químicas, pero para el presente trabajo resultan de interés las relacionadas con el cambio de color de los reactivos o la formación de productos coloreados. Por ello, el presente apartado se centrará solamente en ese tipo de reacciones.

Un ejemplo de este tipo de reacciones químicas bastante conocido se puede llevar a cabo con una tira de indicador universal. Si se sumerge en un medio ácido, la tira mostrará un color entre rojo y amarillo, si se sumerge en un medio neutro se coloreará de verde y si se introduce en un medio básico mostrará un color entre violeta y azul. Es decir, dependiendo del medio con el que interactúa, el cambio en las propiedades ópticas de la tira es diferente [8].

Ilustración 31: Tiras de indicador universal (izquierda) y colores que muestran según la característica del medio según [8].

3.2.1 Indicadores

En general, los indicadores son medios que muestran determinadas informaciones a través de un cambio en sus propiedades, frecuentemente de las ópticas [8]. Así, permiten el seguimiento de procesos o el control de estados.

En química, los indicadores son sustancias, equipos o medios que permiten la monitorización de reacciones químicas o la caracterización de sistemas químicos a través de un cambio en sus propiedades. En este campo es también muy frecuente que el cambio afecte a las propiedades ópticas del indicador, por ejemplo, a través de un cambio de color [8].

Si se trata de una sustancia, es frecuente que un indicador pueda existir en dos o más formas tautómeras, teniendo estructuras distintas y colores diferentes en cada una de ellas.

Según el tipo de procesos físicos o químicos en los que se use un indicador y sus características intrínsecas se diferencia entre los siguientes tipos de indicadores [33]:

- Indicadores de pH: Se trata de de una sustancia que cambia de color dentro de un pequeño intervalo de la concentración de protones y, en general, al pasar de una disolución ácida a alcalina. El equilibrio de ionización de un indicador ácido se podría expresar de esta forma [33].

$$\underbrace{IndH}_{un\ color} + H_2O \leftrightarrows H_3O^+ + \underbrace{Ind^-}_{otro\ color}$$

  Un ejemplo de este tipo de indicadores es el naranja de metilo, un colorante azoderivado que cambia de rojo a amarillo entre un pH de 3.4 y 4.4 [33].

- Indicadores redox: Son sustancias que experimentan un cambio de color según su estado de oxidación (oxidado o reducido). Si en el cambio de estado electroquímico de un indicador redox participa un protón, la variación del color dependerá también del pH. Un ejemplo de indicador redox es el azul de metileno [33].

- Indicadores complejométricos: Los indicadores complejométricos son colorantes iocrómicos, es decir, alteran su color en presencia de determinados iones. La variación en el tono tiene lugar a causa de la formación de complejos débiles entre el indicador y un tipo de ion específico. Éstos tienen unas propiedades ópticas distintas a las del indicador aislado. Un ejemplo sería la hematoxilina, empleada para la detección de iones de cobre [33].

- Indicadores térmicos: En este caso se obtiene un cambio de color según la temperatura. Hay dos formas de obtener este tipo de indicadores, una mediante el cristal líquido y otra mediante colorantes. En el segundo caso, la absorción de radiación electromagnética de una determinada longitud de onda da como resultado un cambio en las propiedades ópticas del indicador. Esta alteración viene asociada con reacciones pericíclicas, isomerizaciones cis-trans, transferencias de electrones, procesos de disociación y otros efectos debidos la absorción de radiación [33].

En el presente trabajo se emplearán las tiras de nitratos, que permiten averiguar la cantidad de este tipo de iones que se encuentra en una solución a partir de un cambio de color que se puede evaluar ópticamente o con la ayuda de un dispositivo adecuado.

3.3 Marco teórico electrónico

Una vez presentadas las bases teóricas físicas y químicas resulta interesante dedicar un apartado a los equipos electrónicos que se utilizaron en este trabajo. Así, a continuación se describirán el sensor RGB, el Arduino Uno, los motores paso a paso, el driver para los motores y los finales de carrera y su funcionamiento.

3.3.1 Sensor RGB TCS34725

Para el desarrollo del presente trabajo se proporcionó el sensor RGB TCS34725 de Adafruit. Se trata de un dispositivo electrónico que detecta el color de un objeto o de la luz incidente y lo convierte en un valor digital en la escala RGB [17]. Así, proporciona información sobre la cantidad de rojo, verde y azul de un objeto así como de la intensidad de la luz incidente. Además, contiene un filtro de infrarrojos que compensa la influencia de esa radiación en las medidas y es capaz de obtener la temperatura de color a partir de los datos obtenidos [17]. A continuación, se explicará el funcionamiento del TCS34725 así como algunas de sus características funcionales.

Este sensor consta de un array de 3x4 fotodiodos (ilustración 32), cuatro convertidores ADC (analog to digital converter), registro de datos, una máquina de estado, un LED blanco y una interfaz I^2C para la comunicación [17].

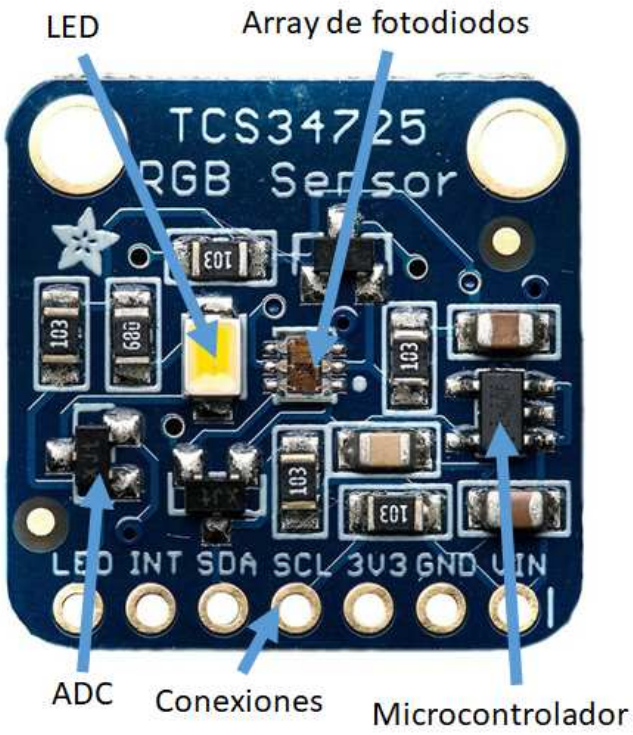


Ilustración 32: Sensor TCS34725 y algunas de sus componentes a partir de [17].

El array de fotodiodos está compuesto por doce fotodiodos. Antes de continuar, cabe tener en cuenta que un fotodiodo es un dispositivo semiconductor que convierte luz en corriente eléctrica [34]. Suele constar de una unión PN o PIN. La parte N tiene un exceso de electrones, por lo que contiene electrones libres, mientras que la parte P tiene carencia de electrones, por lo que tiene huecos libres. Cuando un fotón con la suficiente energía incide en el fotodiodo excita a un electrón, dando lugar a la formación de un hueco (ilustración 33). Si la absorción ocurre cerca de la zona de transición del fotodiodo, los huecos se mueven hacia el ánodo y los electrones hacia el cátodo debido al campo eléctrico en esa zona. De esta forma se genera corriente eléctrica [34].

Ahora bien, en el sensor TCS34725, tres de los fotodiodos tienen un filtro rojo, tres uno azul, tres uno verde y los últimos tres uno transparente (ilustración 34). Todos los fotodiodos constan adicionalmente de un filtro de infrarrojos. De esta forma, cada filtro deja pasar únicamente la luz de un color (es decir, de una determinada longitud de onda), por lo que, dependiendo del tono de lo percibido por el sensor, la señal de cada grupo de fotodiodos será mayor o menor [17]. Por ejemplo, un objeto azul activará en mayor medida los fotodiodos con el filtro del mismo color que el resto.

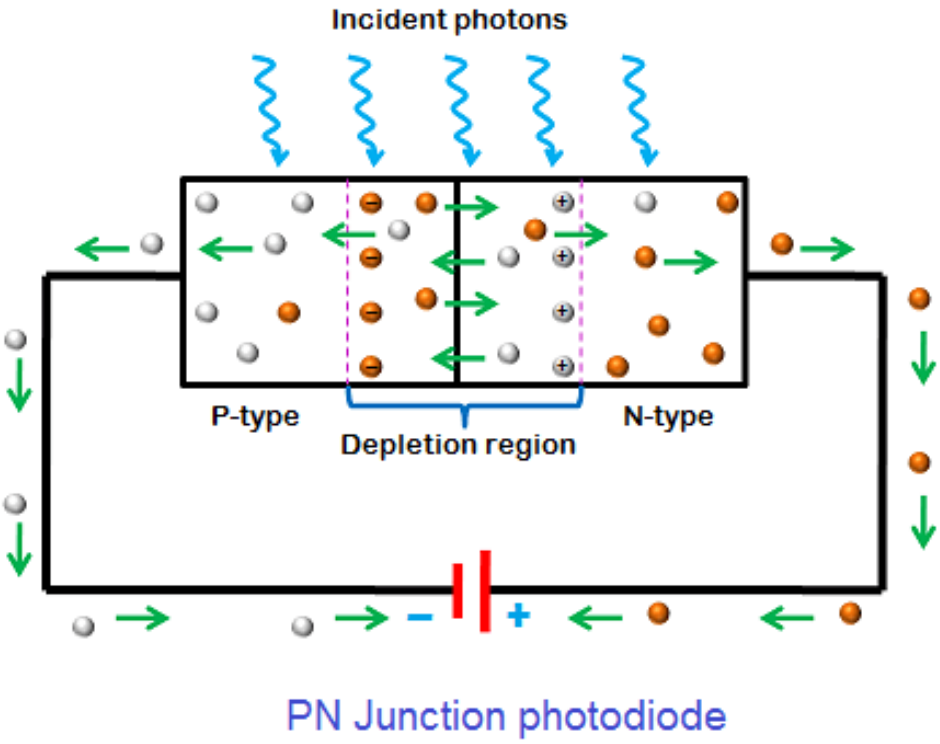


Ilustración 33: Funcionamiento de un fotodiodo según [34].

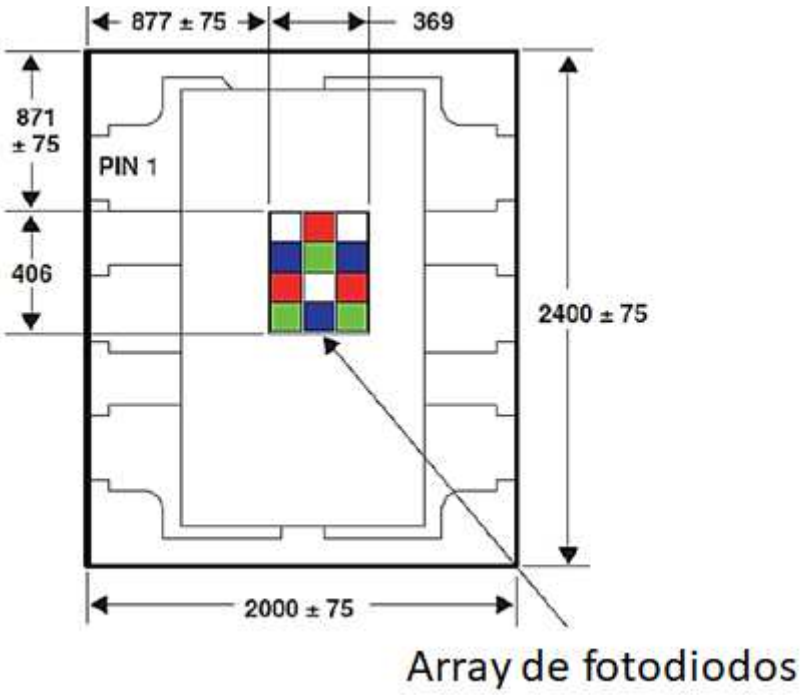


Ilustración 34: Disposiciones y medidas del array de fotodiodos, extraído de [17]. La unidad de todas las dimensiones se encuentra dada en micrómetros.

Habitualmente, se iluminará un objeto con el LED blanco del sensor y se recogerá la luz reflejada. A través de los filtros, los fotodiodos se excitarán según la cantidad de luz incidente. La señal de cada grupo de fotodiodos será recogida, interpretada y transformada de forma que pueda ser interpretada por el software. Dicho procedimiento se resume en la ilustración 35, en la que se obtendría como resultado un valor de rojo elevado, un cierto valor de verde y prácticamente nulo de azul.

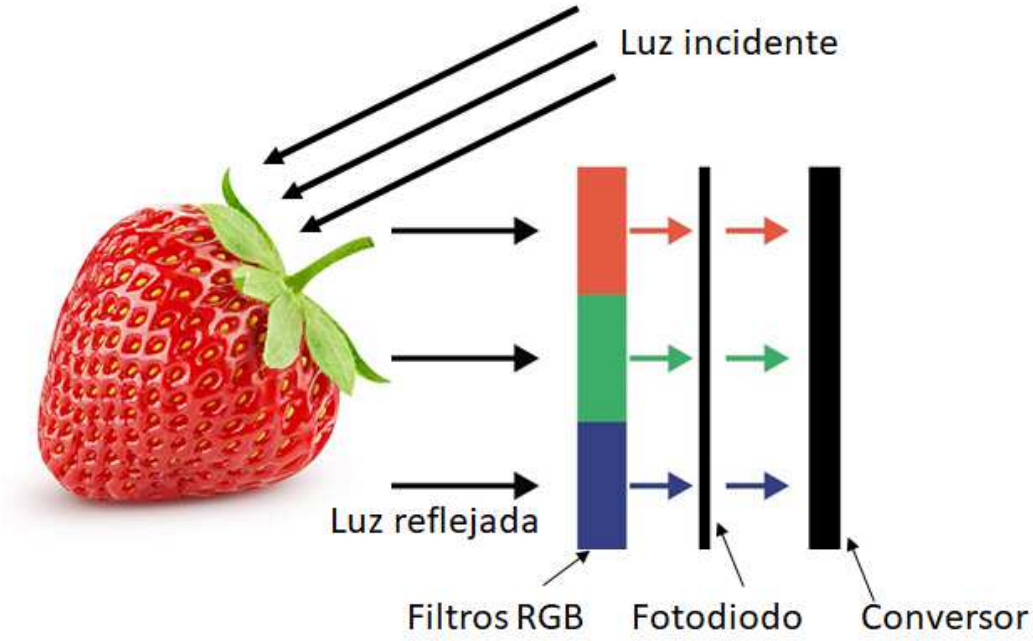


Ilustración 35: Detección del color en el sensor TCS34725 según de [17].

Los cuatro convertidores ADC transforman simultáneamente la corriente amplificada de los fotodiodos en valores digitales de 16 bits. Seguidamente, los datos obtenidos son guardados en el registro. Todos los procesos internos del sensor son regulados por la máquina de estado del sensor [17].

La comunicación con el sensor se realiza a través de un doble bus serial I^2C. El protocolo admite tres tipos de operación, la lectura, la escritura y una combinación de ambas. Además, el usuario tiene la opción de habilitar una señal externa de interrupción. La comunicación con el sensor tiene lugar a través de las conexiones del mismo, explicadas a continuación [17].

- LED: Si el valor digital en este pin es 1, el LED del sensor se encenderá, si éste es 0 permanecerá apagado.

- INT: Pin para la interrupción externa, que se activará si el valor digital de este pin es 1.
- SDA y SCL: A través de estos pines tiene lugar la transmisión de los datos medidos por el sensor.
- 3v3: Pin para el suministro de la alimentación con 3.3 V.
- GND: Es el pin correspondiente a la "tierra" del sensor.
- VIN: Es el pin de alimentación, al que va conectado el suministro de energía para el funcionamiento del sensor.

Una vez explicado el funcionamiento del sensor se puede pasar algunas de sus características principales. Resulta de interés que no ha de estar expuesto a temperaturas mayores a 40 grados ni a humedades superiores al 90 %. Asimismo, es importante saber que el voltaje de alimentación no debería superar los 3.3 V. El LED puede ser activado y desactivado a voluntad. Según la aplicación, es posible modificar el tiempo de integración y la ganancia [17].

La ganancia es una magnitud adimensional que expresa la relación de la amplitud de la señal de salida respecto a la señal de entrada, mientras que el tiempo de integración describe la duración de la toma de datos para una medida [35].

3.3.2 Arduino UNO

Arduino UNO es un producto de la empresa electrónica Arduino, conocida por el desarrollo de soluciones de software y de hardware fáciles de utilizar y de código abierto.
Este producto, que se encuentra entre los más sencillos de esta empresa, consiste en una placa que contiene un microcontrolador, varios puertos de entrada y de salida, tanto análogos como digitales, y el circuito electrónico correspondiente impreso [36]. En la siguiente imagen se puede observar un Arduino UNO.
En la tabla 1 se encuentran las características principales de este producto, que son de vital importancia para un correcto empleo del mismo.

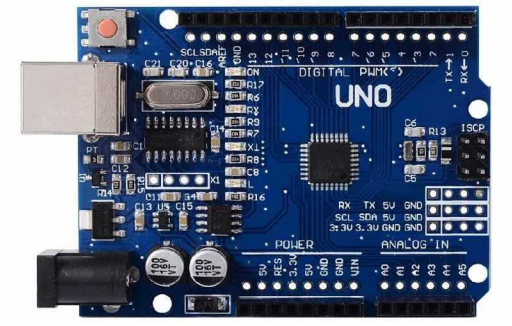


Ilustración 36: Arduino UNO, extraído de [36].

Microcontroller	ATmega328P
Operating Voltage	5 V
Input Voltage (recommended)	7-12 V
Input Voltage (limit)	6-20 V
Digital I/O Pins	14
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz
Built-In Led Pin Number	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Tabla 1: Características del Arduino UNO, extraída de [36].

En la siguiente imagen se observan las partes relevantes del Arduino UNO explicadas brevemente. Se puede ver que se cuenta con 13 pines digitales, algunos de los cuales están también destinados a la comunicación por I^2C o SPI. Además, hay 6 pines analógicos, de los que 2 están reservados para la transmisión y recepción de datos con el protocolo I^2C. El resto de pines están destinados a la alimentación. Se halla un puerto USB para la comunicación con el ordenador, así como un conector para el suministro de energía. También se encuentra el microcontrolador ATmega328P y el oscilador de cuarzo que regula la frecuencia a la que funciona la placa [36].

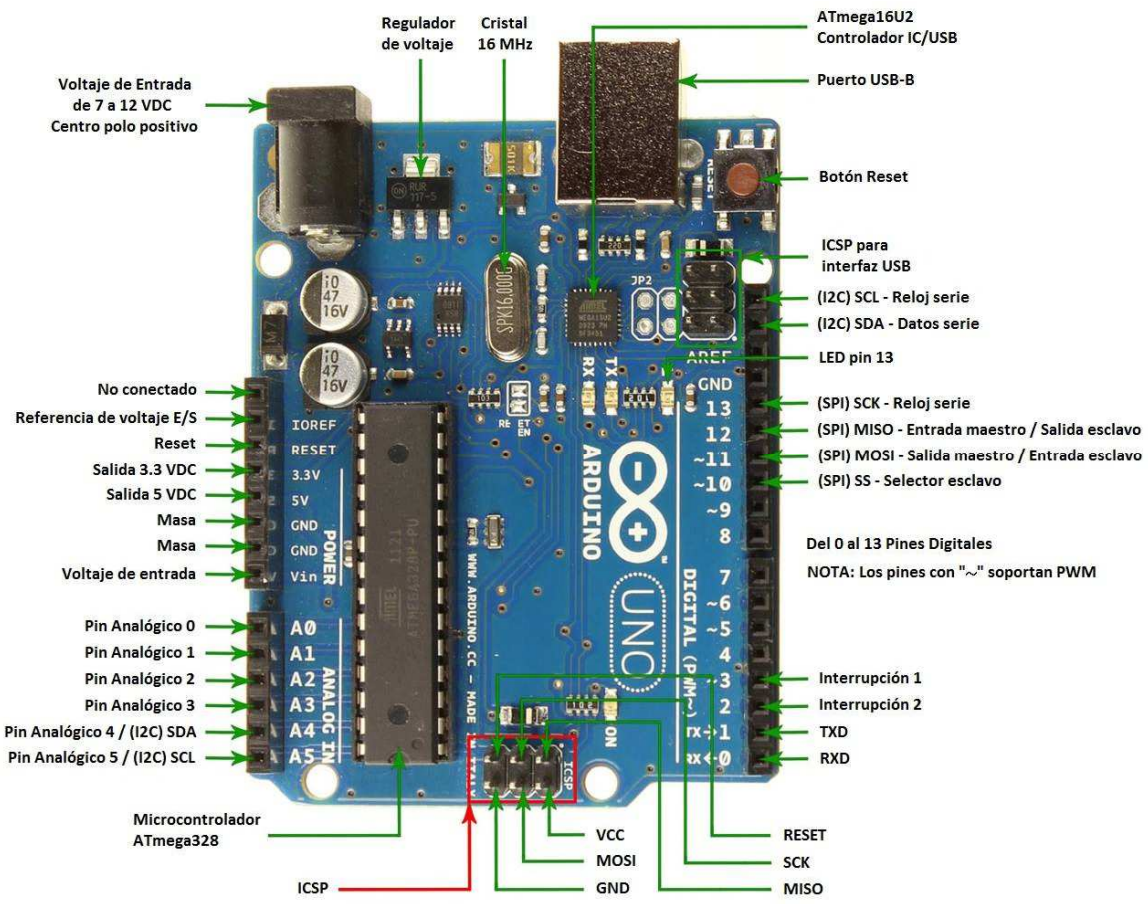


Ilustración 37: Partes del Arduino UNO, imagen adaptada de [36].

Para el presente trabajo resulta interesante conocer con más detalle algunas de las características clave de este producto, tales como algunos rasgos del microcontrolador, la alimentación de la placa, la comunicación y la memoria, que serán explicadas a continuación.

– Microcontrolador ATmega328P

  Es el núcleo del Arduino UNO. Se trata de un circuito integrado programable que puede ejecutar las órdenes que se almacenan en él. Para ello, consta de CPU, memoria y pines de entrada y de salida. Este dispositivo de 8 bits está fabricado según la arquitectura del tipo RISC. Algunas de sus características más relevantes son la frecuencia de funcionamiento de 16 MHz y sus 32 KB de memoria flash [37]. En la siguiente imagen se muestra un microcontrolador ATmega328P.

Ilustración 38: Microcontrolador Atmega328P, extraído de [37].

– Alimentación:

  Existen varias opciones para alimentar la placa Arduino UNO. En primer lugar, es posible suministrar la energía desde una fuente de alimentación externa a través del conector tipo “jack” o desde el pin Vin. En segundo lugar, también es posible abastecer al Arduino UNO desde el puerto USB [36].

  Hay que tener en cuenta que en este caso sólo se podrá contar con 5 V, mientras que la primera opción permite llegar a voltajes de hasta 20 V. Sin embargo, si la alimentación es externa, el fabricante recomienda no superar los 12 V y proporcionar como mínimo 7 V [36].

  Asimismo, es posible obtener el voltaje de la alimentación externa desde el pin IOREF y un voltaje de 3.3 ó 5 V desde los pines correspondientes [36].

– Memoria:

  El microcontrolador ATmega328P cuenta con tres tipos de memoria: la memoria flash, la memoria SRAM y la memoria EEPROM [37].

  La memoria flash contiene el programa compilado del usuario así como la información requerida para la inicialización del mismo. No es volátil, por lo que la interrupción súbita de la alimentación no supone una pérdida de datos. El espacio disponible para la memoria flash es de 32 KB, de los cuales 0.5 KB están reservados al programa de inicialización. Es de interés que se puede ejecutar pero no modificar un programa desde esta memoria [37].

En la memoria SRAM se encuentran las variables generadas por el programa del usuario durante su ejecución. Esta memoria es volátil, por lo que una interrupción de la alimentación supondría la pérdida de la información almacenada en ella. Se le reserva un espacio de 2 KB y es accesible sólo para el programa del usuario [37].

Finalmente, la memoria EEPROM, con un tamaño de tan sólo 1 KB, es sólo de lectura y no volátil. Se emplea para guardar los datos a largo plazo, dado que los mantiene incluso tras la ejecución de un reset. Este tipo de memoria es programable de forma externa [37].

A modo de resumen, en la siguiente imagen se encuentran los tres tipos de memoria del microcontrolador ATmega328P así como su tamaño.

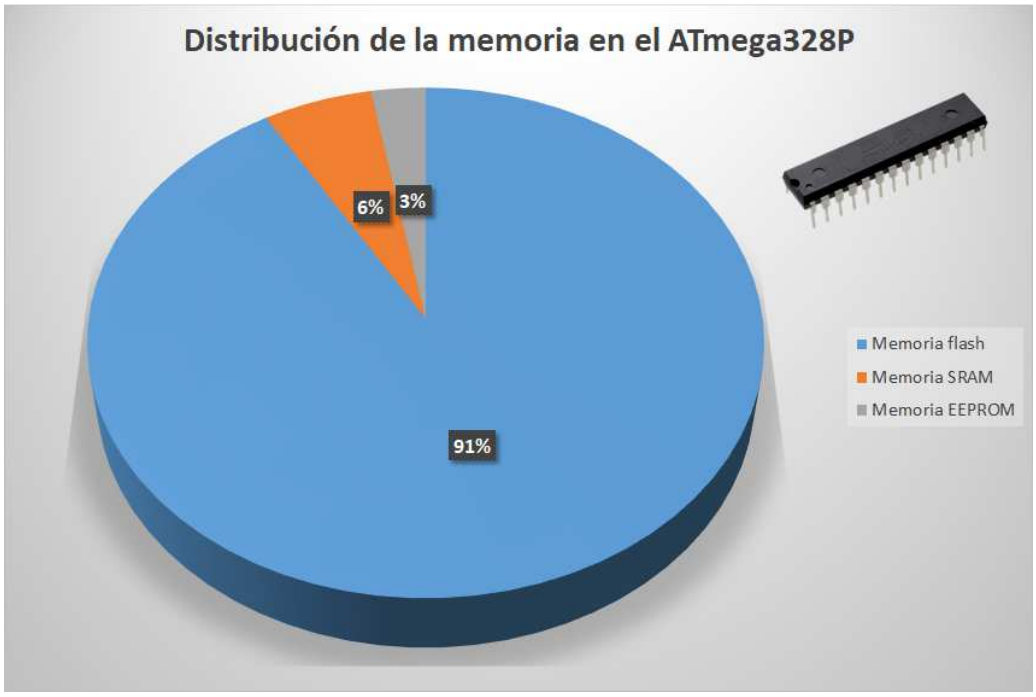


Ilustración 39: Distribución de la memoria en el Atmega328P según [37].

3.3.3 Motores paso a paso

A continuación, se explicarán los conceptos básicos relativos a los motores paso a paso en general, pasando a continuación a exponer el funcionamiento y la clasificación de los mismos. Seguidamente, se mencionarán brevemente las ventajas y las desventajas de este tipo de motores y se finalizará con un breve resumen de las características de los motores de los que se dispuso para el presente trabajo.

En primer lugar, se puede definir a los motores paso a paso como motores síncronos en los que el rotor es controlado por un campo magnético que se encuentra en el estator y se mueve en intervalos angulares constantes. Dicho de otra forma, el rotor salta de una posición a otra. Como se puede ver en la siguiente imagen, constan de un rotor con un imán permanente y un estator con varios grupos de bobinas [38].

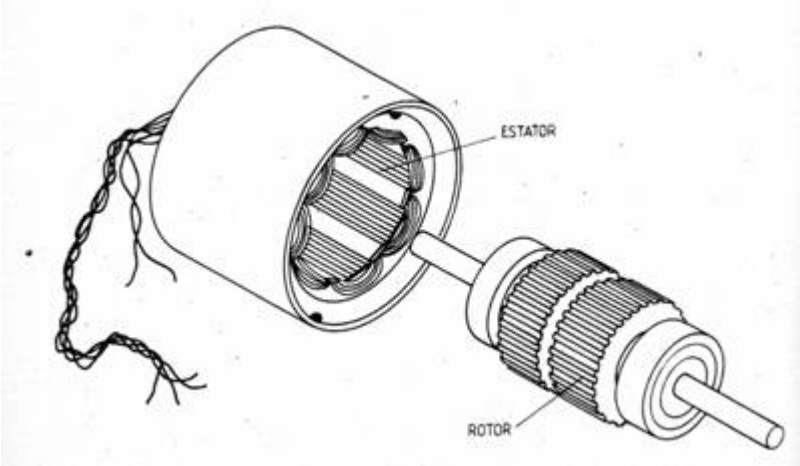


Ilustración 40: Esquema de un motor paso a paso (híbrido) según [38].

Por lo tanto, se trata de un dispositivo electromecánico que transforma impulsos eléctricos en desplazamientos angulares discretos. Para ello, el sistema de bobinas asegura que el rotor sólo pueda girar la distancia equivalente al recorrido entre las bobinas continguas en cada impulso. A este intervalo se le llama paso. El movimiento de rotación se consigue variando la alimentación de los grupos de bobinas de forma que el motor tenga que avanzar un paso por impulso para lograr el equilibrio electromagnético. De esta forma, la rotación del motor tiene lugar mediante los saltos de una posición discreta a otra gracias a los pulsos enviados a las bobinas del estator [38].

En la siguiente imagen se encuentra un esquema que resume la explicación anterior. Se corresponde con la secuencia de una fase, lo que significa que se activa una bobina para lograr un movimiento del rotor. La bobina activada está resaltada en rojo y el estator está representado de color azul.

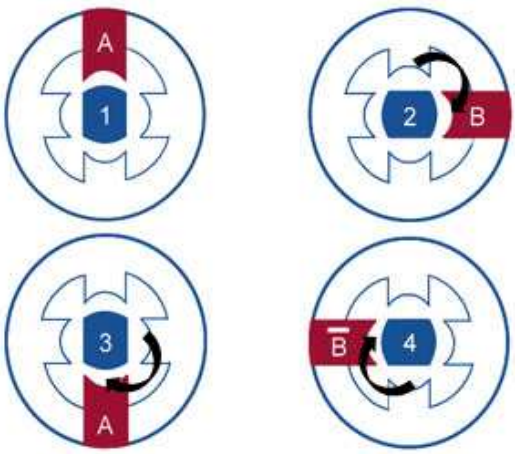


Ilustración 41: Esquema de la secuencia de una fase de [39].

Sin embargo, existen más formas de hacer girar un motor paso a paso. La secuencia de una fase no consume mucha corriente, pero presenta la desventaja de que el motor no gira con mucha fuerza. Este problema se soluciona con la secuencia de dos fases, en la que se activan dos bobinas correlativas por paso. Por ello, el campo magnético generado es más intenso y el par del motor es mayor, aunque el consumo también aumenta [38]. En la próxima imagen se encuentra el esquema de funcionamiento de un motor paso a paso con secuencia de dos fases.

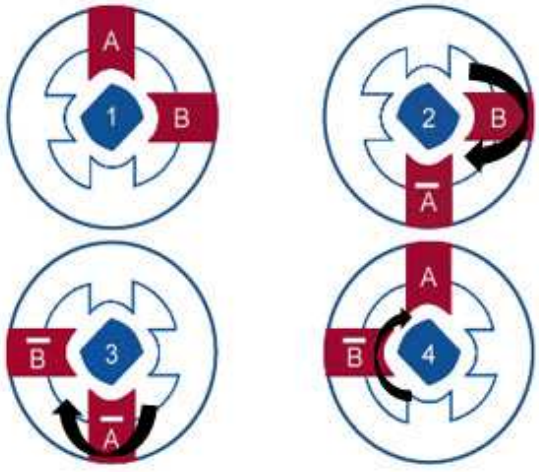


Ilustración 42: Esquema de la secuencia de dos fases de [39].

Además, hay una tercera opción llamada secuencia de medio paso. Consiste en activar de alternativamente una y dos bobinas, de forma que en cada pulso se avanza medio paso. Es la más indicada para aplicaciones que requieran de alta precisión, puesto que hay más

puntos discretos por vuelta. Sin embargo, puede presentar algunos problemas si la carga mecánica se encuentra cerca del máximo soportable por el motor [38]. En la siguiente imagen se encuentra un esquema del funcionamiento de la secuencia de medio paso.

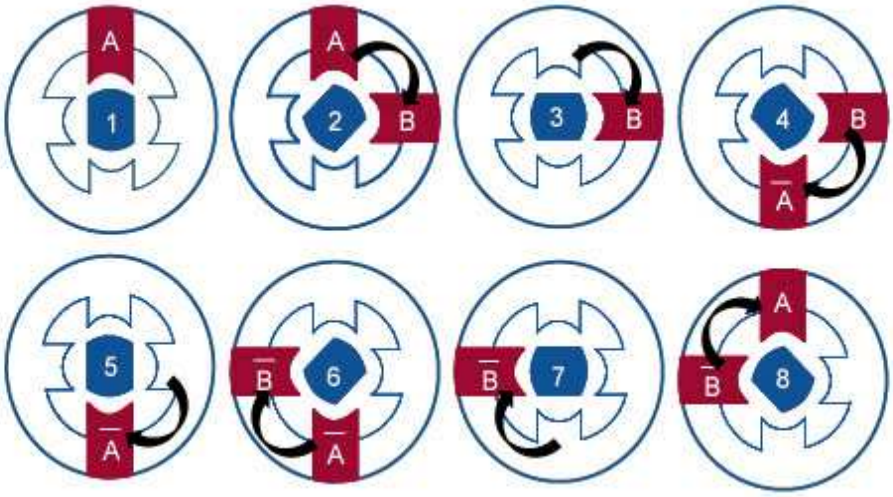


Ilustración 43: Secuencia de medio paso de [39].

Las tres secuencias permiten guiar con precisión el movimiento del motor, puesto que es posible conocer el ángulo del rotor siempre y cuando se mantenga la alimentación. Si se desea controlar un motor paso a paso hay que programar la secuencia de excitación de las bobinas en el orden correcto, aunque existen librerías de software que ya contiene toda la información necesaria para ello [38, 39].

Una vez conocidas las características básicas de este tipo de motores, se puede pasar a explicar su clasificación. En la literatura se encontraron tres posibles tipos de un motor paso a paso: el motor de pasos de reluctancia variable, el motor de pasos de rotor de imán permanente y el motor de pasos híbrido [38].

En el primer caso, el motor cuenta con un rotor dentado multipolar de hierro y un estator devanado del mismo material. El giro tiene lugar cuando uno o varios dientes del rotor son atraídos por una bobina activa para lograr una disminución de la reluctancia. Destaca por su rápida respuesta y la poca inercia que soporta, así como por encontrarse el eje “suelto” si el motor está apagado, dado que se pierde la magnetización del rotor y del estator [38].

En el segundo caso, el motor de pasos con rotor de imán permanente, la parte móvil del motor está fabricada con un material ferromagnético. Así, cuando se interrumpe la alimentación del motor, tan sólo el estator pierde su magnetismo, de forma que el rotor queda fijado en su posición. Este motor presenta la desventaja de que el número de polos magnéticos está limitado y, por lo tanto, su precisión también. Su gran ventaja radica en el gran par motor (momento de fuerza) que puede alcanzar [38].

El tercer caso, el motor híbrido, trata de reunir las ventajas de los otros dos tipos de motor, es decir, la alta precisión del de reluctancia variable y el elevado par del de imán permanente. El estator, dentado en su interior, cuenta con varias bobinas para su magnetización. La clave se encuentra en el rotor, que cuenta con un imán permanente dentado. Éste se encuentra dividido en dos secciones longitudinalmente, que se corresponden con el polo norte y el sur del imán [38]. Además, los dientes de ambas secciones están desplazados, de forma que el hueco de una sección se encuentra frente al diente de la otra, como se muestra en la siguiente imagen.

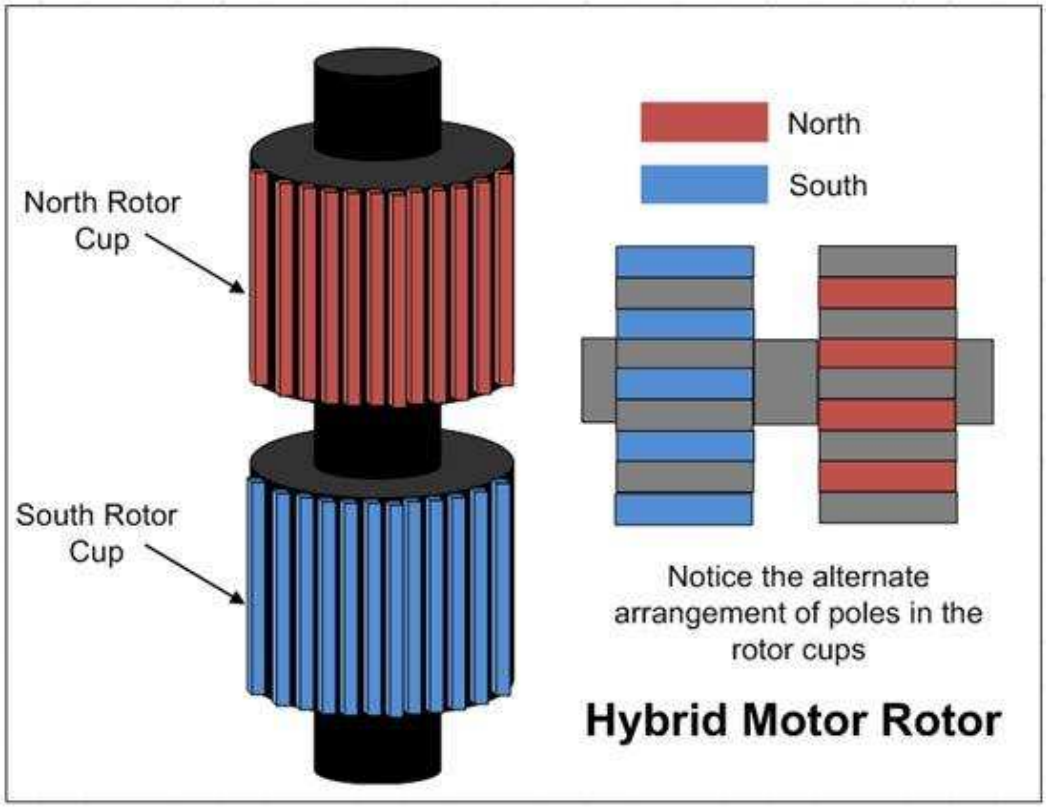


Ilustración 44: Rotor de un motor paso a paso híbrido según [38].

La rotación del motor tiene lugar cuando los polos norte o sur del rotor son atraídos por su contrario en el estator. Mediante la variación de la alimentación de las bobinas del estator y, por ende, de su magnetismo, tiene lugar el control del giro del eje [38], como se muestra en la siguiente imagen.

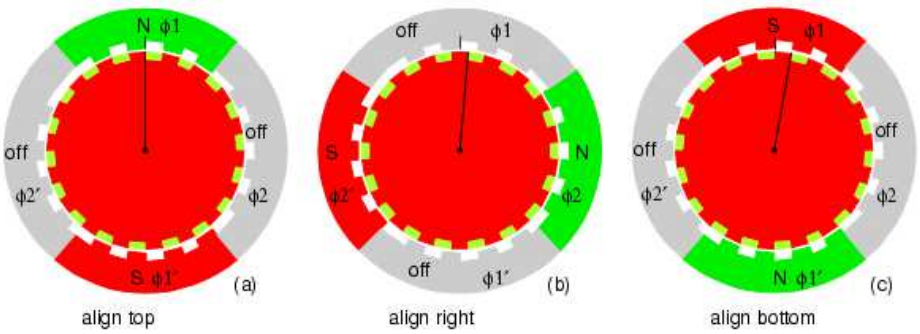


Ilustración 45: Control magnético de la rotación en un motor paso a paso híbrido según [38].

Los motores paso a paso híbridos son los más utilizados hoy en día, pues combinan una alta precisión con un par motor aceptable [39].

Por otro lado, los motores paso a paso también se pueden clasificar como unipolares o bipolares [40]. El estator de los unipolares consta de cuatro bobinas mientras que el de los bipolares cuenta con dos. La diferencia en el funcionamiento consiste en que en el motor unipolar se activa una bobina por pulso, mientras que en el bipolar se activan varias a la vez. Por ello, el par (la fuerza) de un motor bipolar será mayor que la de uno unipolar, aunque su control digital será algo más complejo. En la siguiente imagen se muestran de forma esquemática las diferencias entre ambos tipos de motor. Nótese que, mientras que el motor bipolar tiene cuatro terminales, el unipolar cuenta con seis o cinco, en caso de que las conexiones denominadas como común se unan [40].

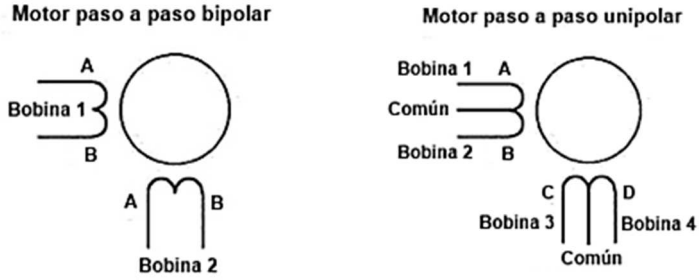


Ilustración 46: Motor paso a paso bipolar y unipolar, extraído de [40].

Una vez tratados los rasgos y los tipos de motores paso a paso se puede pasar a mencionar brevemente sus puntos fuertes y débiles.

A consecuencia de sus características, los motores paso a paso presentan la ventaja de que se pueden posicionar con alta precisión mediante el control digital, lo que los convierte en ideales para un gran número de aplicaciones relacionadas con la robótica. Además, se obtiene un máximo par del motor a bajas velocidades. Como par del motor se entiende el momento de fuerza que ejerce un motor sobre su eje de transmisión [40].

No obstante, este tipo de motores presentan también algunas desventajas. Una de ella es que, si se desea mantener el motor en una posición, se ha de mantener el grupo de bobinas pertinente activado, de modo que se da la paradoja de que el consumo en reposo puede ser mayor al del motor rotando. Si se tiene en cuenta que la alimentación constante de una bobina puede equivaler prácticamente a un cortocircuito, es frecuente que manteniendo una posición fija demasiado tiempo el motor se caliente en exceso [40]. Se podría pensar que basta con apagar la fuente de alimentación, lo que lleva a la segunda desventaja. Ésta consiste en que no es posible conocer en qué punto de giro se encuentra el motor justo cuando arranca, es decir, en el momento en el que se desconecta la alimentación, se pierde por completo la posición del rotor. Por ello, es muy frecuente combinar este tipo de motores con finales de carrera para establecer un punto de referencia cuando comienzan a funcionar [40]. La tercera desventaja es que, si bien presentan un par bastante elevado a bajas velocidades, a velocidades altas el par no es muy bueno. Dicho de otra forma, los motores paso a paso no son aptos para llevar mucha carga a velocidades elevadas [40].

Una vez explicadas brevemente las características, el funcionamiento, la clasificación y los puntos fuertes y débiles de los motores paso a paso es hora de dedicar unas líneas a los motores que fueron proporcionados para la realización del presente trabajo. Se contó con el modelo 28BYJ-48, con el ST-PM35-15-11C y con el SM-42BYG011-25. En la siguiente imagen se encuentran los tres motores.

Ilustración 47: 28BYJ-48, SM-42BYG011-25 y ST-PM35-15-11C (de izquierda a derecha), imágenes de [9].

Es importante mencionar que se requieren drivers para el control de estos motores, dado que el voltaje que necesitan para funcionar no debe interferir con la alimentación de la placa que los controla [39, 40]. Así, el driver actúa como barrera entre la alimentación de los motores y la placa, enviando las órdenes de control procedentes de la misma al motor y suministrándole la energía eléctrica que proviene de una fuente externa. Para el motor 28BYJ-48 se cuenta con el driver UNL2003, mientras que para los otros dos se emplea el L298N. Ambos drivers se encuentran en la imagen situada a continuación.

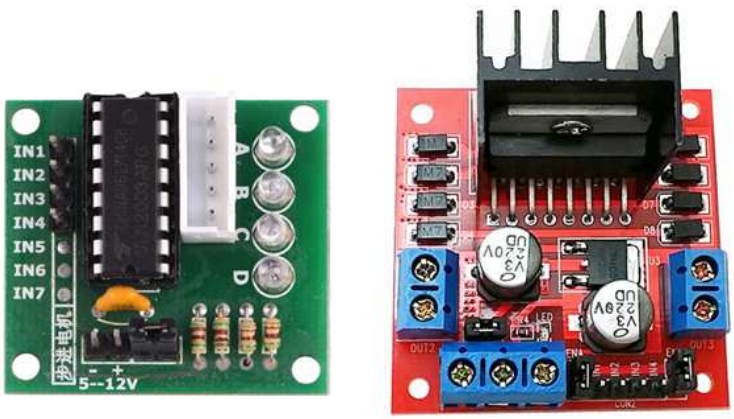


Ilustración 48: Driver UNL2003 y L298N (de izquierda a derecha), imágenes de [9].

En la siguiente tabla se encuentran resumidas algunas de las características más relevantes de los tres motores de los que se dispone. Se puede observar que el 28BYJ-48, con 4096 pasos por vuelta gracias a la reducción 1/64, es el más preciso de los tres y, por lo tanto,

el más lento. El más rápido es el ST-PM35-15-11C (48 pasos por vuelta). No obstante, es también el que menos fuerza tiene. En este aspecto, el SM-42BYG011-25 supera ampliamente a los otros dos motores [41, 42, 43].

Características	**28BYJ-48** 	**ST-PM35-15-11C** 	**SM-42BYG011-25** 
Tipo	Unipolar	Bipolar	Bipolar
Ángulo/paso	5.625º	7.5º	1.8º
Reducción	1/64	Sin reducción	Sin reducción
Pasos/vuelta	32, 4096 con reducción	48	208
Secuencia de control recomendada	1 fase	2 fases	2 fases
Voltaje recomendado	5 V	12 V	12 V
Corriente recomendada	55 mA	400 mA	330 mA
Diámetro de eje	5 mm	3 mm	5 mm
Par de torsión en tracción	300 g/cm	100 g/cm	2.3 kg/cm
Driver	UNL2003	L298N	L298N
Longitud, altura, anchura (mm, aproximado)	29 x 28 x 37	26 x 43 x 35	58 x 42.3 x 42.3

Tabla 2: Características principales de los motores según [41, 42, 43].

Una vez vistas algunas de las principales características de los motores de los que se dispone, se puede dar por finalizado el presente apartado.

3.3.4 Finales de carrera

Anteriormente (en la sección 3.3.3) se mencionó que una de las desventajas de los motores paso a paso consistía en la imposibilidad de conocer la posición exacta del rotor en el momento del arranque. Una posible solución consiste en incorporar finales de carrera para establecer un punto de referencia para el motor.

Un final de carrera es dispositivo que se coloca al final del recorrido de un elemento móvil con la finalidad de saber cuándo éste ha llegado a un punto determinado. Normalmente se colocan en los extremos, de forma que el elemento móvil los activa al llegar a ellos [44]. En la siguiente imagen se muestran algunos de estos dispositivos.

Ilustración 49: Distintos tipos de finales de carrera encontrados en [9].

Constan de un accionador unido a una serie de contactos. Hay accionadores de lengüeta, de bisagra, de palanca con rodillo, de varilla y de pulsador, entre muchos otros. En el interior de un final de carrera se puede encontrar un interruptor normalmente abierto (NA), normalmente cerrado (NC) o un conmutador, dependiendo de la aplicación objetivo [44]. En la siguiente imagen se muestra, de forma esquemática, el interior de un final de carrera.

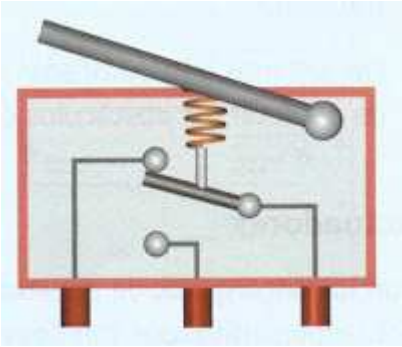


Ilustración 50: Constitución interna de un final de carrera según [44].

Al ser activados de forma mecánica, emiten o dejan de emitir una señal electrónica que puede ser utilizada para contar, posicionar, parar o iniciar una secuencia [44].

De las tres patillas que suele tener un final de carrera, una se corresponde con el modo normalmente abierto (NA o NO), otra con el modo normalmente cerrado (NC) y una tercera al contacto fijo (C). En la siguiente imagen se muestra el funcionamiento de un final de carrera. En reposo se obtendría un uno lógico de la patilla NC y un cero lógico de la NA, mientras que una vez activado el final de carrera se obtendría un cero lógico en la patilla NC y un uno lógico en la NA [44].

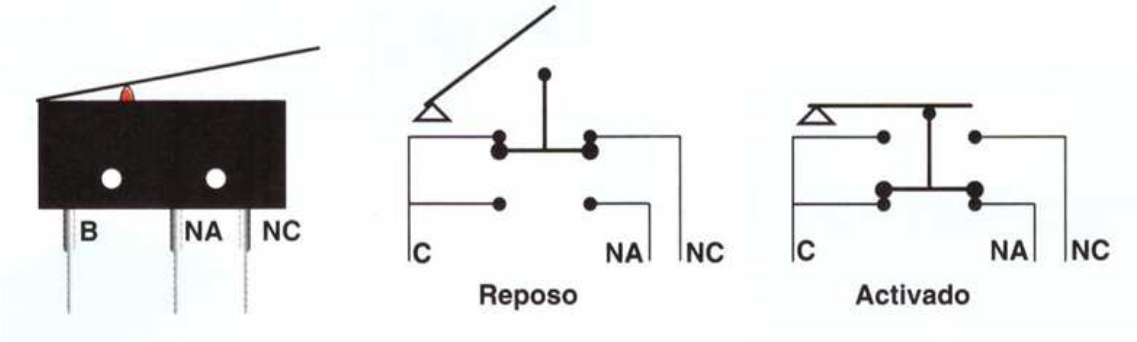


Ilustración 51: Patillas del final de carrera, conexiones en reposo y conexiones en activo (de izquierda a derecha), extraído de [44].

Y así, se da por concluída la sección destinada a los conceptos básicos relacionados con la electrónica.

3.4 Software

Tras haber explicado el marco teórico físico, químico y electrónico se concluirá el capítulo con una breve exposición del software empleado durante el desarrollo del prototipo. En primer lugar, se presentará la plataforma de programación de Arduino, conocida como Arduino IDE. Seguidamente, se pasará a dar una idea general sobre las características y posibilidades de la GUI de MATLAB y se finalizará con el software de diseño SOLIDWORKS.

3.4.1 Arduino IDE

En este apartado se comenzará con la definición de IDE, se enumerarán las características principales del IDE de Arduino y se acabará explicando brevemente la estructura del código generado mediante este software.

En primer lugar, en la literatura se define IDE (del inglés Integrated Development Environment) como un entorno de desarrollo integrado que consta de una serie de herramientas de programación [45]. Dicho de otra forma, se trata de un programa destinado al desarrollo de software, para lo cual ha de contar con un editor de código, un compilador, un depurador y una interfaz gráfica para el usuario. El IDE de Arduino incorpora además las herramientas para subir el programa compilado a la memoria flash de la placa [45].

Los programas diseñados con este IDE tienen la extensión “ino” y deben ser guardados en una carpeta con su mismo nombre [45].

Seguidamente, se mostrarán algunas de las características y herramientas del software de Arduino más relevantes para el presente trabajo.

A continuación, se muestra el aspecto de la interfaz de usuario con algunas de sus partes señaladas. En el editor de software el usuario escribe su programa, que puede verificar, guardar, compilar y subir a la placa de Arduino mediante los controles señalados. La consola de errores ayuda a encontrar y corregir los fallos que se puedan cometer en la escritura del programa o que puedan tener lugar si hay problemas con la comunicación con la placa Arduino [45].

Ilustración 52: IDE de Arduino con algunas de sus partes según [45].

Es muy importante seleccionar la placa y el puerto adecuados para cargar un programa correctamente en la placa Arduino. El puerto se selecciona mediante la ruta **Herramientas > Puerto: "COMX (Placa de Arduino)" > COMX** [45], como se muestra en la siguiente imagen.

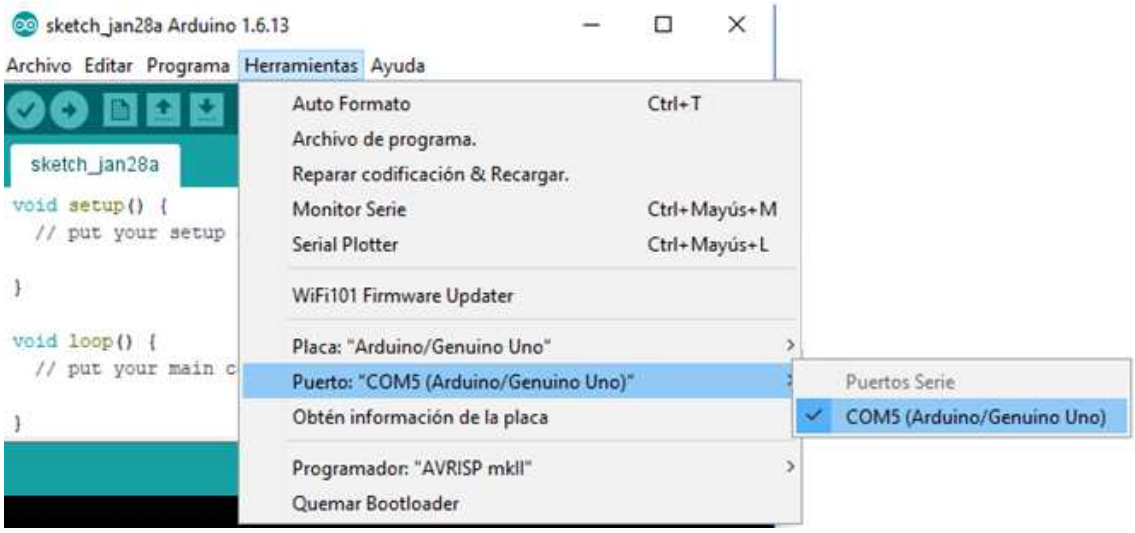


Ilustración 53: Selección del puerto en el IDE de Arduino según [45].

La placa se selecciona mediante la ruta **Herramientas > Placa: "Placa de Arduino" > Placa de Arduino** [45], que se observa en la siguiente captura de pantalla.

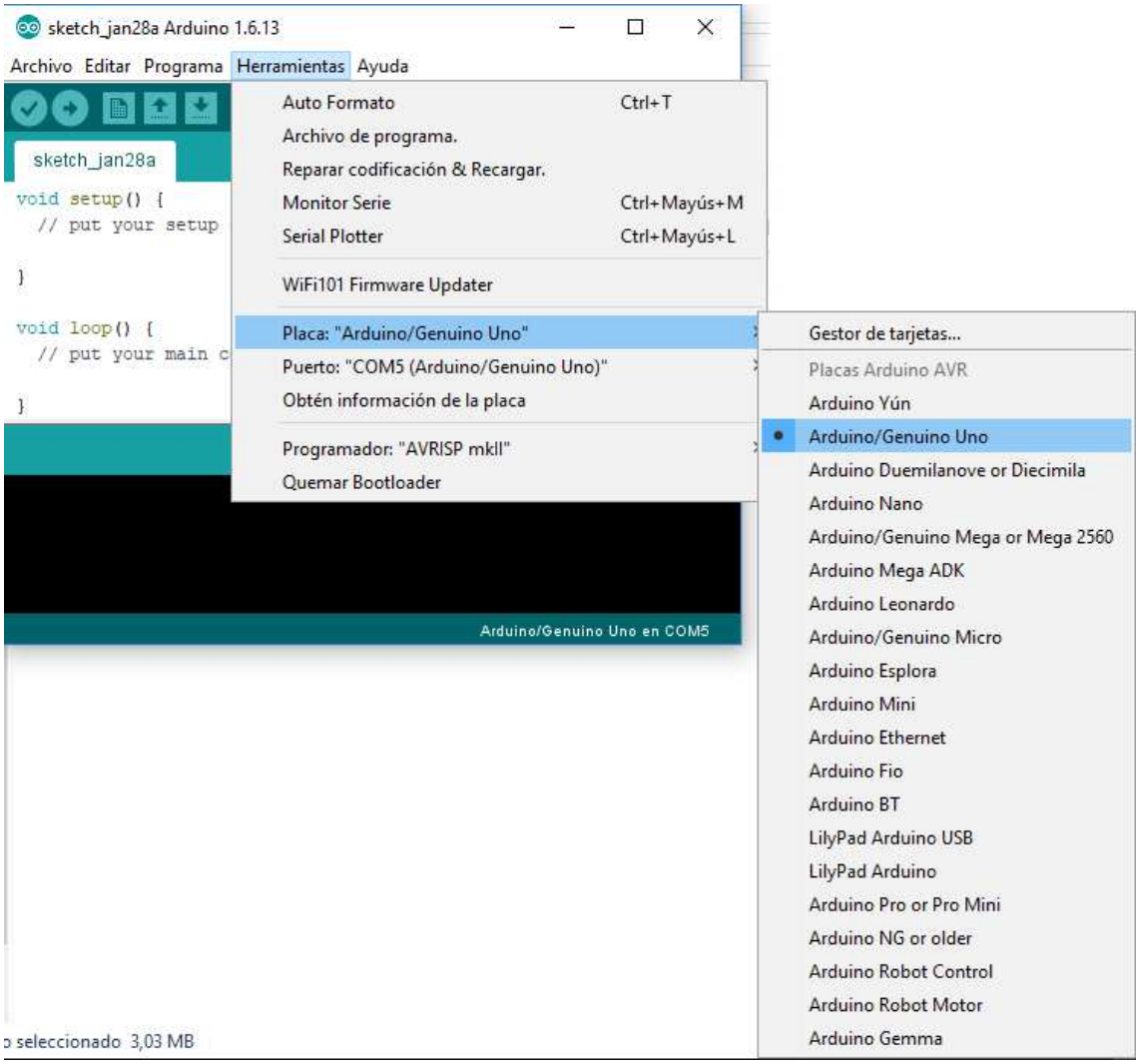


Ilustración 54: Selección de la placa en el IDE de Arduino según [45].

Cuando se sube un programa se usa el bootloader de 0.5 KB de tamaño de la memoria flash que se mencionó en el apartado 3.3.2. Durante su ejecución, el led integrado en la placa parpadea, indicando que está teniendo lugar la carga e inicialización del nuevo software diseñado por el usuario [45].

Otra herramienta de interés es el gestor de librerías, que permite agregar, eliminar o actualizar las librerías del IDE. Esto se lleva a cabo desde la ruta **Programa > Incluir Librería > Gestionar Librerías**. Una librería se define como un conjunto de implementaciones funcionales codificadas en un lenguaje de programación que ofrecen una interfaz simplificada y bien definida para la funcionalidad objetivo. Por ejemplo, una librería para

el control de motores paso a paso evita que el usuario tenga que programar la secuencia de control del motor, puesto que incluirá una función que contendrá dicha secuencia y que el usuario podrá emplear para hacer funcionar su sistema [45].

Otro punto a tener en cuenta es que la comunicación entre el ordenador y la placa Arduino tiene lugar a través de un cable USB. El monitor serie, que se encuentra en la siguiente captura de pantalla, es una herramienta que permite visualizar los datos recibidos del Arduino a través del puerto serial y enviar datos a la placa [45, 36].

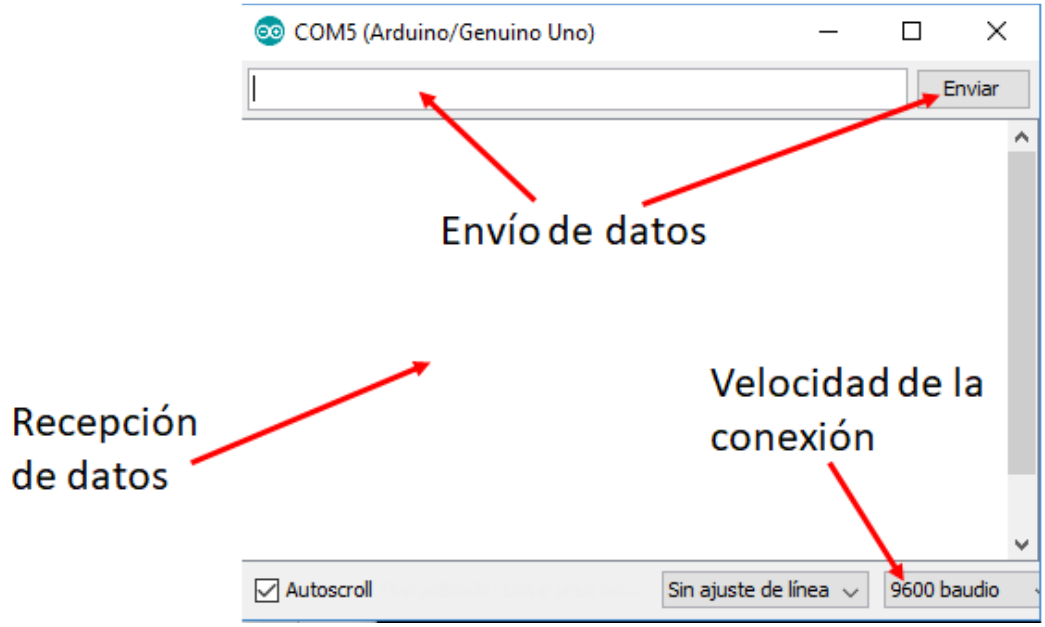


Ilustración 55: Monitor serie del IDE de Arduino según [45].

Para programar en el IDE de Arduino hay que utilizar un subconjunto del lenguaje de programación C. Este lenguaje es apreciado por su eficiencia y es empleado sobre todo en la creación de software de sistemas y aplicaciones.

Todo programa diseñado en este entorno presenta tres elementos clave: la declaración de variables y de funciones, la configuración de Arduino y el bucle principal [45].

En la declaración de variables se asigna un valor a un nombre que Arduino guardará para utilizarlo o alterarlo posteriormente. Las variables que se definan al inicio del programa serán accesibles en todo momento, mientras que las que se declaren dentro de una función sólo existirán dentro de la misma [45]. Hay varios tipos de variables, algunos de los cuales se encuentran en la siguiente tabla.

Denominación	**Característica**
int	Número entero entre -2^{31} y 2^{31}.
long	Número entero entre -2^{63} y 2^{63}.
float	Número decimal en precisión simple de 32 KB.
double	Número decimal en precisión doble de 64 KB.
boolean	Variable con dos estados posibles: Verdadero o falso.
char	Variable que contiene un símbolo o un carácter.
string	Variable que contiene una cadena de texto.

Tabla 3: Algunos tipos de variables y sus características según [46].

Al inicio del programa es frecuente especificar también las librerías que se necesitarán. Asimismo, es habitual definir funciones en este apartado. Las funciones son líneas de código que serán ejecutadas cuando se escriba su nombre en el programa, de forma que se puede evitar tener que escribir varias veces la misma secuencia de comandos si se define una función que las contenga. Para ello, basta con seguir la siguiente estructura: *void NombreDeLaFunción( ){ Código }*. Si se quiere emplear el código de esta función, bastará con escribir *NombreDeLaFunción( )* en el programa, siempre y cuando ésta halla sido correctamente definida [45].

En la configuración de Arduino (*void setup( )*) se escribe el código necesario para la inicialización del programa. Esta sección se ejecuta sólo una vez. Aquí se definen los pines que se utilizarán y de qué forma (entrado o salida), así como las características de la conexión con el ordenador [45].

Finalmente, en el bucle principal (*void loop( )*) se definen todas las instrucciones y comandos para que el programa funcione correctamente. Mientras la placa de Arduino permanezca alimentada, esta sección se ejecutará de forma infinita [45].

Con estas líneas queda brevemente presentado el IDE de Arduino.

3.4.2 MATLAB y GUIDE

MATLAB es una herramienta de software matemático de pago que ofrece un entorno de desarrollo (IDE) con un lenguaje de programación propio basado en el lenguaje C. Por ello, cuenta con un editor de código, un compilador, un depurador y una interfaz gráfica para el usuario. Además, tiene un gran número de prestaciones y complementos adicionales que lo convierten en un programa muy complejo, completo y potente [47].

Sin embargo, para el presente trabajo basta con conocer sus elementos básicos, que son el editor de código, la ventana de comandos, el espacio de trabajo (workspace) y algunas de sus herramientas más específicas como el comando “deploytool” y el programa GUIDE [47].

El editor de código permite escribir, modificar y guardar programas en el lenguaje de MATLAB.

A través de la ventana de comandos se pueden ejecutar órdenes o recibir información sobre la ejecución del programa y de los posibles errores [47].

El espacio de trabajo hace referencia a la ubicación en la que se guardan las variables generadas durante la sesión de MATLAB, que serán accesibles desde la ventana situada en la interfaz del programa. En la siguiente imagen se observa la interfaz de MATLAB con los tres elementos explicados anteriormente señalados. Además, se presenta un sencillo programa para mostrar el funcionamiento de este software matemático. Se puede observar que, en primer lugar, se ejecutó la línea “a=1;” en la ventana de comandos. Esta orden generó la variable “a” con el valor “1” en el espacio de trabajo. A continuación se escribió en el editor de código un programa que asigna el valor “3” a la variable “b” y asigna a la variable “c” el valor de la suma de “a” y “b”. Seguidamente, se ejecutó el código desde la ventana de comandos escribiendo el nombre del archivo (Untitled) y presionando la tecla “Enter”. Tras la ejecución del programa, en el espacio de trabajo hay tres variables: “a”, que fue definida desde la ventana de comandos, y “b” y “c”, que fueron generadas por el programa “Untitled”.

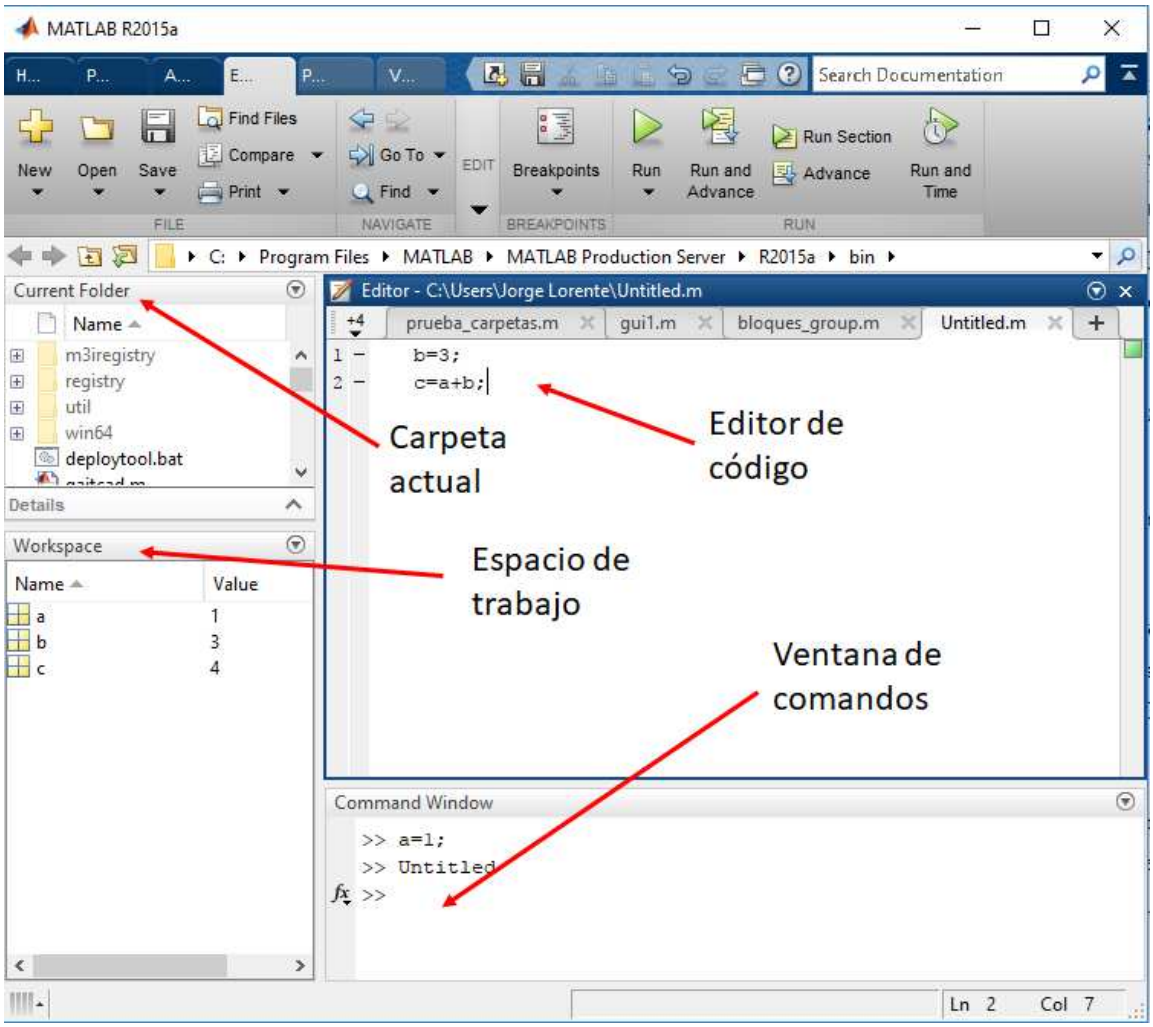


Ilustración 56: Interfaz de MATLAB con programa de ejemplo.

Mediante su complemento GUIDE (Graphical User Interface Development Environment), MATLAB ofrece al usuario la posibilidad de diseñar aplicaciones para automatizar tareas y cálculos, llamadas GUIs. Las GUIs (del inglés Graphical User Interface) son interfaces gráficas que permiten un manejo sencillo de aplicaciones de software, eliminando la necesidad de aprender un lenguaje de programación o de escribir comandos a fin de ejecutar un determinado programa [47]. Un ejemplo de GUI sería un programa que imita a una calculadora, evitando que el usuario tenga que realizar las operaciones matemáticas a mano.

En la siguiente imagen se muestra una captura de pantalla de la interfaz de GUIDE.

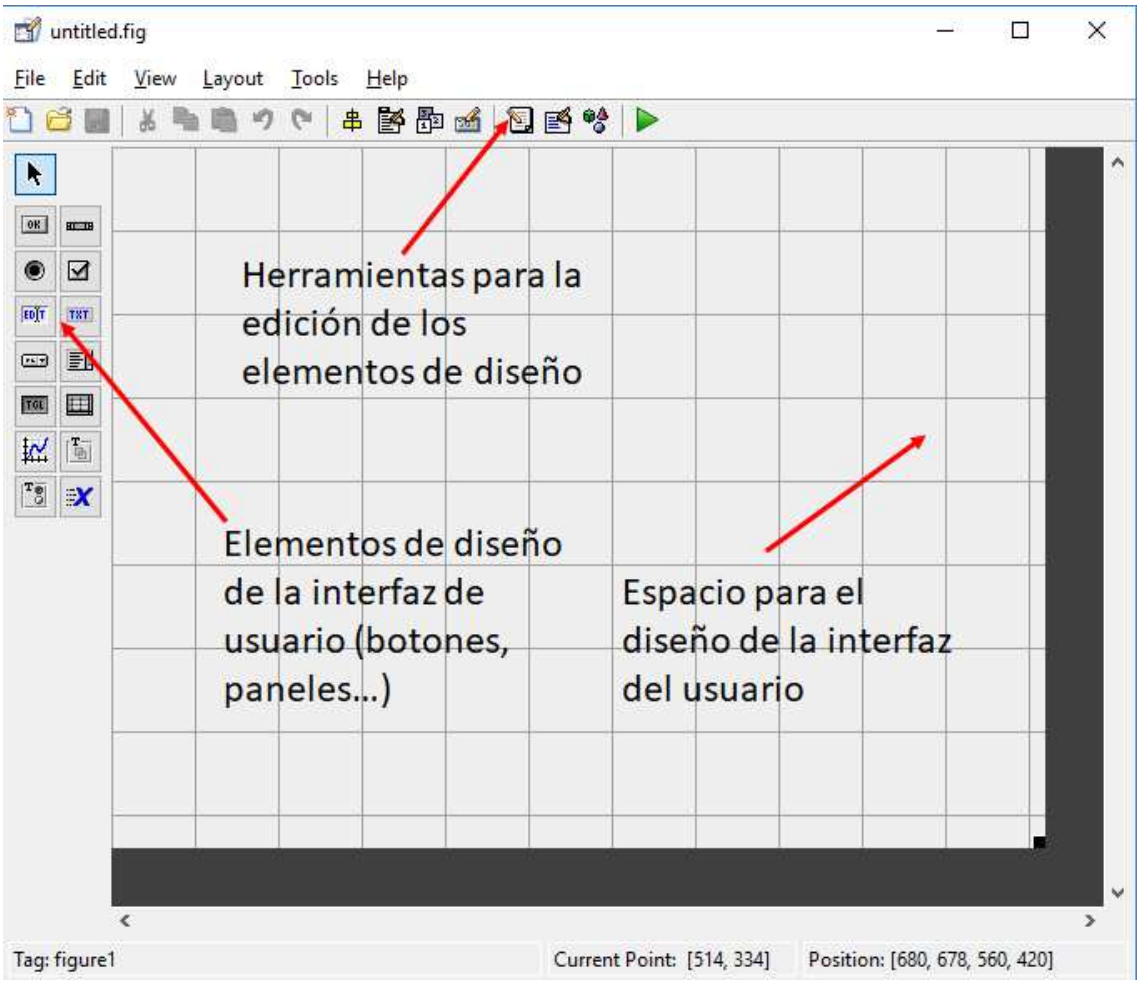


Ilustración 57: Interfaz de GUIDE y algunos de sus elementos básicos.

Así, GUIDE permite desarrollar de forma interactiva aplicaciones personalizadas. Cuenta con un editor de diseño con el que se puede elaborar gráficamente la interfaz de usuario. Cabe la posibilidad de agregar cuadros de diálogo, controles de interfaz de usuario (botones y controles deslizantes) y contenedores (paneles y grupos de botones) y de editar sus características a placer. Una vez acabada la interfaz, la herramienta genera de forma automática el código MATLAB necesario para construir la interfaz, el cual se puede modificar para programar el comportamiento de la aplicación [47].

Finalmente, mediante el comando “deploytool” de MATLAB se puede convertir la GUI diseñada mediante GUIDE a formato “exe” mediante un compilador diseñado con tal fin, de forma que el producto final pueda ser instalado en cualquier ordenador independientemente de si el usuario tiene MATLAB o no [47]. A continuación se muestra una captura de pantalla de la ejecución del comando “deploytool” y las diferentes opciones que proporciona.

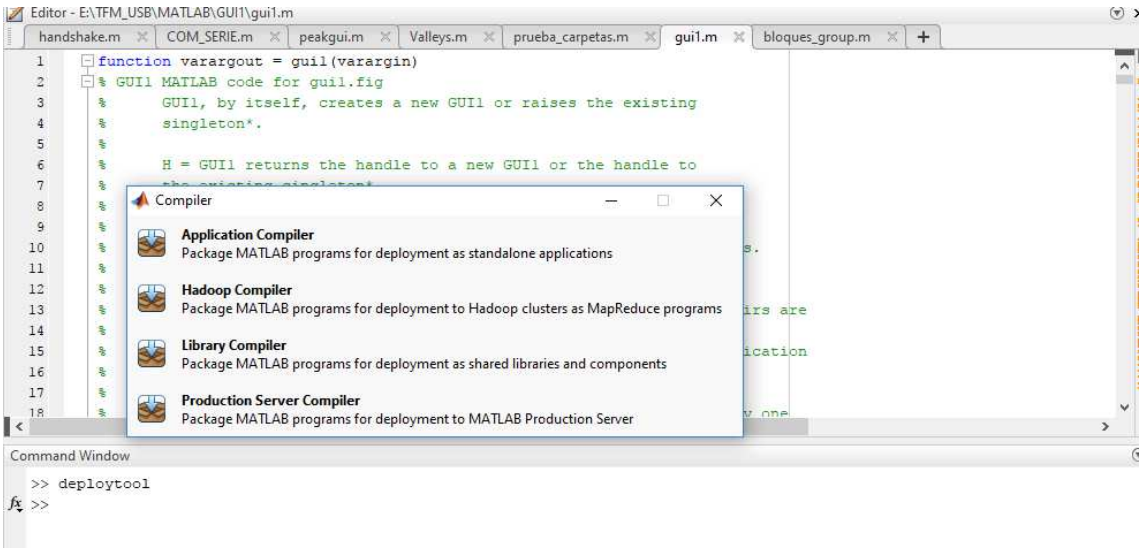


Ilustración 58: Comando “deploytool“ para la transformación del código MATLAB en ejecutable “exe”.

Por consiguiente, para el desarrollo de una aplicación sencilla e intuitiva que permita al usuario manejar el prototipo sin tener necesidad de conocer el código que la hace funcionar, en el presente trabajo se seguirán los siguientes pasos [47]:

- Diseño de la interfaz gráfica mediante la herramienta GUIDE de MATLAB.
- Generación del código MATLAB para construir la interfaz gráfica diseñada (automático).
- Programación del código MATLAB necesario para que la aplicación funcione como se pretende.
- Conversión del código MATLAB a formato “exe” mediante la herramienta de MATLAB “deploytool” para una fácil instalación y ejecución de la aplicación por parte del usuario final.

3.4.3 SOLIDWORKS

El SOLIDWORKS es un programa de modelado mecánico en dos y tres dimensiones basado en el sistema de diseño paramétrico. La naturaleza paramétrica de este software implica que las dimensiones y las relaciones geométricas serán las que definan la forma de la pieza diseñada y no al revés [48]. Mediante este programa se pueden generar modelos, ensamblajes y planos, como se puede ver en la siguiente imagen.

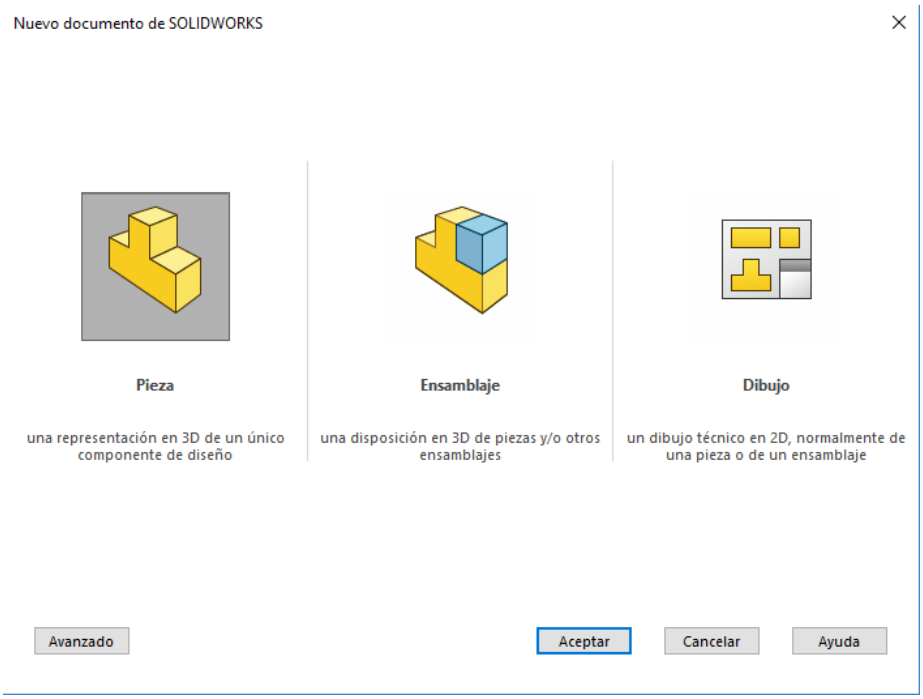


Ilustración 59: Pantalla de selección de nuevo documento en SOLIDWORKS.

La interfaz del programa variará dependiendo del tipo de documento que se elija [48].

El SOLIDWORKS funciona a partir de parámetros, que aplican restricciones cuyos valores determinan la forma o geometría del modelo o del ensamblaje. Éstos pueden ser numéricos, como la longitud de una recta o el diámetro de un círculo, o geométricos, como las relaciones de tangencialidad o de concentricidad. Con la finalidad de capturar la intención del diseño, los parámetros pueden interactuar entre sí por medio de las relaciones [48].

Como intención del diseño se entiende al comportamiento que el creador espera del modelo si varía determinados parámetros. Por ejemplo, si se está diseñando una lata de refresco, se desea que el agujero para beber se encuentre en la parte superior del modelo,

independientemente de la altura o el tamaño del mismo. SOLIDWORKS permite al usuario asignar el agujero a la superficie de la parte superior y mantendrá el agujero allí aunque cambien las dimensiones de la lata, respetando así la intención del diseño.

Este programa cuenta con una serie de características y operaciones para el modelado de piezas y ensamblajes.

El diseño de una pieza en SOLIDWORKS suele comenzar con un croquis bidimensional sobre un plano espacial a elegir. Se cuentan con varias herramientas para su elaboración, como puntos, líneas, arcos, conos e incluso ranuras. El siguiente paso consiste en especificar las dimensiones del croquis para definir el tamaño y la forma de la pieza en desarrollo [48]. Se pueden emplear relaciones para aplicar atributos como simetría, paralelismo, tangencialidad o perpendicularidad al croquis [48]. En la siguiente imagen se muestra un croquis en dos dimensiones con sus medidas ya definidas que fue realizado con el SOLIDWORKS.

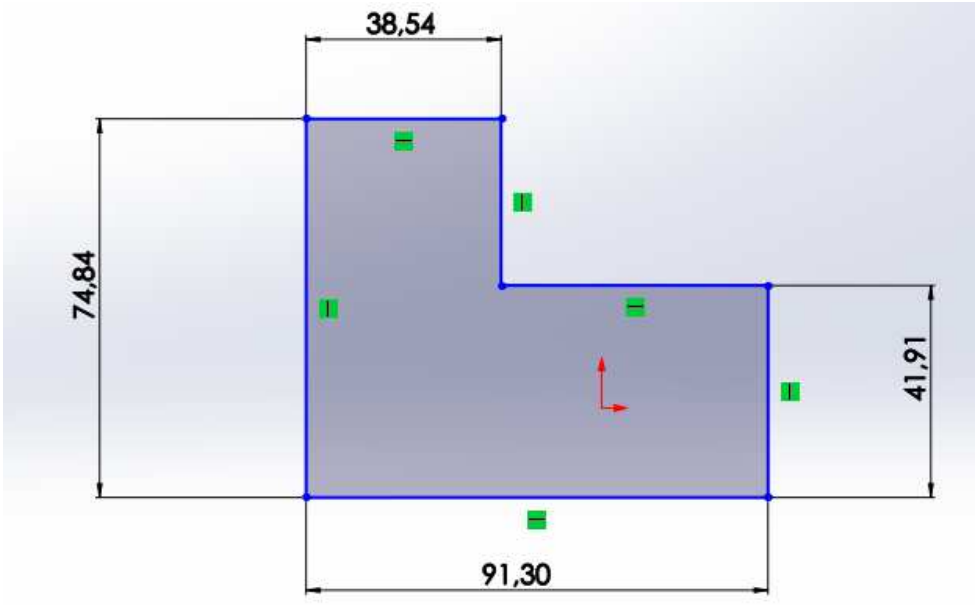


Ilustración 60: Croquis bidimensional generado con SOLIDWORKS

A continuación, se genera una pieza tridimensional a partir del bosquejo bidimensional. Para ello, se puede extruir el croquis en una dirección o alrededor de un eje. Si se ha generado un croquis sobre una pieza tridimensional, es posible eliminar material de la misma [48]. En la siguiente imagen se muestra cómo a partir del mismo croquis se pueden generar diferentes piezas mediante las operaciones de extrusión y de revolución.

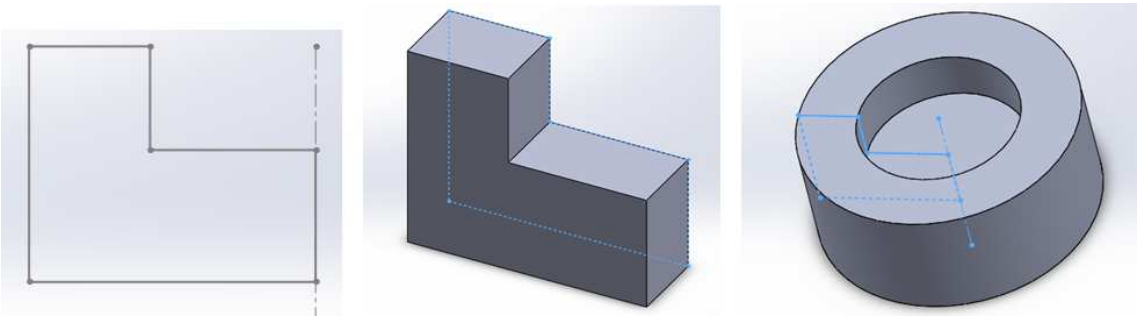


Ilustración 61: Croquis original, resultado de la extrusión y resultado de la revolución (de izquierda a derecha).

Una vez obtenido el modelo tridimensional, se le pueden aplicar operaciones como el añadido de chaflanes o la extrusión de cortes, entre muchas otras. A continuación, se muestra un ejemplo de ello. La extrusión de corte es una operación en la que se define un croquis para eliminar material en una pieza [48].

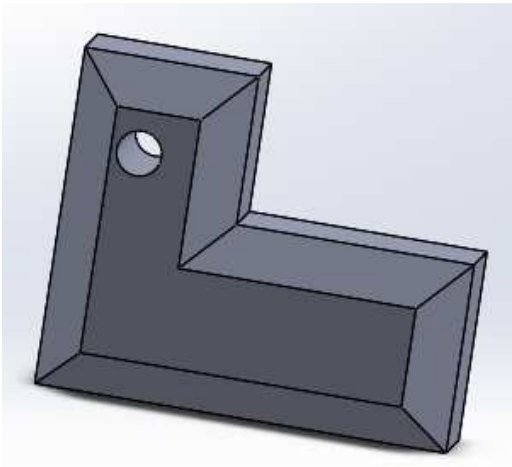


Ilustración 62: Resultado del añadido de un agujero (extrusión de corte) y de chaflanes a la pieza de la figura anterior.

Para modelar un ensamblaje se añaden piezas para lograr una construcción determinada. La posición relativa entre las componentes del diseño se define a través de las relaciones de posición: tangencialidad, colinearidad, coincidencia, perpendicularidad, concentricidad y distancia. Gracias a ellas el modelo puede ser construido con rapidez y sencillez [48].

En la siguiente imagen se muestra el procedimiento del ensamblaje en base a un ejemplo. Mediante la función “Insertar componentes” se cargan las piezas en el archivo del diseño [48]. Como se puede observar, se definió que la cara interna del aro tenía que coincidir

con la parte alargada del clavo y que la parte inferior de la cabeza del clavo debía coincidir con la cara superior del aro. Así, la posición relativa de ambas piezas quedó perfectamente fijada.

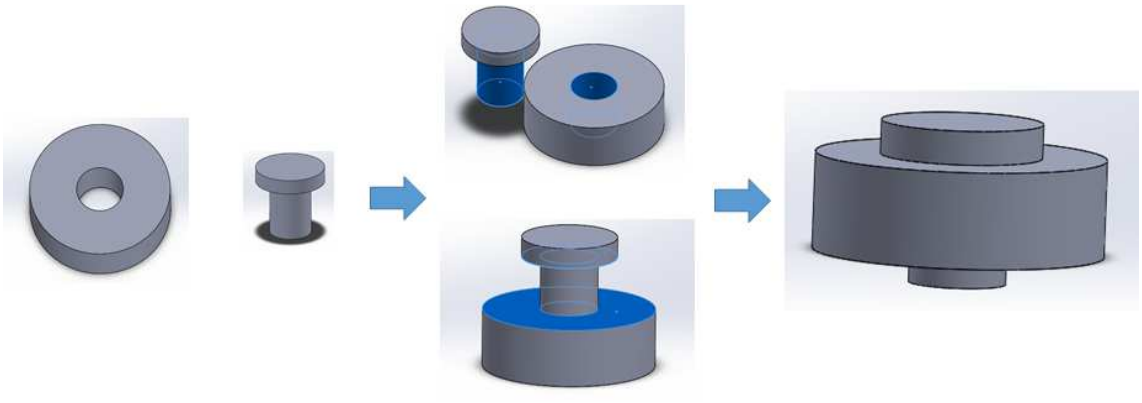


Ilustración 63: Piezas empleadas en el ensamblaje de ejemplo, relaciones de posición y resultado final (de izquierda a derecha).

Finalmente, cabe señalar que el SOLIDWORKS también permite generar planos a partir de piezas o ensamblajes. Las vistas se generan automáticamente a partir del modelo y se pueden añadir notas, dimensiones y tolerancias [48]. En la siguiente imagen se puede observar el plano generado a partir del ensamblaje de la figura anterior.

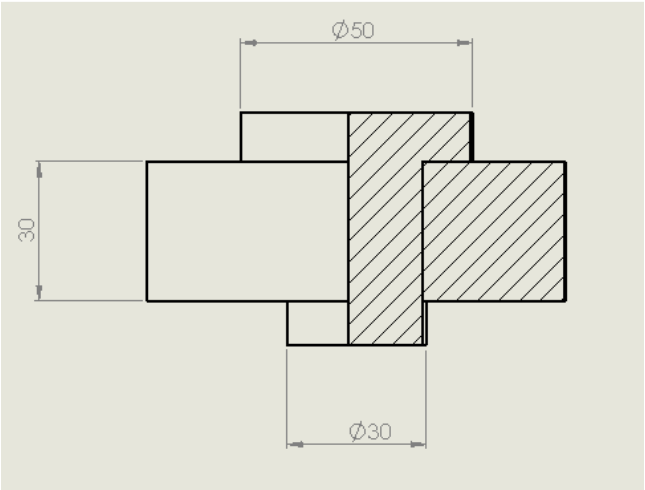


Ilustración 64: Plano del ensamblaje de ejemplo.

Aunque se podría profundizar mucho más en este programa de modelado mecánico, para el presente trabajo serán de utilidad únicamente las funcionalidades que han sido brevemente presentadas en este apartado.

4 Desarrollo del prototipo

Los conceptos básicos explicados en el anterior capítulo serán empleados aquí para resolver el problema objeto del presente trabajo.

Para ello, en primer lugar, es preciso conocer el comportamiento del sensor RGB TCS34725. Así, será posible diseñar un sistema en el que pueda operar en condiciones óptimas.

En segundo lugar, y con la misma finalidad, hay que caracterizar los motores paso a paso de los que se consta. El objetivo es elegir el que mejor se adecúe a los requerimientos del prototipo.

Una vez conocido el comportamiento del sensor RGB y de los motores, es posible diseñar un sistema mecánico que integre ambos elementos para lograr un escáner de bajo coste. Para ello, hay que diseñar o comprar las piezas necesarias, fabricarlas si es necesario y, finalmente, montarlas. Dentro de este apartado hay que tener también en cuenta el espacio y la función que tendrán los elementos electrónicos.

Cuando se halle acabado el sistema mecánico se pueden añadir los componentes electrónicos pertinentes.

Para acabar, hay que diseñar un programa para poder controlar el prototipo. Todos estos pasos serán desarrollados a continuación con más detalle.

4.1 Caracterización del sensor TCS34725

El funcionamiento y las características del sensor RGB TCS34725 fueron presentados en el apartado 3.3.1. Con la finalidad de construir un escáner RGB que pueda integrar este sensor de forma efectiva y eficiente, es preciso conocer el comportamiento del mismo. Para ello, se comprobó la combinación óptima de los parámetros modificables del sensor, la iluminación óptima de la muestra, la reproducibilidad de las medidas, la resolución de las mismas, la distancia y el ángulo entre el sensor y la muestra y la ubicación y la forma del área de detección.

A fin de llevar a cabo las mediciones, se colocó el sensor sobre un montaje de madera de forma que quedase situado horizontalmente. Se tomaron los datos mediante la placa Arduino UNO. El esquema de conexiones se encuentra a continuación. Como se puede observar, la alimentación se proporciona a través del cable USB. El pin 3v3 suministra la energía al sensor, GND es la toma de tierra y los pines SDA y SCL son los necesarios para la comunicación por I2C.

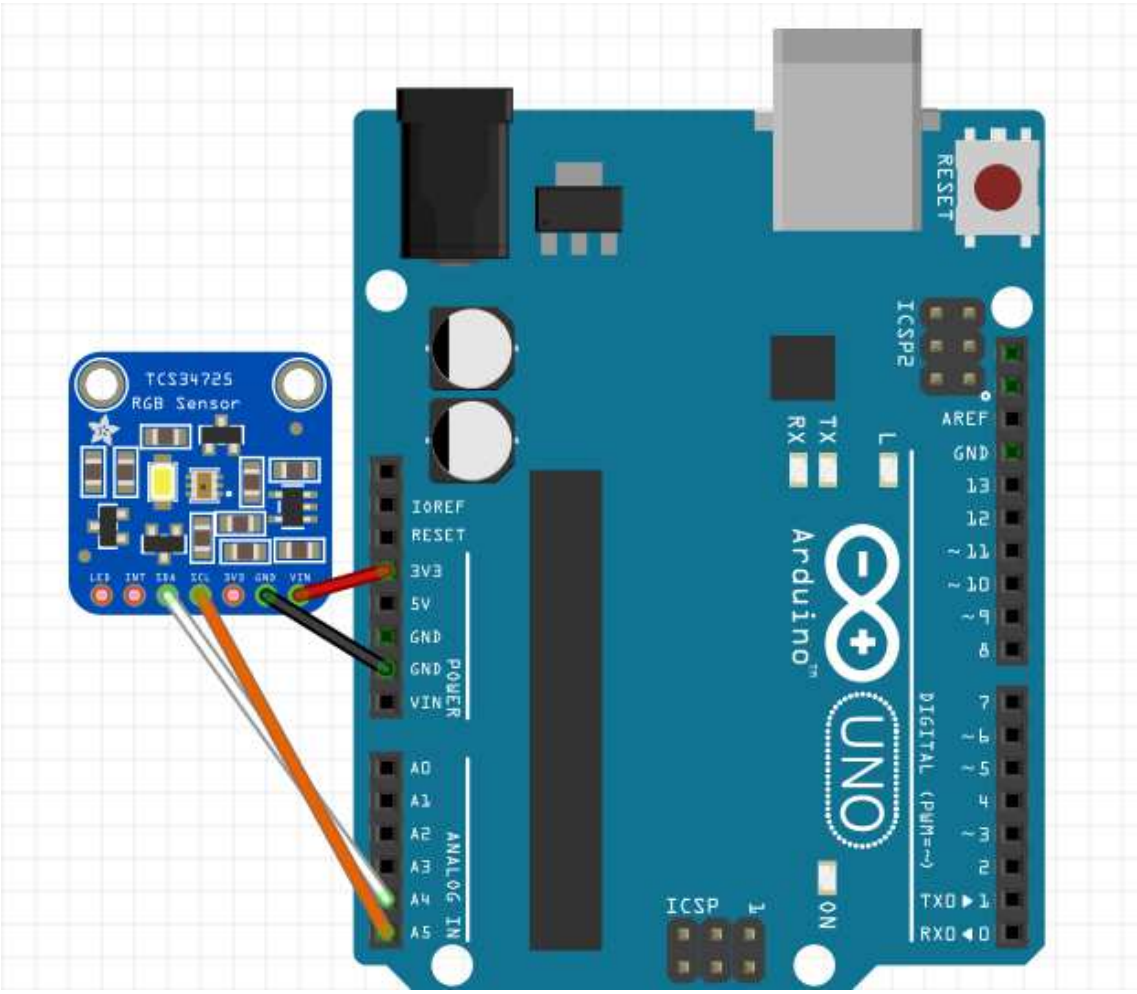


Ilustración 65: Esquema de conexiones para la caracterización del TCS34725.

Una vez visto el esquema de conexiones, se explicará brevemente el funcionamiento del programa de Arduino que fue utilizado para la toma de datos del sensor a partir de su diagrama de bloques, situado a continuación. Como se puede ver en el mismo, en primer lugar se crea el objeto TCS34725 y se definen el tiempo de integración y la ganancia. Además, se inicialización la comunicación con el ordenador, tras lo cual se comprueba se el sensor está conectado.

De ser así, se entra en el bucle principal del programa, en el cual se toman los datos de las coordenadas R, G, B y C (C contiene los datos de todas las coordenadas). Seguidamente, el programa calcula la temperatura de color y la iluminación, muestra todos los resultados por el monitor serial y vuelve a ejecutar el bucle. En el anexo A se encuentra el código completo en Arduino correspondiente este diagrama de bloques.

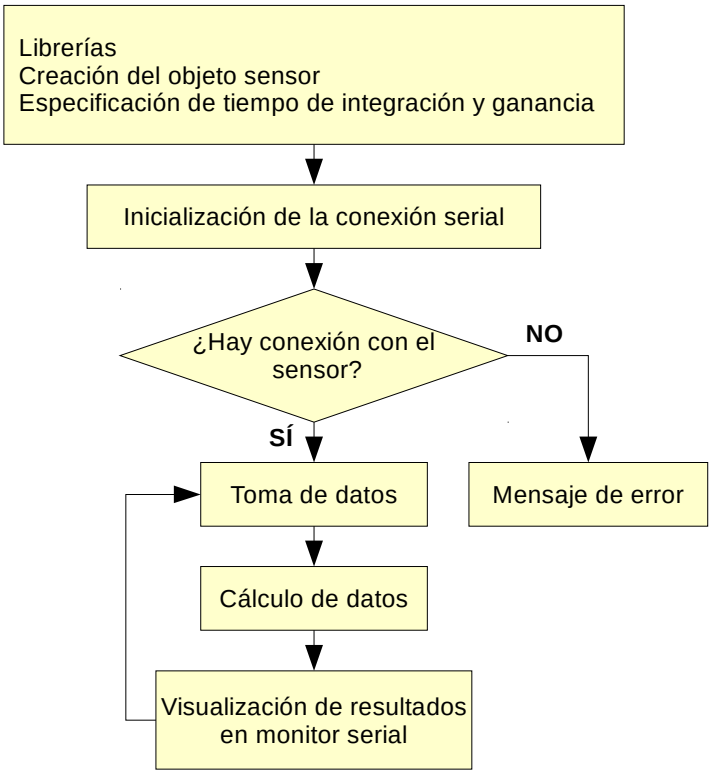


Ilustración 66: Diagrama de bloques del programa empleado para la caracterización del TCS34725.

4.1.1 Tiempo de integración y ganancia

Los parámetros del TCS34725 que es posible modificar desde el software son el tiempo de integración y la ganancia. Como se ha visto en el apartado 3.3.1, el tiempo de integración aumenta la duración de la exposición del sensor a la señal, por lo que es de gran ayuda para medir en entornos de baja iluminación. Asimismo, conviene recordar que la ganancia es un factor adimensional que se multiplica con la señal de salida obtenida. De ello se puede deducir que, a la vez que permite una caracterización más precisa de los resultados, también dilata los errores cometidos en las medidas.

Para encontrar la combinación óptima de estos parámetros, se colocó una muestra blanca sobre el sensor y se cambió el tiempo de integración y la ganancia desde el software. En la siguiente imagen se muestra la línea de código que hay que modificar con la finalidad de alterar ambas variables.

```
/* Initialise with specific int time and gain values */
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_700MS, TCS34725_GAIN_1X);
```

Ilustración 67: Modificación del tiempo de integración y la ganancia desde el software.

Se toma una muestra blanca como referencia dado que, en la escala RGB, las tres coordenadas tienen sus valores máximos para este color. En el caso del TCS34725, que funciona a 16 bits, los mejores resultados serán aquellos que más se acerquen a 65535 sin saturar el sensor. En la tabla 4 se pueden observar algunos de los resultados de las medidas. La estabilidad se define según el comportamiento de las cifras de las medidas. Un fondo de color verde en la tabla significa que la cifra es estable, amarillo significa predeciblemente inestable y rojo representa una cifra muy inestable. Un fondo naranja en la tabla indica una estabilidad absoluta independientemente de la muestra, es decir, el sensor se ha saturado. A partir de la ganancia 4x se obtiene saturación para todos los tiempos de integración, por lo que no se encuentran representados en la tabla.

	Ganancia							Ganancia						
	1x			Estabilidad				4x			Estabilidad			
TI (ms)	R	G	B	m	c	d	u	R	G	B	m	c	d	u
2,4	121	188	167					492	770	676				
24	1208	1885	1672					4891	7671	6729				
50	2538	3960	3512					10267	16111	14129				
101	5200	8112	7197					21017	33001	28928				
154	7741	12080	10720					**31284**	**49115**	**43050**				
700	**30978**	**48328**	**42899**					65535	65535	65535				

Tabla 4: Resultados de la caracterización de los parámetros del sensor.

Tras probar múltiples combinaciones quedan dos opciones: 700 ms de tiempo de integración sin ganancia o 154 ms de tiempo de integración con el factor 4 de ganancia. Puesto que la medición se ha realizado en un entorno con baja iluminación (la muestra tapa el sensor) y se desea minimizar el error en los resultados, la primera opción es la más indicada. La inestabilidad de las decenas y las unidades supone un error máximo del 0.3 %

en las medidas, que es asumible. Así, el sensor requerirá de 700 ms para llevar a cabo una medida.

4.1.2 Iluminación

Para este apartado se contó con una muestra con los colores negro, blanco, rojo, verde y azul. Se realizaron lecturas de cada color en distintas condiciones de iluminación: a oscuras, con el LED del sensor, con luz ambiente e iluminando la muestra desde detrás con el LED del móvil. Para ello se colocó la zona a medir directamente sobre el sensor. A continuación, se puede ver la muestra con la que se realizaron las medidas.

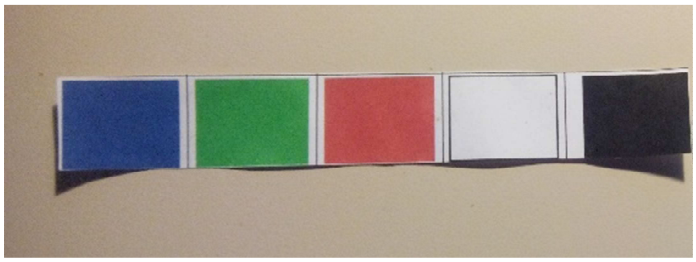


Ilustración 68: Muestra para la caracterización de la iluminación durante las medidas.

Los resultados sin los valores del negro (dado que en este color no se puede valorar el resultado) se encuentran en la siguiente tabla. Un buen resultado significa un valor máximo lo más próximo a 65535 en la coordenada correspondiente. Por ejemplo, el blanco ha de tener valores elevados en sus tres coordenadas y el azul en la de su color.

	Blanco			Rojo			Verde			Azul		
Iluminación	R	G	B	R	G	B	R	G	B	R	G	B
Led	**31110**	**47430**	**42075**	**14790**	**11220**	**10200**	**8415**	**20145**	**12495**	**5610**	**11985**	**15555**
Luz ambiente	4335	3570	2040	2550	510	510	765	1530	510	510	510	510
Oscuridad	0	0	0	0	0	0	0	0	0	0	0	0
Luz del móvil	**46155**	**64770**	**45645**	**17595**	**6120**	**4590**	**7650**	**21675**	**7905**	**3570**	**7650**	**11475**

Tabla 5: Resultados de la caracterización de la iluminación.

Teniendo esto en cuenta, los mejores valores se obtienen para la iluminación con el LED del móvil. Sin embargo, no hay que olvidar que las medidas se han realizado con un folio que permite pasar gran cantidad de luz. Es posible, sin embargo, que en el prototipo se

midan muestras con transparencias variables, en cuyo caso los resultados variarían dependiendo de la anchura y composición de la muestra. Para evitar esta variabilidad, se elige la opción del LED del sensor para la iluminación, ya que sus resultados son también muy buenos y, al realizarse la medida por reflexión, no hay dependencia de la transparencia de la muestra.

4.1.3 Reproducibilidad

En todo sistema de medidas es de vital importancia la reproducibilidad de los datos obtenidos. Para caracterizar la reproducibilidad del TCS34725 se generaron 12 muestras de distintos colores por ordenador y se imprimieron en folios convencionales. En nueve de ellas, el color cubrirá completamente el sensor. El las otras tres, se imprime un punto de 3 mm de diámetro en rojo, verde y azul, respectivamente. Para las medidas, dicho punto se colocará de la forma más precisa posible sobre el array de fotodiodos.

Se realizaron 20 medidas por muestra. Se llevaron a cabo en distintos días (una medida por día) y reiniciando el programa de Arduino. En la tabla 6 se muestran los resultados. Para el cálculo de la desviación relativa media se eliminó la muestra 5, dado que se arrugó durante el transcurso de las medidas, lo que falseó los resultados. Las muestras 7, 8 y 9 se valoran aparte, dado que son distintas (punto de 3 mm). En general, la desviación relativa se encuentra en torno al 1.3 % como máximo. Esto cambia en las muestras 7, 8 y 9, donde las pequeñas variaciones en la colocación de una medida a otra, debidas a la preparación manual, conllevan un aumento de la desviación relativa hasta el 2.2 %. En ambos casos se puede calificar los resultados como muy buenos.

4.1.4 Resolución

Otro punto a tener en cuenta es hasta dónde llega la diferenciación de colores por parte del sensor. Para ello, existe la limitación de la generación de muestras, para la cual se cuenta únicamente con la escala RGB de 8 bits (0-255). Además, la calidad de la muestras también depende de la de la impresora y la tinta empleadas.

	Desviación relativa (%)		
	R	G	B
Muestra 1	1.82126	0.77239	0
Muestra 2	2.0712	1.33677	1.43297
Muestra 3	1.37782	1.60865	1.83327
Muestra 4	1.20888	1.51131	1.46042
Muestra 5	2.03877	4.02788	2.96086
Muestra 6	1.84558	1.49151	1.83084
Muestra 7	0.96502	2.74595	2.32668
Muestra 8	2.8102	2.12315	1.55948
Muestra 9	2.07413	1.79477	2.24959
Muestra 10	0.71036	0.70764	0.81234
Muestra 11	0.75357	0.81482	0.79389
Muestra 12	0.64274	0.80135	0.66014
Media 7.8,9	1.94979	2.22129	2.04525
Media resto	1.30393	1.13055	1.10298

Tabla 6: Resultados de la caracterización de la reproducibilidad del TCS34725.

De todas formas, se hizo una serie de 26 muestras (de 0 a 250 en la escala RGB) por ordenador que se diferenciaban en 10 puntos en las tres coordenadas RGB respecto a la siguiente (véase la ilustración 69).

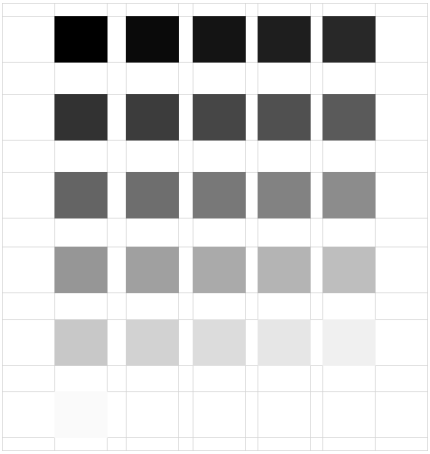


Ilustración 69: Patrón de muestras para comprobar la resolución.

Los resultados de las medidas realizadas se muestran en la siguiente gráfica. Se puede observar que existe una correcta diferenciación de todos los puntos, si bien esta empeora un poco en los extremos. Asimismo, es posible realizar una regresión lineal que describa

la relación de lo diseñado por ordenador frente a lo medido con el sensor. De estos resultados se puede deducir que existe una resolución de 10 puntos en la escala RGB de 8 bits, es decir, del 4.4 %.

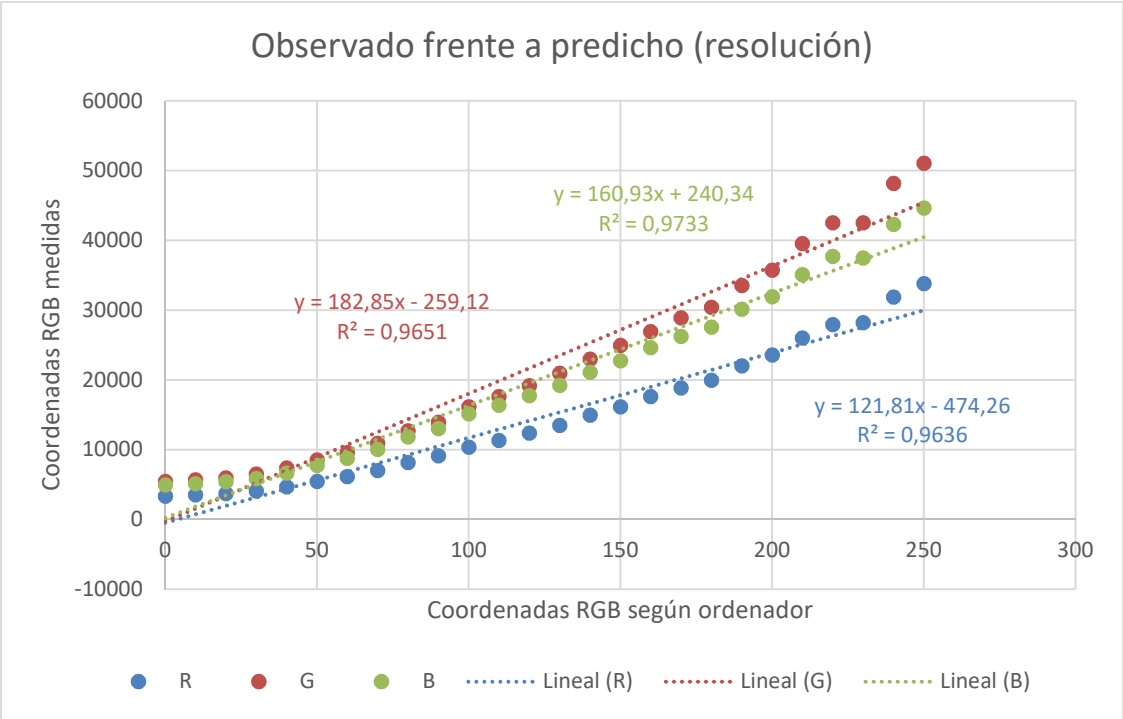


Ilustración 70: Gráfica de lo predicho frente a lo observado para el patrón de muestras para la caracterización de la resolución del TCS34725.

La siguiente pregunta es: ¿hay una resolución mayor? Para averiguarlo, se llevó a cabo un segundo patrón de muestras. Éste consta de 11 zonas que van del 100 al 110 en las tres coordenadas de la escala RGB, es decir, cada muestra es un punto mayor a la anterior en todas sus coordenadas (véase la siguiente imagen).

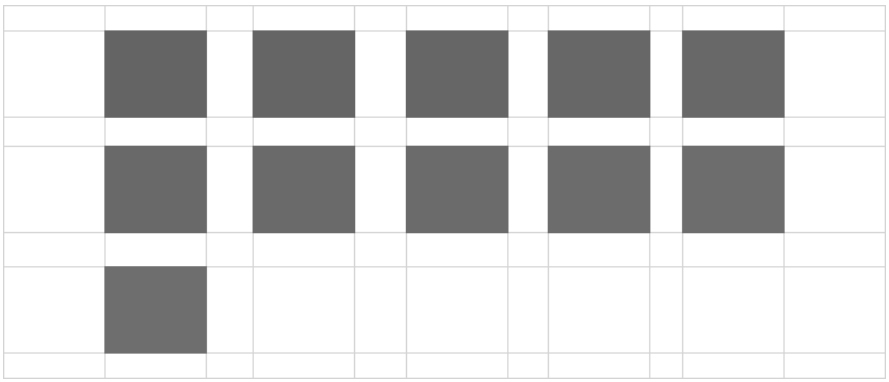


Ilustración 71: Segundo patrón de muestras para la caracterización de la resolución.

En la gráfica con los resultados, que se muestra a continuación, se puede observar que hay una clara diferenciación de todas las muestras. Es más, al encontrarse en la zona óptima de diferenciación según las primeras medidas, la relación de linealidad entre los predicho y lo medido es mucho más clara. Esto se refleja en el coeficiente de determinación, que ronda 0.99 para las tres coordenadas.

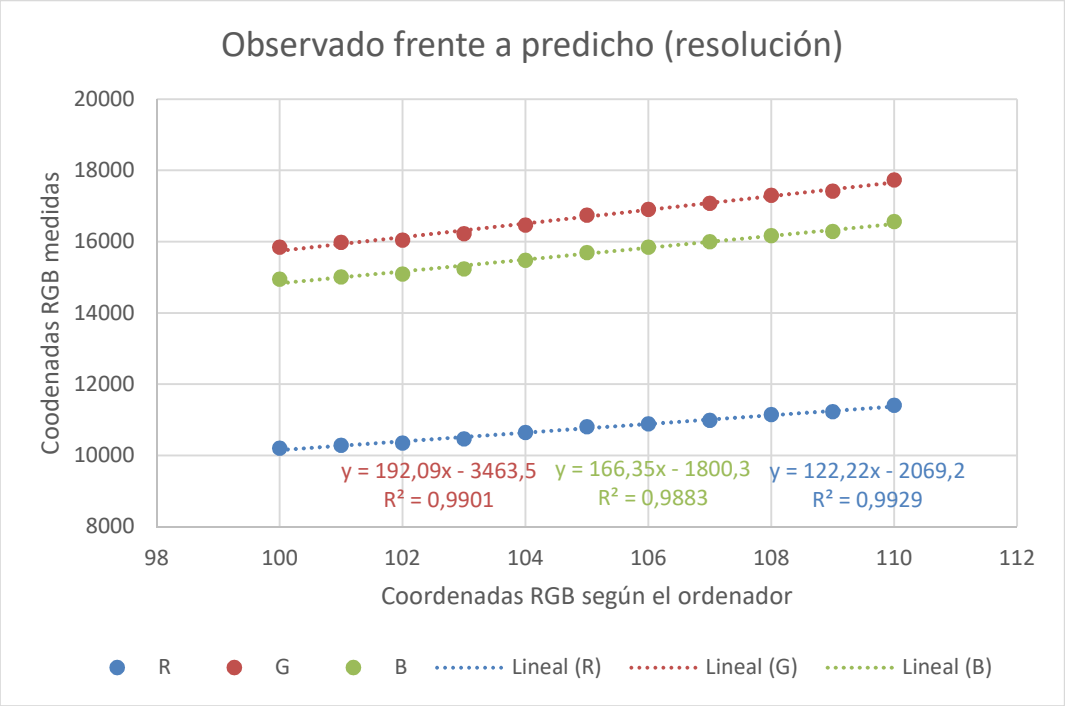


Ilustración 72: Gráfica de lo predicho frente a lo observado para el segundo patrón de muestras para la caracterización de la resolución del TCS34725.

Por lo tanto, se puede afirmar que existe una resolución de un punto en la escala RGB de 8 bits, es decir, el TCS34725 muestra una resolución de aproximadamente 0.45 %. Es posible que la resolución sea mayor, sin embargo, no se consta de los medios necesarios para comprobarlo. Además, la resolución obtenida es más que suficiente para el presente trabajo.

4.1.5 Distancia entre el sensor y la muestra

Para las mediciones anteriores se colocó la muestra directamente sobre el sensor. En este apartado se pretende averiguar la distancia óptima del sensor respecto al objeto que se pretende analizar. Con esa finalidad, se colocó una muestra de color blanco a distintas distancias del sensor. La óptima será aquella en la que los valores de las coordenadas

RGB sean máximos. En la siguiente imagen se muestra el montaje con el que se efectuaron las medidas.

Ilustración 73: Montaje para la averiguación de la distancia óptima entre el sensor y la muestra.

Los mejores resultados se obtienen a 3 y 4 mm de distancia. En la ilustración 74 se observa que este parámetro depende de la colocación del LED en el sensor, puesto que si la muestra se aleja más de la cuenta recibe demasiada luz ambiente y si se acerca demasiado bloquea la luz del LED, impidiendo que la luz reflejada llegue al array de fotodiodos. Nótese la mayor sensibilidad del sensor al verde que al azul y al rojo.

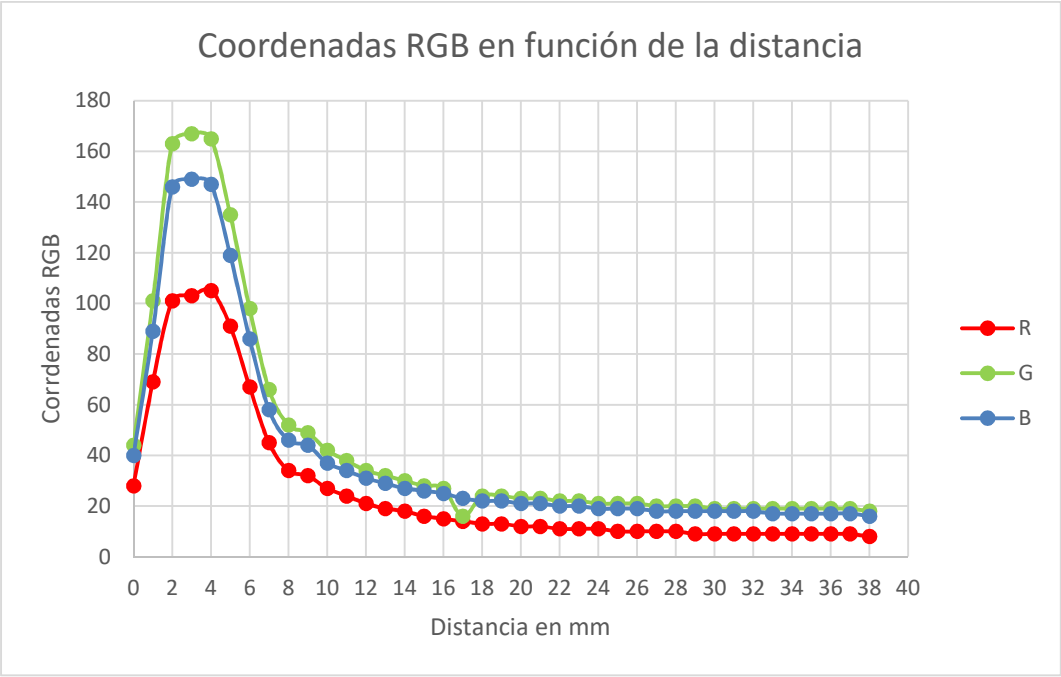


Ilustración 74: Coordenadas RGB para una muestra blanca en función de la distancia al sensor.

4.1.6 Ángulo del sensor respecto a la muestra

En el datasheet del TCS34725 se indica que la muestra ha de encontrarse paralela al sensor para obtener resultados óptimos en las medidas. En este apartado se realizó la comprobación a esa afirmación. Para ello, se utilizó el montaje que se muestra a continuación con una muestra blanca. De nuevo, los valores máximos son los mejores resultados.

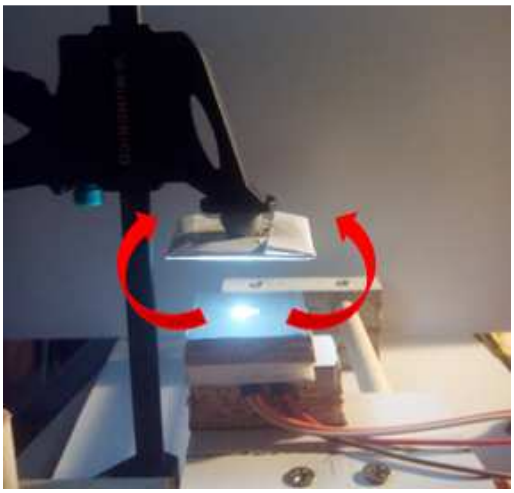


Ilustración 75: Montaje para la caracterización del ángulo de lectura.

Los resultados permiten llegar a la conclusión de que, como era de esperar, la muestra debe encontrarse paralela respecto al sensor para obtener mejores lecturas (véase la ilustración 76).

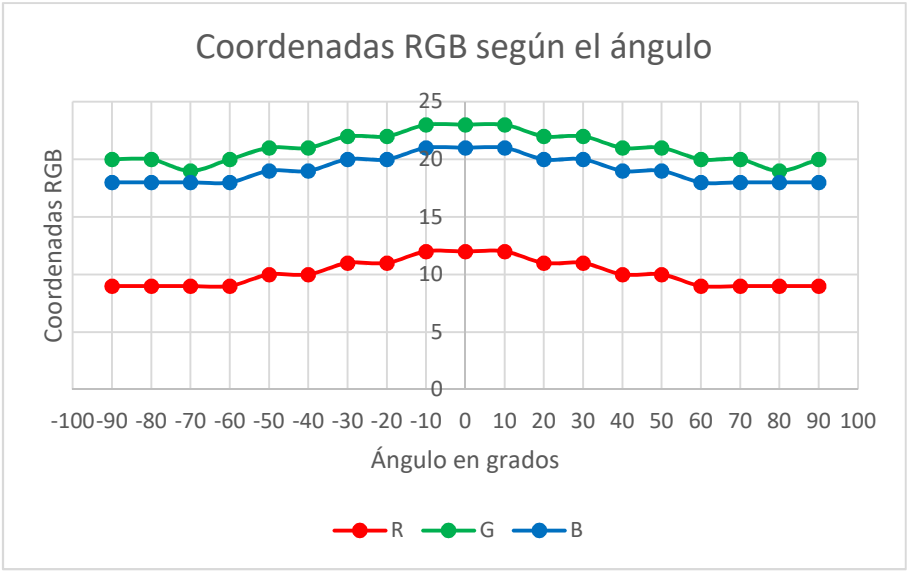


Ilustración 76: Resultados de la caracterización del ángulo óptimo de la muestra respecto al sensor.

4.1.1 Caracterización del área de detección

Para caracterizar el tamaño y la ubicación del área de detección se llevaron a cabo mediciones con muestras con puntos negros de distinto diámetro sobre un fondo blanco. El punto se mueve sobre el sensor hasta que se obtiene un valor mínimo en la lectura. Así, se sabe que el punto ha sido percibido en su totalidad por el sensor, dado que en la escala RGB el valor del negro es cero en las tres coordenadas.

Se elaboró un patrón de muestras de diez círculos negros de 1 mm a 10 mm de diámetro (ilustración 77).

Ilustración 77: Patrón de muestras para la caracterización del área de detección.

En la primera tanda de medidas se trabajó con el sensor al aire. Los resultados se muestran en la gráfica situada a continuación. Se puede observar como a partir de los 7 mm el área de detección está completamente tapada por la muestra. Para los puntos de menor diámetro, el aumento en los valores RGB significa que se está incorporando cierta cantidad de fondo al resultado. Cuanto mayor es el valor RGB, mayor es también la cantidad de fondo incorporada. La detección tiene lugar de forma aceptable desde los 10 mm a los 4 mm. De esta gráfica se podría deducir un área de detección de 7x7 mm.

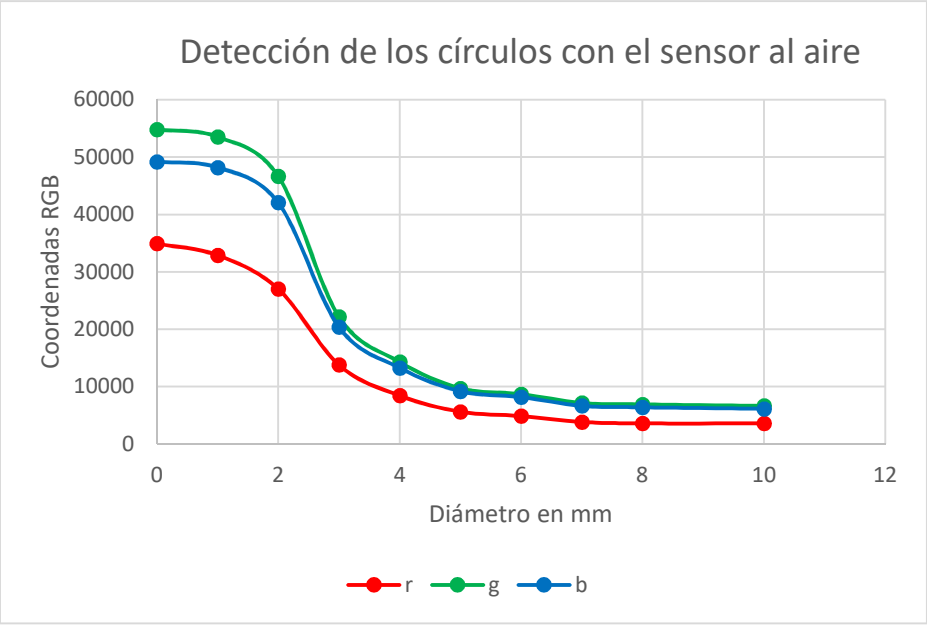


Ilustración 78: Coordenadas RGB en función del diámetro de la muestra.

Teniendo en cuenta la posición del LED y el array de fotodiodos en el sensor, que ocupan un rectángulo de 7 mm de anchura y 3 mm de altura, se asume que el área de detección se encuentra caracterizada por dicho rectángulo. Para comprobarlo, se repitieron las medidas del patrón de círculos cubriendo el sensor con una ventana negra con una abertura de 7x3 mm sobre la supuesta área de detección. Los resultados fueron idénticos a los de la gráfica anterior, con lo que se probó la suposición sobre el tamaño y la ubicación de la zona de medición. En la siguiente imagen se muestra el área de detección del TCS34725 marcada con un rectángulo rojo. Con un rectángulo verde se señala la zona en la que se obtuvieron los mejores valores para diámetros inferiores a 4 mm.

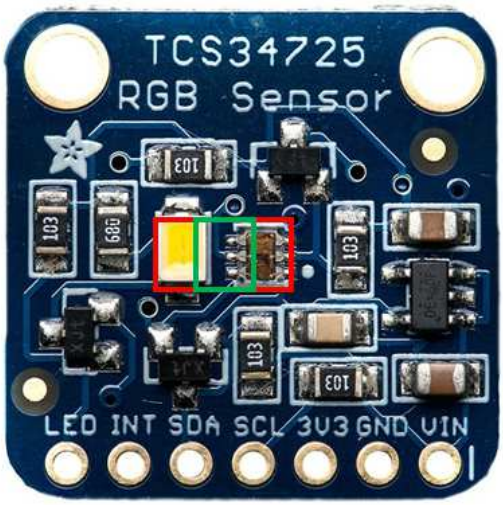


Ilustración 79: Área de detección (rojo) y zona de detección de alta precisión (verde).

La pregunta que cabe resolver es si el área de detección no se limitará solamente al rectángulo verde. Para resolver esta duda se prepararon tres ventanas con cartulina negra, una con una abertura de 6x3 mm, otra con una de 3x3 mm y una última con una de 2x2 mm (véase la siguiente imagen).

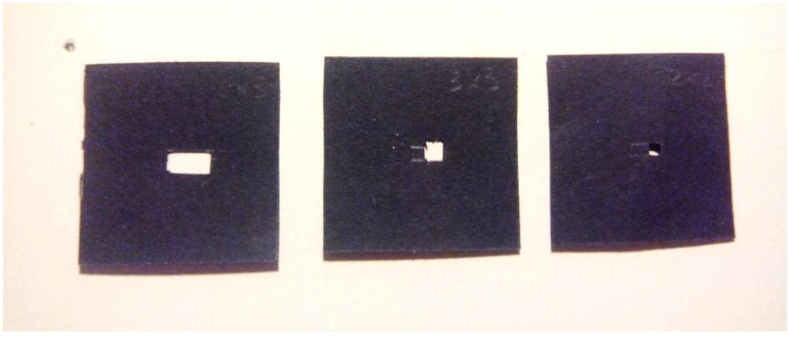


Ilustración 80: Ventana de 6x3 mm, de 3x3 mm y de 2x2 mm (de izquierda a derecha).

A continuación, se repitieron las medidas con el patrón de círculos cubriendo el sensor con cada una de estas ventanas.

En la siguiente gráfica, que se corresponde con los resultados de la ventana de 6x3 mm, se puede observar que la estabilización de los valores RGB debida a que la zona de detección se halla completamente cubierta tiene lugar para un diámetro inferior al de la gráfica correspondiente al sensor al aire (6 mm). Esto indica que se ha reducido el tamaño de la zona de detección. Además, los valores obtenidos son en algo menores antes de la estabilización, lo que se debe a que se ha reducido levemente la iluminación por medio de la ventana.

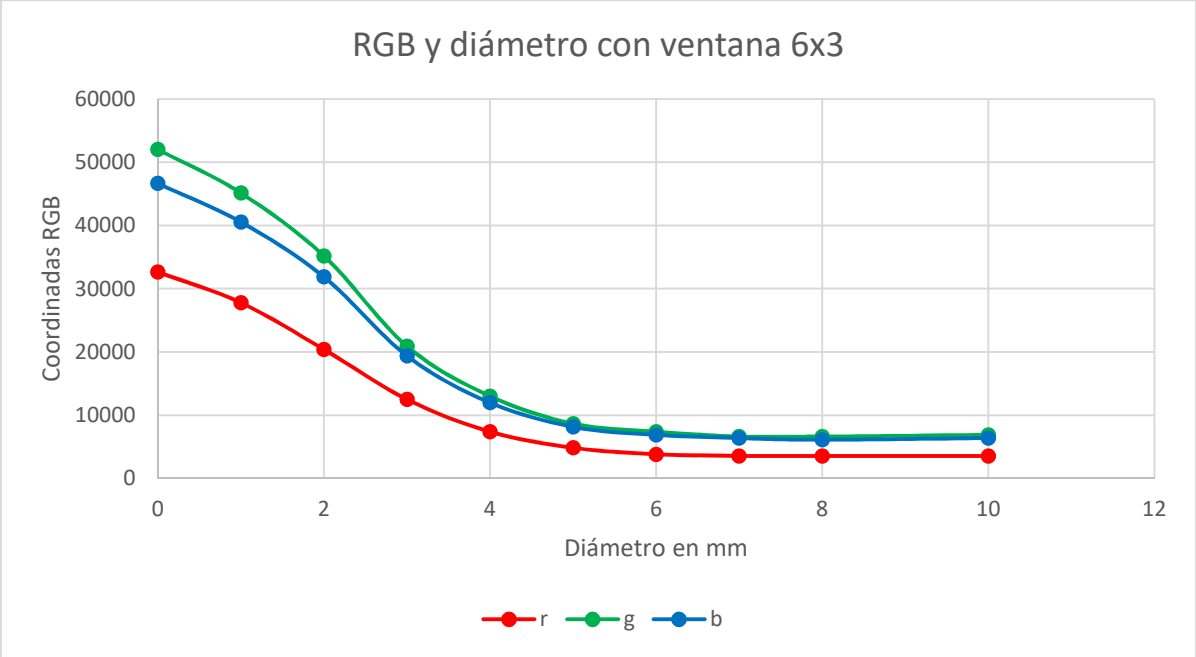


Ilustración 81: Resultados del patrón de círculos con la ventana de 6x3.

De estos resultados se puede deducir que el área de detección queda correctamente caracterizada por el rectángulo rojo de 7x3 mm visto anteriormente. Sin embargo, para asegurarse, se revisarán los resultados de la ventana de 3x3 mm.

A continuación se encuentra la gráfica de la ventana de 3x3 mm. La estabilización de las medidas tiene lugar mucho antes, dado que se incorpora una menor cantidad de fondo a los resultados. Sin embargo, la disminución de la iluminación debida a la reducción de la ranura conlleva una importante pérdida de resolución. La diferencia entre el fondo blanco (punto de 0 mm) y el negro disminuye notablemente. En el sensor al aire el fondo blanco

tiene un valor de verde de 54825, mientras que en la ventana de 3x3 mm este valor se ve reducido a 34935.

Por lo tanto, queda demostrado que el área de detección queda descrita por el rectángulo de 7x3 mm visto anteriormente. En esas condiciones se da la combinación óptima de iluminación de la muestra y recepción de información por parte del array de fotodiodos.

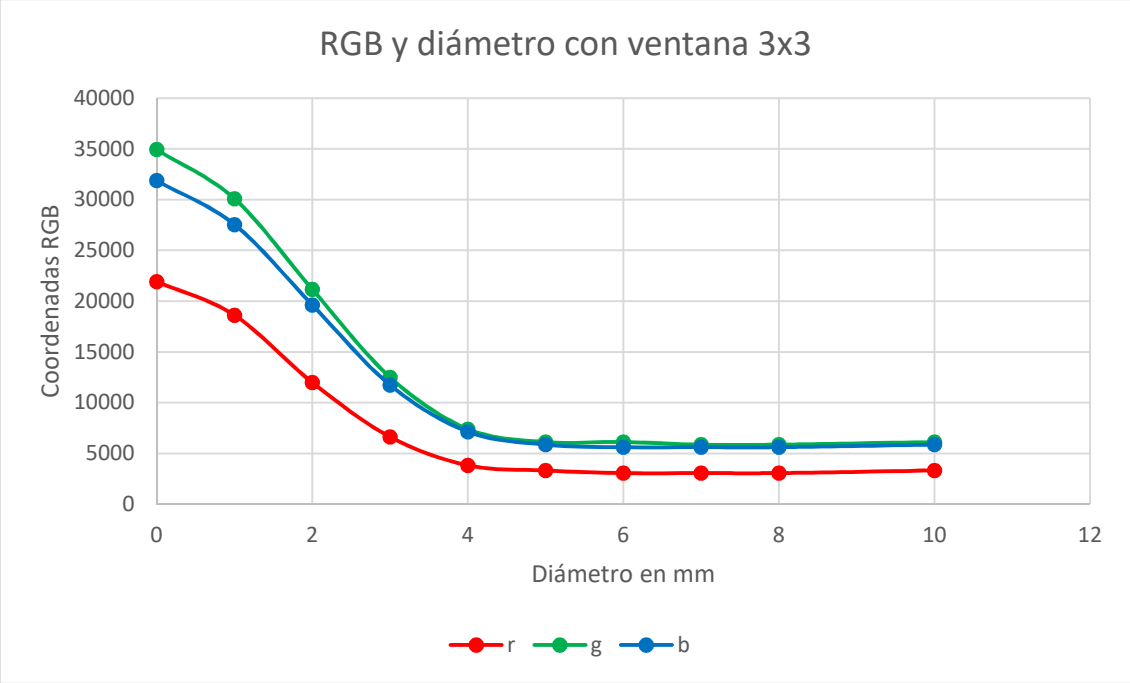


Ilustración 82: Resultados de la ventana de 3x3 mm.

Así, una vez caracterizado el sensor RGB TCS34725, se puede pasar a buscar el motor que mejor se adecúe a las necesidades del prototipo que se pretende desarrollar en el presente trabajo.

4.2 Caracterización de los motores paso a paso

Ya sabidas las características y las condiciones óptimas de trabajo del sensor RGB, es hora de hacer lo mismo con los motores de los que se dispone. Además, habrá que elegir aquél cuyas características mejor sirvan a los intereses del presente trabajo.

Para ello, el primer paso es revisar las características generales de los motores, brevemente explicadas en el apartado 3.3.3. Las más interesantes a la hora de llevar a cabo un descarte son los pasos por vuelta, el par motor, el voltaje y la corriente (véase la siguiente tabla).

Modelo	**28BYJ-48** 	**ST-PM35-15-11C** 	**SM-42BYG011-25** 
Pasos por vuelta	4096	48	208
Par de torsión en tracción	300 g/cm	100 g/cm	2.3 kg/cm
Voltaje	5 V	12 V	12 V
Corriente	55 mA	400 mA	330 mA

Tabla 7: Repetición de algunas de las propiedades más importantes de los motores disponibles.

El primer modelo muestra un elevadísimo número de pasos por vuelta en comparación con los otros. Ello se debe a la reducción 1/64 con la que cuenta y que no se encontró la manera de eliminar. Por ello, el 28BYJ-48 funcionará a velocidades muy lentas, puesto que con cada impulso eléctrico sólo se avanzará un paso. Teniendo esto en cuenta, es posible descartar este modelo como el adecuado para el presente trabajo.

Así, quedan el ST-PM35-15-11C y el SM-42BYG011-25. Ambos funcionan en condiciones similares de voltaje y de corriente. La diferencia radica en la fuerza, que es mucho mayor en el segundo modelo. Además, el ST-PM35-15-11C, con un número de pasos por vuelta muy bajo, es adecuado para funcionar a velocidades altas. Otro contraste se encuentra en el tamaño: el SM-42BYG011-25 es bastante más voluminoso. A primera vista, sin embargo, ambos parecen adecuados para el prototipo que se pretende diseñar, por lo que, para poder decidirse, habrá que probarlos en acción.

Con tal fin, se conectaron ambos motores según la ilustración 83.

Como se puede observar, esta vez no basta con la alimentación del Arduino para hacer funcionar el sistema. Para suministrar 12 V al motor, se puede optar por una batería o conectar una fuente de alimentación con salida de 12 V a la red eléctrica.

Una vez realizado el montaje del sistema para caracterizar el sistema, se escribieron dos programas en Arduino para comprobar el funcionamiento de los motores. El primer programa hace girar el motor en un sentido de forma continuada. El segundo programa simula la realización de medidas. Para ello, gira durante unos segundos, para durante un segundo (el sensor requiere de 700 ms para realizar una medida) y repite la operación hasta que se interrumpa la alimentación.

En la ilustración 84 se muestra el diagrama de bloques del primer programa. Al inicio del mismo se definen las librerías que se necesitarán. Se eligió la librería “Accelstepper”, dado que incorpora muchas más funciones para hacer funcionar los motores paso a paso que la librería “Stepper” de Arduino, funciones que encontrarán aplicación en el desarrollo del prototipo. A continuación, se crea el objeto Accelstepper y se definen el número de conexiones y sus pines correspondientes.

Seguidamente, se define la velocidad en pasos por segundo. La documentación de la librería “Accelstepper” recomienda una velocidad máxima de 1000 pasos por segundo.

En el bucle principal se hace girar el motor mediante el comando adecuado. Se puede encontrar el programa completo y explicado en el anexo B.

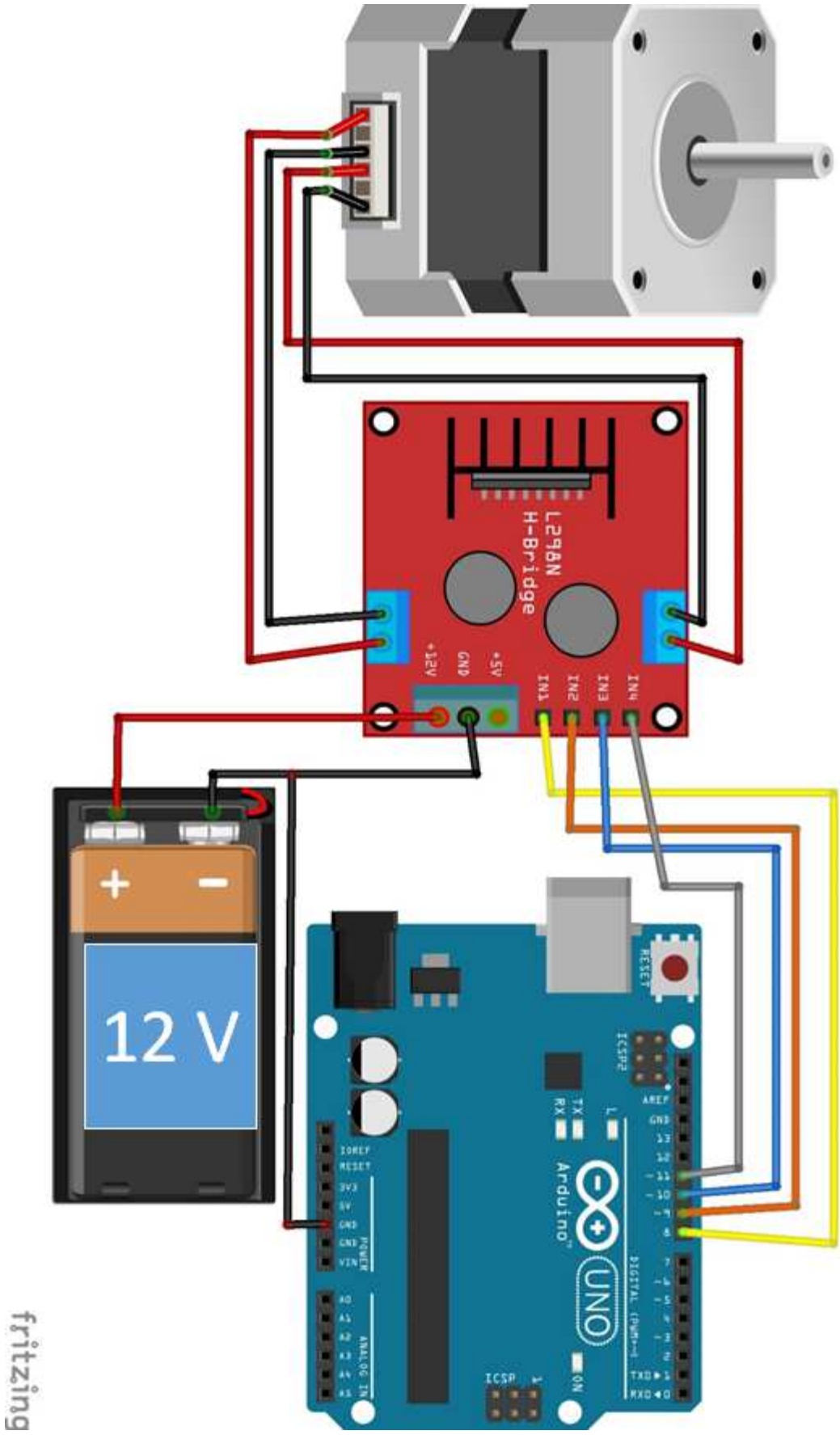


Ilustración 83: Esquema de conexiones para la caracterización del motor.

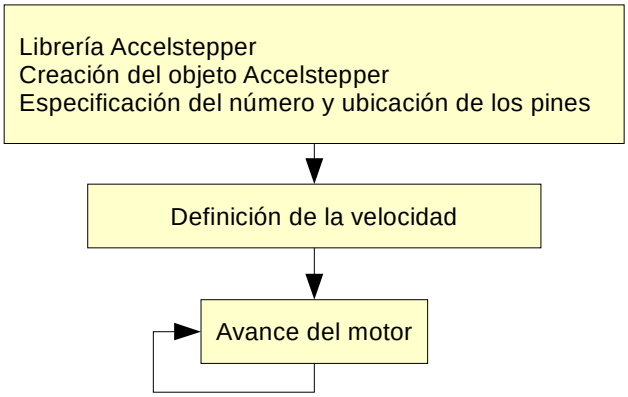


Ilustración 84: Diagrama de bloques del programa para hacer avanzar los motores paso a paso.

Para el segundo programa, cuyo diagrama de bloques se encuentra en la ilustración 85, se siguen los mismos pasos al inicio, es decir, se incluye la librería “Accelstepper”, se crea el objeto “Accelstepper”, se definen el número de conexiones, sus pines y la velocidad del motor. Lo que varía es el bucle principal, en el que, cada cierto número de pasos, se mantiene la posición durante un segundo para simular la toma de datos.

Con el primer programa ambos motores funcionan muy bien a todas las velocidades. Se mantienen en funcionamiento sin que surjan problemas.

Sin embargo, durante la ejecución del segundo programa se observó cierta tendencia del modelo ST-PM35-15-11C a sobrecalentarse. Se trató de remediar dicho comportamiento desde el software, pero sin éxito. En cambio, el modelo SM-42BYG011-25 apenas se calentó durante la ejecución del segundo programa.

Hay que recordar que, como se mencionó en el apartado 3.3.3, mantener un motor paso a paso en una posición supone provocar un cortocircuito si la resistencia de las bobinas es baja, como es el caso en el motor ST-PM35-15-11C. Con ello, quedaría explicado el sobrecalentamiento del motor y este modelo descartado para el prototipo.

Para llevar a cabo la caracterización de forma completa, se midió la temperatura de ambos motores durante la ejecución del segundo programa mediante un termopar. Mientras que el SM-42BYG011-25 mantiene su temperatura constante, el ST-PM35-15-11C alcanza los 70ºC en menos de cinco minutos, lo cual no sólo daña el motor (se recuerda que hay un imán permanente en el rotor) sino también podría perjudicar los circuitos adyacentes (ilustración 86).

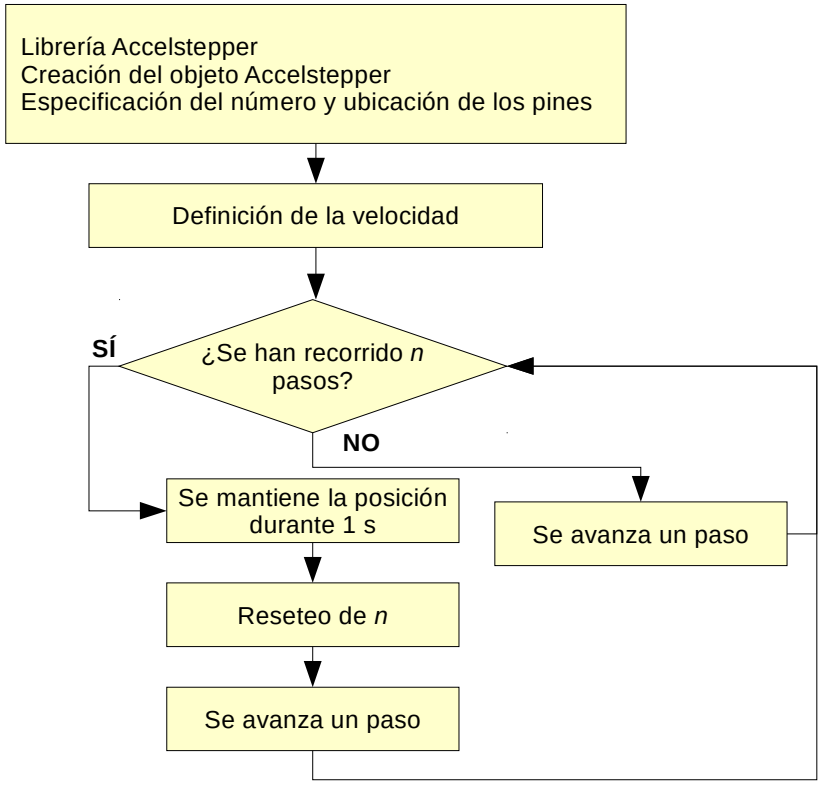


Ilustración 85: Programa para simular la toma de datos parando el motor cada cierto número de pasos.

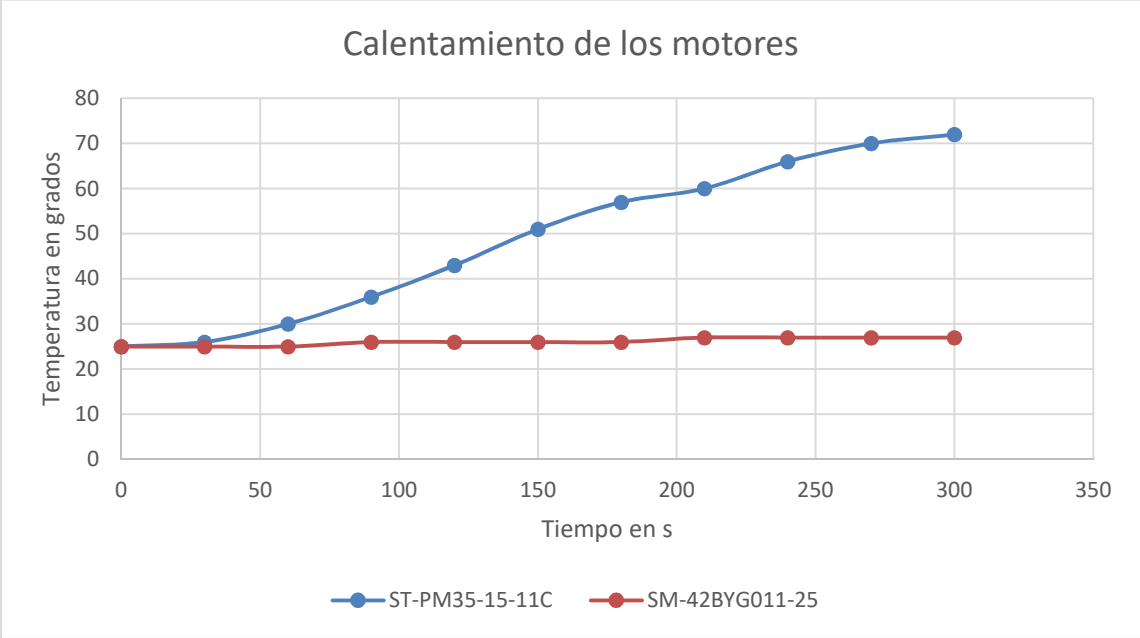


Ilustración 86: Calentamiento de los motores durante la ejecución del segundo programa.

Por lo tanto, de la caracterización de los motores que fueron proporcionados se extrae la conclusión de que el indicado para el presente trabajo es el modelo SM-42BYG011-25.

4.3 Sistema mecánico y electrónico

Una vez conocidas las condiciones de trabajo y las características del sensor RGB y el motor paso a paso escogido, hay que diseñar y fabricar un sistema mecánico y electrónico que emplee ambas componentes para el análisis de muestras químicas. Para ello, en primer lugar, hay que realizar un diseño conceptual de lo que se pretende lograr. En segundo lugar, hay que modelar, fabricar o comprar las piezas necesarias para construir el diseño. En tercer lugar, se ha de montar el sistema mecánico. Una vez realizados estos pasos se puede pasar a la integración del sistema electrónico en el diseño, que, no obstante, conviene planificar de antemano. En este apartado se explicará con detalle el proceso seguido para llegar a un sistema electromecánico funcional para el prototipo final.

4.3.1 Diseño conceptual

En esta fase del desarrollo de un producto se han de identificar los objetivos y requerimientos del sistema a diseñar. Se sientan las bases de cómo va a ser el producto, los materiales que se utilizarán para su fabricación, el concepto de los mecanismos que lo hacen funcionar y cómo lo va a utilizar el usuario.

El objetivo principal es fabricar un producto de bajo coste para el análisis de muestras de índole química. Aparte del coste de producción, es importante también que el sistema sea flexible, es decir, que admita distintos tipos de aplicaciones. Como punto de partida, se sabe que ha de emplearse el sensor RGB TCS34725. Asimismo, la lectura de la muestra ha de ser preferentemente lineal. También es importante que el resultado final sea fácil de usar.

Se buscó una opción que cumpliese todos esos requerimientos fijándose en los productos existentes en el mercado (véase el capítulo 2). Se llegó a la conclusión de que podría ser interesante solucionar el problema basándose en un escáner comercial. El resultado final consistiría en una caja con conexión con el ordenador. El usuario podría levantar la tapa y depositar la muestra sobre un cristal destinado a tal fin. Bajo ese cristal y una vez cerrada la tapa, el sensor se movería linealmente, tomando los datos de la muestra. Para manejar

el “escáner”, se contaría con un software especializado, que habría que diseñar en el presente trabajo. A continuación, se muestra un esquema en el que se puede apreciar la idea.

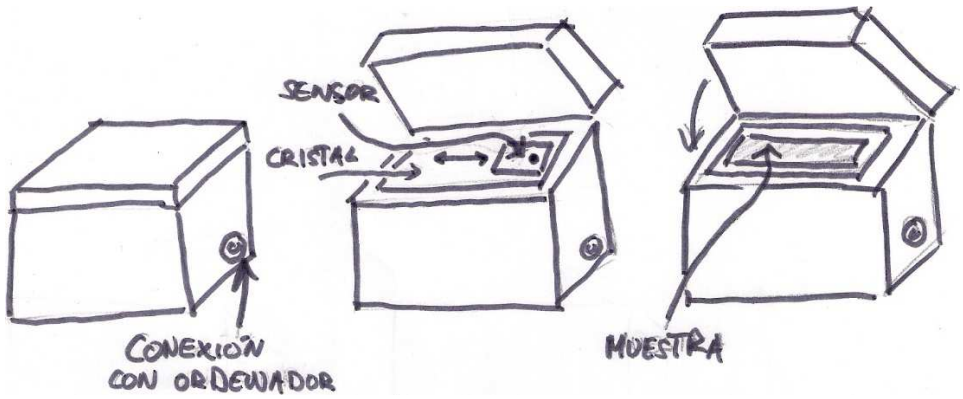


Ilustración 87: Concepto general del producto a desarrollar.

Uno de los primeros problemas que hay que solucionar consiste en transformar la rotación del eje del motor en un movimiento lineal para el sensor. Hay que tener en cuenta que las muestras a medir tendrán una longitud máxima de 5 cm. En la literatura se encontraron múltiples opciones: la leva, el piñón-cremallera, las correas, el mecanismo tornillo-tuerca y el sistema biela-manivela.

Una leva es un elemento de forma no circular que va unido al eje de un motor. Dado su forma, al girar empuja un mecanismo colocado en su “diámetro”, llamado seguidor. Es éste el que experimenta una traslación lineal a causa de la rotación de la leva (véase la siguiente imagen). Este sistema se descarta, puesto que para la traslación del motor se requeriría una leva demasiado grande. Además, el movimiento lineal generado no es completamente uniforme y hay demasiada vibración debido a la rueda loca.

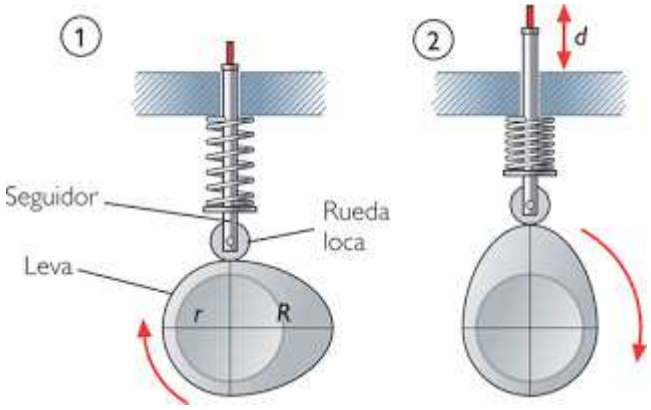


Ilustración 88: Funcionamiento de una leva.

El piñón-cremallera consiste en una rueda dentada (piñón) que gira sobre un mecanismo lineal dentado, haciendo que éste se desplace (véase la siguiente imagen). Parece una buena opción para el prototipo, sin embargo, no se encontraron en el mercado las piezas adecuadas. Se podrían fabricar las piezas, no obstante, al tratarse de un sistema tan sensible y del que depende el funcionamiento del prototipo, se evaluaron otras opciones antes de decantarse por esta. También sería posible encargar las piezas, pero el coste sería elevado, lo que contradice el concepto de bajo coste del producto que se pretende desarrollar.

Ilustración 89: Funcionamiento del mecanismo piñón-cremallera.

También es posible enganchar una correa al eje y a un rodamiento y obtener así un movimiento lineal, como en una cinta transportadora. Esta opción se descarta porque ya existen múltiples ejemplos en el mercado. Además, la fabricación de las piezas necesarias para la sujeción del sensor a la correa podría dar problemas. Por otra parte, la precisión que permiten las correas es algo inferior a la deseada. Los sistemas de actuadores lineales con correas suelen ser empleados en la fotografía, por lo que tienen un tamaño demasiado grande como para resultar de interés para el presente trabajo. En la ilustración 90, se observa un actuador de ese tipo de la empresa DiMotion (correa de color azul claro).

Ilustración 90: Actuador lineal de DiMotion.

El mecanismo biela-manivela consta de dos piezas articuladas entre sí, como se observa en la siguiente imagen. Si la biela va acoplada a una guía, permite transformar el giro de la manivela en un desplazamiento lineal de esa pieza. Este sistema es óptimo para motores, sin embargo, en el prototipo se requeriría demasiado espacio. Además, dadas las reducidas dimensiones que se pretende que tenga el mismo, no existen piezas en el mercado de ese tamaño, por lo que habría que fabricarlas o pagar un precio elevado para su producción

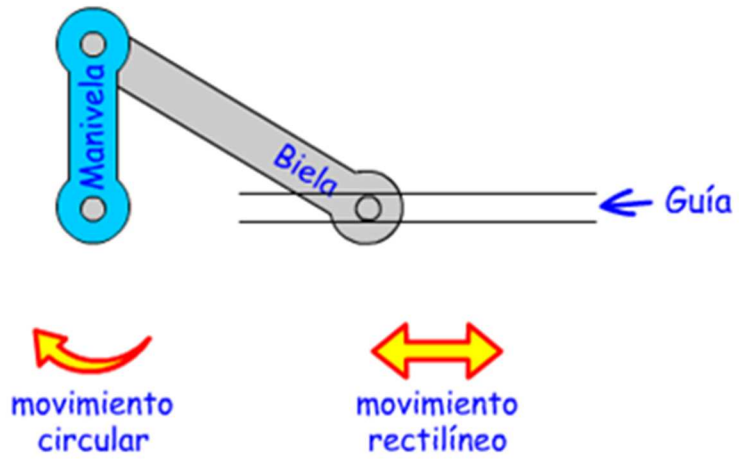


Ilustración 91: Funcionamiento del sistema biela-manivela.

Finalmente, está la opción del sistema tornillo-tuerca. Un tornillo consiste principalmente en una varilla roscada y una tuerca es una pieza con un agujero roscado. Al girar el tornillo manteniendo fija la tuerca tiene lugar un desplazamiento lineal del tornillo. A continuación, se muestra el funcionamiento del mecanismo tornillo-tuerca.

Ilustración 92: Mecanismo tornillo-tuerca. Por vuelta se avanza una distancia equivalente al paso del tornillo.

Dado que el sistema tornillo-tuerca es ampliamente utilizado para la unión entre dos o más piezas, existe en el mercado una gran variedad de estos productos a bajo coste. Además, permite tener un control preciso del movimiento del sensor, puesto que cada vuelta del motor significará un desplazamiento equivalente al paso del tornillo. Por estas razones, para el desarrollo del prototipo se eligió este mecanismo para la transformación de la rotación del motor en la traslación lineal del sensor.

Una vez solucionado el problema de la transformación de movimiento, hay que pensar en cómo implementarla en el prototipo.

Como tornillo se tomará una barra roscada, que es fácil encontrar en cualquier ferretería. Dicha barra roscada se acoplará al eje del motor. Para ello, será necesario construir una pieza que transmita la rotación del motor a la barra roscada y la sujete, como se muestra en el siguiente esquema.

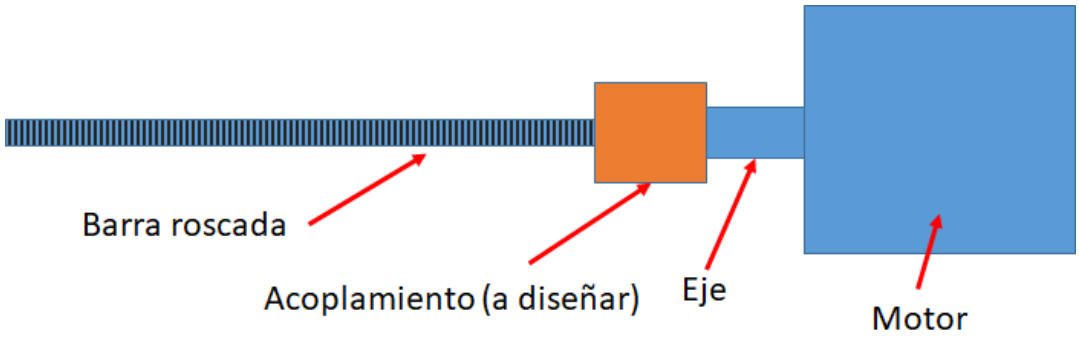


Ilustración 93: Esquema de la transmisión del giro del motor a la barra roscada.

Se pasa ahora a la tuerca. Con la finalidad de que se produzca un desplazamiento lineal de la misma, hay que bloquear su giro. Una opción consiste en fabricar una plataforma en cuyos lados se puedan incrustar dos tuercas. El bloqueo de giro se llevará a cabo mediante dos guías que se colocarán a la izquierda y a la derecha de la tuerca, respectivamente. Sobre dicha plataforma se colocará el sensor. Hay que tener en cuenta que el movimiento de la plataforma no será exactamente lineal, ya que se dará un cierto bandeo debido a la disposición de las guías. Sin embargo, puesto que la durante la toma de datos el sistema se encontrará quieto, dicho bandeo no supondrá un problema. En el siguiente esquema se resume la explicación anterior.

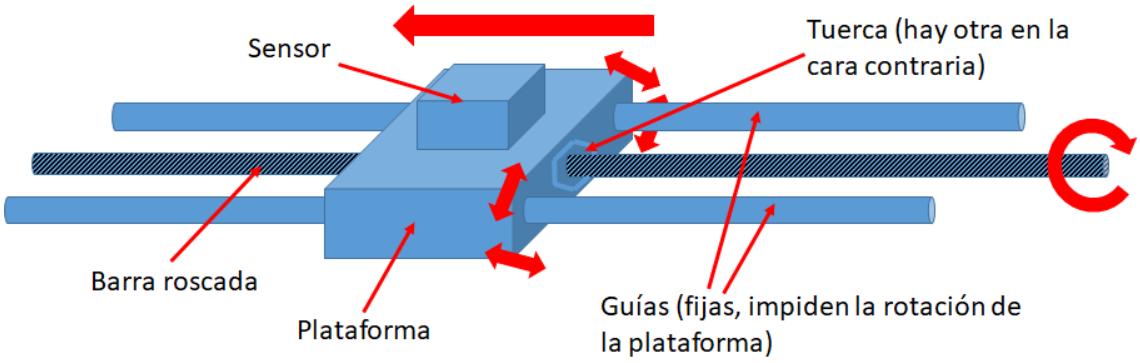


Ilustración 94: Esquema con el concepto para el movimiento lineal del sensor.

Una vez se tiene una idea aproximada del concepto del mecanismo del prototipo, hay que centrarse en las piezas necesarias para su construcción. Se requerirán dos soportes para fijar las guías y establecer el inicio y el final del recorrido del sensor. Con esa finalidad, se pondrá además un final de carrera en cada soporte. En ambas piezas para fijar las guías interesa colocar un rodamiento para facilitar la rotación de la barra roscada. Además, hay que construir un soporte para fijar el motor. Todo este sistema se deberá fijar sobre una superficie plana.

En resumen, mediante este sistema mecánico será posible obtener un movimiento lineal del sensor a través del mecanismo tornillo-tuerca. Además, se conocerá cuando el recorrido de escaneo se ha completado gracias a los finales de carrera. Con ello, queda establecido el diseño conceptual del prototipo. A continuación, se muestra una imagen que resume el diseño propuesto.

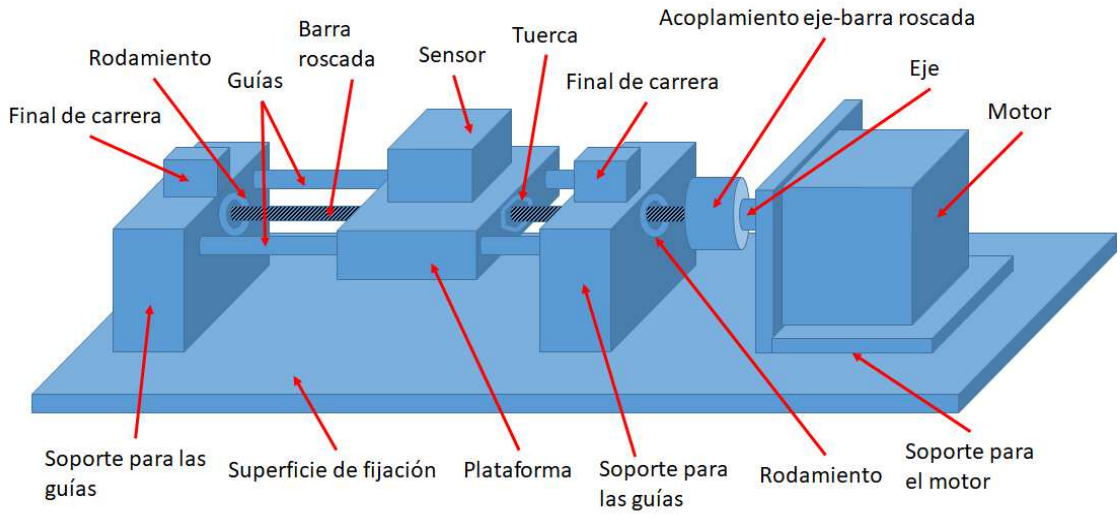


Ilustración 95: Diseño conceptual del sistema mecánico para el prototipo.

Este sistema será montado en el interior de una caja con espacio suficiente para las componentes eléctricas (placa Arduino UNO y driver L298N). En dicha caja se realizarán los agujeros necesarios para la conexión del sistema con el ordenador y con una fuente de alimentación. Se montará un cristal en la parte superior, destinado a la colocación de las muestras. Sobre dicho cristal se acoplará una tapa con la finalidad de aislar las muestras durante el escaneo. Todo ello se muestra en la siguiente imagen.

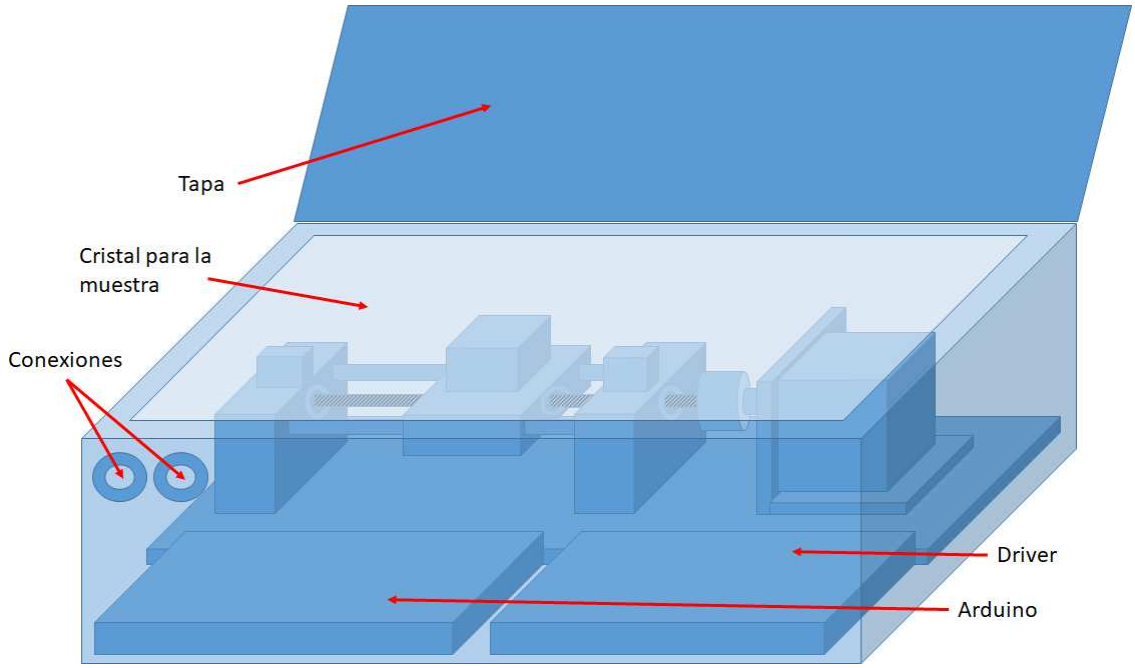


Ilustración 96: Diseño conceptual del prototipo.

4.3.2 Materiales

Una vez se conoce de aproximadamente la forma que tendrá el producto final, toca concretar su desarrollo. Para ello, hay que hacerse con las piezas correspondientes. Se pueden comprar o fabricar, según se estime conveniente. En la siguiente tabla se encuentran los materiales que fueron comprados o proporcionados, junto con algunas de sus características clave y su procedencia.

Producto	**Características clave**	**Procedencia**
Finales de carrera (x2)	8x6x7 mm	Electrónica Gimeno S.A
Rodamientos (x2)	Diámetro externo 10 mm, diámetro interno 3 mm	RS Components, DIE
Barra roscada	M3, paso 0.5 mm	Leroy Merlin
Barra cilíndrica (guía)	Diámetro 4 mm	Leroy Merlin
Tornillos	M2, M2.5 y M3	Leroy Merlin, DIE
Arandelas	M2.5	DIE
Caja	190x109x90 mm, marrón, de plástico rígido	RS Components, DIE
Tablones de madera	Grosor 16 mm	DIE
Fuente de alimentación	12 V	RS Components, DIE
Cable USB	Comunicación serial	DIE
Conector para fuente de alimentación	Conector para chasis, patillas para tierra y voltaje	Electrónica Gimeno S.A
Arduino UNO	Véase 3.3.2	DIE
SM-42BYG011-25	Véase 3.3.3	DIE
L298N	Véase 3.3.3	DIE

Tabla 8: Material empleado para la construcción del prototipo y su procedencia.

Los tablones de madera se utilizaron para crear una superficie plana para el sistema mecánico, puesto que la base de la caja estaba ligeramente combada. En la tabla anterior no se ha incluido el cristal para la toma de muestras, puesto que se utilizó el de un reloj estropeado. Los precios de cada componente se pueden encontrar en el capítulo 7, dedicado al presupuesto.

4.3.3 Diseño de piezas del sistema mecánico.

Una vez presentado el material que se requerirá para la construcción del prototipo, hay que diseñar las piezas para el sistema mecánico. Ello se debe a que en el mercado no fue posible encontrar mecanismos que pudiesen servir para la construcción del prototipo objeto del presente trabajo. Por ello, se optó por la fabricación propia de los elementos necesarios. Tras el diseño mediante el software SOLIDWORKS, los modelos serán producidos con una impresora 3D. En este apartado se mostrará el resultado final de cada pieza con sus partes relevantes señaladas. En el anexo E se pueden encontrar los planos de todas las piezas.

En primer lugar, se muestran los soportes para las guías. Ambos cuentan con agujeros para la sujeción mediante tornillos a la base de madera, con un alojamiento para los finales de carrera, con un hueco para los rodamientos y con agujeros para sujetar las guías mediante tornillos. La diferencia entre los dos soportes se encuentra en los agujeros para las guías, en uno no atraviesan el soporte (para servir de tope), mientras que en el otro sí.

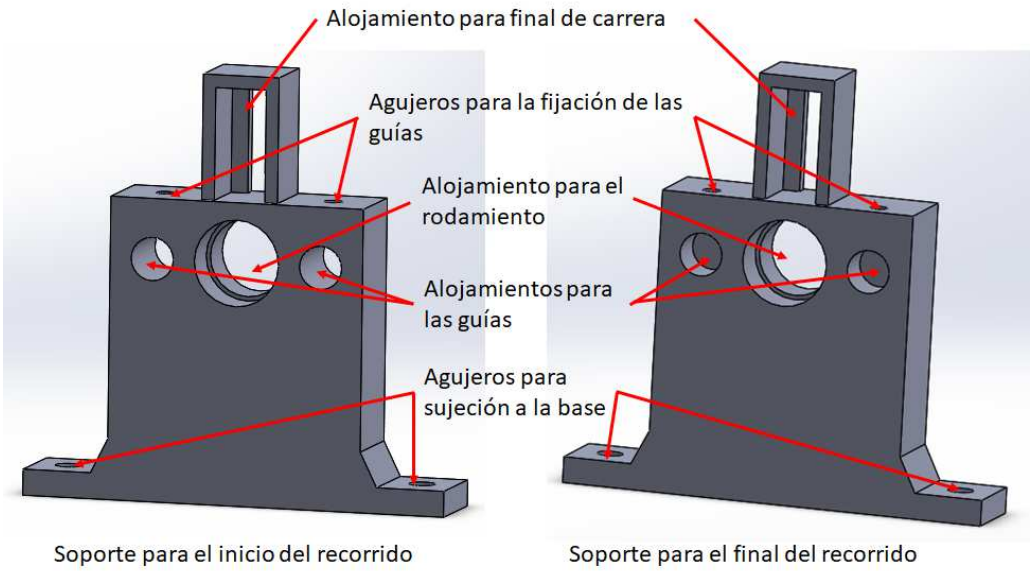


Ilustración 97: Soportes para las guías.

A continuación, se puede observar el soporte para el motor. Cuenta con varios agujeros para su fijación a la base y la sujeción del motor al soporte.

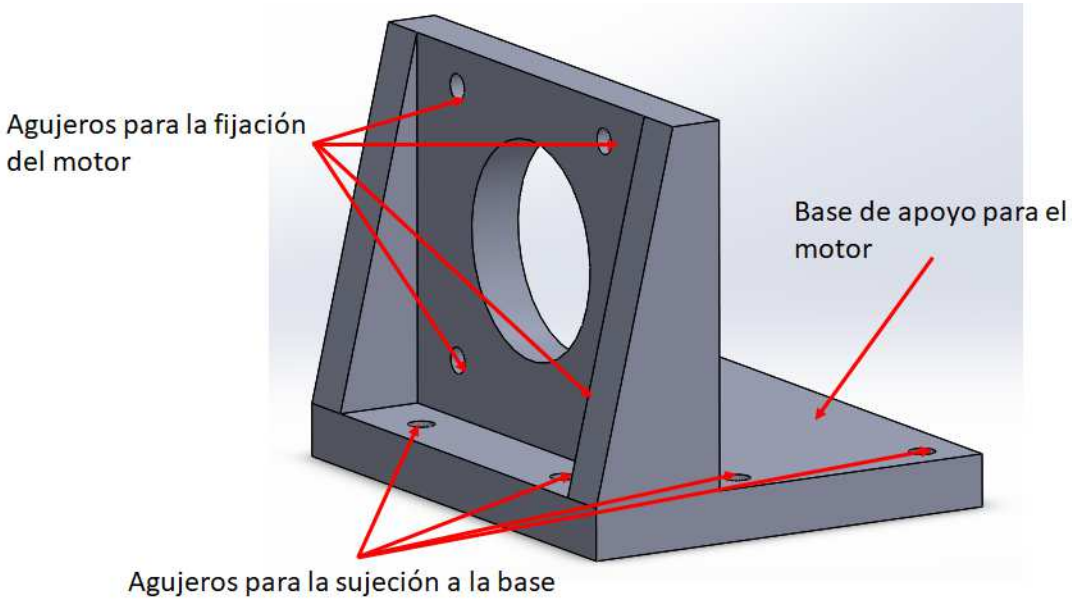


Ilustración 98: Soporte para el motor.

En tercer lugar, se muestran la plataforma y la pieza de sujeción del sensor. La primera consta de dos huecos en los que se insertará una tuerca M3. Además, tiene dos agujeros para las guías. La segunda tiene un hueco con la forma del sensor, dos agujeros para su fijación y un hueco para pasar los cables. Su función consiste en proteger el sensor y hacer contacto con los finales de carrera al inicio y al final del recorrido.

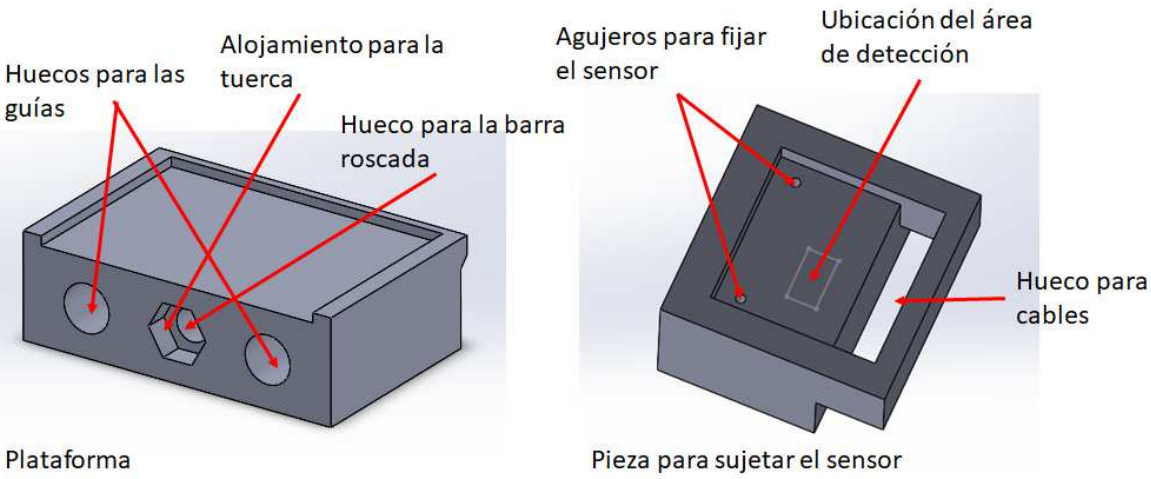


Ilustración 99: Plataforma y pieza para sujetar el sensor.

La siguiente pieza que se presentará será el acoplamiento encargado de la transmisión del giro del eje del motor a la barra roscada. Tendrá forma cilíndrica. En el extremo del motor

se realizó un agujero con un diámetro ligeramente menor a 5 mm (el diámetro del eje), de forma que la pieza se mantenga fija al eje por presión. En el otro extremo del acoplamiento se realizó una incisión alargada. Se deberá modificar manualmente el extremo de la barra roscada para que quepa en ese hueco. A continuación, se muestra el modelo 3D del acoplamiento.

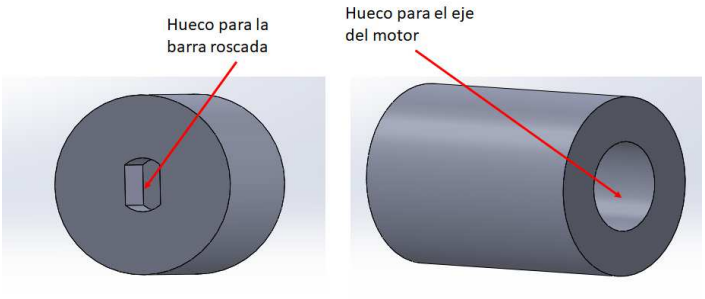


Ilustración 100: Acoplamiento. Cara para la barra roscada (izquierda) y para el eje del motor (derecha).

Aunque la caja comprada cuenta con una tapa, ésta impide la colocación de la muestra a la distancia óptima respecto al sensor. Se podría mecanizar la tapa, sin embargo, el esfuerzo no compensaría la calidad del resultado. Por ello, se modeló una pieza que permita una óptima colocación de la muestra. En la imagen situada a continuación se observa su modelo 3D. Se pueden apreciar los agujeros para la fijación a la caja, así como el hueco destinado al cristal sobre el que se colocará la muestra. Se cuenta con dos ranuras para la colocación de dos bisagras que permitan colocar una tapa extra para aislar la muestra durante la medición.

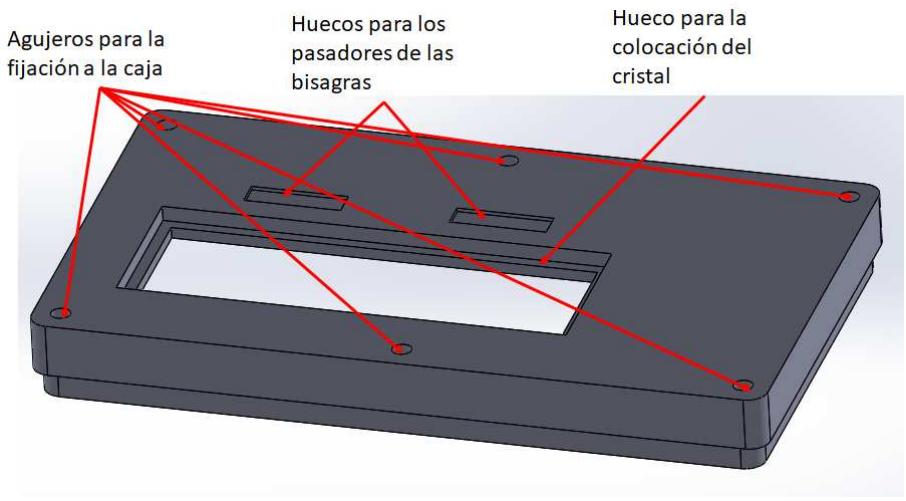


Ilustración 101: Tapa para la colocación de las muestras.

Así, las últimas piezas que queda por modelar son la tapa para cubrir la muestra y el pasador para poder abrir y cerrar la tapa. La imagen situada a continuación muestra ambos modelos.

Ilustración 102: Tapa para la muestra (izquierda) y pasador para la tapa (derecha).

4.3.4 Fabricación de las piezas diseñadas

Una vez se cuenta con los modelos 3D de todas las piezas del sistema mecánico, hay que fabricarlas. Para ello se contó con la colaboración del taller de impresoras 3D de la escuela de diseño industrial de la UPV (ETSID). En la impresión 3D se sintetiza un objeto tridimensional mediante la extrusión por capas a alta temperatura de un material termoplástico sobre una superficie caliente. Para el presente trabajo se utilizó el modelo Zortrax m200, que se observa en la siguiente imagen.

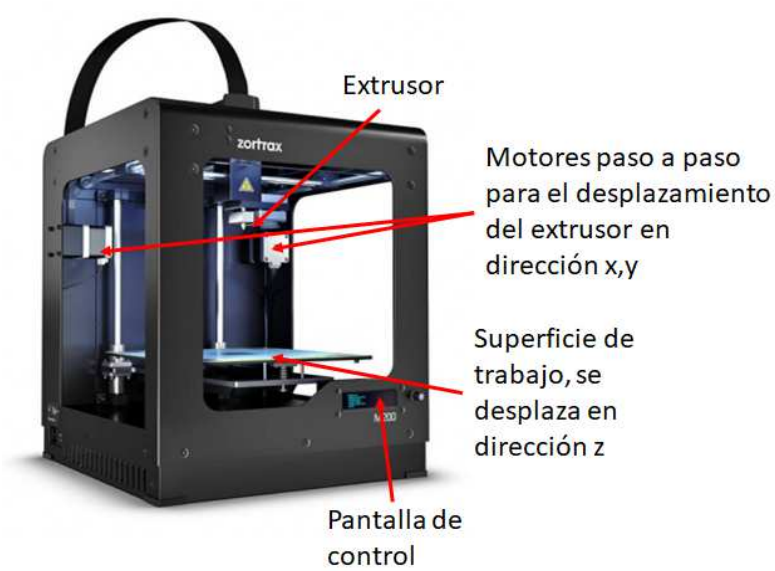


Ilustración 103: Impresora 3D Zortrax m200.

En el taller de las impresoras 3D se podía elegir distintos materiales para la elaboración de las piezas. Se eligió el Z-ULTRAT por sus características mecánicas y ópticas, que se consideraron como aceptables para el sistema mecánico del prototipo. Algunas de las propiedades de ese material se encuentran en la tabla 9.

Como se puede ver, ofrece unas propiedades mecánicas excelentes para el desarrollo de prototipos. Sus características más destacables son su durabilidad, su alta dureza, su bajo nivel de deformación y su resistencia a la distorsión durante la impresión.

Propiedad	**Valor**
Densidad	1.08 gr/cm^3
Dureza (Rockwell)	110
Resistencia a la tracción	42 MPa
Módulo de tracción	1.95 GPa
Módulo de flexión	2.02 GPa
Punto de reblandecimiento	128 ºC (carga de 5 kg)

Tabla 9: Algunas características del termoplástico Z-ULTRAT.

Una vez se sabe el material y la impresora que se utilizarán, hay que convertir los archivos de la pieza a formato “stl” para poder imprimirlos. Este paso se puede realizar desde la opción “Guardar como” del SOLIDWORKS.

A continuación, hay que generar un archivo que prepare los modelos para su impresión con la Zortax m200. Para ello, se cuenta con el programa Z-Suite en el taller de impresoras de la ETSID, mediante el cual se ajustan parámetros tales como el material elegido, el espesor de la capa, la velocidad de impresión o el relleno de la pieza. Además, se puede modificar la colocación del modelo.

En este sentido, con la finalidad de minimizar la cantidad de material empleado, interesa colocar la mayor cantidad de superficie plana sobre la base de impresión.

Se eligió el Z-ULTRAT como material, un espesor de capa de 0.09 mm, la velocidad de impresión predeterminada y un relleno medio de las piezas.

Una vez generado el archivo para la Zortrax m200, se comprueba que el extrusor y la placa térmica esté limpios, se aplica laca a la última para evitar que la pieza se combe durante la impresión y se da comienza a la fabricación de las piezas. El extrusor alcanza una temperatura máxima de 180 ºC y la placa 84 ºC, por lo que no interesa acercar la mano durante el proceso.

La impresión se efectuó en tres tandas: en la primera, se fabricaron los soportes y el acoplamiento, en la segunda, la tapa para las muestras y en la tercera, la tapa para cubrir la muestra y los soportes para las bisagras. El tiempo total de impresión fue de unas 27 horas y la cantidad de material empleado ronda los 200 gramos. La siguiente imagen se tomó durante la impresión de la primera tanda. Se puede observar cómo la impresora trabaja por capas y la disposición del relleno de las piezas.

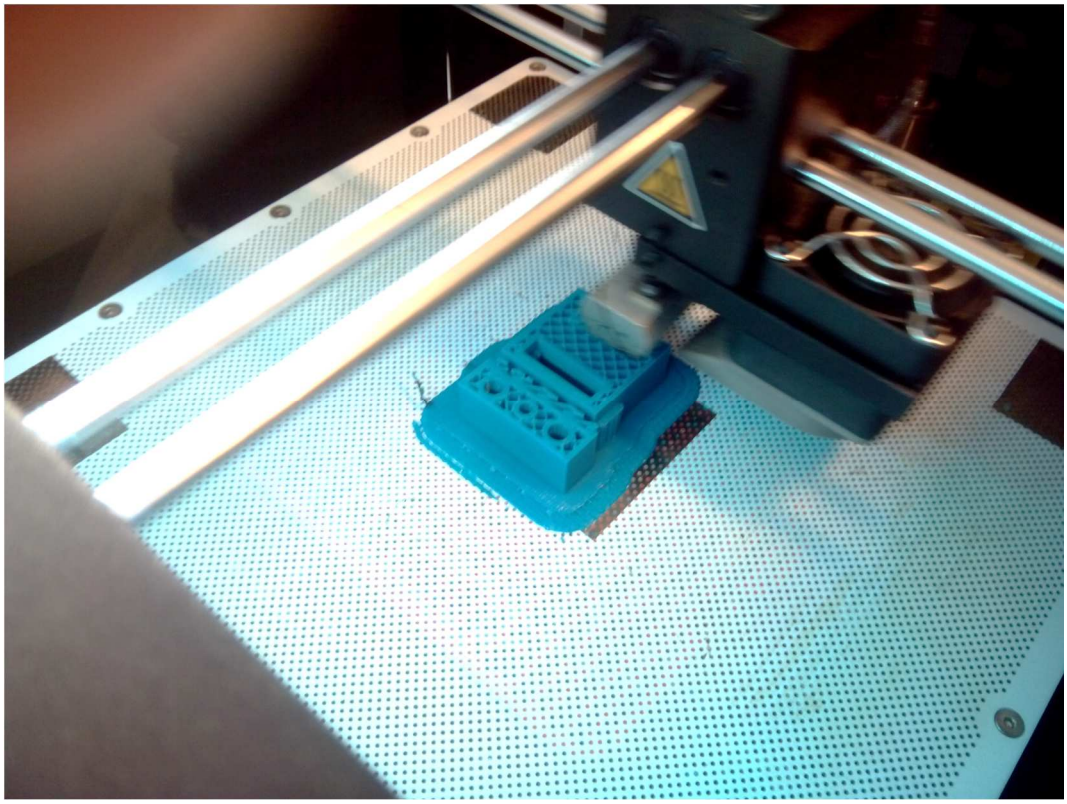


Ilustración 104: Zortrax m200 en proceso de impresión.

Una vez ha finalizado la impresión, hay que esperar a que baje la temperatura de la placa. A continuación, se pueden extraer las piezas con ayuda de una espátula.

Seguidamente, se muestran las imágenes correspondientes con los resultados de las impresiones. La segunda dio varios problemas relacionados con el tamaño de la pieza, por lo que se tuvo que repetir.

Ilustración 105: Resultado de la primera tanda de impresión.

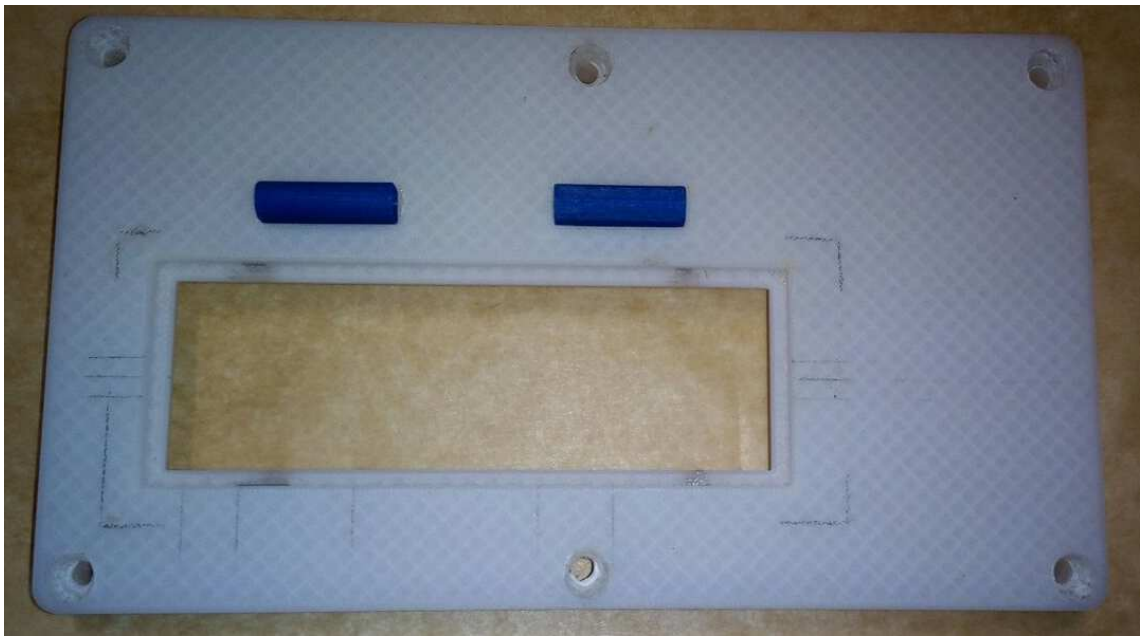


Ilustración 106: Resultado de la segunda impresión, con los pasadores de la tercera ya colocados.

Ilustración 107: Resultado de la tercera impresión.

4.3.4 Montaje

Una vez se han obtenido todas las piezas y materiales necesarios para llevar a cabo la construcción del prototipo, cabe realizar el ensamblaje de las mismas. En este apartado se explicará brevemente el proceso que se siguió para ello.

En primer lugar, se preparó la superficie sobre la que irá montada el mecanismo mecánico. Para ello, se trabajó un trozo de madera, dándole el tamaño justo para que cupiese en la caja adquirida, dejando un hueco para el Arduino UNO y señalando la ubicación de los soportes, como se muestra en la ilustración 108.

A continuación, se cortó la barra roscada y las barras de latón para las guías. Un extremo de la barra roscada se pulió para poder introducirlo en el acoplamiento. El resultado se puede ver en la ilustración 109.

En el siguiente paso se fijaron los finales de carrera y los rodamientos a los soportes para las guías. Además, se introdujeron las tuercas en la plataforma, se pego ésta al soporte del sensor y se atornilló el TCS34725 a dicho soporte. A las tuercas y los rodamientos se les aplicó algo de lubricante. El resultado se encuentra en la ilustración 110.

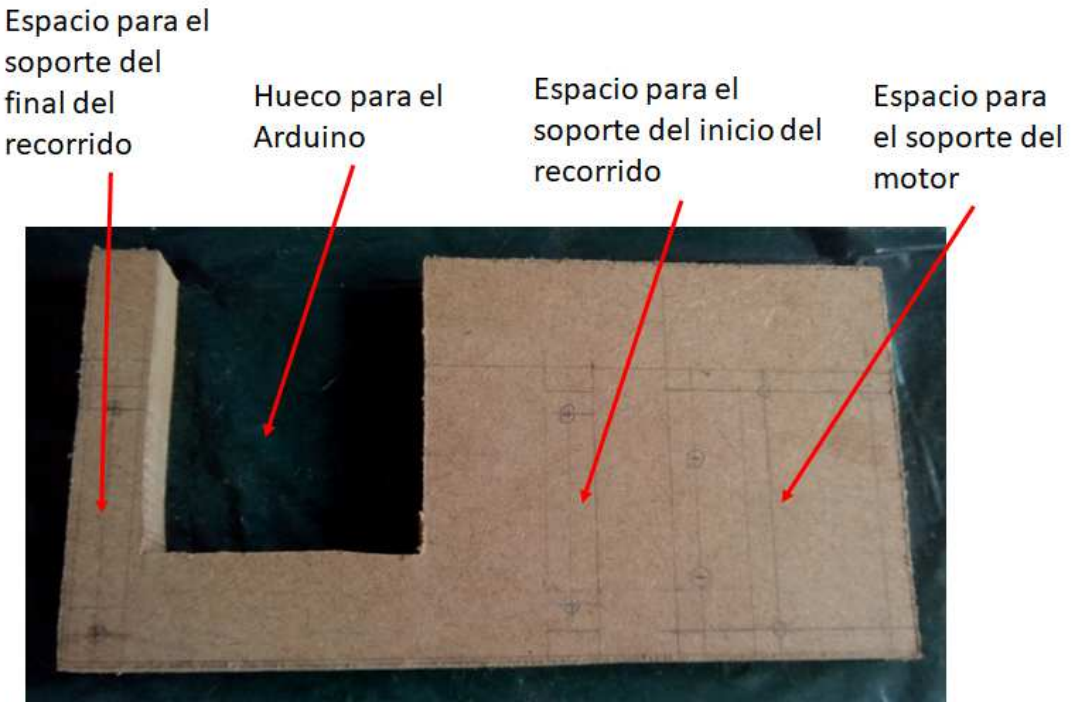


Ilustración 108: Base del sistema mecánico del prototipo.

Ilustración 109: Guías y barra roscada.

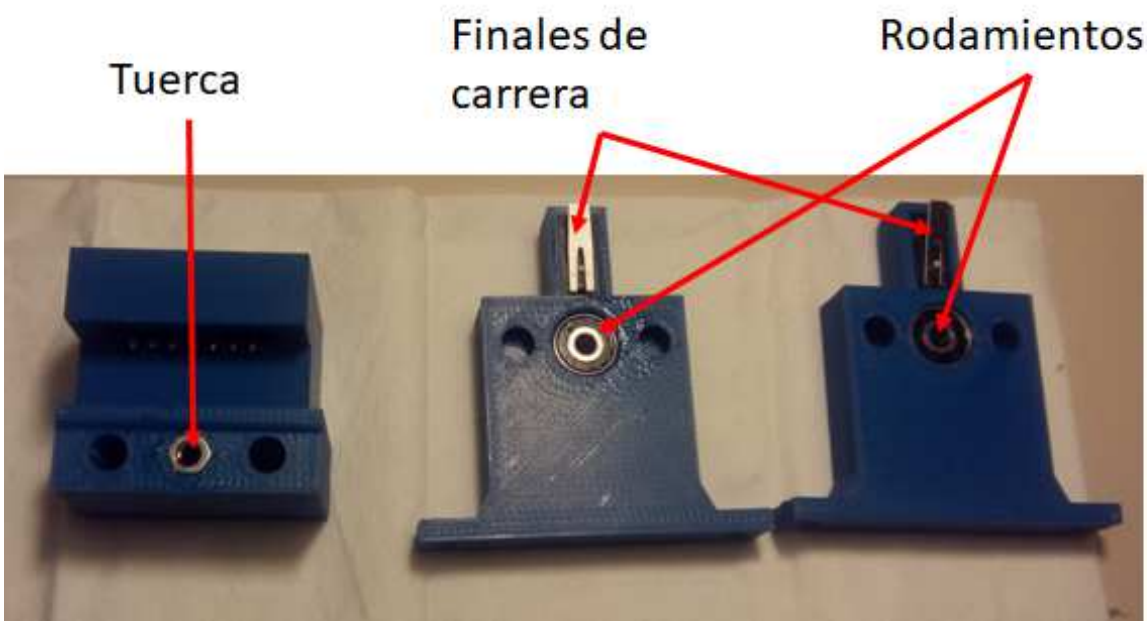


Ilustración 110: Plataforma con soporte para el sensor y soportes de las guías con rodamientos y finales de carrera.

A continuación, se introducen la barra roscada y las guías en la plataforma, se colocan en los soportes los extremos de las guías y se atornillan a la base de madera con arandelas. Seguidamente, se coloca el acoplamiento en el eje del motor (por el lado correspodiente), se fija el motor a su soporte y se atornilla este último a la base de madera, para lo cual habrá que introducir primero el extremo de la barra roscada trabajado con tal fin en la pieza de acoplamiento. Una vez está montado el sistema, se fijan las guías con ayuda de unos tornillos en la parte superior de los soportes. En la ilustración 111 se muestra la comparación entre el modelo de SOLIDWORKS y el montaje mecánico real.

El siguiente paso consistió en realizar los agujeros necesarios para las conexiones con la fuenta de alimentación y el ordenador en la caja. Se colocó el conector de chasis en el agujero correspondiente. El resultado se muestra en la ilustración 112.

Antes de completar el montaje, hay que ocuparse de las conexiones eléctricas. En la imagen 113 se observa un esquema de conexiones y una fotografía del sistema.

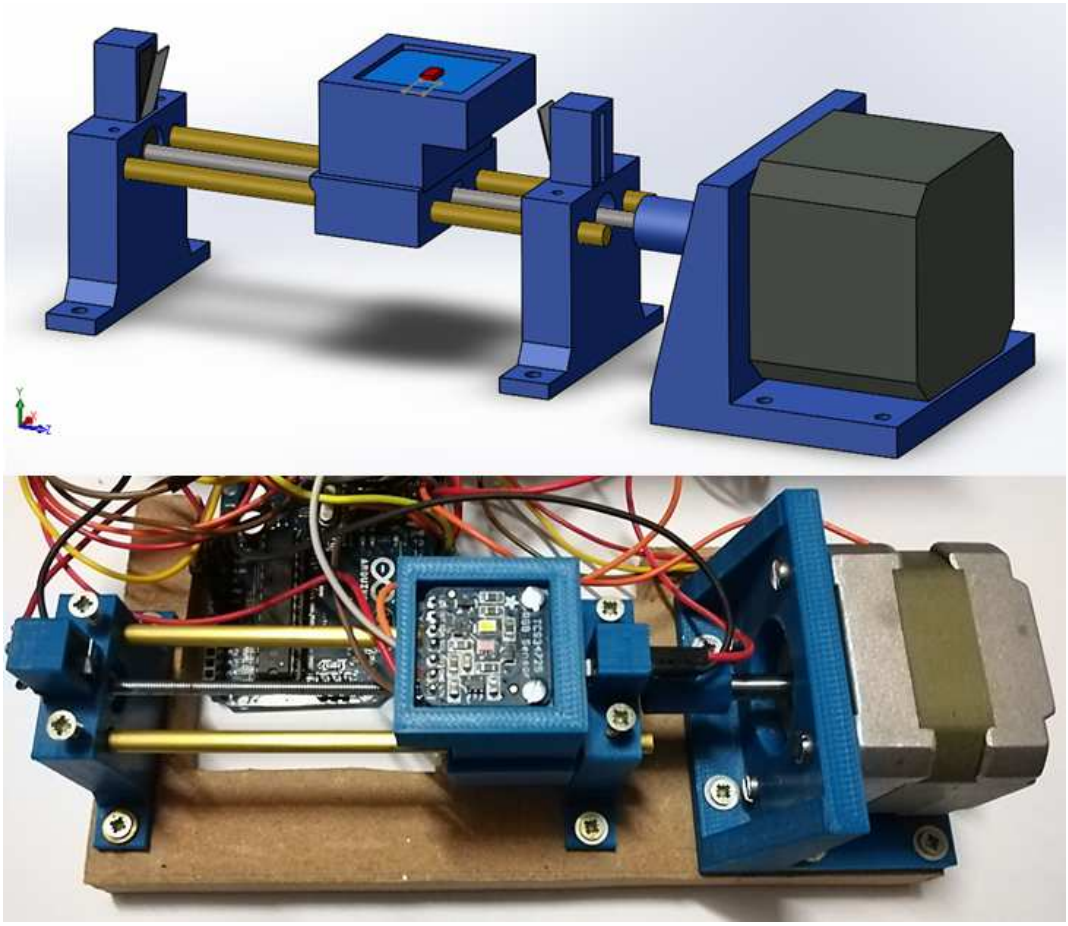


Ilustración 111: Modelo 3D del sistema mecánico (arriba) y montaje real (abajo).

Ilustración 112: Conexiones para la comunicación y alimentación del prototipo.

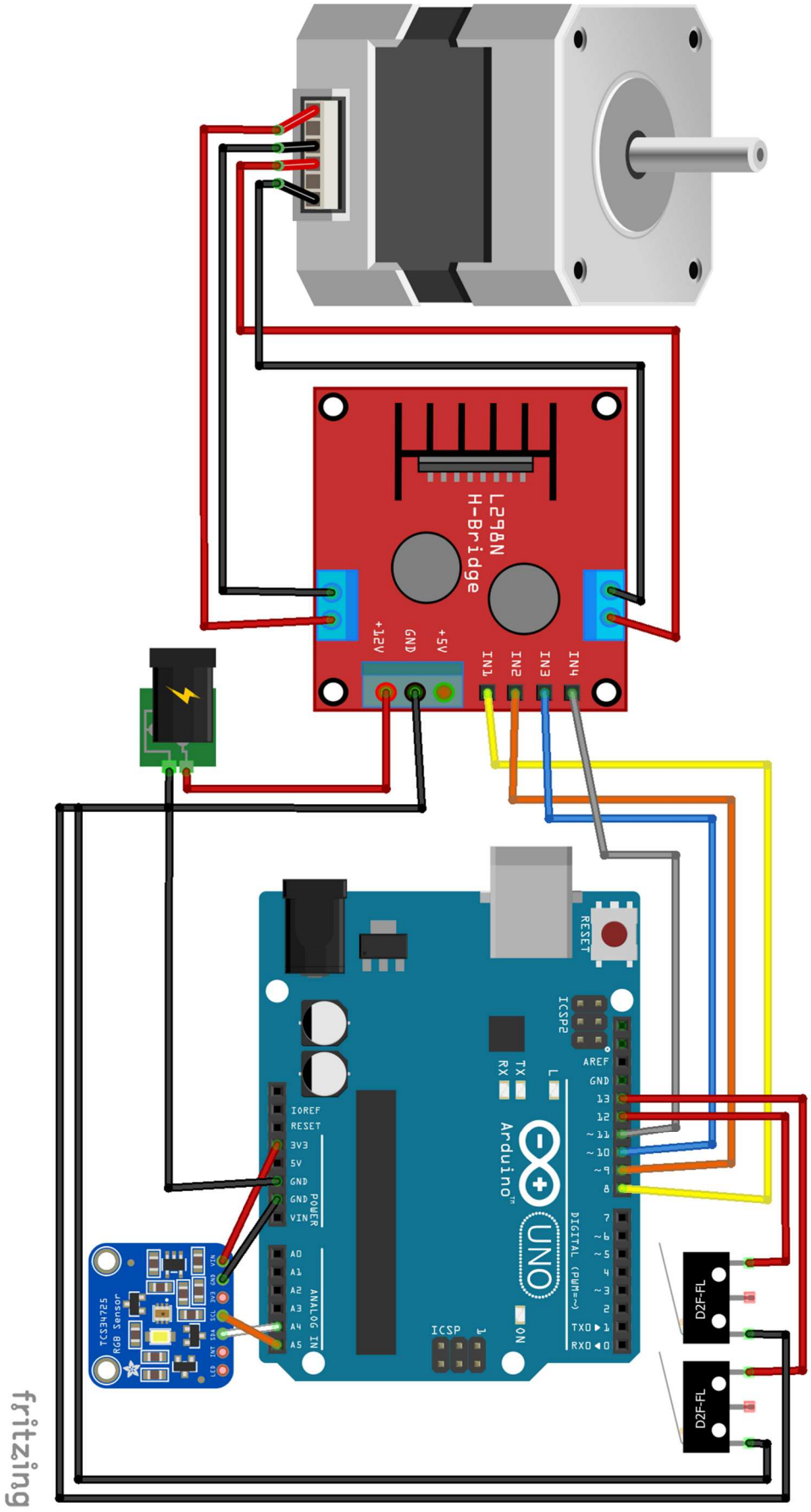


Ilustración 113: Esquema de conexiones del sistema.

Como se puede observar en la imagen anterior, el suministro de la energía para el motor tiene lugar desde un conector. Los finales de carrera se conectan a dos salidas digitales, de las que habrá que activar la resistencia “pull-up”. El final de carrera del pin 13 se colocó al inicio del recorrido y el del pin 12 al final. Las conexiones del motor y del sensor son idénticas a las que se empleó en sus respectivas caracterizaciones.

Una vez se han montado los componentes electrónicos y conectado los cables, se puede insertar el sistema mecánico, el Arduino UNO y el driver L289N en la caja, donde no sobra mucho espacio, como se puede ver en la siguiente fotografía.

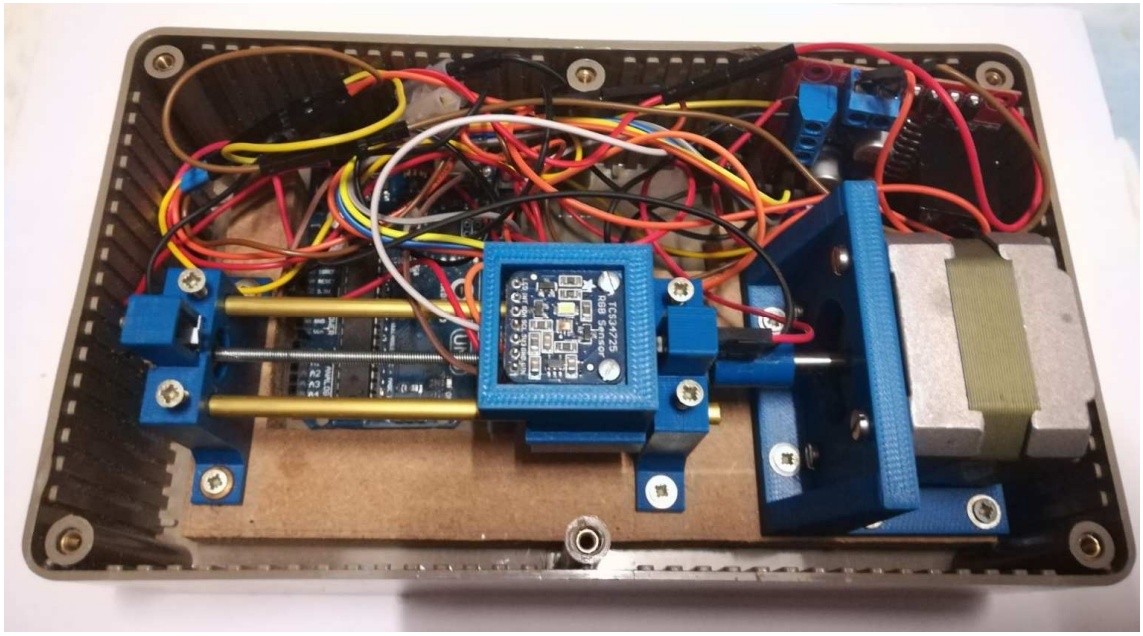


Ilustración 114: Sistema mecánico y electrónico en la caja.

A continuación, se pegan los pasadores para las bisagras en la tapa para la colocación de las muestras, se pone el cristal y se atornilla la tapa. Adicionalmente, se preparó una cartulina para orientar al usuario en la colocación de las piezas.

Finalmente, se coloca la tapa para aislar las muestras. Como eje de la bisagra se usó un trozo del excedente de la barra roscada, que se mantiene en su sitio mediante tuercas autoblocantes. Para presionar la muestra contra el cristal y hacer de fondo del escáner se preparó un trozo de cartón pluma.

Para finalizar el apartado, en la siguiente imagen se muestra el prototipo montado.

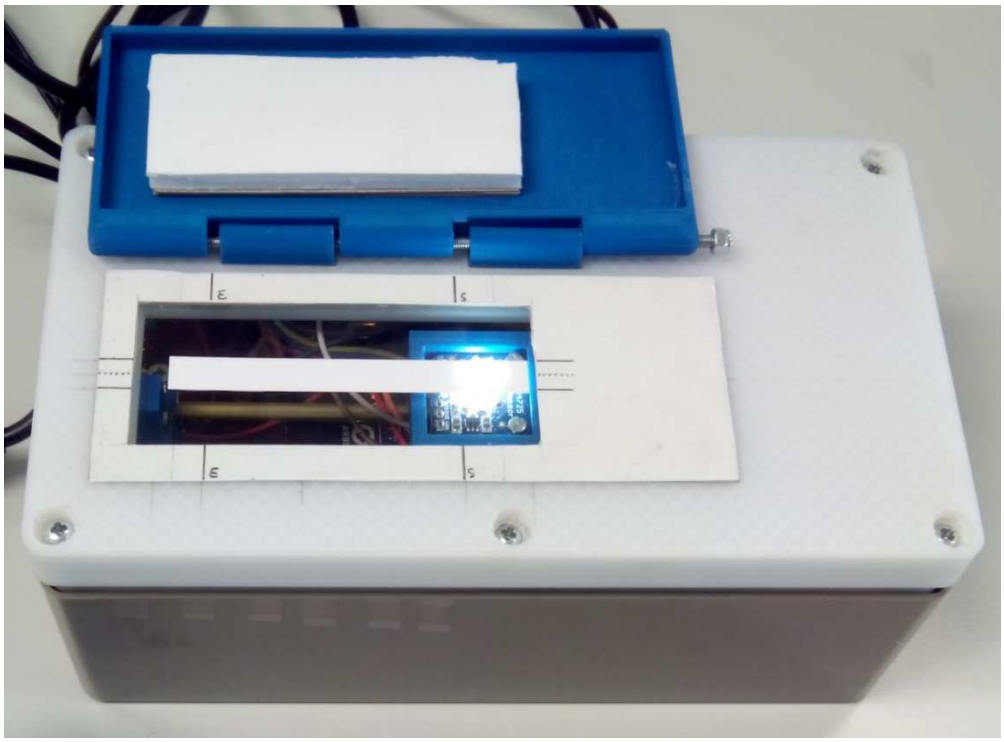


Ilustración 115: Prototipo montado con muestra sobre el cristal.

4.4 Software

Como resultado de los apartados anteriores, se cuenta con un prototipo de escáner que ahora hay que hacer funcionar. Para ello, se elaboró un programa en Arduino que se encarga del funcionamiento del sistema y una interfaz de usuario en MATLAB para el manejo del mismo. En los siguientes apartados se explicará el funcionamiento de cada uno de ellos.

4.4.1 Sofware de Arduino

En este apartado se explicará el funcionamiento del software programado para Arduino mediante diagramas de flujo.

El primer paso consiste en definir las librerías que se necesitarán. Para el control del motor se eligió la librería “Accelstepper”, para el sensor la “Adafruit_TCS34725” y la “Wire” para administrar las conexiones de los pines.

A continuación, se definieron las constantes y las variables que se utilizarán posteriormente y se creó el objeto “motor” (se establece el número y la ubicación de las conexiones) y “sensor” (se establece el tiempo de integración y la ganancia).

Seguidamente, se encuentran definidas una serie de funciones que permiten que el prototipo ejecute las siguientes acciones: el escaneo, la lectura única, la calibración RGB, el calibrado del recorrido, la validación de los datos escaneados y la comprobación de la conexión con el motor y con Arduino, que serán explicadas con detalle más adelante.
En el "setup" del programa se inicializa la conexión serial, se define los pines para los finales de carrera y se activan sus resistencias "pull-up". Seguidamente, se inicializa el sensor y se concreta la velocidad y aceleración del motor.

En el bucle principal se comprueba continuamente si han sido enviado órdenes desde MATLAB. La recepción del carácter "1" activa la función de escaneo (calibración del recorrido, escaneo y validación de los datos), un "2" la de calibrado RGB, un "3" el reset de la posición del motor, el "4" comprueba si el motor está conectado, mediante el "5" MATLAB comprueba que el Arduino se encuentre conectado y el "6" activa la lectura única.

En el siguiente diagrama de flujo se muestra el funcionamiento básico del programa de Arduino para controlar el prototipo.

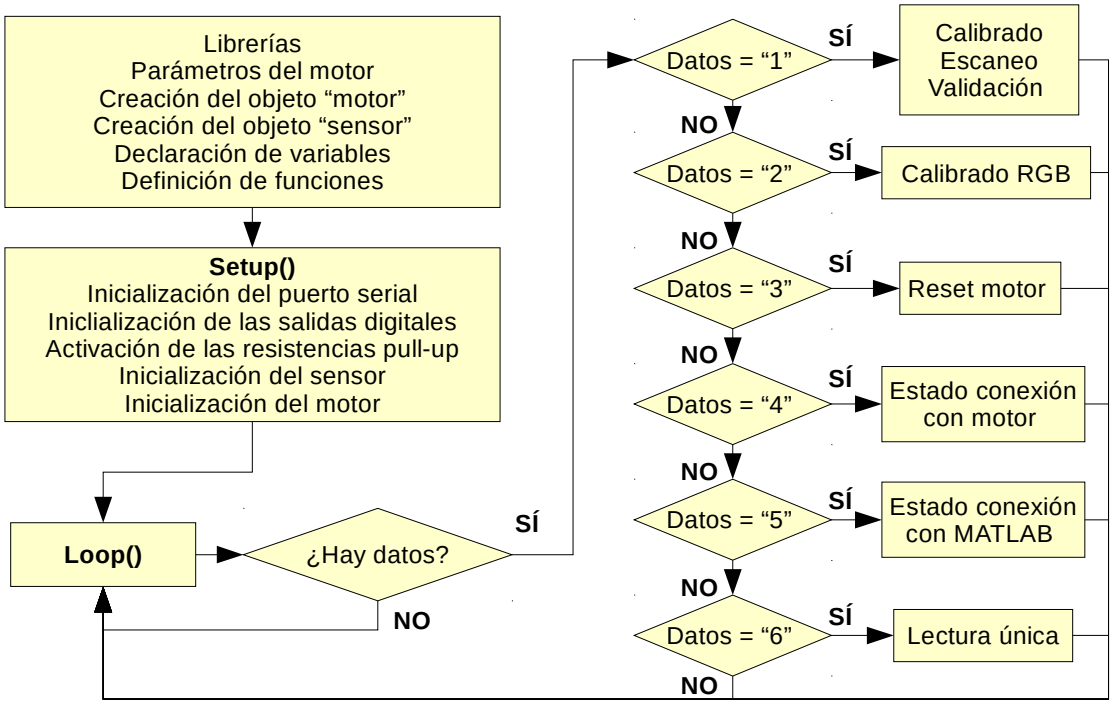


Ilustración 116: Diagrama de flujo del programa de Arduino.

A continuación, se pasará a explicar de forma más detallada el funcionamiento de cada una de las funciones.

- *Función de calibración del recorrido:*

  Al inicio de la calibración, se comprueba si se ha calibrado con éxito con anterioridad. Si no es así, se inicia el procedimiento de calibración. Con tal fin, se comprueba si el sensor se encuentra al inicio del recorrido. Ese es el caso si se recibe un valor digital alto del pin 13, correspondiente al final de carrera situado al principio. De no ser así, se hace retroceder el sensor hasta que hay contacto con ese final de carrera. A continuación, se define la posición del sensor como “0” y se avanza hasta el segundo final de carrera. Una vez hay contacto con éste, se guarda la posición como “Xmax” y se retrocede hasta el primer final de carrera.

  Durante el movimiento del sensor se informa a MATLAB cada dos segundos de la acción que se está llevando a cabo.

  Si al finalizar esta función hay contacto con el primer final de carrera, ello significa que la calibración se ha realizado con éxito. Por lo tanto, se informa a MATLAB y se guarda la posición actual del sensor para el posterior cálculo del error.

  Si, por el contrario, no hay contacto con el primer final de carrera, no se ha podido calibrar. Se informa a MATLAB, que mostrará un mensaje de error al usuario.

  En la ilustración 117 se encuentra el diagrama de bloques de la función de calibración del recorrido. Los puntos de inicio y de final están coloreados en azul claro.

- *Función de escaneo:*

  Esta función comienza comprobando si se ha calibrado correctamente y si hay contacto con el final de carrera del inicio del recorrido. Si se cumplen ambas condiciones, se activa la toma de datos. Para ello, el sensor avanza hacia el segundo final de carrera, parándose cada cierto número de pasos para tomar datos, enviarlos a MATLAB y registrar la lectura. Se toman datos cada 0.5 mm milímetros (paso de la barra roscada), es decir, cada vuelta completa del eje del motor. Una vez se ha llegado al final del recorrido, informa de ello a MATLAB, guarda la

posición y retrocede hasta el primer final de carrera. El diagrama de flujo de la ilustración 118 resume la explicación anterior.

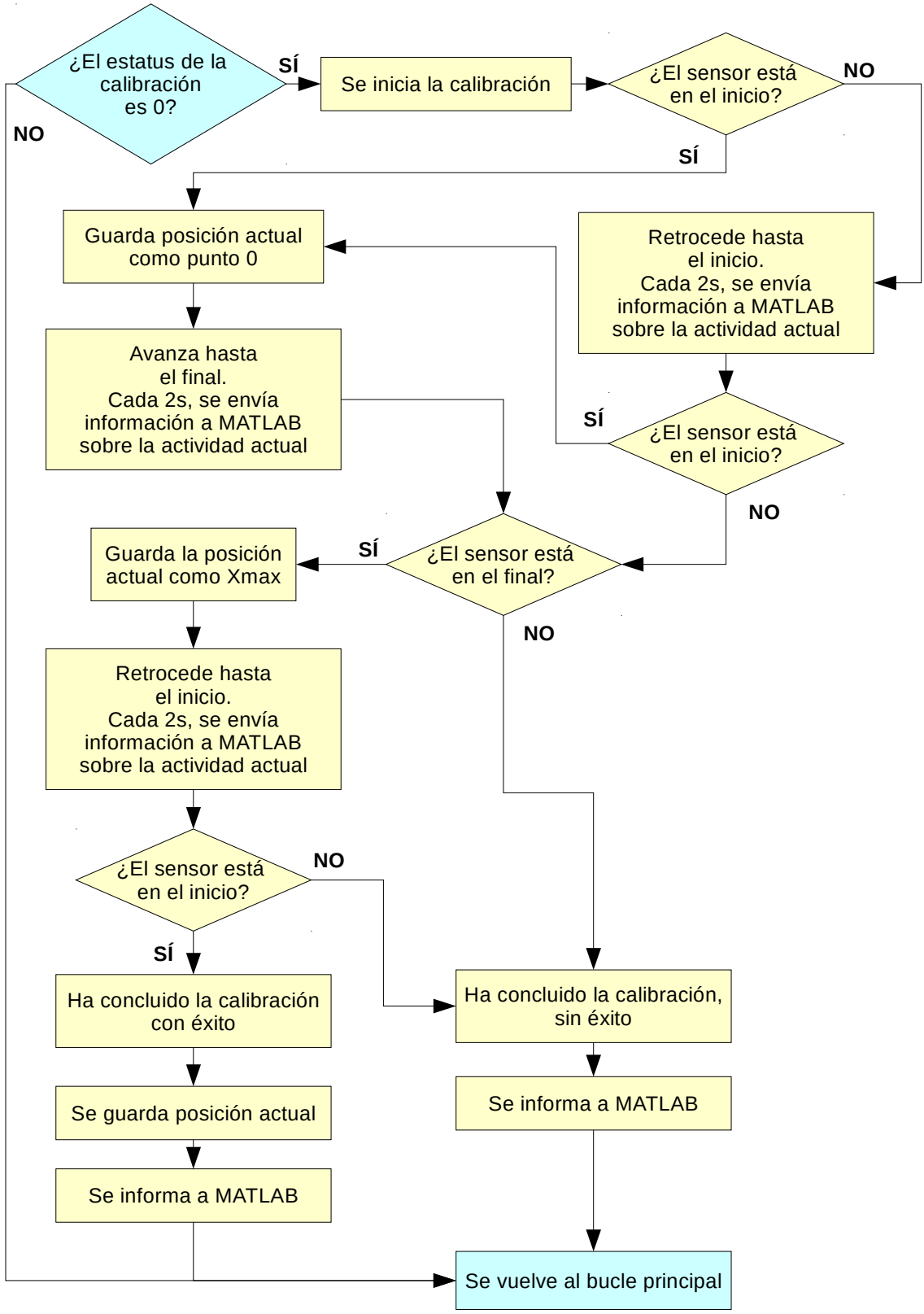


Ilustración 117: Diagrama de flujo de la función de calibración del recorrido.

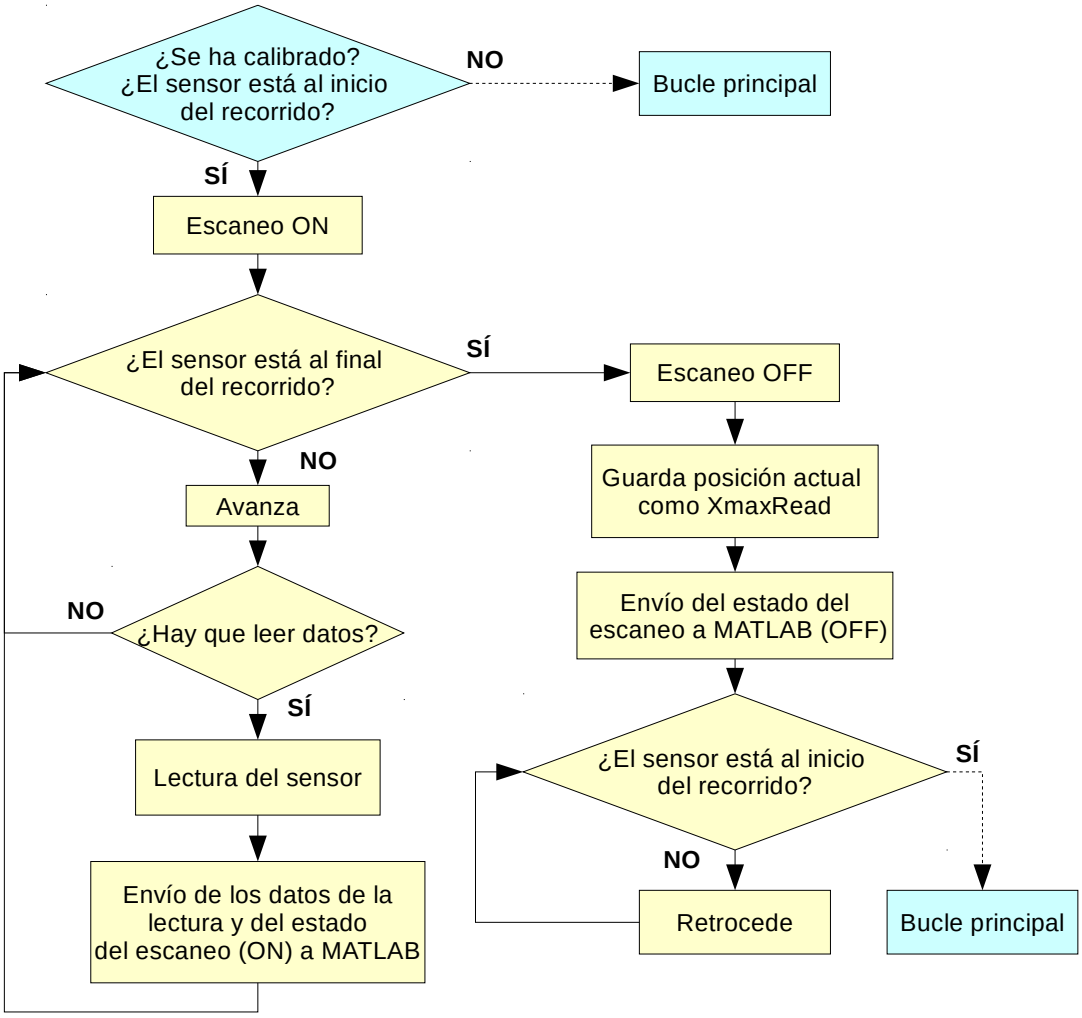


Ilustración 118: Diagrama de flujo de la función de escaneo.

– *Función de validación:*

  Esta función permite evaluar la calidad de los datos tomados mediante el sensor y controla los pasos que ha perdido el motor durante el escaneo. Para ello, se comprueba si se ha calibrado correctamente y si el sensor se encuentra al inicio del recorrido. Si se cumplen ambas condiciones, se calcula el error cometido durante el escaneo, para lo cual se suma la diferencia entre la distancia del recorrido medida en la calibración y la observada en el escaneo a la diferencia entre la posición "0" al inicio y al final de la calibración. Si este error es inferior a 208 pasos, es decir, medio milímetro de recorrido, se toman los datos como buenos. Si no, se informa a MATLAB de que hay que realizar una nueva toma de datos.

  La siguiente ecuación resume el cálculo del error.

$$[Xmax(calibración) - Xmax(escaneo)] + [Xo(inicio\ calibración) - Xo(final\ calibración)] < 208$$

En el diagrama de flujo situado a continuación se observa el funcionamiento de la función de validación.

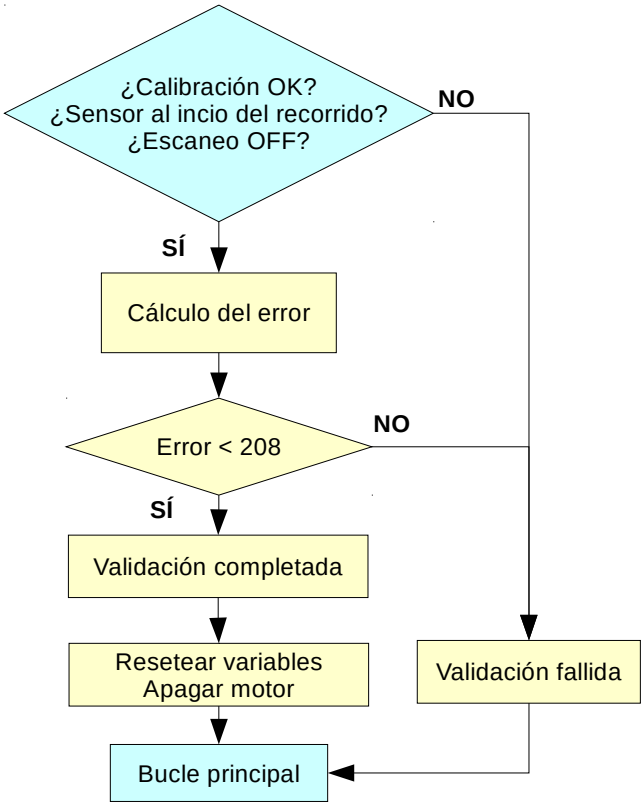


Ilustración 119: Diagrama de flujo de la función de validación.

– *Función para el reset de la posición del motor:*

El objetivo de esta función es que el sensor se coloque al inicio del recorrido. Esta función se incorpora por si tiene lugar una interrupción de la alimentación mientras el sensor se encuentre a mitad de camino entre los dos finales de carrera.

Simplemente hace retroceder el motor hasta que hay contacto con el primer final de carrera, como se muestra en el siguiente diagrama de bloques.

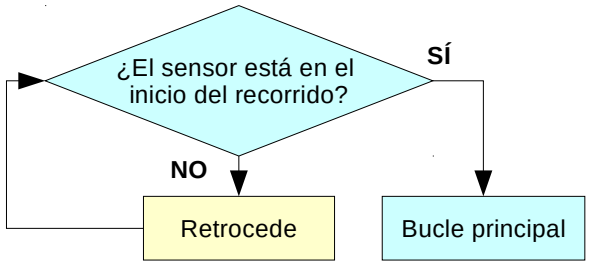


Ilustración 120: Diagrama de flujo para el reset de la posición del motor.

– *Función para comprobar si hay conexión con el motor:*

  Este código pretende establecer el estado de la conexión del motor y comunicarlo a MATLAB. Para que funcione, se parte de la condición de que el sensor se encontrará muy cerca del primer final de carrera. De no ser así, habrá que resetear la posición del motor.

  En primer lugar, se define un contador. Si el sensor se encuentra en contacto con el primer final de carrera, se hace avanzar hasta que no haya contacto o pasen dos segundos. Si cambia la señal del primer de carrera, significa que el motor se ha movido y, por lo tanto, está conectado. Si no cambia la señal no hay conexión con el motor. En ambos casos se comunica el resultado a MATLAB y se modifica el contador.

  La otra opción es que el sensor no esté en contacto con el final de carrera. En este caso se hará retroceder el motor hasta que haya contacto o pasen dos segundos sin contacto. De nuevo un cambio en la señal del final de carrera indica que hay conexión con el motor. El diagrama de flujo se encuentra en la ilustración 121.

– *Función para comprobar la conexión con MATLAB:*

  Con la finalidad de comprobar la conexión con Arduino, el programa de MATLAB enviará la orden que active esta función al inicio de su ejecución. Si recibe la respuesta correcta desde Arduino, se sabrá que la conexión es correcta.

  Para ello, lo único que hará este programa al activarse será enviar un “1” por el puerto serial, para lo cual no es necesario elaborar un diagrama de bloques. Si la conexión con Arduino no existe, será imposible recibir respuesta de esta función, por lo que se sabrá inmediatamente que algo falla en la comunicación.

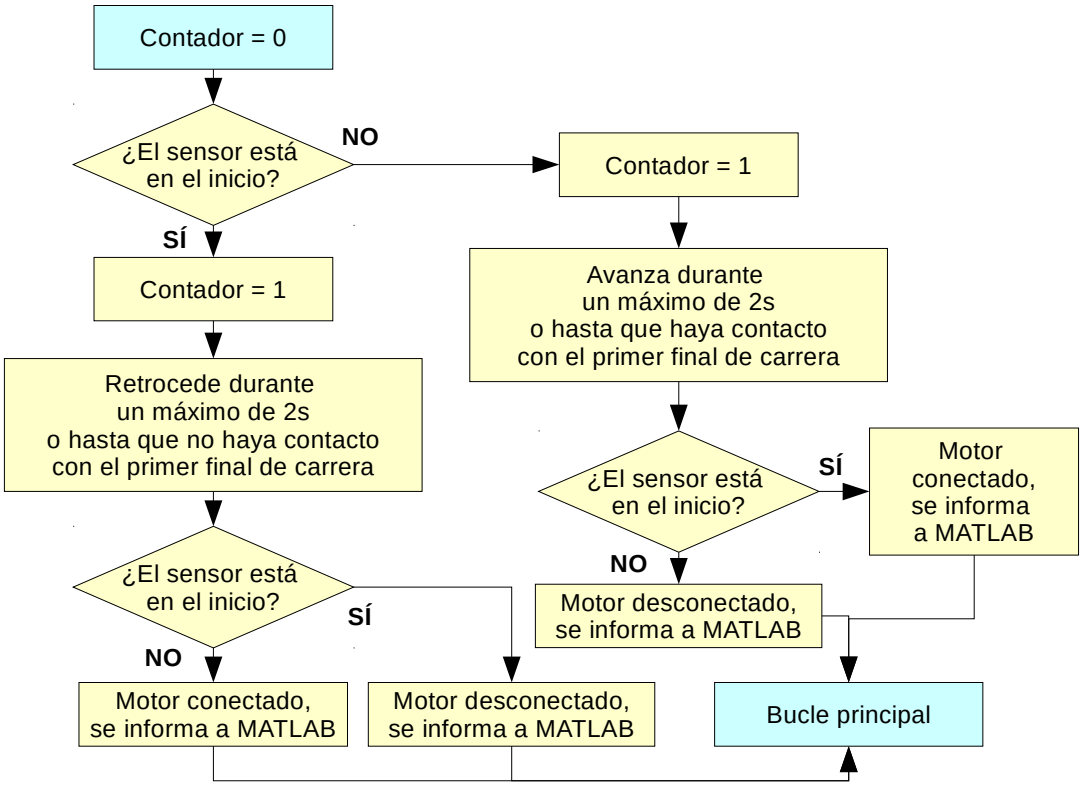


Ilustración 121: Diagrama de flujo del programa para comprobar si el motor está conectado.

– *Función de calibración RGB:*

Este parte del programa está destinada al establecimiento de unos valores de referencia del rojo, verde, azul y negro para el tratamiento de los datos en MATLAB. Para ello, se toman los datos del sensor y se envían por conexión serial al programa de MATLAB, como se muestra en el siguiente diagrama de flujo.7

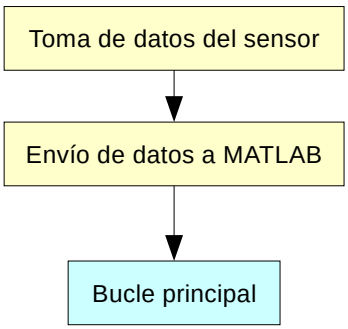


Ilustración 122: Diagrama de flujo del programa de calibración RGB.

– *Función de lectura única:*

Este código es muy parecido al anterior, puesto que está destinado a tomar los datos del sensor y enviarlos a MATLAB. La diferencia radica en que esta función también envía los datos de la temperatura de color, la iluminación y la coordenada c. Las dos primeras han de ser calculadas en Arduino anteriormente.

El siguiente diagrama de flujo resume la explicación anterior.

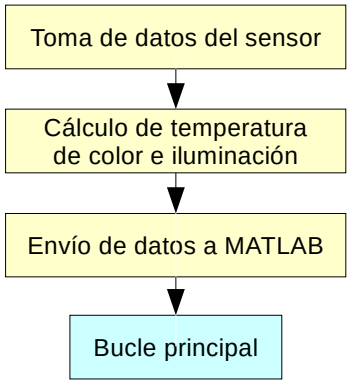


Ilustración 123: Diagrama de flujo del código para la lectura única.

Así, queda explicado el funcionamiento general del programa que se escribió para Arduino. El código completo y explicado se encuentra en el anexo C.

4.3.4 Software de MATLAB

Este apartado está destinado a explicar el desarrollo de una interfaz de usuario para el manejo del prototipo mediante el software MATLAB.

Con esa finalidad, se comenzará con el diseño gráfico de la interfaz, que se realizó mediante el complemento GUIDE de MATLAB. Se continuará con la generación de del código automático de la interfaz, para pasar seguidamente a la explicación de las distintas partes del programa. Para acabar, se mostrará el procedimiento que se siguió para convertir el software obtenido a formato “exe” y se enumerarán los pasos a seguir para instalar el programa acabado en cualquier ordenador.

Diseño de la interfaz gráfica con GUIDE

Como se ha mencionado anteriormente, el primer paso de este apartado consistirá en la elaboración de una interfaz gráfica para el manejo del prototipo. En el punto 3.4.2 se habló brevemente del software que se empleará con tal fin. Para recapitular, se trata de una ventana a la que se pueden arrastrar los distintos controles (botones, casillas de marcación) y editar sus propiedades visuales.

En la parte superior del programa se colocó el título, un botón de ayuda y un botón para cerrar la ventana de la interfaz (véase la siguiente imagen).

Ilustración 124: Parte superior de la interfaz gráfica.

Para la interfaz a desarrollar se optó por agrupar las distintas funciones en paneles, explicados a continuación.

– *Panel de conexiones:*

  Contiene los controles necesarios para administrar las conexiones y las comunicaciones del programa. Para ello, cuenta con un botón para reintentar la conexión, uno para resetear la posición del motor y tres ventanas que muestran el estado de la comunicación con el puerto USB, con Arduino y con el motor (véase la siguiente imagen).

Ilustración 125: Panel de conexiones.

– *Panel de escaneo:*

  Como su nombre indica, desde este panel se controla la toma de datos del dispositivo. Para ello, cuenta con un botón de escaneo, uno para añadir los datos de un

nuevo escaneo al anterior, uno para cargar datos en el programa y un último para guardar los datos en formato Excel (véase la siguiente imagen).

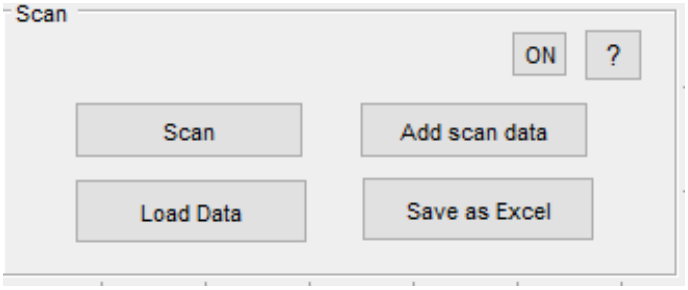


Ilustración 126: Panel de escaneo.

– *Panel de visualización y calibración:*

Desde este panel se pueden graficar los resultados del escaneo. Para ello, se cuenta con un botón para generar gráficas y seis variables a seleccionar. Además, se pueden cargar datos. También se puede aplicar una calibración a los datos del escaneo y guardarla en formato Excel (véase la siguiente imagen).

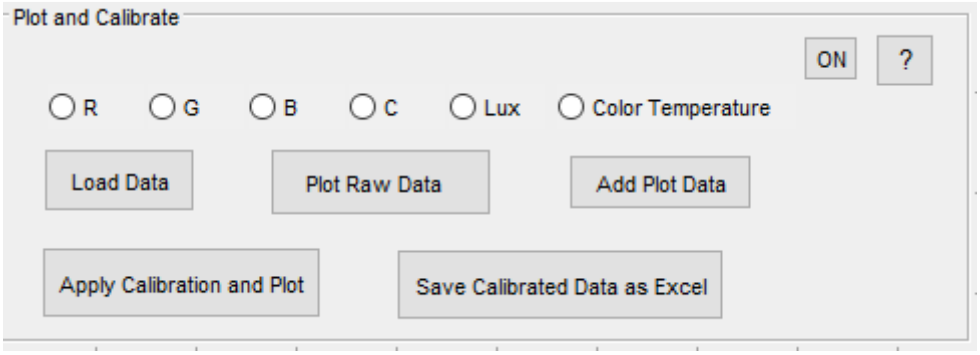


Ilustración 127: Panel de visualización y calibración RGB.

– *Panel de calibración RGB:*

Contiene cuatro botones para la lectura de los patrones de rojo, verde, azul y negro. Un recuadro indica cuando se ha completado la lectura. También cuenta con un botón para guardar la calibración, como se muestra a continuación.

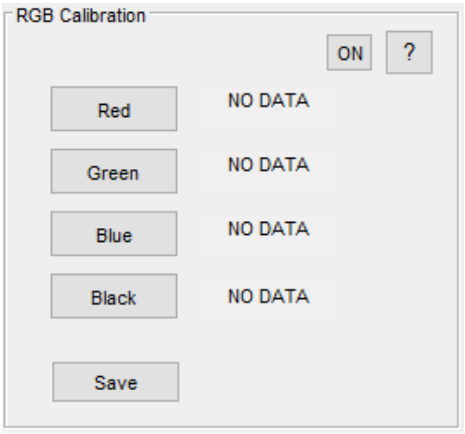


Ilustración 128: Panel de calibración RGB.

– *Panel para el análisis de los datos/panel de mínimos:*

En el programa se le llama panel de mínimos, puesto que está pensado para encontrar dichos valores en los datos obtenidos. Con tal fin contiene un botón para cargar datos, un panel para el ajuste de la resolución, un botón para buscar mínimos, otro para graficarlos, uno para guardar los resultados en una hoja Excel y, finalmente, uno para copiar los mínimos al portapapeles del ordenador. En éste último se puede elegir si se desea mantener la etiqueta de los datos (R, G y B), como se muestra en la siguiente imagen.

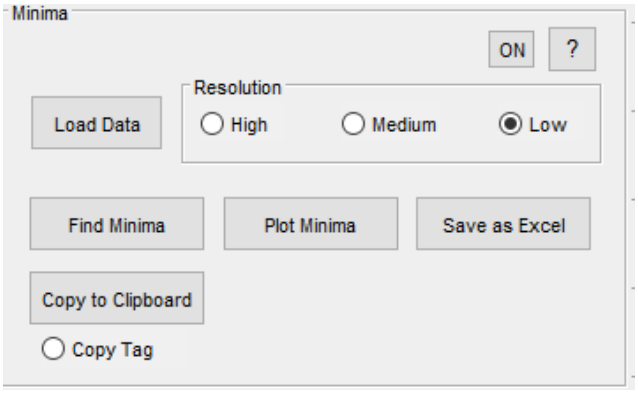


Ilustración 129: Panel para el análisis de los datos.

– *Panel de lectura única:*

Pensado para una toma rápida y sencilla de los datos, este panel cuenta con un botón de lectura y un espacio reservado a la visualización de los datos. También

tiene un control para copiar los resultados al portapapeles del ordenador, con o sin su etiqueta (véase la siguiente ilustración).

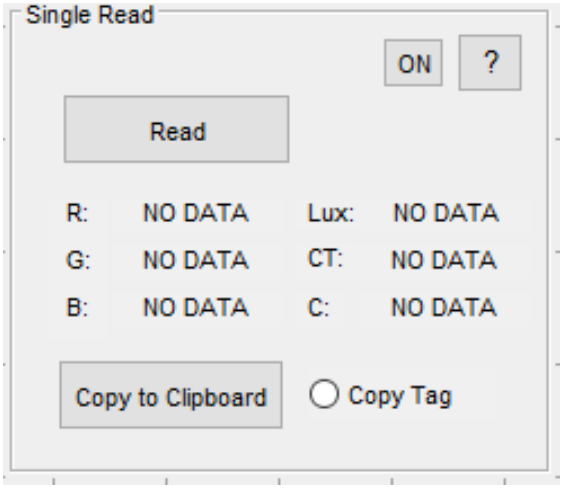


Ilustración 130: Panel de lectura única.

Para finalizar, se muestra una captura de pantalla de la interfaz resultante.

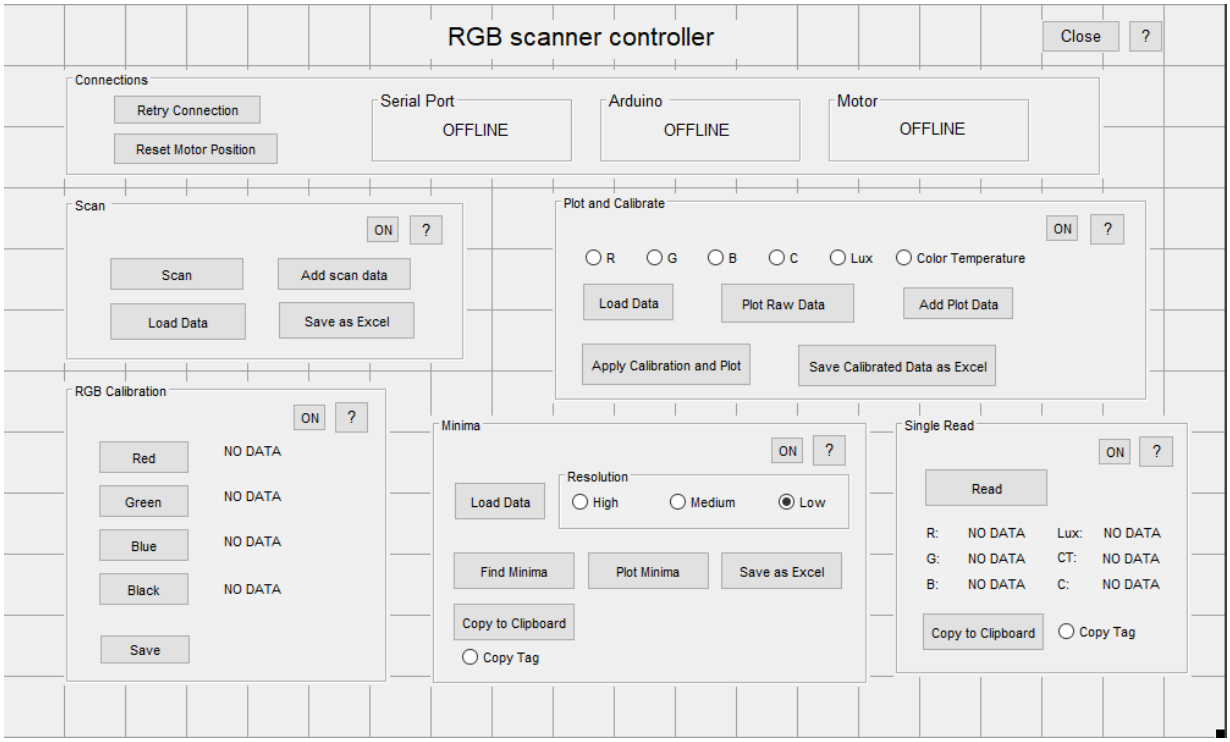


Ilustración 131: Interfaz elaborada en MATLAB GUIDE.

Código MATLAB

El mero hecho de guardar la interfaz realizada en el apartado anterior genera un archivo de código MATLAB cuyos controles, sin embargo, no realizan ninguna acción. El objetivo de este apartado es modificar el programa para poder controlar el prototipo y analizar sus lecturas. El código completo se puede encontrar en el anexo D. Aquí se explicará únicamente su funcionamiento con la ayuda de diagramas de flujo cuando éstos sean necesarios.

Para ello, antes de centrarse en el funcionamiento de los paneles anteriormente explicados, conviene echar un vistazo al funcionamiento general del programa. El código de la interfaz de usuario comienza con una inicialización, en la que se crean las variables y los archivos necesarios para el correcto funcionamiento del programa. A continuación, se comprueban las conexiones. Según el estado de las mismas, será posible activar determinados paneles. Finalmente, se esperan acciones del usuario, es decir, la activación de un panel o de una función que esté fuera de los paneles (botón de “Salir” y de “Ayuda”). Para evitar el cruce de datos, se escribió el programa de forma que sólo pudiese haber un panel activo. Además, mientras el programa esté trabajando, se bloquean los controles. En el diagrama de flujo de la ilustración 132 se muestra el funcionamiento general del programa.

Una vez se ha visto el funcionamiento general del programa, se puede centrar la atención en el código de sus funciones individuales. Se comenzará con el código de inicialización, para pasar después a los botones “libres” y a los diferentes paneles.

- *Código de la inicialización:*

  El primer paso de la inicialización es borrar todas las variables que pueda haber en el espacio de trabajo de sesiones anteriores. A continuación, se comprueba si hay un sistema de carpetas para guardar los datos de las lecturas. Si no lo hay, se crea uno. A continuación, se borran todos los puertos activos y se desactivan los controles de usuario. En este punto tiene lugar la generación de la interfaz gráfica del programa. Seguidamente, se pasa a comprobar el estado de las conexiones.

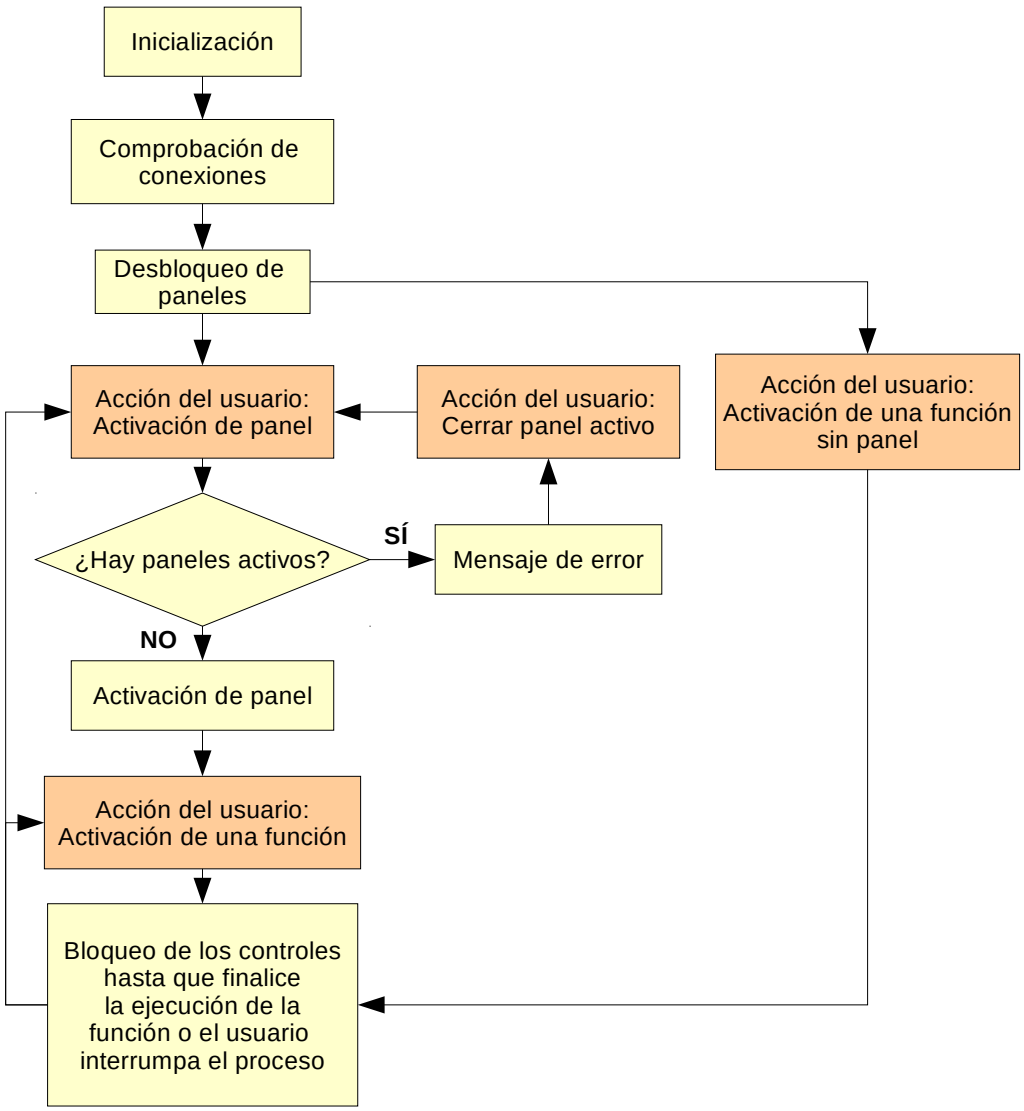


Ilustración 132: Diagrama de flujo del programa de MATLAB.

Para ello, en primer lugar se comprueba si hay conexión con el puerto serial, en segundo lugar si dicha conexión es con el programa correcto de Arduino y en tercer lugar si hay conexión con el motor. En función del estado de las conexiones se podrán activar determinados paneles. Concretamente, si no hay ninguna conexión se podrán utilizar los paneles para el tratamiento y la visualización de los datos, si hay conexión con el puerto serial y con Arduino se podrán usar además los del calibrado RGB y la lectura única y, finalmente, si también se encuentra conectado el motor será posible desbloquear todos los paneles. El estado de conexiones se muestra en el panel destinado a ellas. El diagrama de flujo que corresponde al código de inicialización se muestra en la ilustración 133.

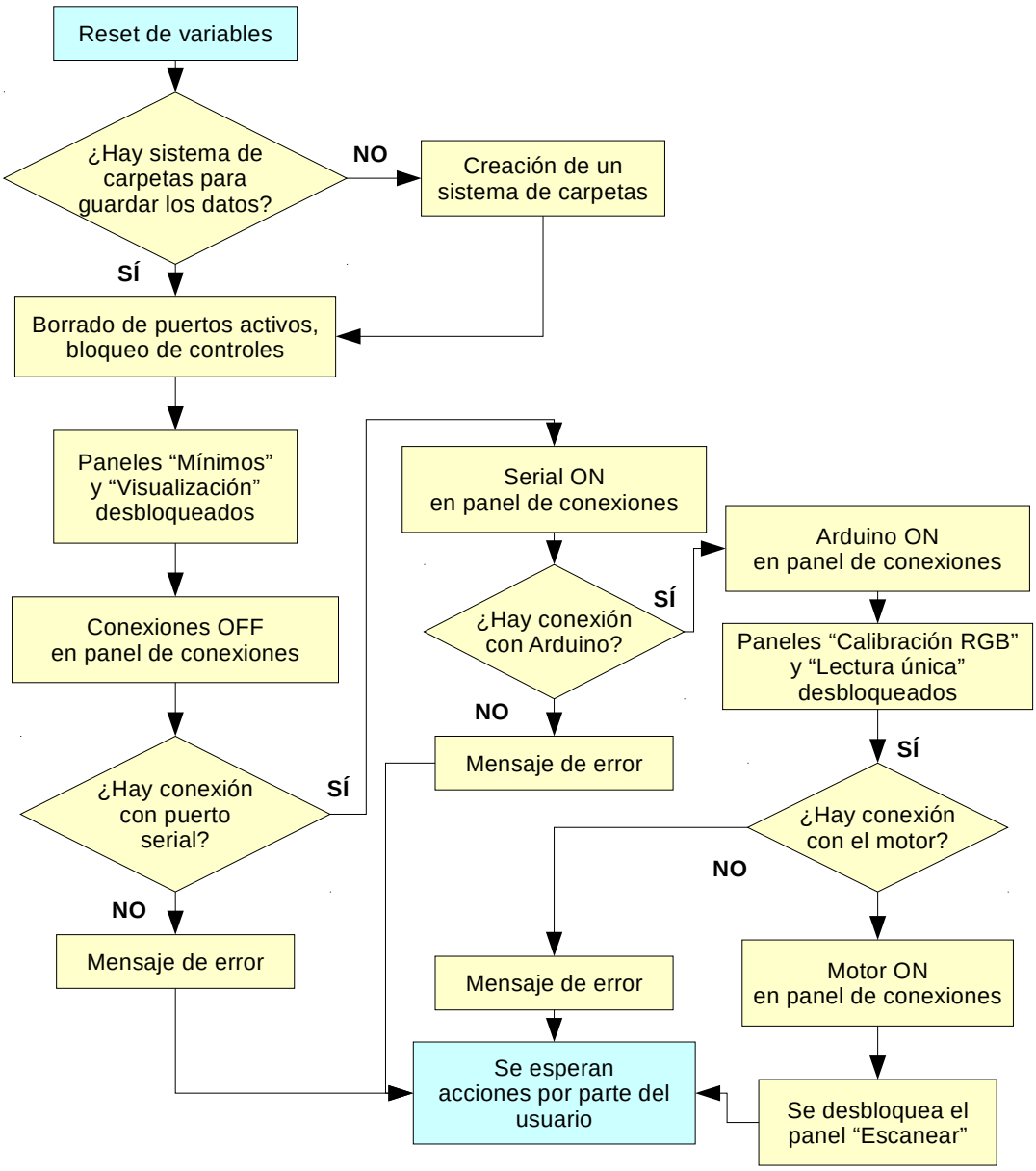


Ilustración 133: Inicialización del programa de MATLAB.

– *Código de los botones “libres”:*

Son el botón de “salir” y el de “ayuda”, situados a la derecha del nombre del programa.

El botón de ayuda muestra un mensaje en el que explica brevemente cómo hacer funcionar el programa. Por ello, no se requiere un diagrama de bloques para su explicación.

El código del botón de salir genera un cuadro de diálogo que pregunta al usuario si es verdad que pretende salir. Si se selecciona la opción “no”, se volverá a la

interfaz del programa. Si se selecciona “sí”, se borrarán los puertos activos y todas las variables y se cerrará la interfaz de usuario. Aunque también se puede salir mediante la cruz roja situada en la esquina superior derecha superior, se recomienda usar la función aquí explicada, puesto que resetea todas las variables que haya generado el programa durante su ejecución. A continuación, se muestra el diagrama de bloques del botón “salir”.

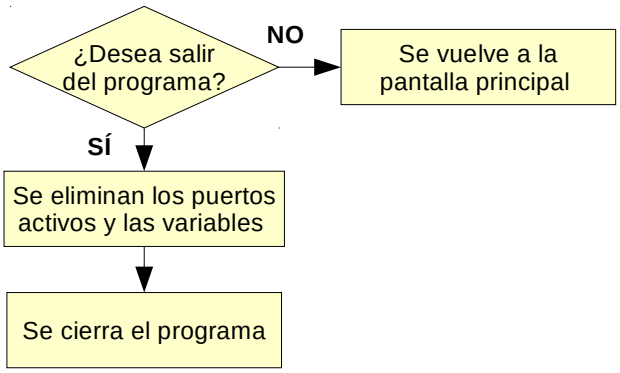


Ilustración 134: Diagrama de bloques del botón “salir”.

– *Código del panel de conexiones:*

Como se ha visto anteriormente, el estado de las conexiones se regula desde la inicialización, aunque se muestre en este panel. Quedan los botones de “reintentar la conexión” y “resetear la posición del motor”.

En el primer caso, se repite la rutina de comprobación de las conexiones, ya explicada con anterioridad.

En el segundo caso, se envía por serial el carácter “3”, que activa en el Arduino el programa que hace que el motor vuelva al inicio del recorrido. Cuando esto ha ocurrido se recibe el carácter “1” de Arduino. Entonces, se informa al usuario de que se ha llevado a cabo el reseteo del motor con éxito.

– *Código de escaneo:*

El panel del escaneo cuenta con cuatro botones: uno para escanear, otro para añadir datos, otro para cargar datos y un último para guardar los resultados en una hoja Excel.

Los dos últimos constan únicamente de una función de MATLAB y sólo realizan la acción ya mencionada, por lo que no requieren más explicación.

El botón del escaneo envía por serial a Arduino una orden para activar la rutina de lectura de datos. Seguidamente, se recibe el resultado de la calibración. Si ésta se ha realizado con éxito, comienza la toma de datos, que Arduino envía a MATLAB. Una vez finalizado el escaneo, tiene lugar la validación de las lecturas en Arduino, que comunica el resultado a MATLAB. Si éste es positivo, se guardan los datos en el sistema de archivos del software. A continuación se encuentra el diagrama de flujo del botón de escaneo.

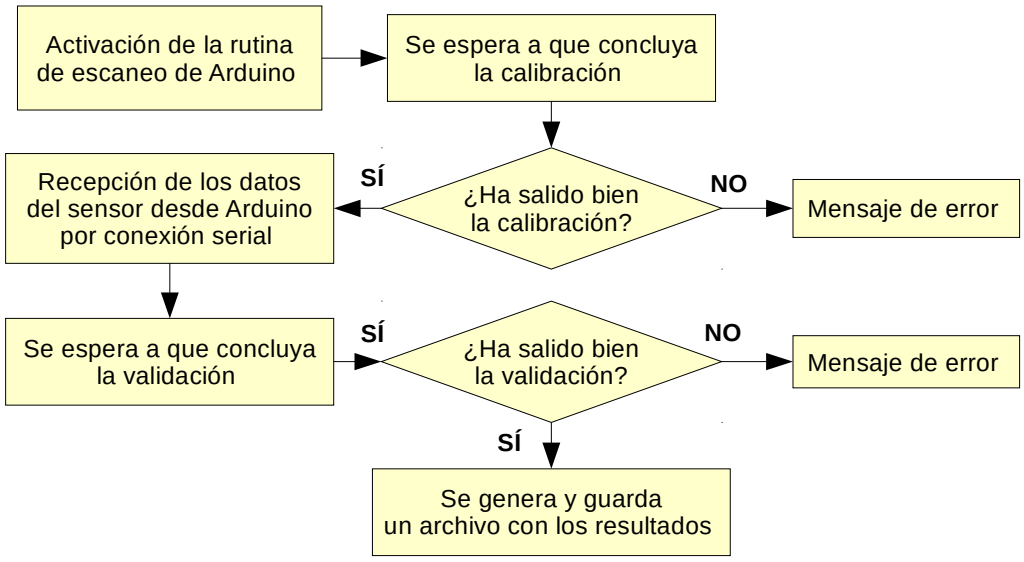


Ilustración 135: Diagrama de flujo del botón de escaneo.

El código del botón “añadir datos escaneados” es prácticamente idéntico. La diferencia radica en que los datos obtenidos se añaden a los tomados en la lectura anterior antes de guardarlos.

La jerarquía es la siguiente: en un principio sólo se pueden ejecutar los botones “cargar datos” y “escaneo”. Una vez cualquiera de estos dos se ha accionado, se tiene acceso a los otros dos botones.

– *Código de visualización y calibración:*

Este panel consta de seis botones circulares para elegir las variables que se mostrarán en las gráficas, un botón para cargar datos, uno para generar gráficas, uno

para añadir datos, otro para aplicar una calibración y un último para guardar los datos calibrados en una hoja Excel.

El funcionamiento del botón para cargar datos es el mismo que se ha visto en la explicación del panel anterior.

El código para generar gráficas funciona de la siguiente forma: se comprueban las variables que ha seleccionado el usuario y se elabora la gráfica correspondiente a partir de los datos de las lecturas.

La función de añadir datos permite juntar los datos de un archivo a seleccionar con los que ya hay en el espacio de trabajo.

La función de calibrado y graficado aplica un ajuste de los datos a partir de un archivo de calibración a elegir. Dicho archivo contiene los valores de las lecturas de patrones de color rojo, verde, azul y negro. Si se tiene en cuenta de que el espacio RGB es equiparable al cartesiano, se comprende que los datos obtenidos pueden ser expresados mediante vectores con las componentes R, G y B. En este código se aplica una transformación de coordenadas a los datos del escaneo para expresar los datos en un sistema con los vectores base del archivo de escaneo. Es decir, se ajustan los datos para que el rojo no sea (255, 0, 0) sino el valor real obtenido en la calibración. Esto permite hacerse una idea más precisa de los colores escaneados.

Una vez llevada a cabo esta transformación de coordenadas, se grafican los resultados.

Éstos se pueden guardar en una hoja Excel mediante el botón correspondiente.

La jerarquía en este panel es la siguiente: una vez se ha activado el botón de cargar datos o si ya hay datos en el espacio de trabajo, se tiene acceso a todas las funciones menos a la de “guardar datos calibrados en hoja Excel”, que sólo se desbloquea si se lleva a cabo una calibración.

– *Código de calibración RGB:*

El resultado de la ejecución completa de este panel es un archivo de calibración que contiene los colores rojo, verde, azul y negro y que podrá ser utilizado para

un ajuste de los datos mediante la transformación de coordenadas anteriormente explicada.

Para ello, cuenta con un botón para cada uno de los colores y uno para guardar los resultados. La activación del botón de cada color envía por serial a Arduino una orden de lectura única. A continuación, Arduino envía los datos leídos a MATLAB. Entonces, la casilla que se encuentra a la derecha del botón correspondiente se vuelve verde. Sólo se puede generar un archivo de calibración si hay lecturas para todos los colores necesarios, es decir, para el rojo, el verde, el azul y el negro.

A continuación, se muestra el diagrama de flujo del botón rojo, verde, azul y negro (el mismo sirve para los cuatro) y el del botón de guardar.

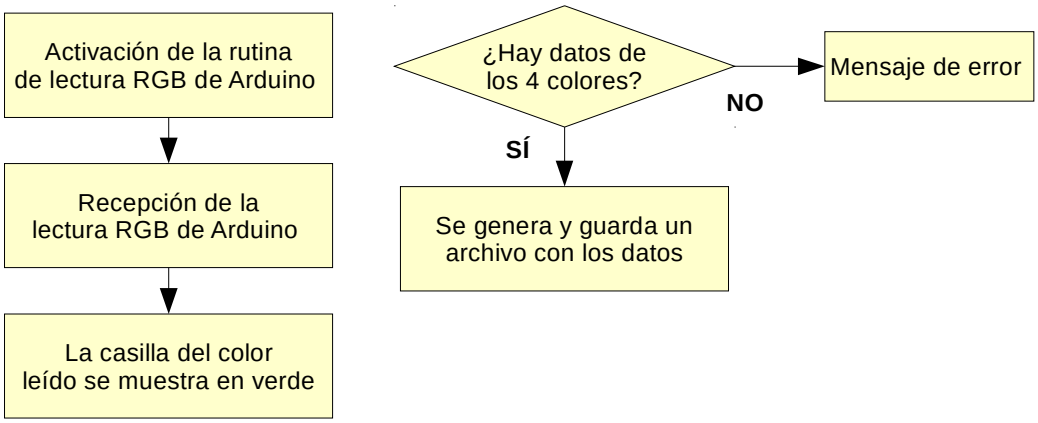


Ilustración 136: Diagrama de flujo de los cuatro botones para la lectura del patrón de color (izquierda) y para la generación de un archivo de calibración (derecha).

– *Código de búsqueda de mínimos:*

Consta de las siguientes rutinas. En primer lugar, la opción de carga de datos, que permite seleccionar un archivo para la extracción de los mínimos. En segundo lugar, el código de “buscar mínimos” extrae los valores mínimos de los datos que hay en el espacio de trabajo según la resolución seleccionada. La resolución se refiere principalmente al tamaño de los mínimos en relación con la media de los datos. En tercer lugar, el botón “graficar mínimos” genera una gráfica con los

valores R, G y B de los datos del espacio de trabajo con los mínimos que se encontraron anteriormente señalados. Hay un botón para guardar los mínimos en una hoja Excel. Finalmente, también hay una opción que permite copiar los mínimos obtenidos al portapapeles. Si se selecciona la opción “con etiqueta”, cada columna de mínimos llevará su nombre correspondiente (mínimos de la coordenada R, G o B, respectivamente).

La jerarquía en este panel es la siguiente: si hay datos en el espacio de trabajo o se cargan datos se desbloquea la opción de buscar mínimos. Una vez ésta se ha ejecutado se tiene acceso al resto de opciones.

– *Código de la lectura única:*

Al presionar el botón de lectura, se activa la rutina de lectura única de Arduino. Los datos que lee el sensor se reciben por el puerto serial y se muestran en las casillas correspondientes. Se pueden copiar los valores de la lectura al portapapeles, con o sin su denominación gracias a la casilla “etiqueta”. El botón para copiar datos al portapapeles se desbloquea una vez se ha llevado a cabo una lectura o si hay datos de una lectura anterior que no se eliminaron del espacio de trabajo.

Conversión del programa a formato “exe”

Una vez se han explicado las funciones para el manejo del programa para controlar el prototipo, se puede pasar a explicar cómo se llevó a cabo la transformación del resultado a formato “exe”.

Con tal fin se empleó la herramienta “deploytool” de MATLAB. Una vez acabado el código de la interfaz de usuario, se escribe “deploytool” en la consola de comandos de MATLAB y se presiona la tecla Enter.

Se selecciona la opción “compilador de aplicaciones” y en la ventana que aparece se seleccionan los archivos necesarios para que el programa diseñado funcione correctamente.

Adicionalmente, se puede diseñar una pantalla de carga y un icono para la aplicación. A continuación se muestran los que se realizaron para el presente trabajo.

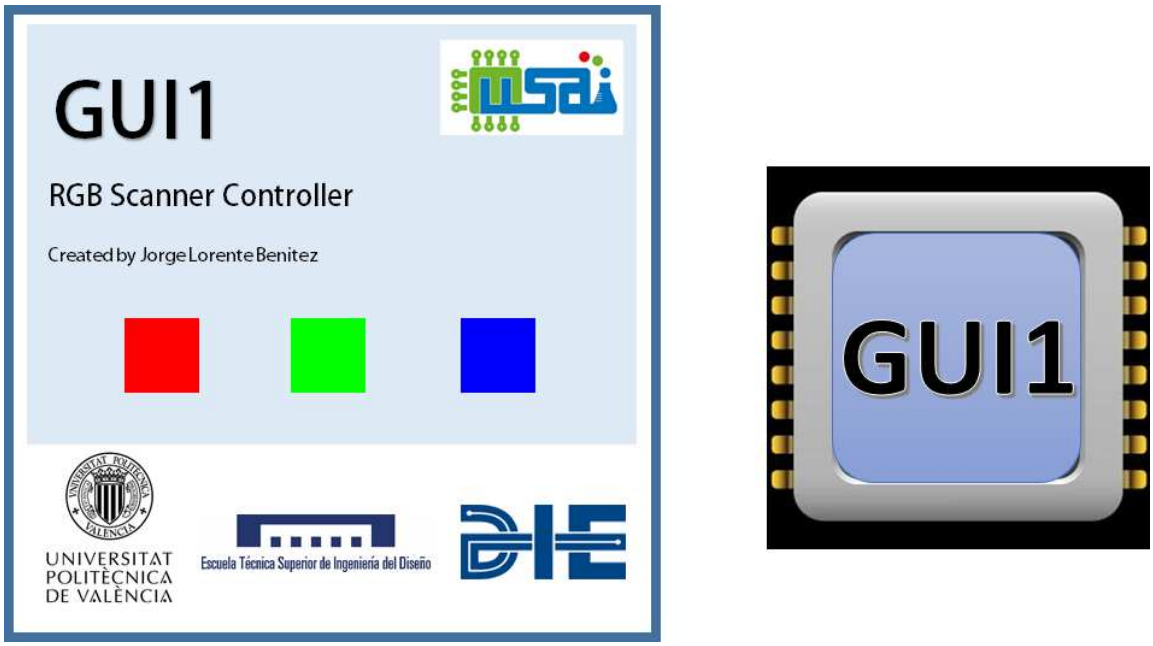


Ilustración 137: Pantalla de carga de la aplicación para manejar el prototipo (izquierda) y su icono (derecha).

Antes de continuar, se puede seleccionar si se desea que en la instalación de la aplicación generada se descarguen los archivos de MATLAB necesarios para su correcto funcionamiento de Internet o que éstos se incluyan en el ejecutable. En caso de seleccionar la segunda opción, el archivo de instalación será de un tamaño bastante mayor.

Una vez hecho esto, se selecciona la opción empaquetar y se espera a que ésta finalice. Como resultado, se obtiene una carpeta con los archivos necesarios para la instalación de la aplicación diseñada en cualquier sistema informático.

5 Caracterización del sistema

En este punto se cuenta con un prototipo completamente funcional y el software necesario para manejarlo. El siguiente paso consistirá en caracterizar el sistema y los resultados que proporcione al análisis de determinadas muestras. En el caso óptimo, los resultados de la caracterización del sistema serán casi idénticos a los de la caracterización del sensor.

Este capítulo se dividirá en dos apartados, en el primero se expondrán los parámetros a exponer y el modo de proceder para ello. En el segundo se mostrarán y discutirán los resultados obtenidos.

5.1 Materiales y métodos

Para la caracterización del sistema se comprobó el comportamiento para las siguientes variables: la detección del fondo, el intervalo de lectura, la distancia entre muestras, la longitud y anchura de las mismas, las dimensiones óptimas de la muestra, la detección del color, la resolución, la transparencia, la reproducibilidad y la iluminación. A continuación, se explica el procedimiento que se siguió para cada parámetro.

Detección del fondo

Para caracterizar la detección del fondo en el prototipo, se realiza un escaneo sin muestra. Es decir, los resultados obtenidos deberían caracterizar la superficie uniforme de cartón pluma de color blanco que se usa para presionar las muestras contra el cristal. Lo óptimo sería obtener tres líneas horizontales.

Para valorar la calidad de la detección del fondo, se calculará la media de los resultados de cada coordenada, su desviación estándar y su desviación estándar relativa. Según la literatura [49], el último parámetro no debería superar el 3 % para una correcta caracterización de un fondo uniforme.

Intervalo de lectura (distancia recorrida entre lecturas)

Como se ha visto el apartado de la caracterización del escáner, el área de detección es bastante grande (3mm x 7mm) en relación con el array de fotodiodos. Con la finalidad de

optimizar el tiempo de escaneo y la precisión de los datos obtenidos, se realizaron medidas de dos muestras en las que se varió la distancia que recorre el sensor entre las lecturas.

Se realizaron medidas con los siguientes intervalos de lectura: 0.25 mm; 0.5 mm; 1.0 mm; 1.5 mm; 2.0 mm; 2.5 mm y 3.0mm. Al ser la longitud del área de detección 3 mm, un intervalo de lectura mayor significaría la pérdida de datos.

Se realizaron las medidas sobre dos muestras, elaboradas con el software Microsoft PowerPoint, impresas en folios blancos convencionales mediante un dispositivo comercial. La muestra 1 consta de rayas de 3 mm de ancho y 10 de altura. El espaciado aumenta de 1mm a 10mm de forma progresiva. Nótese que la muestra no cabría en el escáner, pero que gracias a la función “Añadir escaneo” es posible partir la muestra en varios trozos, leerlos de uno en uno y juntarlos en un archivo.

Ilustración 138: Muestra 1.

La muestra 2 consta de rayas de anchura variable (de 1 mm a 10 mm) separadas 10 mm entre sí.

Ilustración 139: Muestra 2.

Se analizarán las gráficas obtenidas para los distintos intervalos, fijándose en la calidad de los datos en relación con el tiempo de lectura.

Distancia entre muestras

Una vez conocido el intervalo óptimo de escaneo, se pueden tomar los resultados del apartado anterior correspondientes a la muestra 1 y utilizarlos para encontrar la distancia óptima entre muestras. Como se ha mencionado anteriormente, la muestra 1 consta de rayas de 3 mm de anchura y 10 mm de altura dispuestas de forma que la distancia entre ellas varíe de 1 mm a 10 mm.

Se analizarán las gráficas obtenidas para ambas muestras con el fin de determinar la distancia óptima entre muestras. Esta vez, habrá que tener en cuenta la estabilización de los mínimos correspondientes a cada muestra así como la detección del fondo.

Longitud de las muestras

Aquí encuentran aplicación los resultados del apartado del intervalo de lectura para la muestra 2. Ésta consta de rayas de longitud variable (de 1 mm a 10 mm) situadas a una distancia de 10 mm entre sí.

De nuevo, habrá que fijarse en la estabilización de los mínimos y en la correcta detección del fondo.

Anchura de las muestras

En este apartado se comprobará la influencia de la anchura de las muestras en los resultados y se buscará la óptima para trabajar con el sistema. Para ello, se efectuaron medidas de la muestra 3, que consta de varias rayas de una longitud de 3 mm y anchura variable (de 1 a 11 mm) a 10 mm de distancia las unas de las otras. Esta muestra también se imprimió sobre un folio convencional y se preparó mediante el PowerPoint.

Ilustración 140: Muestra 3.

Los parámetros a optimizar en los resultados son la detección del fondo y la estabilización y máxima “profundidad” de los mínimos.

Límites de detección

Una vez caracterizadas las dimensiones de la muestra, cabe comprobar cuál es el límite de tamaño para la obtención de buenos resultados. Para comprobarlo, se realizaron medidas con la muestra 4, que consta de cuadrados con lados de 1 a 10 mm separados 10 mm entre sí.

Ilustración 141: Muestra 4.

A continuación, se evaluarán los resultados de los valores mínimos óptimos y los valores mínimos detectables. En la siguiente imagen se encuentran las muestras a medir. Ambas se realizaron en base a los resultados obtenidos para variables anteriores, que aún no han sido presentados.

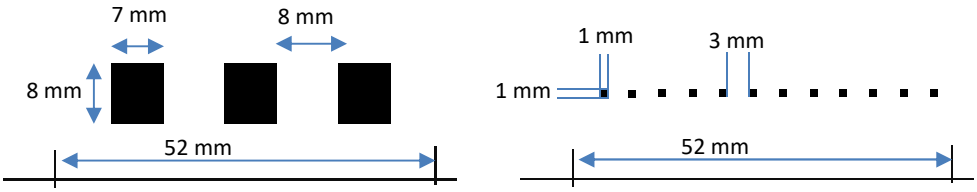


Ilustración 142: Muestra 5 con parámetros mínimos óptimos (izquierda) y muestra 6 con parámetros límite (derecha).

Para evaluar si ha tenido lugar una correcta detección de la muestra, se calculará la media de los mínimos de la misma. A dicha media se le restara el valor de la media obtenida en las lecturas del fondo. El valor calculado se dividirá entre la desviación estándar del fondo. Si el factor que resulte de las anteriores operaciones es mayor a 3, se puede afirmar que la detección de las muestras correspondientes ha sido exitosa, puesto que los resultados no podrían ser confundidos con el error del fondo.

Detección del color

Con el fin de evaluar la capacidad del sensor para diferenciar colores, se llevó a cabo el escaneo de una muestra con los colores del arco iris (muestra 7). Ésta consta de rayas de 3 mm de anchura y 10 de altura. Al final, se le añadió una serie de grises.

Ilustración 143: Muestra 7.

Como en anteriores muestras, ésta se elaboró con el PowerPoint y fue impresa en un folio convencional. En la siguiente tabla, se muestran las coordenadas RGB de cada muestra según PowerPoint en la escala de 8 bits.

	R	**G**	**B**
	192	0	0
	255	0	0
	255	192	0
	255	255	0
	146	208	80
	0	176	80
	0	176	240
	0	112	192
	0	32	96
	112	48	160
	13	13	13
	22	22	22
	58	56	56
	117	113	113
	174	170	170

Tabla 10: Coordenadas RGB de la muestra 7.

Igual que en el apartado enterior, se juzgará que ha tenido lugar un correcta diferenciación del tono correspondiente cuando el valor obtenido de la diferencia entre la lectura del tono y la media del fondo dividido entre la desviación estándar del fondo sea mayor a 3.

Resolución RGB

Al igual que se hizo en la caracterización del TCS34725, se evaluará la resolución en la escala RGB del prototipo. Para ello, se elaboraron varias muestras con el PowerPoint. La primera consta de 7 rayas de 3 mm de longitud y 10 mm de anchura separadas 3 mm entre sí. Se comienza con el color negro (R=0, G=0, B=0) y se va subiendo el valor de las tres coordenadsa en 42.5 puntos por raya, como se muestra a continuación.

Ilustración 144: Muestra 8.

La siguiente muestra es igual que la anterior, pero se comienza en un tono gris (R=G=B=140) y se van aumentando la coordenadas en 10 puntos por raya.

Ilustración 145: Muestra 9.

Finalmente, la tercera muestra está constituída igual que las dos anteriores, pero comienza en un tono gris diferente (R=G=B=200) y se aumenta tan sólo un punto de las coordenadas RGB en cada raya. La diferencia apenas se aprecia a simple vista.

Ilustración 146: Muestra 10.

Para valorar los resultados de las tres muestras, se realizará una recta de calibrado de las coordenadas RGB obtenidas en relación con las generadas por ordenador para cada una. Se ha elegido el color gris por tener una representación equitativa de las tres coordenadas RGB.

Transparencia

De especial importancia para las aplicaciones químicas, para las cuales está pensado el prototipo del presente trabajo, son los cambios en la transparencia en un indicador para caracterizar concentraciones de un determinado analito.

Por ello, en este apartado se probará la capacidad del sensor para distinguir entre distintas transparencias de un mismo tono. La muestra 7 consta de 7 rayas de 3 mm de longitud y 10 mm de anchura separadas entre sí por 3 mm de fondo blanco. La separación se elige para simular la distancia entre muestras que se podría encontrar en una tira reactiva real. El tono es rosa oscuro (R=234; G=0; B=156), la muestra empieza con un 0% de transparencia que se va incrementando en un 12.5 % en cada raya hasta llegar al 75 %.

Ilustración 147: Muestra 11.

En la siguiente muestra se reduce el cambio en la transparencia: se comienza también con una del 0 %, pero el incremento de una raya a otra es del 5 % hasta llegar al 30 %.

Ilustración 148: Muestra 12.

Se realizó una tercera muestra en la que el incremento es de un 1 %, partiendo también de un 0 % en la primera raya.

Ilustración 149: Muestra 13.

Para evaluar las tres muestras, se elaboró una recta de calibrado para cada una con los resultados obtenidos y los valores de transparencia conocidos. En función de la calidad de la recta, se puede deducir también la calidad de las medidas.

Reproducibilidad

De especial importancia en cualquier sistema de medición es conocer la reproducibilidad de los datos obtenidos. Con ese fin se realizaron medidas de varias muestras de un mismo color. Cada muestra consta de siete rayas de un mismo color de 10 mm de anchura y 3 mm de longitud separadas 3 mm entre sí. La muestra 14 es verde (R=0; G=255; B=0), la muestra 15 es roja (R=255; G=0, B=0), la muestra 16 es azul (R=G=0; B=255) y tiene 8 rayas y la muestra 17 es de color rosa (R=234; G=0; B=156).

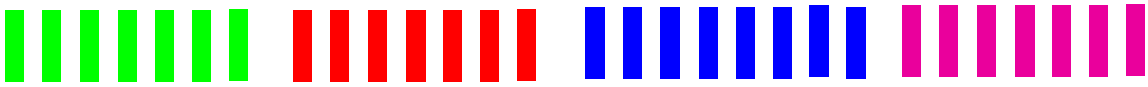


Ilustración 150: Muestra 14, muestra 15, muestra 16 y muestra 17 (de izquierda a derecha).

Dichas muestras fueron realizadas con el PowerPoint e impresas en folios convencionales. Para la evaluación de los resultados, se tendrán en cuenta la desviación estándar y relativa de las rayas de cada muestra. Ésta última no debería superar el 3 %.

Iluminación

El material con el que se fabricó la tapa del prototipo es translúcido, puesto que no había materiales opacos disponibles. Al tratarse de un escáner, la iluminación de la muestra es de vital importancia, por lo que hay que comprobar si las variaciones en la iluminación ambiental tienen repercusión en las medidas. Además, el Arduino y el driver del motor constan de LEDs que iluminan el interior del escáner y podrían influenciar los resultados.

Con la finalidad de averiguar y cuantificar estas interferencias, se realizaron medidas del fondo en distintas condiciones de iluminación: con luz ambiente elevada y los LEDs de los equipos electrónicos, a oscuras con los LEDS, con luz ambiente elevada y con los LEDs cubiertos y, finalmente, a oscuras y con los LEDs cubiertos. El fondo es un trozo de cartón pluma plano y de color blanco.

Se estudiará la desviación estándar y relativa del fondo en las distintas condiciones de iluminación y se eligirá la menor.

5.2 Resultados y discusión

Tras la presentación del procedimiento para la caracterización de determinados parámetros, en este apartado se procederá a mostrar y discutir los resultados obtenidos.

Detección del fondo

El primer paso de la caracterización del sistema consiste en la medida del fondo, es decir, los resultados del escáner sin muestra. Como se muestra en la ilustración 151, se obtiene una línea más o menos plana para todas las variables, de lo que se deduce una correcta detección del fondo. Las pequeñas oscilaciones se deben a imprecisiones debidas a la fabricación manual del sistema, que tienen como consecuencia una ligera inclinación del cristal del escáner, lo cual se refleja sobre todo en la coordenada G.

Si se calcula la desviación estándar de las medidas, se obtiene como máximo 268 en la coordenada verde. Si se tiene en cuenta que la escala del sensor va de 0 a 65535, se puede ver que esta desviación es muy pequeña. Calculando la desviación relativa de las coordenadas se obtiene como máximo un 0.82 %, que pertenece a la coordenada azul. La media del fondo, su desviación estándar σ_f y su desviación relativa $\sigma_{f,rel}$ se encuentran en la tabla 11 para cada coordenada.

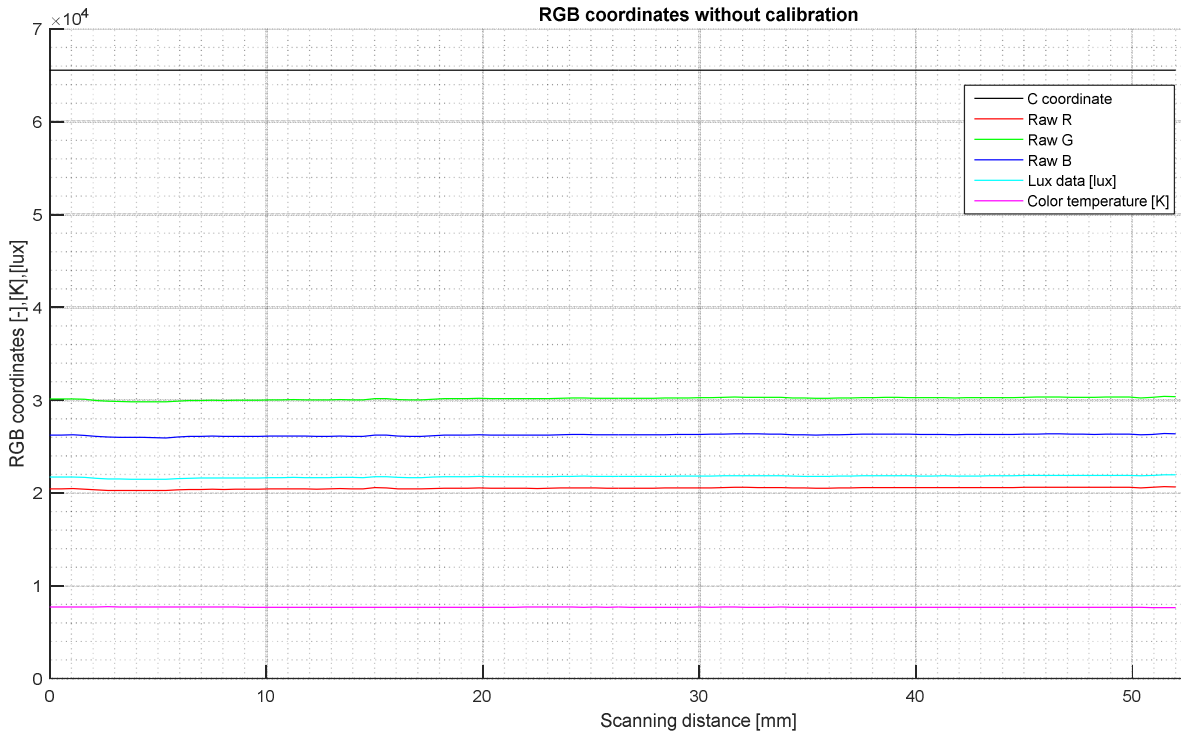


Ilustración 151: Resultados de la detección del fondo.

	R	G	B
Media	24064.102	36411.510	31341.327
σ_f	145.188	267.752	261.668
$\sigma_{f,rel}$ [%]	0.6	0.7	0.8

Tabla 11: Media, desviación estándar y desviación relativa del fondo.

Así, se puede afirmar que la detección del fondo se ha realizado con éxito y con un error menor al 3 % requerido.

Intervalo de lectura (distancia entre lecturas)

Para la muestra 1, los mejores resultados se obtienen para los intervalos de lectura más pequeños: 0.25 mm; 0.5 mm y 1 mm.

Resulta interesante, sin embargo, que las gráficas obtenidas para estos intervalos son muy parecidas. Es de suponer que esto se debe a la superposición de datos que se da puesto que el área de detección es mayor que la distancia recorrida por lectura. Así, todos los puntos de la muestra son leídos más de una vez. Lo que sí varía, sin embargo, es el tiempo de escaneo, como se muestra en la siguiente tabla 12. A menor distancia entre lecturas aumenta el tiempo de escaneo.

Intervalo de lectura	**Tiempo de escaneo**
0.25 mm	6-8 min
0.5 mm	3-4 min
1.0 mm	≈ 2 min

Tabla 12: Relación entre el intervalo de lectura y el tiempo de escaneo.

En la gráfica de 0.25 mm se aprecia cierto ruido en los datos, que desaparece en la gráfica de 0.5 mm (ilustración 152). Para un intervalo de 1.0 mm, la gráfica comienza a mostrar picos, lo que significa que la unión entre los datos empieza a sufrir de la pérdida de información. Teniendo en cuenta que el tiempo de escaneo del intervalo de 0.5 mm es bastante razonable y la calidad de los datos es buena en comparación con el resto, se puede elegir este intervalo como el óptimo para el escaneo. Además, hay que tener en cuenta que 0.5 mm se corresponde con la distancia entre hilos del tornillo, es decir, el intervalo elegido se corresponde con una vuelta completa del motor. A partir de 1.0 mm de recorrido de escaneo, la gráfica obtenida permite diferenciar cada vez menos las rayas, descritas por los mínimos. Además, la ubicación de los mínimos, que debería estabilizarse a lo largo del escaneo (puesto que todas las rayas son del mismo color), muestra variaciones muy grandes. Esto se debe a que la posición del sensor en el momento de la lectura de una raya es aleatoria. Así, en el momento de leer los datos de una raya, es posible que lea mucho blanco por encontrarse desplazado respecto a ella, mientras que en la siguiente podría leer el valor correcto de la raya por encontrarse justo encima. Este efecto aparece independientemente del intervalo de escaneo, pero su repercusión en las medidas aumenta a mayor intervalo de escaneo. En la ilustración 153 se pueden ver los problemas descritos anteriormente. El primer círculo rojo muestra cómo la falta de datos impide diferenciar las primeras rayas y el segundo cómo varía la lectura debido a la integración de demasiado blanco.

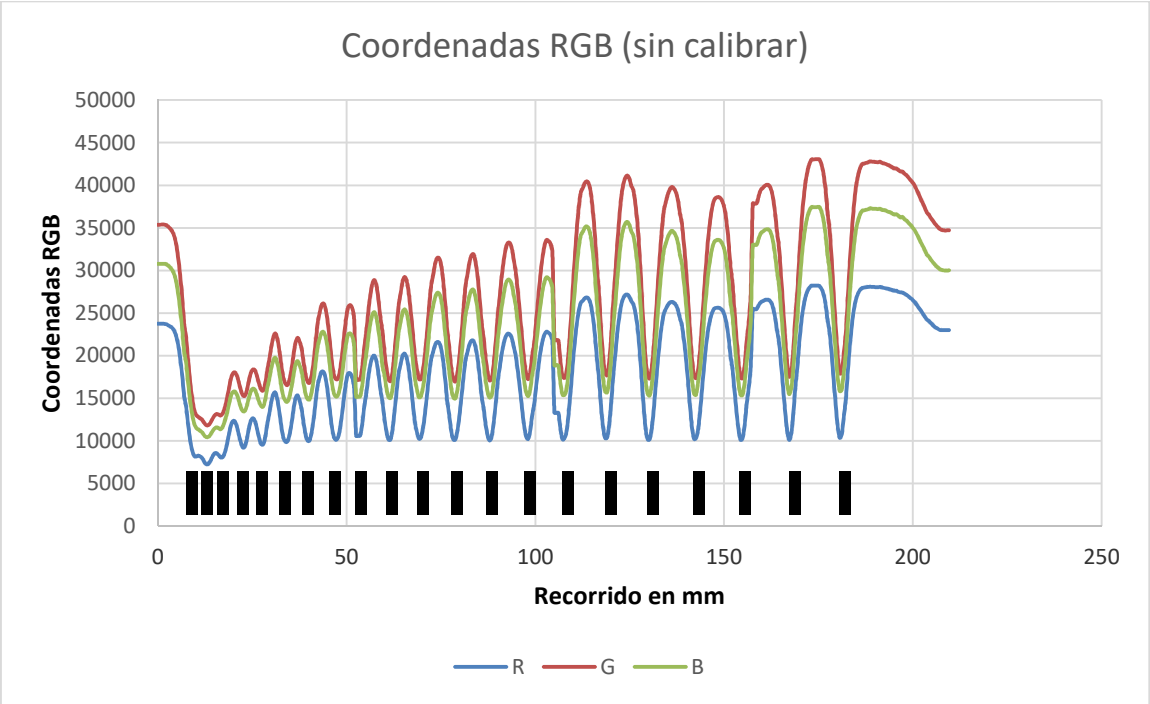


Ilustración 152: Coordenadas RGB de la muestra 1 para un intervalo de escaneo de 0.5 mm.

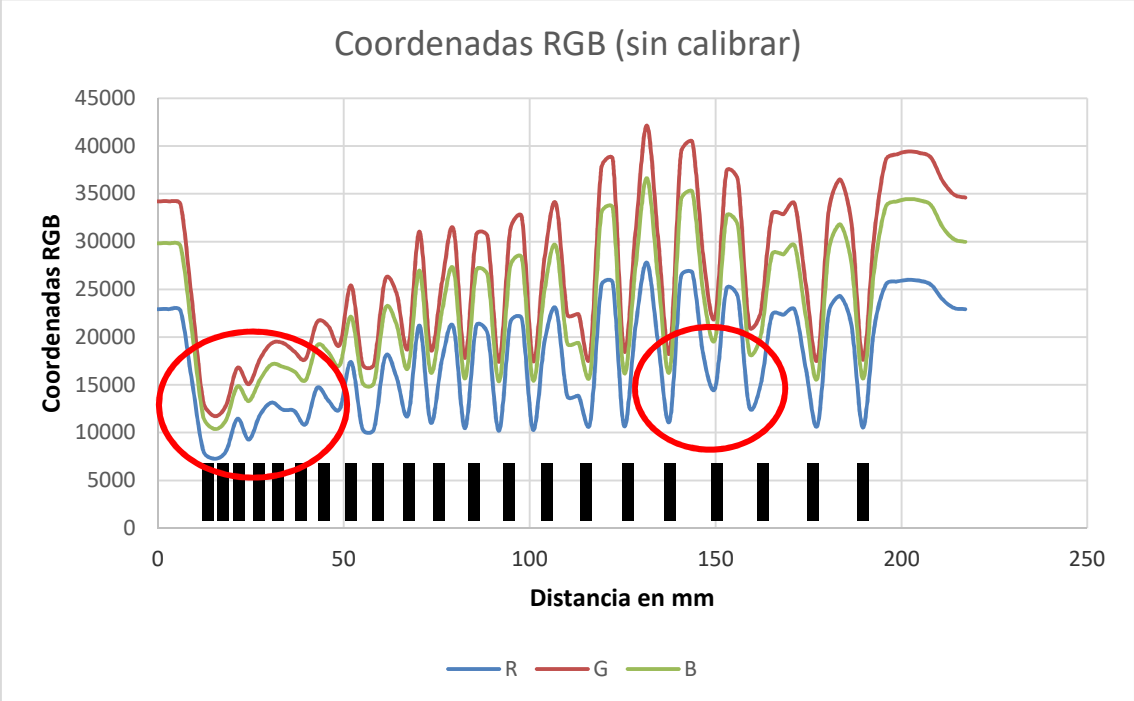


Ilustración 153: Datos de la muestra 1 para un intervalo de escaneo de 3 mm.

Las medidas de la muestra 2 se realizaron con la finalidad de comprobar la capacidad del sistema de distinguir muestras bastante alejadas entre sí. Los resultados muestran que, al contrario que para la muestra 1, todos los intervalos de escaneo permiten diferenciar con claridad las rayas. Sin embargo, las medidas muestran el mismo comportamiento observado con anterioridad: un intervalo de escaneo de 0.25 mm incorpora ruido y a partir de 1 mm las gráficas muestran picos debido a la falta de datos. Por lo tanto, se obtiene como intervalo de lectura óptimo 0.5 mm. A continuación, se muestra la gráfica correspondiente a los resultados obtenidos en la medida de la muestra 2 con el intervalo óptimo de lectura.

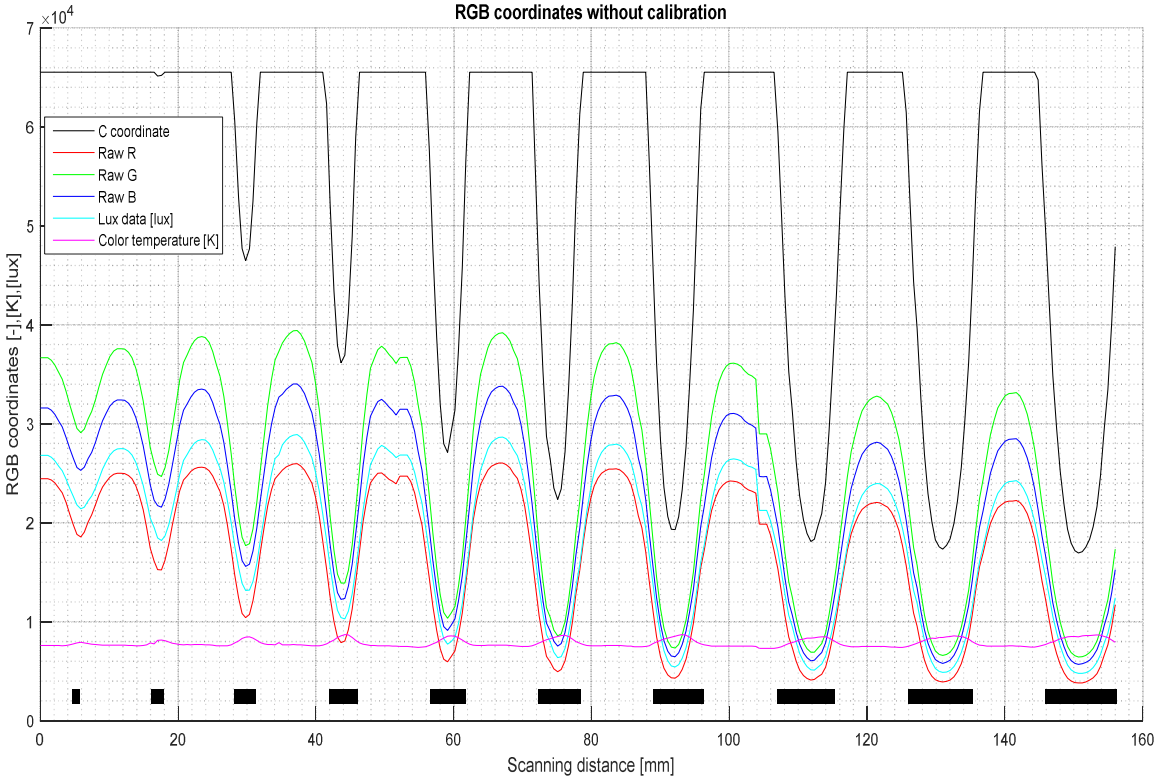


Ilustración 154: Resultados de las medidas de la muestra 2 con un intervalo de lectura de 0.5 mm.

Distancia entre muestras

Los resultados del escaneo de la muestra 1 se muestran de nuevo en la ilustración 155. En primer lugar, llama la atención que casi todos los mínimos se encuentran a la misma altura. Esto es debido a que todas las rayas son del mismo color. La excepción la constituyen las rayas que se encuentran a uno y a dos milímetros de distancia entre sí. La distancia

es tan pequeña que el sensor, al realizar la lectura de una raya, está tomando también lectura de las adyacentes. Al ser éstas de color negro, se obtiene un valor menor al real. Esto ocurre también en la lectura de los espacios blancos entre las rayas. Así, no es posible distinguir con claridad la ubicación de las rayas. Debido a esta contaminación de las lecturas, las distancias de 1 y 2 milímetros no son recomendables para este sistema. En la siguiente imagen los picos correspondientes a los espacios en blanco de estas distancias se encuentran redondeados en rojo.

Los dos siguientes picos muestran unos mínimos que ya se encuentran estabilizados, no obstante, los máximos cambian de valor. Esto significa que sigue existiendo contaminación debido a la integración de datos adyacentes a la zona de interés de la lectura, lo que se refleja en un blanco percibido inferior al real. Dado que el porcentaje del blanco percibido es menor al 50% del blanco real, las distancias de 3 y 4 mm tampoco son aconsejables para mediciones del color, aunque podrían bastar si se pretende conocer únicamente la ubicación aproximada de las muestras de color y el tono predominante de las mismas. Por ello, los picos correspondientes a estas zonas se han coloreado de naranja.

El párrafo anterior se aplica casi en su totalidad a los tres siguientes picos, en los que la única diferencia consiste en que los máximos contienen un porcentaje de las coordenadas del blanco real superior al 50%. Los máximos de estas zonas están rodeados de color amarillo.

En los tres últimos picos se obtiene un porcentaje de las coordenadas del blanco real superior al 80%, lo que significa que el sensor percibe los espacios entre las rayas con una contaminación mínima. Se puede observar que el blanco de la muestra tiene un valor superior al del fondo en sus tres coordenadas, lo que se debe por una parte a su menor distancia al elemento sensor y por otra parte a los distintos materiales de ambos. Dado que tanto las rayas negras como los espacios blancos entre las mismas se hallan suficientemente caracterizados en la gráfica, las distancias de 8, 9 y 10 mm son las más apropiadas para trabajar con este sistema. Por ello, sus máximos se han marcado con círculos verdes.

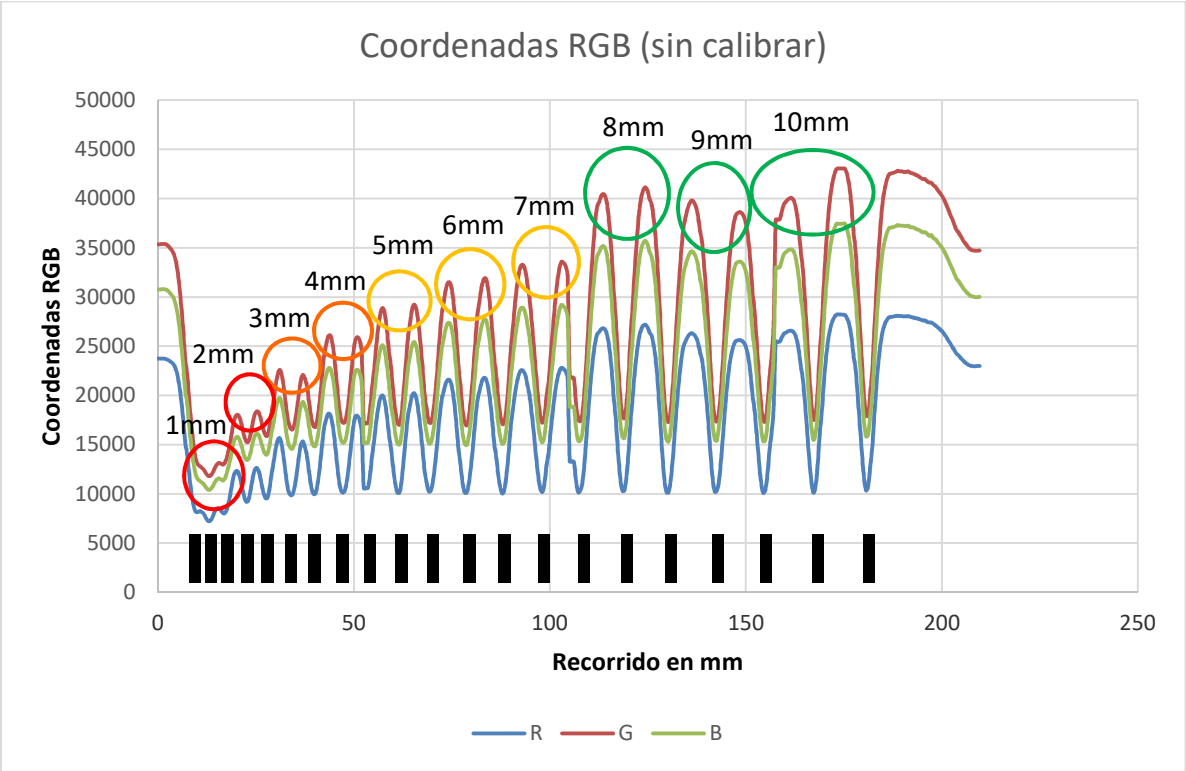


Ilustración 155: Coordenadas RGB de la muestra 1.

Longitud de las muestras

Como se indicó anteriormente, para la caracterización de la longitud se muestra de nuevo el gráfico con los resultados del escaneo de la muestra 2 (ilustración 156). Se observa que todas las rayas son detectadas, sin embargo, los mínimos no se estabilizan hasta llegar al séptimo (raya de 7 mm de anchura). Esto significa que hasta esa longitud las lecturas incorporan fondo, lo que significa que el tono percibido no es exactamente el negro, sino una mezcla del negro y el blanco del fondo. A mayor longitud de la raya, menor es la cantidad de blanco que se incorpora al resultado, hasta que es posible visualizar el color de la raya en la gráfica, lo que ocurre a partir de los 7 mm de anchura. Por ello, la longitud mínima de muestra para percibir un tono con claridad con este sistema es de 7 mm.

Sin embargo, es posible extraer conclusiones sobre cualquier longitud superior a 1 mm siempre y cuando se tenga conocimiento de las características del fondo y se mantenga la longitud de muestra constante, de forma que el fondo incorporado en las lecturas sea siempre el mismo y, por lo tanto, no tenga influencia sobre el problema tratado.

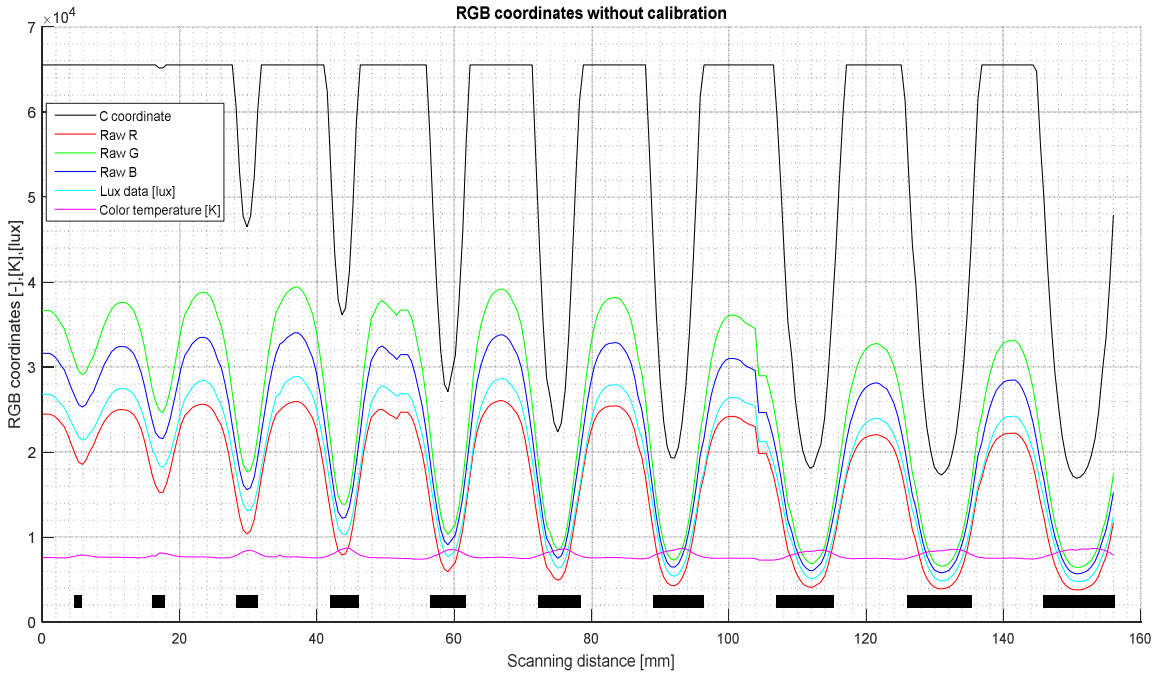


Ilustración 156: Resultados de las medidas de la muestra 2.

Anchura de las muestras

Los resultados de las medidas de la muestra 3 se encuentran en la ilustración 157. Se puede observar una gráfica bastante parecida a la del apartado anterior, dado que aunque el parámetro que varía es distinto, el comportamiento de las lecturas es el mismo. En los primeros valores se incorpora al resultado una gran cantidad de fondo, por lo que el valor de los mínimos es mayor de lo que debería. Los mínimos no se estabilizan hasta llegar al octavo pico, que tiene 8 mm de anchura. Por lo tanto, la anchura mínima para trabajar con una incorporación despreciable de fondo es 8 mm.

Sin embargo, como se ha mencionado anteriormente, es posible trabajar con anchuras de muestra inferiores siempre cuando este parámetro se mantenga constante y el fondo sea uniforme.

Límites de detección

Teniendo en cuenta los apartados anteriores, sería de esperar que la anchura y longitud mínimas de raya se encontrasen entre 7 y 8 mm. Para comprobarlo, se realizaron medidas con la muestra 4, que consta de cuadrados con lados de 1 a 10 mm separados 10 mm entre sí.

En la ilustración 158 se encuentran los resultados de las medidas. Como es de esperar, los mínimos se estabilizan a partir del octavo pico, que pertenece al cuadrado con 8 mm de lado. Por lo tanto, la longitud del lado de un cuadrado mínimo para trabajar con una incorporación despreciable de fondo es 8 mm.

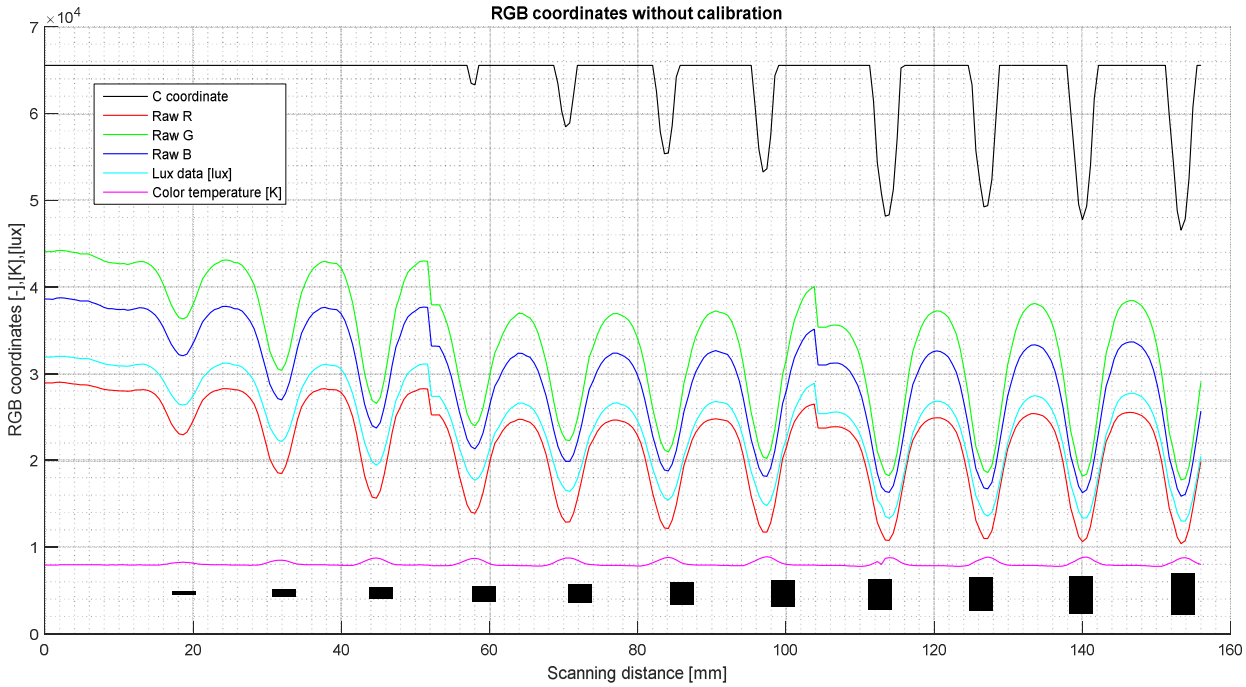


Ilustración 157: Medidas con anchura variable.

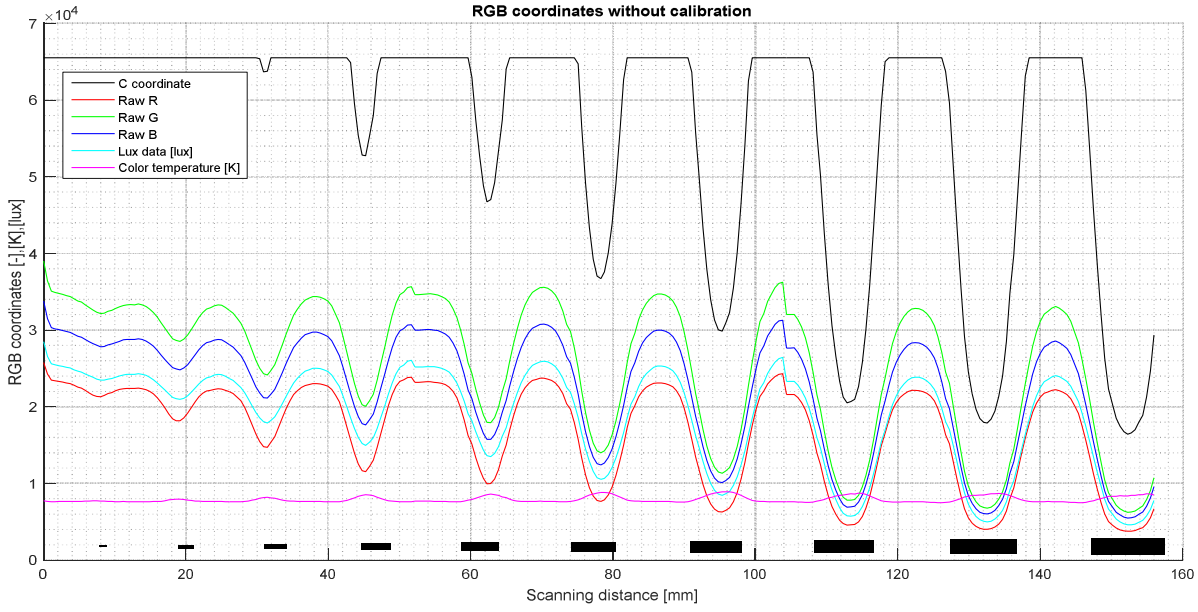


Ilustración 158: Resultados de las medidas de la muestra 4.

Sin embargo, cabe preguntarse hasta qué punto los datos obtenidos se pueden diferenciar del fondo. El motivo es que la señal para el cuadrado de 1 mm es la menor obtenida en las medidas hasta ahora. Con la finalidad de resolver esta cuestión se tomaron los valores del mínimo de este cuadrado, ya que son los menos contaminados por el fondo, y se comparó la diferencia Δ_m de la media del fondo y los valores del mínimo con la desviación estándar σ_f de la media del fondo. Los resultados se encuentran en la siguiente tabla.

	R	G	B
Media del fondo	24064.102	36411.510	31341.327
σ_f	145.188	267.752	261.668
Valores del mínimo	21305	32155	27817
Δ_m	2759.102	4256.510	3524.327
Δ_m/σ_f	19.004	15.897	13.469

Tabla 13: Resultados de los cálculos para la caracterización de la detección del primer mínimo de la muestra 4

Se observa que el cociente entre la diferencia entre la media del fondo y el mínimo con la desviación estándar del fondo es mayor a tres para todas las coordenadas, por lo que se puede afirmar que existe una correcta detección del cuadrado de 1 mm [49]. Por lo tanto, el resto de los datos también son buenos.

Así, las medidas revelan también que es posible trabajar en condiciones no óptimas. A continuación se evaluarán los resultados de los valores mínimos óptimos y los valores mínimos detectables.

Los resultados de las medidas de la muestra 5 se muestran en la ilustración 159. Se puede observar una correcta detección de todos los mínimos, así como un fondo estable y no contaminado por las muestras.
Si se calcula la diferencia Δ_m entre la media de los mínimos y la media del fondo y se divide este valor entre la desviación estándar del fondo σ_f, se demuestra que existe una clara diferenciación de los mínimos, puesto que el factor obtenido es mucho mayor a tres. Esto se puede apreciar en la tabla 14.

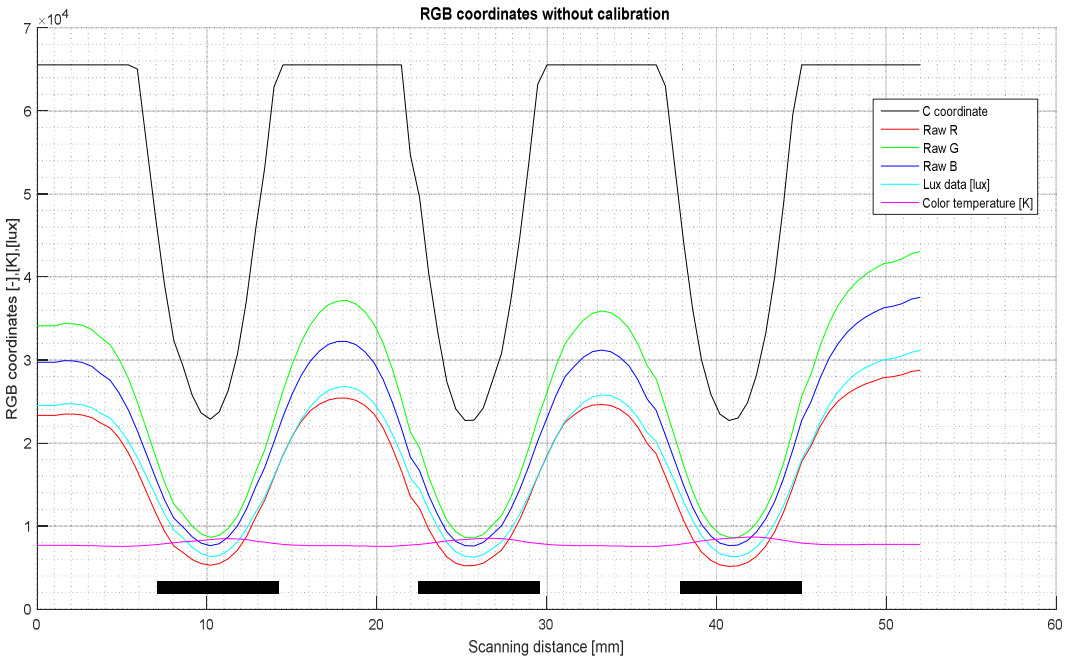


Ilustración 159: Resultados de la muestra 5.

	R	G	B
Media del fondo	33149.092	50948.112	43694.724
σ_f	170.599	307.170	296.796
Media del mínimo	5229.667	8623.333	7635.667
Δ_m	15270.823	21538.310	18592.997
Δ_m/σ_f	89.513	70.118	62.646

Tabla 14: Resultados de la caracterización de los resultados de la muestra 5.

El mismo procedimiento se repite para la muestra 6. En la gráfica correspondiente a sus resultados, situada a continuación, se puede observar que los mínimos son menos pronunciados, lo que indica que se incorpora una gran cantidad de fondo a los resultados. Se puede ver una tendencia ascendente en los mínimos, lo que indica una ligera inclinación ascendente del sistema mecánico en el momento de la medida.

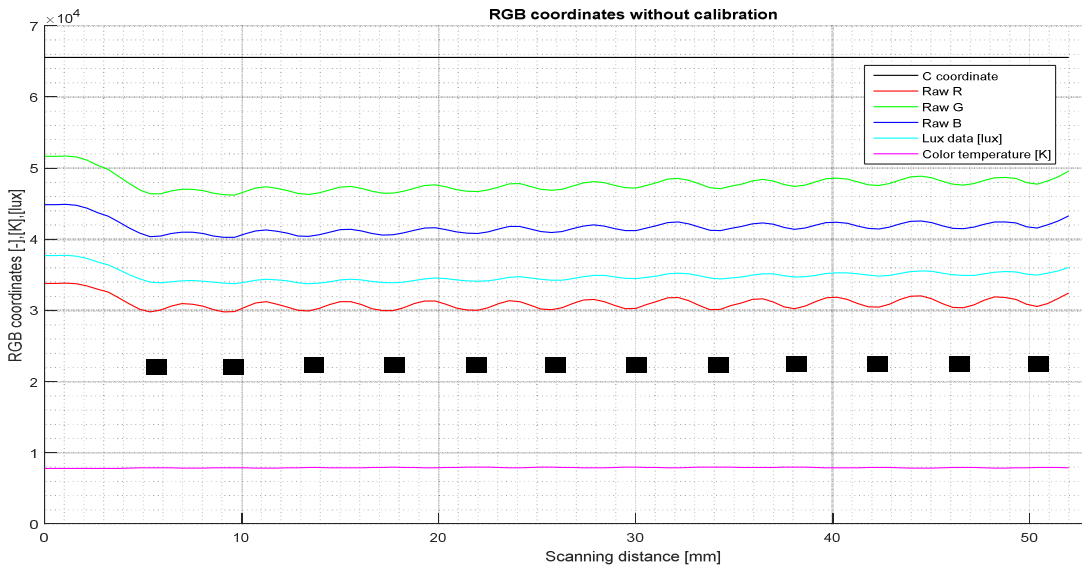


Ilustración 160: Resultados para la muestra 6.

El cálculo de la diferencia Δ_m entre la media de los mínimos y la media del fondo y su división entre la desviación estándar del fondo σ_f demuestra que el factor obtenido es en todos los casos mayor a tres, lo que indica una clara detección de los mínimos en la muestra B. Los resultados se muestran en la tabla expuesta a continuación.

	R	G	B
Media del fondo	33149.092	50948.112	43694.724
σ_f	170.599	307.170	296.796
Media del mínimo	30149.083	46974.167	40987.667
Δ_m	3000.009	3973.946	2707.058
Δ_m/σ_f	17.585	12.937	9.121

Tabla 15: Resultados de la caracterización de los resultados de la muestra 6.

Detección del color

Con el fin de evaluar la capacidad del sensor para diferenciar colores, se llevó a cabo el escaneo de una muestra con los colores del arco iris (muestra 7). Ésta consta de rayas de 3 mm de anchura y 10 de altura. Al final, se le añadió una serie de grises.

Como se puede ver en los resultados, que se encuentran en la siguiente imagen, el sensor distingue con claridad todos los colores, ya que cada raya tiene unos valores mínimos distintos para la mayoría de los parámetros. De los resultados se deduce que para distinguir el tono son importantes las coordenadas RGB, la iluminación y la temperatura de color. La coordenada C, que contiene toda la información anterior codificada, no permite distinguir los distintos colores. Así, queda demostrado que el sensor es capaz de distinguir distintos tonos de color con claridad.

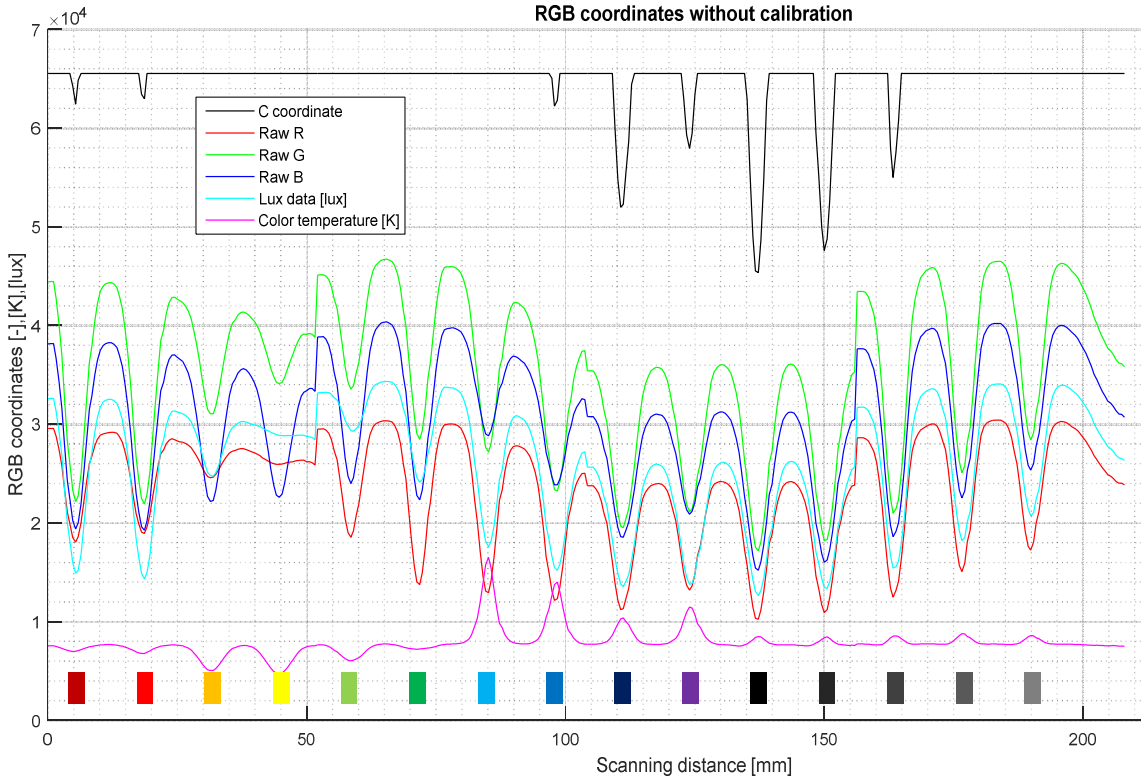


Ilustración 161: Resultados de la muestra 7.

De todos modos, es de ayuda calcular si los colores detectados son diferenciables del fondo. Para ello, se cuenta con la desviación estándar del fondo para cada una de las coordenadas (R, G y B), calculada anteriormente. Seguidamente, se extraen los mínimos de los resultados de la muestra 7, que representan los valores RGB de cada tono. A continuación, se calcula la desviación de estos mínimos con respecto a la media del fondo y se comprueba que ésta sea como mínimo el triple de la desviación del fondo. Esto permitiría distinguir los picos correspondientes a las muestras del ruido de fondo. La siguiente tabla resume los resultados de los cálculos aquí descritos.

	Mínimos: $\overrightarrow{RGB}$			Diferencia: $\Delta_m = \overline{Fd} - C$			Factor: α_{RGB}		
Color	R	G	B	Δ_{mR}	Δ_{mG}	Δ_{mB}	$\frac{\Delta_{mR}}{\sigma_{fR}}$	$\frac{\Delta_{mG}}{\sigma_{fG}}$	$\frac{\Delta_{mB}}{\sigma_{fB}}$
	18074	22167	19394	5990.1	14244.5	11947.3	41.26	53.20	45.66
	18946	21916	19271	5118.1	14495.5	12070.3	35.25	54.14	46.13
	24593	31064	22172	-528.9	5347.51	9169.33	-3.64	19.97	35.04
	25926	34102	22593	-1861.9	2309.51	8748.33	-12.82	8.63	33.43
	18545	33554	24008	5519.1	2857.51	7333.33	38.01	10.67	28.03
	13753	28464	22336	10311.1	7947.51	9005.33	71.02	29.68	34.42
	12937	27178	28823	11127.1	9233.51	2518.33	76.64	34.49	9.62
	12151	23250	23864	11913.1	13161.5	7477.33	82.05	49.16	28.58
	11224	19521	18535	12840.1	16890.5	12806.3	88.44	63.08	48.94
	13192	21135	20885	10872.1	15276.5	10456.3	74.88	57.05	39.96
	10249	17193	15227	13815.1	19218.5	16114.3	95.15	71.78	61.58
	10931	18226	16034	13133.1	18185.5	15307.3	90.46	67.92	58.50
	12481	20985	18608	11583.1	15426.5	12733.3	79.78	57.62	48.66
	15079	25101	22548	8985.1	11310.5	8793.33	61.89	42.24	33.60
	17243	28421	25344	6821.1	7990.51	5997.33	46.98	29.84	22.92

Tabla 16: Resultados de los cálculos para la caracterización de las medidas de la muestra 5.

En la tabla 16 Δ_m expresa la diferencia entre la media del fondo $\overline{Fd}$ de una coordenada y los mínimos de la coordenada RGB correspondiente, simbolizados por C. Además, Δ_{mR}, Δ_{mG} y Δ_{mB} son las diferencias entre la media del fondo y los mínimos de la muestra 5 para cada coordenada. Los factores α_{RGB} de se obtienen dividiendo la diferencia Δ_m de cada coordenada entre la desviación estándar σ_f correspondiente obtenida en las medidas del fondo. Como se observa en la tabla, estos factores son en todos los mínimos iguales o mayores a 3, por lo que se puede afirmar que existe una diferenciación clara de los datos con respecto al fondo.

Resolución RGB

Para la caracterización de la resolución RGB se comienza con la muestra de menor resolución (muestra 8), cuyos resultados se encuentran en la siguiente gráfica.

Se observa que el comportamiento es el esperado: los tonos más oscuros presentan unos valores inferiores en todas sus coordenadas RGB que los más claros. Conviene tener en cuenta que la detección de la última raya es algo imprecisa.

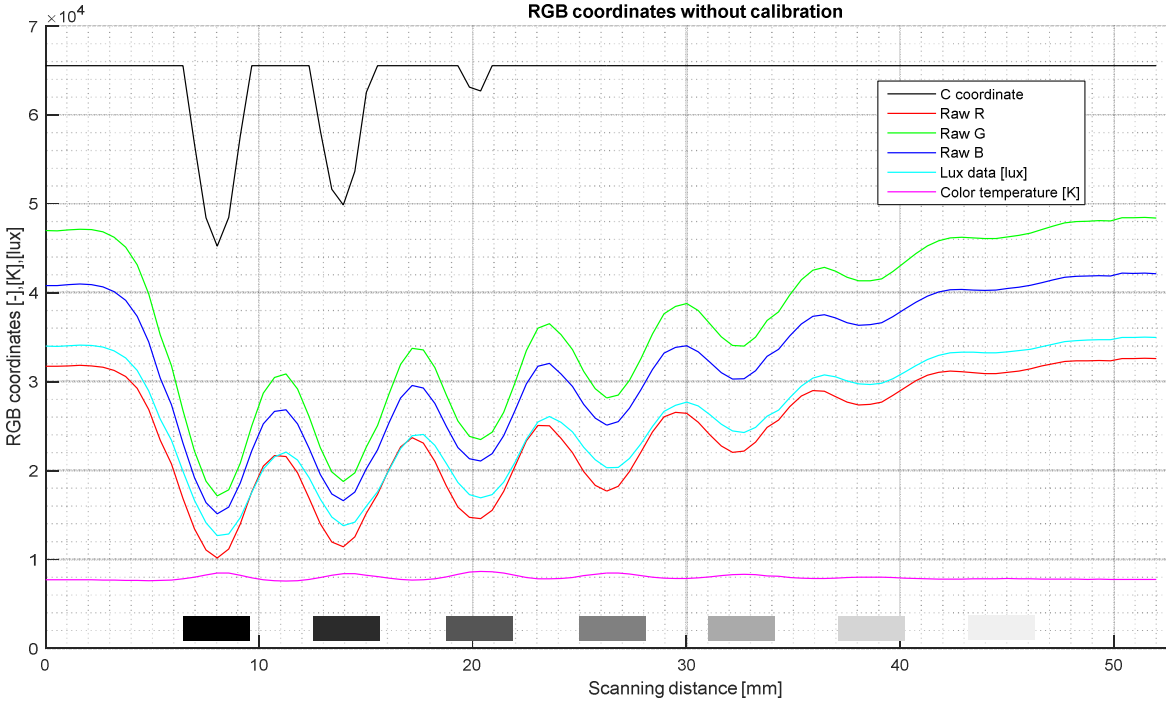


Ilustración 162: Resultados de la muestra 8.

El siguiente paso consiste en realizar una recta de calibrado con los resultados obtenidos y los valores RGB que se empleó en la elaboración de las muestras por ordenador. Para evaluar la calidad de dicha regla se tomará el coeficiente de determinación, que debería acercarse lo máximo posible a 1. Sin embargo, se considerará todo resultado superior a 0.7 como “aceptable”.

En la siguiente gráfica se muestran las rectas de calibrado de la muestra 8 con sus coeficientes de determinación. Aproximadamente se obtiene un 0.97 en las tres coordenadas, lo que es un resultado excelente. Sin embargo, si se calculan las rectas de calibrado sin el primer y el último valor, dado que éstos integran una mayor cantidad de fondo que el resto, se obtiene un coeficiente de determinación mínimo del 0.985 en la coordenada roja. Así, eliminando los valores más contaminados por el fondo se llega al 0.99 en el coeficiente de determinación. Por ello, se puede afirmar sin ninguna duda que la resolución de 42.5/255 es detectada sin problemas.

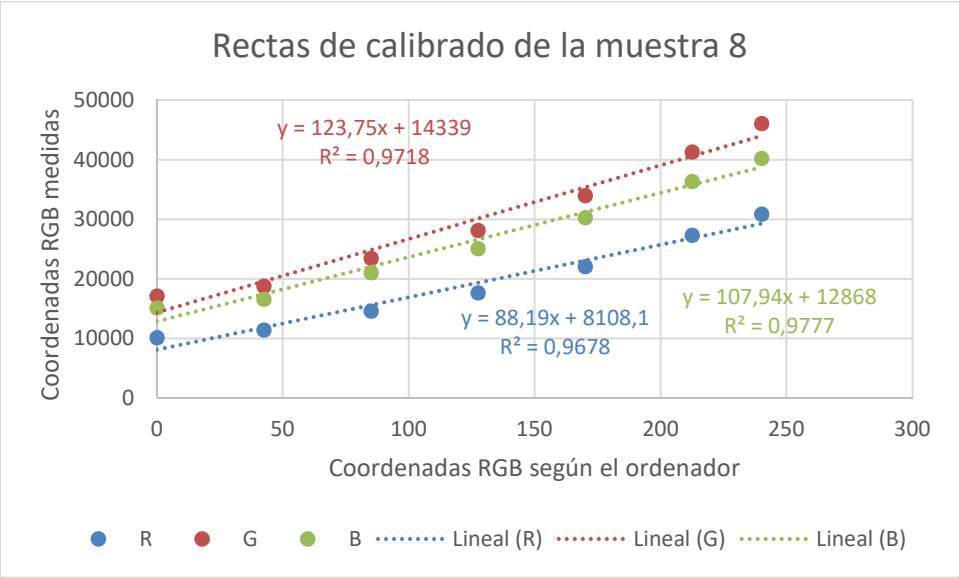


Ilustración 163: Rectas de calibrado de la muestra 8.

Seguidamente, se repitieron los mismos pasos con la muestra 9. A continuación, se muestran los resultados del escaneo. Se observa que, igual que antes, la tendencia ascendente de los mínimos es la que cabría esperar.

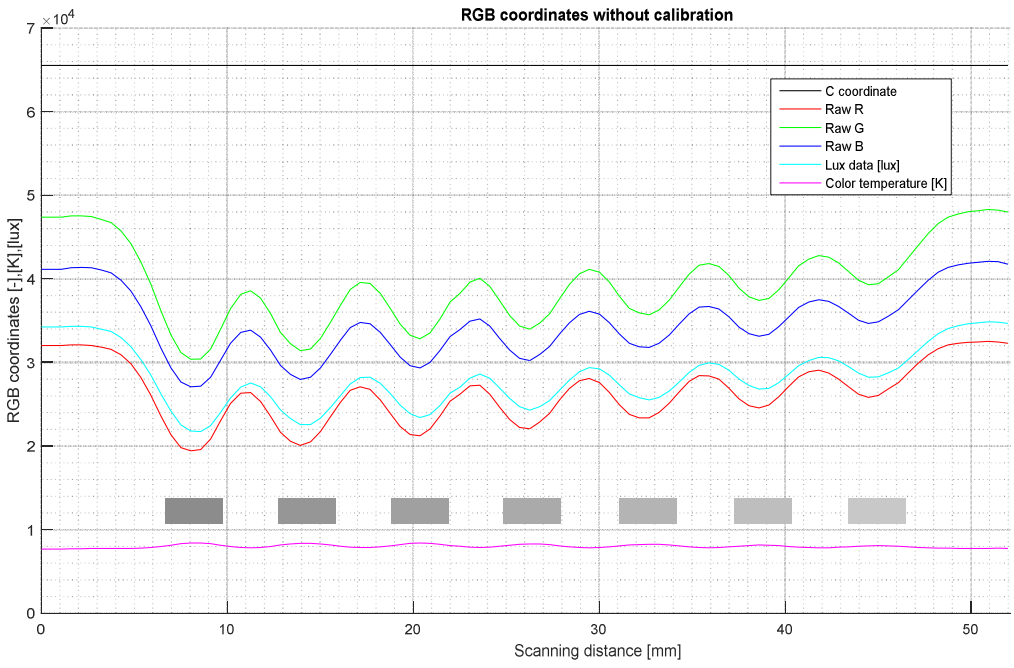


Ilustración 164: Resultados del escaneo de la muestra 9.

En la siguiente gráfica se encuentran los resultados de la regresión lineal de la muestra 9. El coeficiente de determinación es excelente. Si se eliminan el último y el primer mínimo, éste mejora, pero no es necesario.

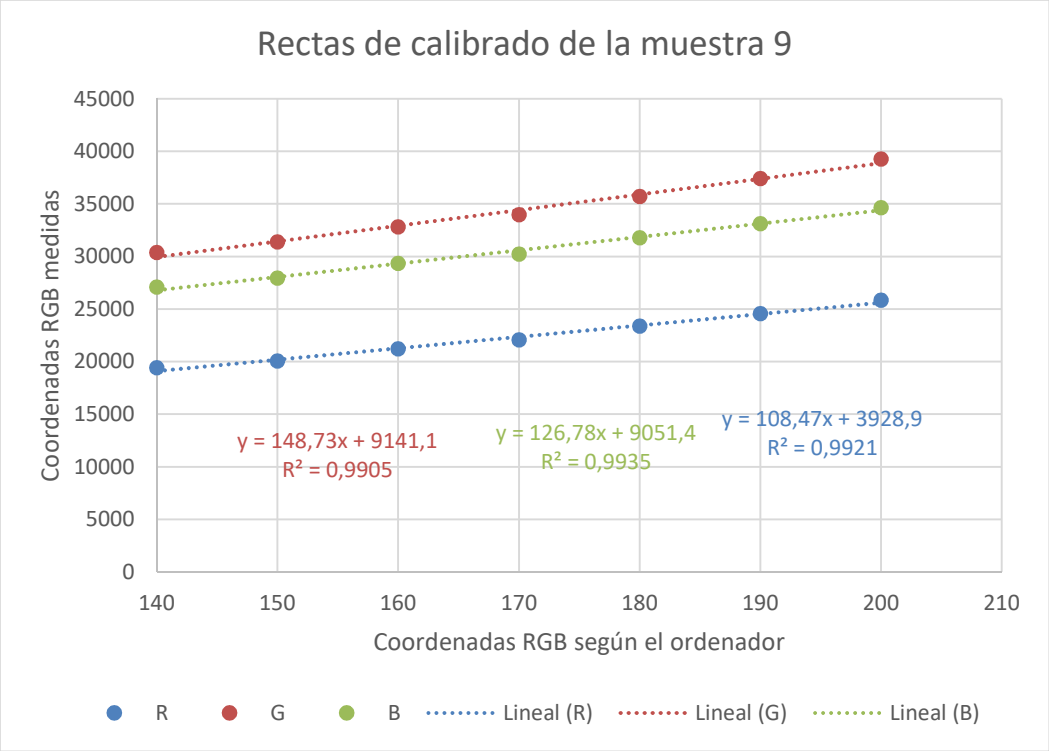


Ilustración 165: Rectas de calibrado de la muestra 9.

De ello se concluye que la resolución es mayor a 10/255.

En el último paso para la caracterización de la resolución se repitió el procedimiento seguido en la muestra 8 y 9 para la muestra 10.

A continuación, se muestra el resultado del escaneo de la muestra 10. Se puede observar que la tendencia ascendente que cabría esperar ya no es apreciable a simple vista.

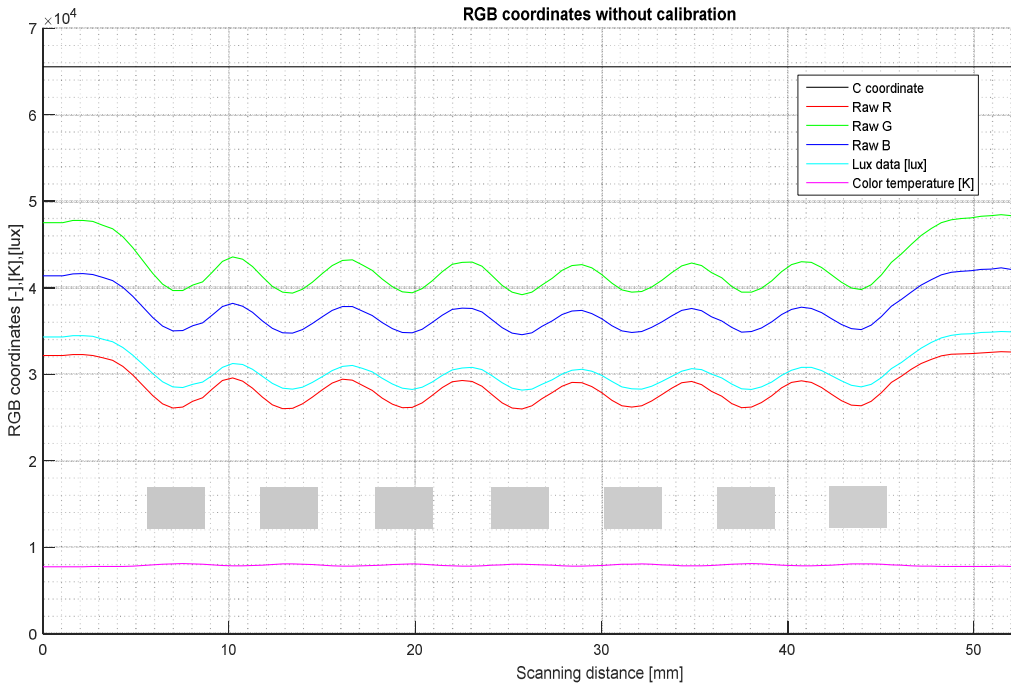


Ilustración 166: Resultados del escaneo de la muestra 10.

La regresión lineal de las tres coordenadas revela que ya no existe apenas correlación entre los datos medidos y los predichos, puesto que el coeficiente de determinación ni siquiera llega a 0.5. Las rectas de calibrado de muestran en la siguiente gráfica.

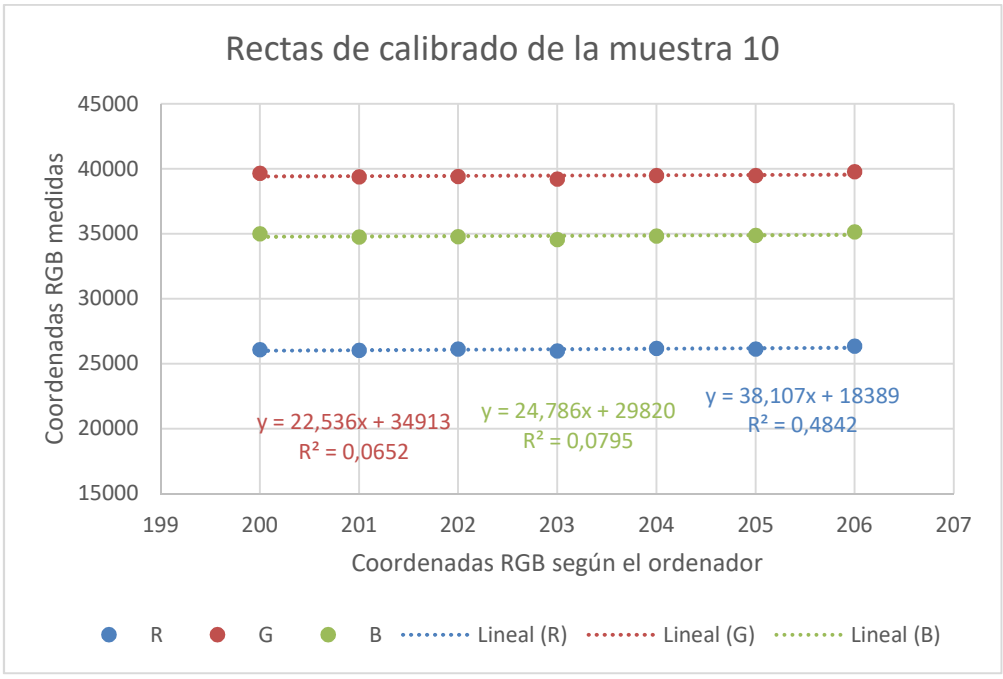


Ilustración 167: Rectas de calibrado de la muestra 10.

De ello se pueden deducir dos cosas. En primer lugar, no se tiene una resolución de 1/255. En segundo lugar, el sensor no está trabajando en condiciones óptimas, lo que se podría deber, entre otras cosas, a la imprecisión en la construcción del prototipo, dado que está se llevó a cabo manualmente.

Aunque una resolución de 10/255 (equivalente a aproximadamente un 4 %) es más que suficiente, se preparó una muestra adicional en la que, partiendo de un tono gris (R=G=B=140) se aumenta el valor de las tres coordenadas de 5 en 5 hasta llegar a 230. A continuación se encuentran los resultados del escaneo de dicha muestra.

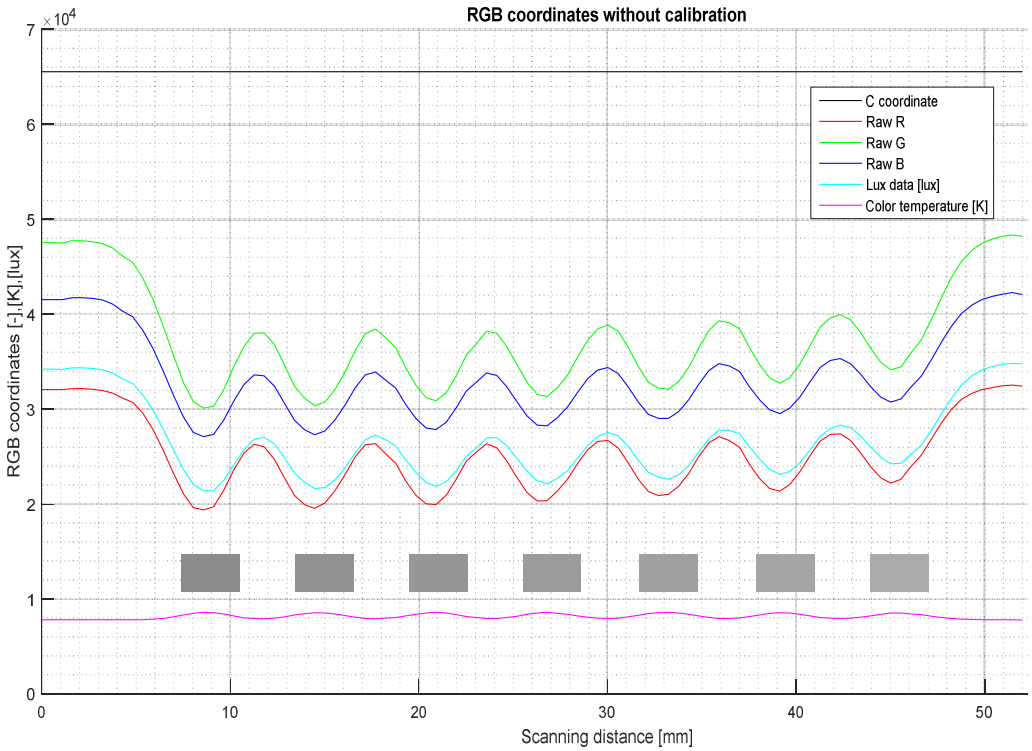


Ilustración 168: Resultados del escaneo de la muestra adicional.

La regresión lineal de los resultados proporcionó las siguientes rectas de calibrado. El coeficiente de determinación es en todos los casos mayor a 0.9, lo que se trata de un muy buen resultado.

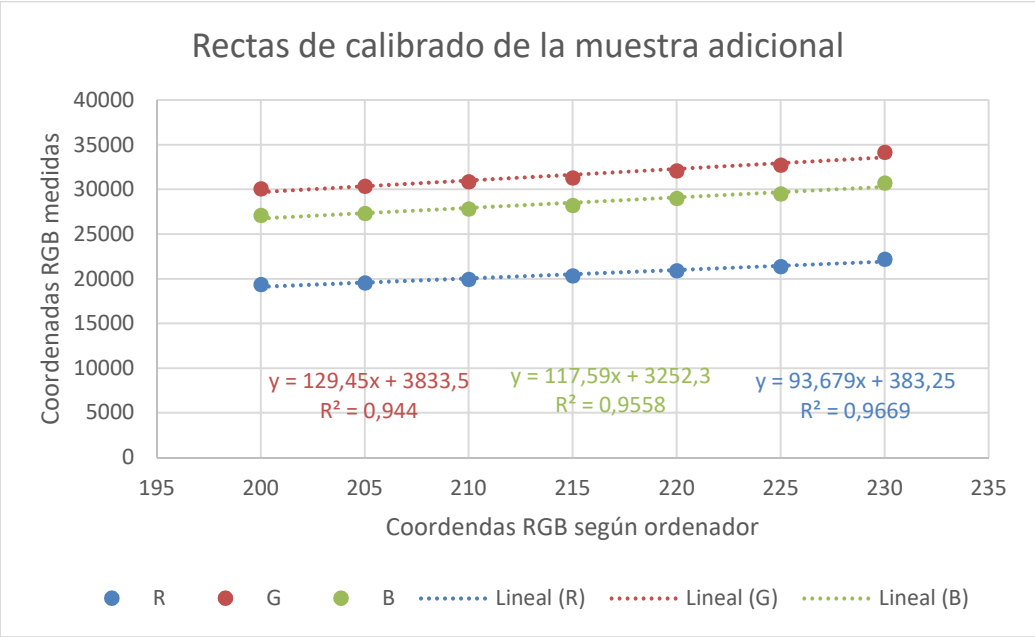


Ilustración 169: Rectas de calibrado de la muestra adicional.

En conclusión, se puede afirmar que la resolución RGB del prototipo resultante del presente trabajo es suficiente siempre y cuando no se requieran resoluciones mayores al 5/255, o dicho de otra forma, al 2 %.

Transparencia

Los resultados de la muestra 11 se encuentran en la gráfica situada a continuación. Se puede ver que la tendencia de las medidas es la correcta, puesto que un aumento de la tranparencia supone una mayor claridad y, por lo tanto, un incremento de las coordenadas RGB. Sin embargo, los puntos extremos de cada medida no son fiables, puesto que integran una mayor cantidad de fondo que el resto de medidas. Se observa también que la detección de transparencias a partir del 62.5 % (penúltimo mínimo) empieza a perder el carácter lineal. Como era de esperar, el hecho de que las rayas estén a tan poca distancia entre sí tiene como consecuencia que el fondo no sea detectado con claridad. Sólo en los espacios entre medidas tiene lugar una mejor caracterización del mismo. Por ello, no es recomendable integrar los picos para obtener los valores RGB. Sería de más ayuda tomar los valores de los mínimos, en los que la contaminación de las muestras adyancentes y del fondo es mucho menor.

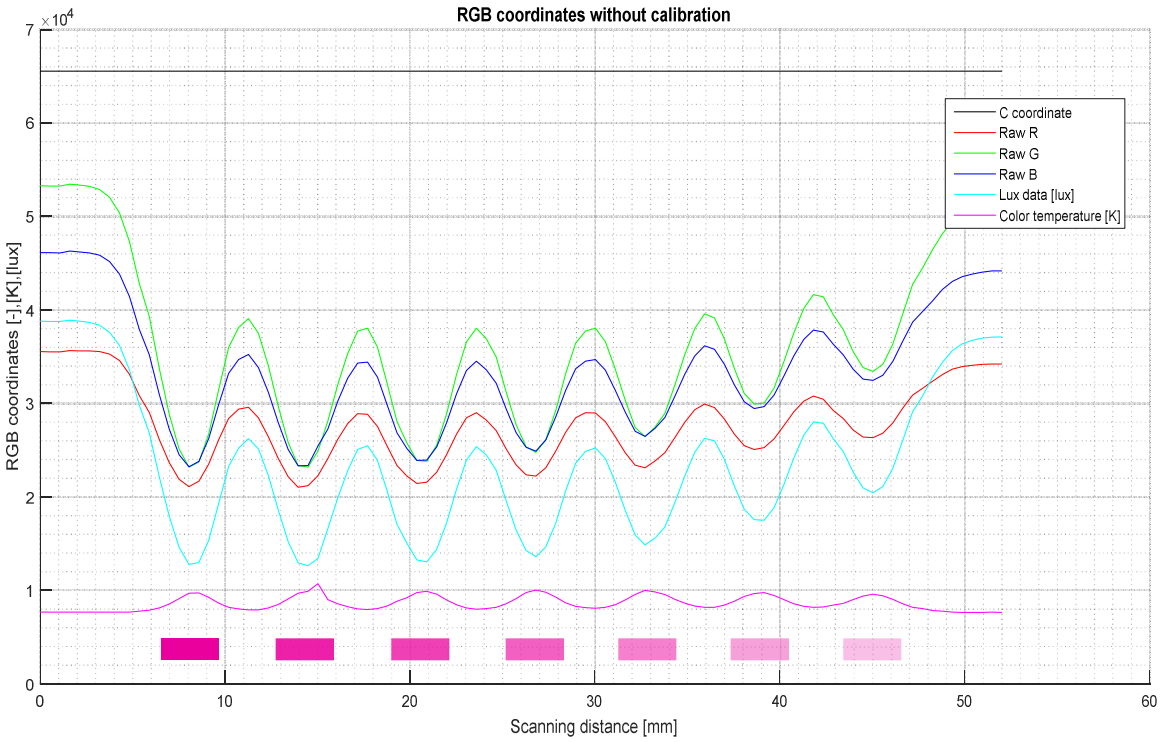


Ilustración 170: Resultados de la muestra 11.

De esta gráfica se extrajeron los mínimos y se compararon con los valores reales de transparencia. Si se incorpora el primer punto, se obtiene un coeficiente de determinación mínimo de 0.907 en la regresión lineal, un resultado bastante bueno. Sin embargo, si se elimina el primer punto, el coeficiente de determinación mínimo se sitúa en torno al 95 %, un valor excelente.

En la siguiente gráfica se muestra la regresión lineal llevada a cabo sin el valor del primer mínimo y sus resultados para la evaluación de la muestra 12. En una medida real, para evitar el error se podría poner dos muestras iguales al inicio y al final de la tira y eliminar los valores de esas lecturas al final.

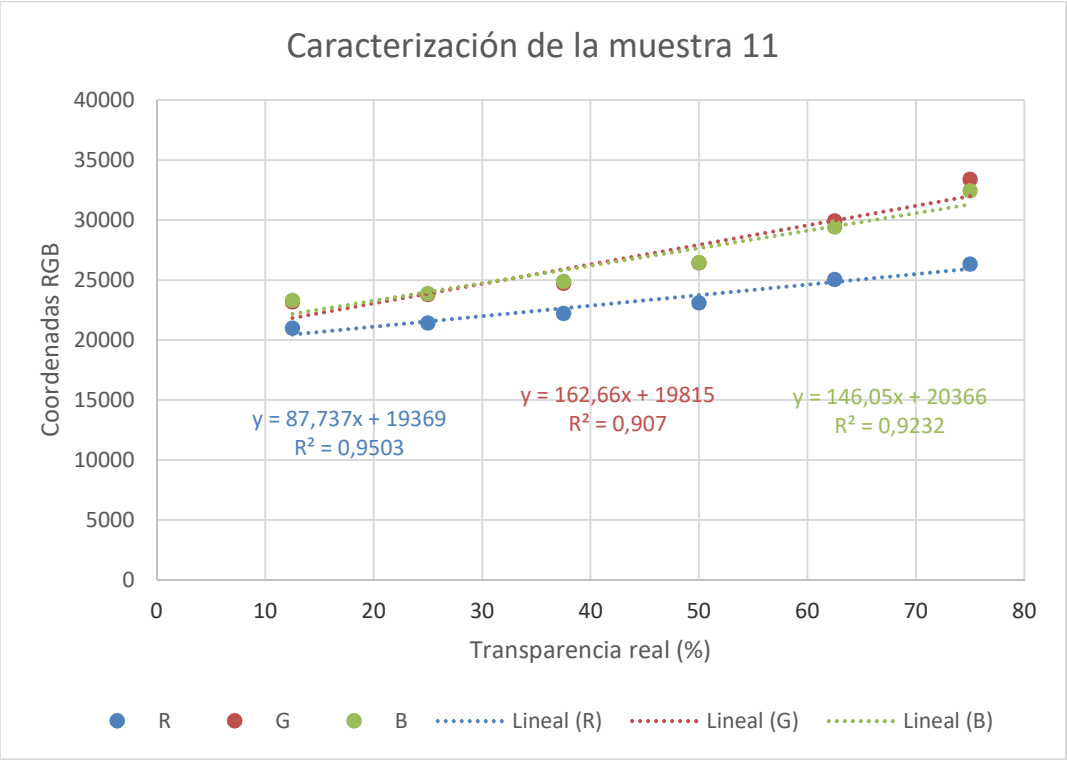


Ilustración 171: Resultados de la caracterización de la muestra 11.

En segundo lugar, se presentarán los resultados de la muestra 12. Como se ha explicado anteriormente, en dicha muestra la transparencia aumenta un 5 % por raya. A continuación, se muestra la gráfica obtenida del escaneo de la muestra 12.

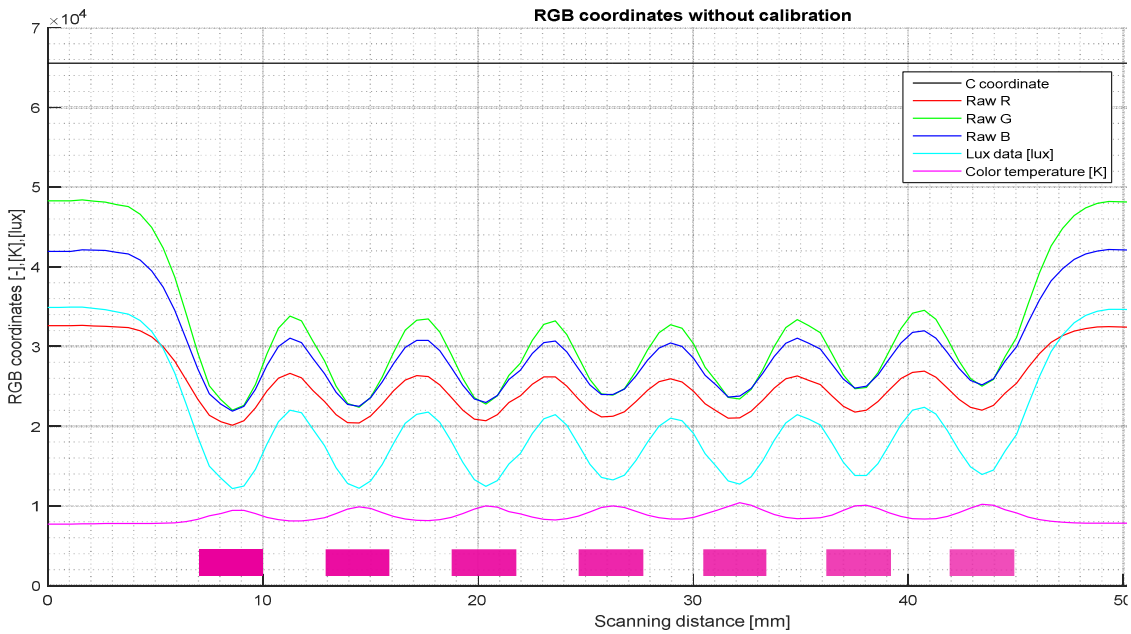


Ilustración 172: Resultado del escaneo de la muestra 12.

El siguiente paso consiste en calcular las rectas de calibrado, que se muestran en la siguiente gráfica. El coeficiente de determinación superior a 0.9 indica que es posible identificar muestras con una resolución de transparencia del 5 %.

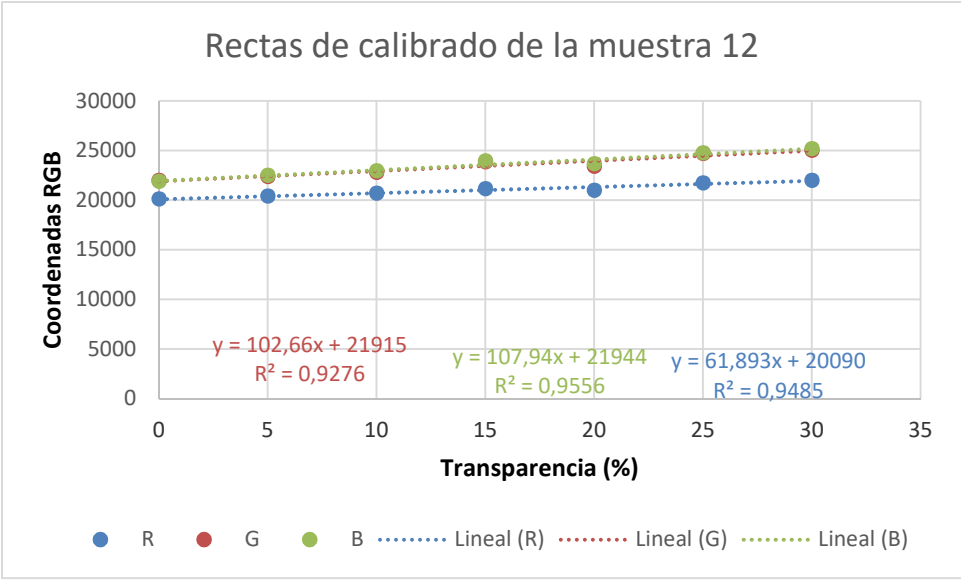


Ilustración 173: Rectas de calibrado de la muestra 12.

Finalmente, se midió una muestra en la que la transparencia aumenta tan sólo un 1 % de una raya a otra (muestra 13), llegando al límite de lo que el software disponible puede generar. Los resultados del escaneo se muestran en la siguiente imagen.

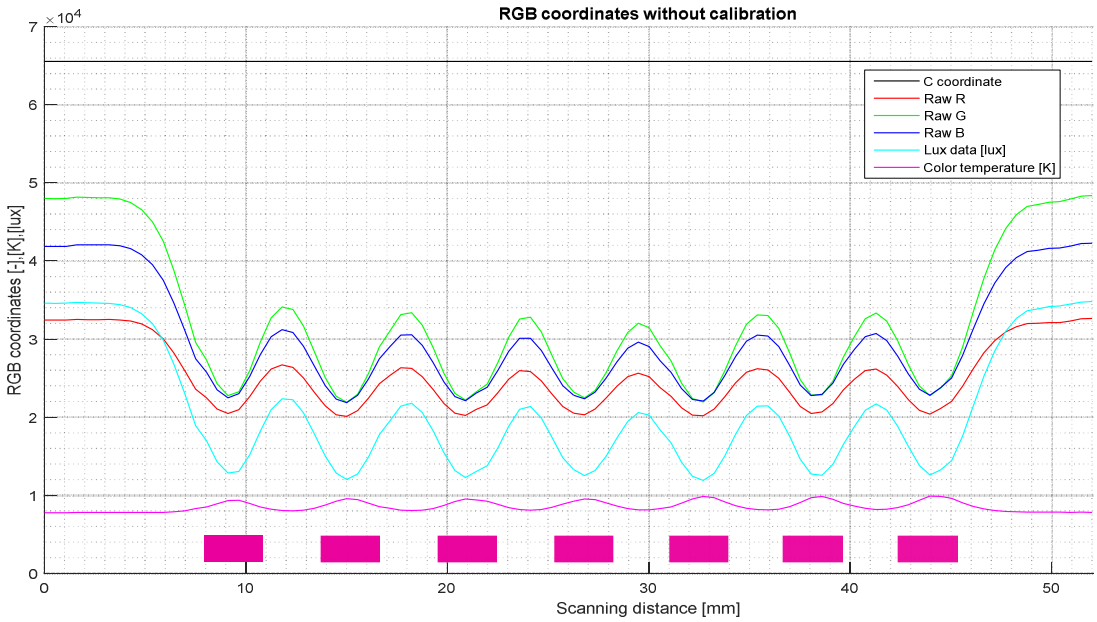


Ilustración 174: Resultados del escaneo de la muestra 13.

Las rectas de calibrado obtenidas de esa muestra, que se encuentran a continuación, revelan que no es posible distinguir diferencias en la transparencia del 1 % con el prototipo, dado que el coeficiente de determinación no llega ni siguiera a 0.5.

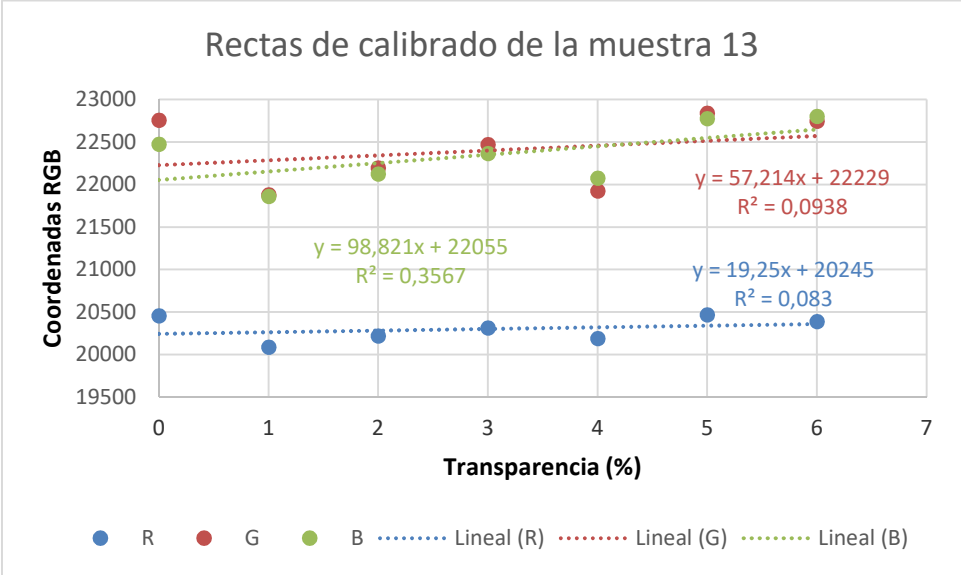


Ilustración 175: Rectas de calibrado de la muestra 13.

En conclusión, es posible utilizar el prototipo siempre y cuando la resolución requerida en la transparencia no sea superior al 5 %.

Reproducibilidad

En la siguiente imagen se encuentran los resultados de las cuatro muestras. Se puede ver que las oscilaciones entre los mínimos de cada medida son muy pequeñas, lo que indica una reproducibilidad aceptable. La desviación en las medidas del rosa parece ser algo mayor, lo que se deba seguramente a que la muestra se dobló ligeramente durante las medidas.

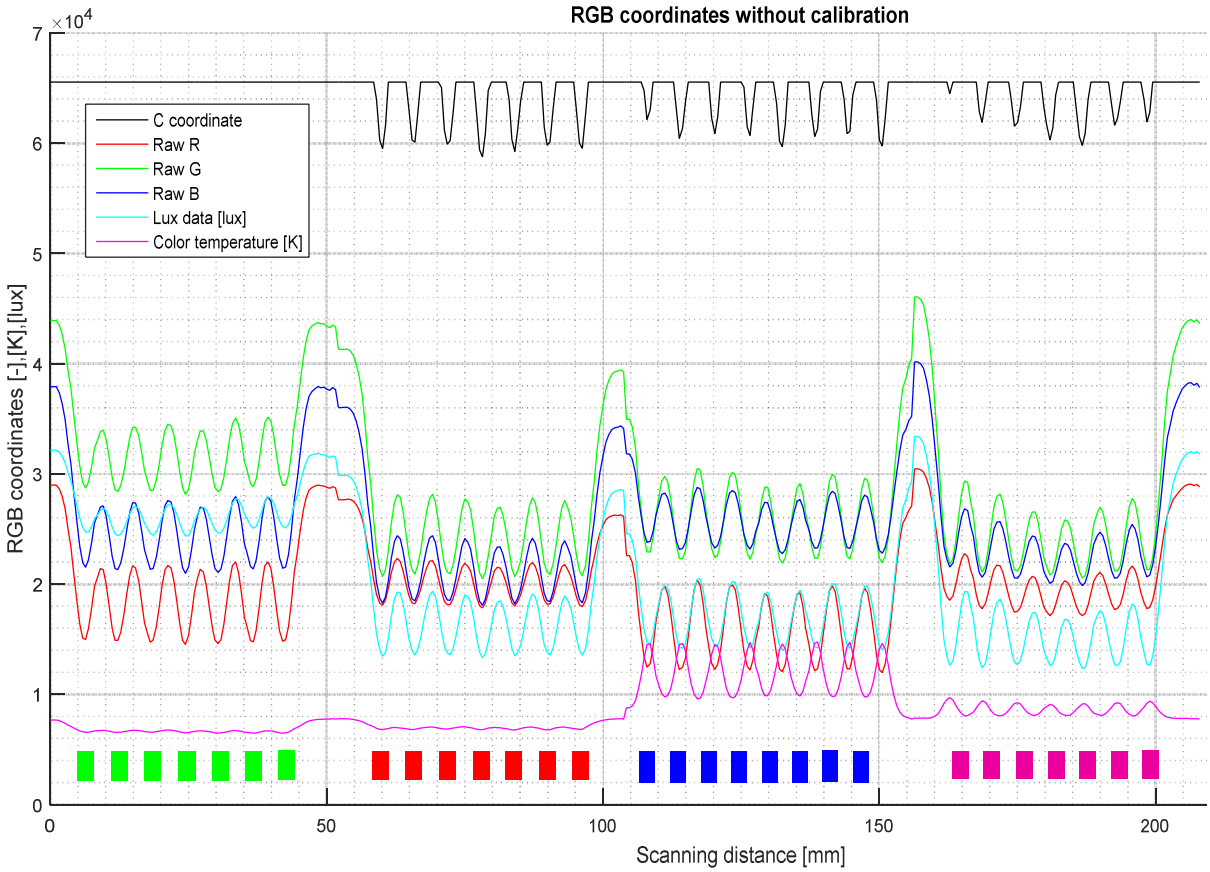


Ilustración 176: Resultados de las medidas de las muestras 8, 9, 10 y 11.

Para hacerse una mejor idea de la reproducibilidad que permite el prototipo, es de ayuda extraer los mínimos de cada muestra y compararlos entre sí. Debido a que integran una mayor cantidad de fondo que el resto de las medidas, el primer y el último mínimo de cada color se elimina.

A continuación, se calculan las desviaciones estándar y la desviación relativa de las coordenadas de cada color. Los resultados se encuentran en la siguiente tabla.

	Verde			**Rojo**			**Azul**			**Rosa**		
	R	G	B	R	G	B	R	G	B	R	G	B
Media	14703	28447	21208	18079	20778	18336	12177	22237	23121	17440	20972	20324
Desviación estándar	146.35	247.48	186.3	151.0	194.5	197.1	81.70	198.9	205.7	297.7	272.5	332.5
Desviación relativa(%)	0.995	0.87	0.878	0.835	0.936	1.075	0.671	0.894	0.889	1.707	1.299	1.636

Tabla 17: Resultado del cálculo de la desviación estándar de los mínimos de las muestras 8, 9, 10 y 11.

Como se puede ver en la tabla, la desviación relativa no llega al 2%, por lo que el sistema satisface las condiciones requeridas de reproducibilidad. La desviación relativa media es aproximadamente del 1.05%.

Iluminación

A continuación, para una mejor visualización de las diferencias, se colocan los resultados de las cuatro medidas en el mismo gráfico.

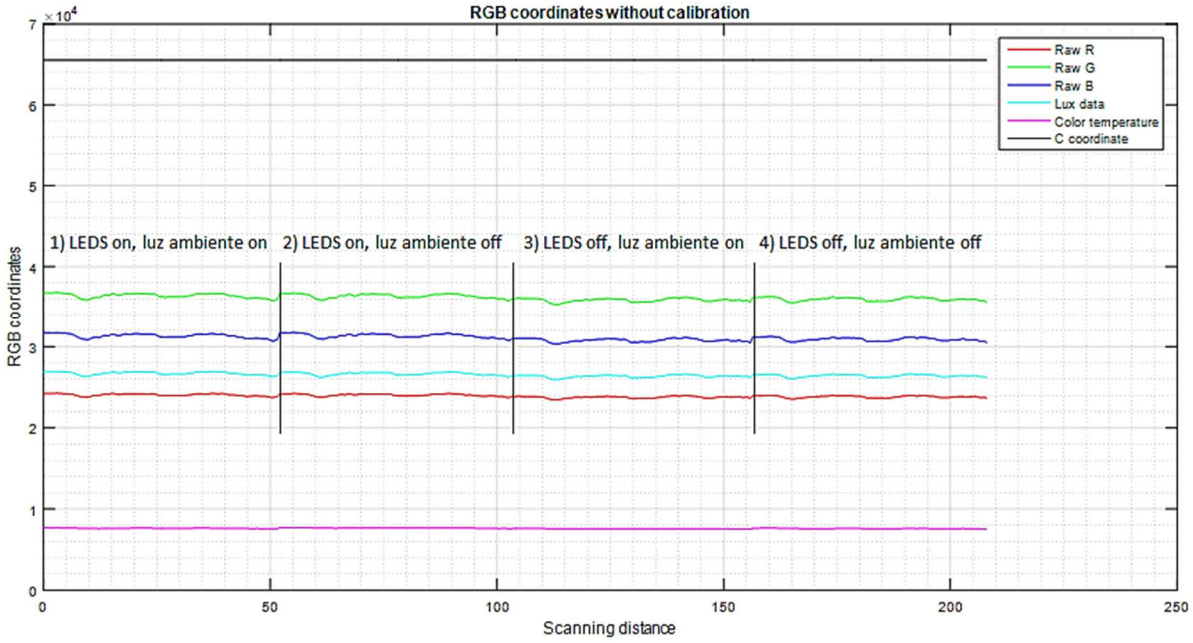


Ilustración 177: Medidas para la caracterización de la influencia de la iluminación

Se puede observar que las medidas son muy parecidas entre sí. Se puede apreciar incluso que la forma de las gráficas coincide, aunque sus valores mínimos y máximos aparecen levemente desplazados. Al hallarse comprimidos los datos en el eje x, aparecen oscilaciones más marcadas que en el apartado de las medidas del fondo. Si se calcula la desviación relativa de los datos, se observa que ésta no supera el 1.1%. Así, la variación de la iluminación no tiene gran repercusión en las medidas.

Con el fin de establecer diferencias de forma clara entre las diferentes medidas, se pueden ampliar los datos de una coordenada de color. Dado que las tres varían de igual forma de una medida a otra, es indiferente la coordenada elegida. Se elige la coordenada azul, mostrada a continuación junto con algunos de sus puntos relevantes.

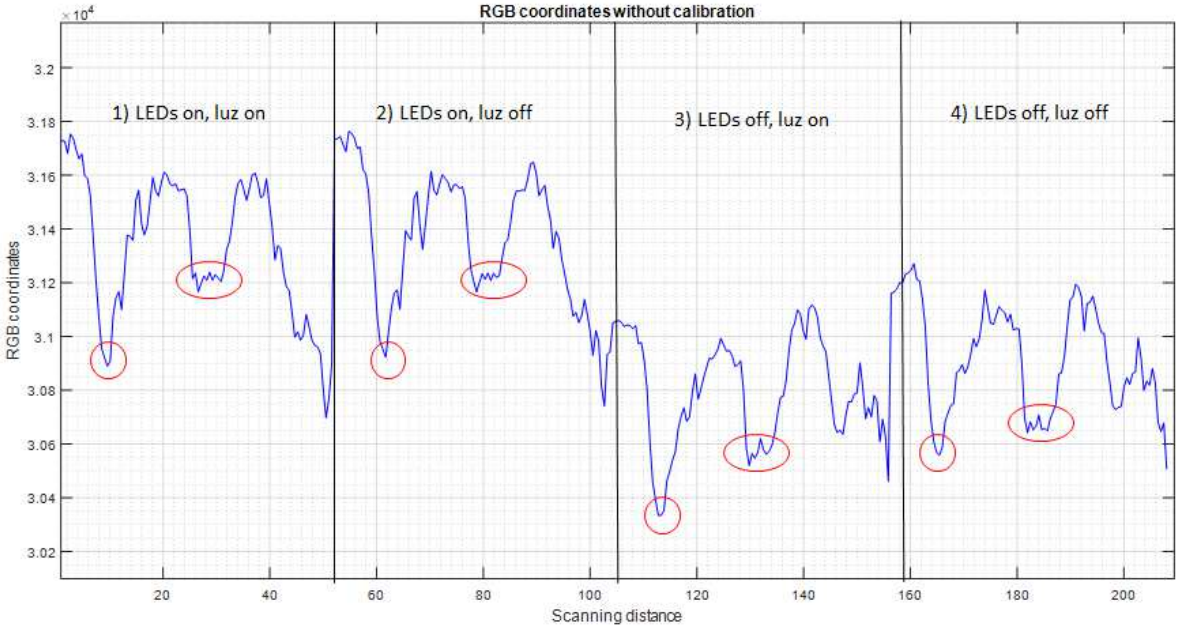


Ilustración 178: Coordenada azul de las cuatro medidas.

Al fijarse en los puntos relevantes, hay que tener cuidado en no tender hacia los extremos, puesto que, al tratarse de los puntos en los que se juntan los datos de distintas medidas, se podrían extraer conclusiones erróneas.

Se puede observar que las dos primeras medidas son muy parecidas entre sí. No obstante, los valores de la segunda son ligeramente superiores a los de la primera, lo cual se pone de manifiesto en las zonas marcadas. De este comportamiento se puede deducir que si los LEDs están encendidos la repercusión de la luz ambiente en las medidas es ínfima.

Esta afirmación se ve reforzada por las dos siguientes medidas, en las que los valores de las gráficas disminuyen notablemente. También es de interés que los valores de la cuarta son mayores que los de la tercera. Teniendo en cuenta de que en ambas medidas los LEDs están cubiertos y la única diferencia es la luz ambiente, se puede deducir que, en ausencia de la luz de los LEDs, la influencia de la luz ambiente en las medidas es mucho mayor. Cabe señalar que la forma de las cuatro gráficas es muy parecida, así que las diferencias en la iluminación no cambian la relación entre los datos de las medidas tomadas, sino únicamente su ubicación en el eje y.

A continuación, se presentan la media, las desviaciones estándares σ_f y las desviaciones relativas $\sigma_{f,rel}$de cada medida.

Medida	1 (LEDS on, luz on)			2 (LEDS on, luz off)		
	R	G	B	R	G	B
Media	24064.102	36411.510	31341.327	24037.663	36371.153	31359.541
σ_f	145.188	267.752	261.668	135.310	251.263	249.367
$\sigma_{f,rel}$ [%]	0.6	0.7	0.8	0.6	0.7	0.8
Medida	3 (LEDS off, luz on)			4 (LEDS off, luz off)		
	R	G	B	R	G	B
Media	23761.980	35862.245	30793.898	23824.888	35975.959	30919.551
σ_f	118.580	213.911	201.172	118.671	208.877	199.653
$\sigma_{f,rel}$ [%]	0.5	0.6	0.7	0.5	0.6	0.6

Tabla 18: Media y desviaciones de las medidas en distintas condiciones de iluminación.

Se puede observar que las desviaciones son equiparables a pesar de los cambios en la iluminación. Por lo tanto, se puede afirmar que sólo tiene lugar un desplazamiento de los datos en dirección del eje y, manteniendo la gráfica su forma en las diferentes medidas. La media de los datos sí que muestra una variación algo mayor que concuerda con la gráfica explicada anteriormente.

En conclusión, con la finalidad de eliminar en la medida de lo posible la influencia de la luz ambiente, interesa trabajar con los LEDs de los equipos electrónicos descubiertos. Además, el fondo es de color blanco, lo que significa que los valores más elevados son los que mejor lo describen. De ello se deduce que, según las medidas, lo óptimo sería

trabajar a oscuras con los LEDs a descubierto. No obstante, siempre y cuando las condiciones de iluminación se mantengan más o menos constantes, es posible obtener resultados reproducibles con este equipo.

Conclusiones

Para una mejor visualización de los resultados obtenidos en este capítulo se elaboró la siguiente tabla.

Parámetro	**Conclusiones**
Detección de fondo	Correcta, desviación relativa inferior al 1%
Intervalo de lectura	Óptimo 0.5 mm
Distancia entre muestras	>2 mm posible si se mantiene constante, óptimo >7 mm
Longitud (en paralelo a la dirección de escaneo del sensor)	>1 mm posible si se mantiene constante, óptimo >6 mm
Anchura (perpendicular a la dirección de escaneo del sensor)	>1 mm posible si se mantiene constante, óptimo >7 mm
Diferenciación de colores	Correcta
Diferenciación de tonos	Se distinguen diferencias de hasta el 5% en la escala RGB con R^2>0.9
Diferenciación de transparencia	Se distinguen diferencias de hasta el 5% en la transparencia con R^2>0.9
Reproducibilidad	Correcta, desviación relativa del 1.05%
Iluminación	Poca repercusión (desviación relativa menor a 1.1%). Mejor mantenerla constante, óptimo con LEDs encendidos y baja luz ambiente

Tabla 19: Resumen de los resultados de la caracterización del prototipo.

6 Aplicación: Determinación del contenido de nitratos mediante la lectura de tiras reactivas

6.1 Introducción

Una vez caracterizado el comportamiento del prototipo, se probó su adecuación para una aplicación real: la detección de nitratos mediante tiras reactivas.

Los nitratos (NO_3^-) son un conocido contaminante, dado que su reducción produce nitritos, que pueden reaccionar a nitroaminas, que son cancerígenas. Por ello, la determinación de su contenido en agua potable es de vital importancia.

En este capítulo se comprobará si el prototipo desarrollado se puede emplear para la detección de nitratos, para lo cual se compararán los resultados obtenidos con los que proporcione el reflectómetro comercial RQFlex.

6.2 Material y métodos

En este apartado se presentarán en primer lugar los materiales que se emplearon para la determinación de nitratos. En segundo lugar, se explicará el procedimiento que se siguió.

El RQFlex 10 de Merck es un sistema comercial rápido y de sencilla utilización para el análisis cuantitativo de gran cantidad de analitos orgánicos e inorgánicos mediante el uso de las tiras correspondientes que proporciona su empresa. Combinados, el reflectómetro y la tira reactiva miden la concentración de una sustancia basándose en la luz que se refleja en una zona de reacción. Para una mayor exactitud, se realizan dos medidas simultáneamente y se calcula la media de ambas. La lectura de los resultados se lleva a cabo a través de la pantalla LCD que incorpora el RQFlex 10, aunque es posible comunicar los datos directamente al ordenador. La fuente de iluminación son LEDs. El RQFlex permite guardar hasta cinco métodos de análisis (para diferentes analitos), pero para el presente trabajo sólo es relevante el de los nitratos. A continuación, se encuentra una fotografía del RQFlex 10.

Ilustración 179: Reflectómetro RQFlex 10.

Además, se cuenta con las tiras reactivas de nitratos del RQFlex (Merck, producto 110020), que funcionan según la reacción explicada a continuación. En primer lugar, los iones nitrato son reducidos a iones nitrito. En presencia de un tampón ácido, los iones nitrito forman una sal de diazonio con una amina aromática. Ésta reacciona con N-(1-naftil)-etilendiamina dando un azocolorante violeta rojizo. La concentración de éste se determina reflectométricamente y se usa para el cálculo de la cantidad de nitratos. Como se ha mencionado anteriormente, hay dos zonas de reacción, cuyos resultados se promedian en el RQFlex para una mayor exactitud. En teoría, con las tiras es posible medir concentraciones de 5 a 225 mg/l de nitratos. En la siguiente imagen se observan dos tiras, una que no ha reaccionado (zonas reactivas en blanco) y otra que sí (zonas reactivas rosas). Se define para las futuras medidas las zonas reactivas de la izquierda como zona 1 y las de la derecha como zona 2. Es decir, partiendo de la siguiente imagen, la tiras serán leídas por el prototipo de izquierda a derecha.

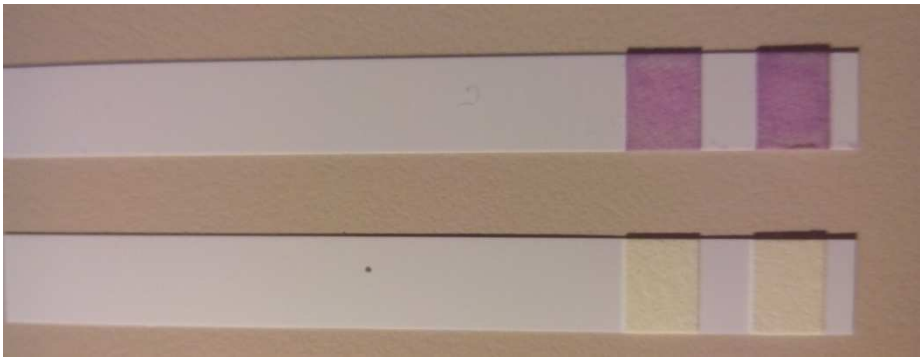


Ilustración 180: Tiras reactivas para la determinación de nitratos tras ser sumergidas en una solución con nitratos (arriba) y sin nitratos (abajo).

Finalmente, se tiene el prototipo obtenido como resultado de este trabajo, basado en el sensor RGB TCS34725. Tomando como punto de partida la relación existente entre la concentración de nitratos y la intensidad del color de la zona reactiva de la tira después de la reacción, se parte de la idea de que sería posible obtener la información sobre la concentración según las coordenadas RGB medidas.

Para ello, se prepararon disoluciones en agua destilada con nitrato potásico en las siguientes concentraciones: 0 mg/l, 25 mg/l, 50 mg/l, 100 mg/l, 150 mg/l, 200 mg/l, 250 mg/l, 300 mg/l, 350 mg/l, 400 mg/l, 450 mg/l y 500 mg/l. Los resultados se encuentran en la siguiente imagen.

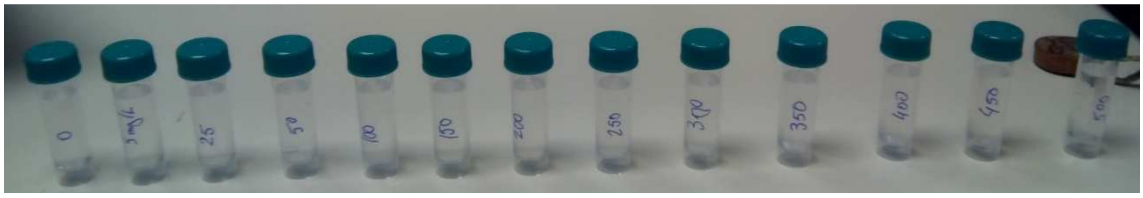


Ilustración 181: Disoluciones de agua destilada con distintas concentraciones de nitratos.

Para llevar a cabo las medidas, se sumergen dos tiras reactivas simultáneamente en una disolución, se deja que reaccionen y se miden en el sistema comercial y el prototipo a la vez. Para ello, se habrá de esperar un minuto antes de introducir la tira en el RQFlex, tal y como indica el protocolo de dicho sistema, mientras que la tira se introducirá en el prototipo nada más comience la reacción, ya que éste último tiene un tiempo de calibración de aproximadamente 1 minuto.

Puesto que las tiras reactivas sólo miden hasta 225 mg/l, basta con analizar las disoluciones que no superen esa concentración de nitratos, aunque se midieron algunas más para visualizar la saturación de las tiras. En la siguiente imagen se observa la colocación de la tira reactiva a seguir en las medidas.

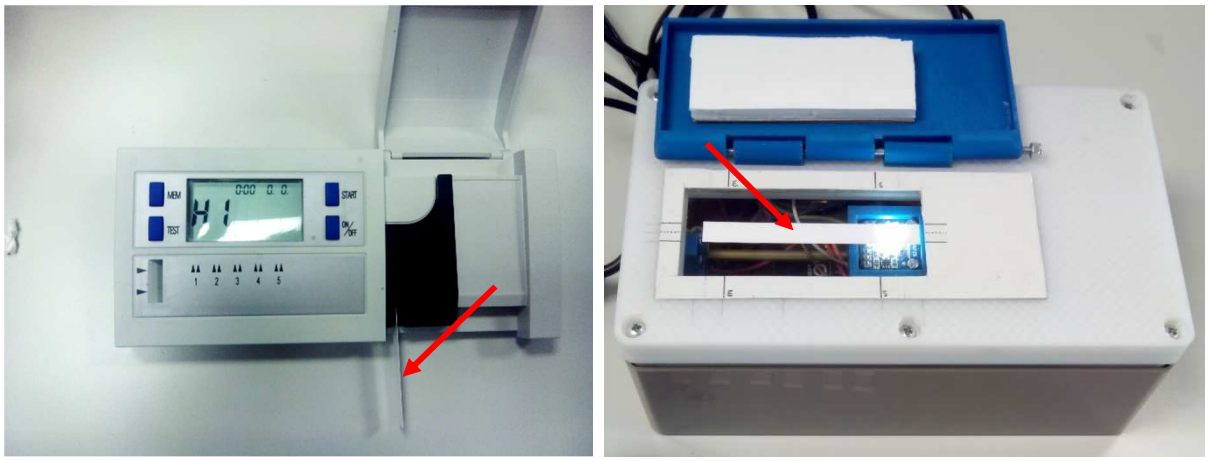


Ilustración 182: Colocación de la tira reactiva en el sistema comercial (izquierda) y en el prototipo (derecha).

6.3 Resultados y discusión

En primer lugar, se muestran los resultados obtenidos con el sistema comercial (ilustración 183). Se puede observar que se obtiene una recta de calibrado casi perfecta, con un coeficiente de determinación muy cercano a 1. A partir de los 200 mg/l de concentración real de nitratos tiene lugar una saturación, puesto que el reactivo de las tiras se ve desbordado. Por ello, se obtiene siempre el mismo tono, correspondiente con la saturación máxima (225 mg/l).

Con el prototipo se obtiene para cada tira una gráfica como la que se muestra aen la ilustración 184. Se observan dos mínimos, el primero pertenece a la primera zona reactiva de las tiras y el segundo a la segunda. Además, es posible observar donde comienzan y terminan las zonas reactivas y la tira, dado que, al ser de distinta anchura, proyectan pequeñas sombras al ser escaneadas.

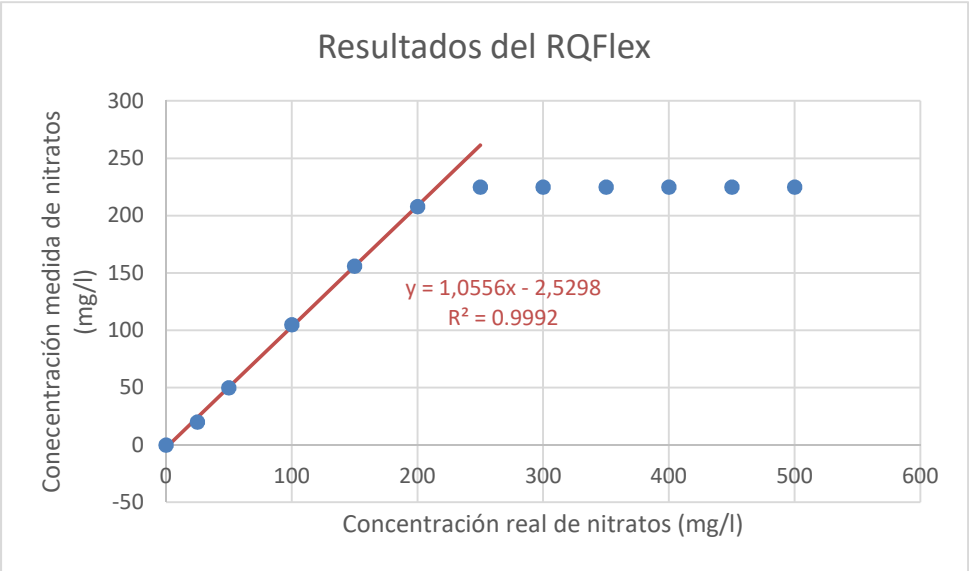


Ilustración 183: Observado frente a predicho para el sistema comercial.

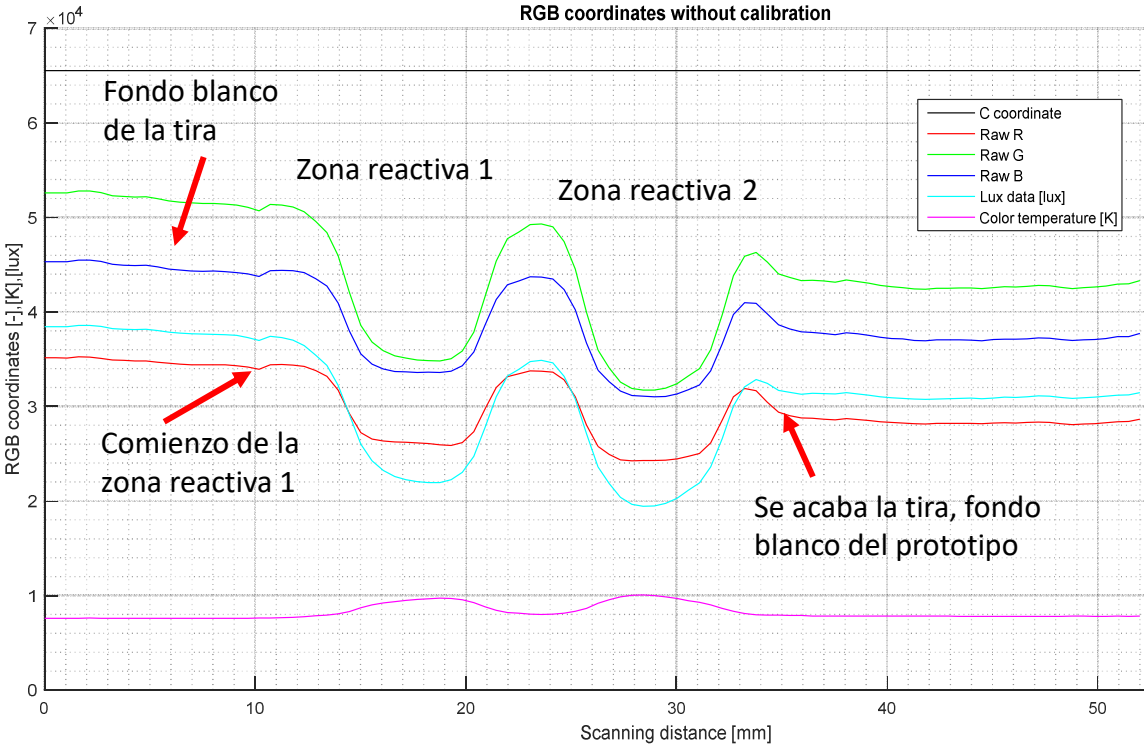


Ilustración 184: Resultados del escaneo de la tira con 50 mg/l de nitratos.

Para proceder, se extraen y se analizan los mínimos de cada zona reactiva para las diferentes concentraciones con la ayuda del software diseñado, puesto que se trata de los valores menos contaminados por el fondo. Los resultados del prototipo son similares para

las dos zonas reactivas y se puede apreciar que la región lineal coincide con la correspondiente al equipo comercial, por lo que a modo de ejemplo se muestran sólo los resultados de la primera zona reactiva.

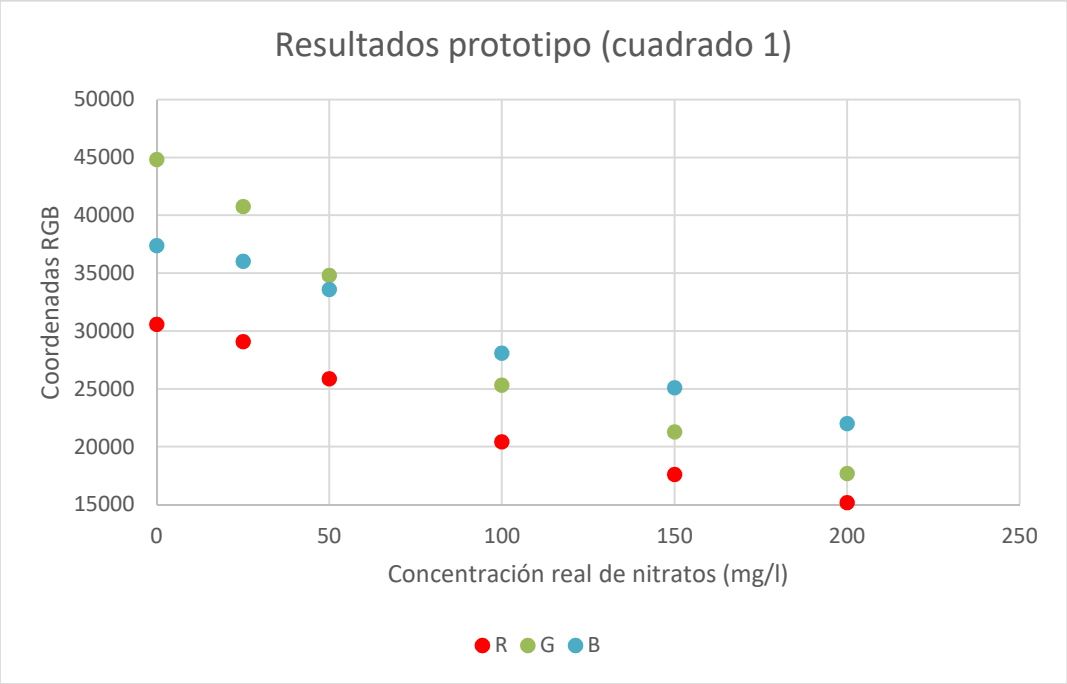


Ilustración 185: Valores de los mínimos de la primera zona reactiva.

A continuación, se elaboró un modelo para extraer con rapidez los valores de la concentración de nitratos a partir de las coordenadas RGB.

Para ello, el primer paso consiste en realizar una regresión lineal de cada coordenada RGB mediante el método de los mínimos cuadrados. En un eje se tendrá el valor real de la concentración y en el otro las coordenadas RGB de ese punto. En las siguientes gráficas se encuentran las regresiones lineales obtenidas para ambas zonas reactivas (ilustraciones 187 y 188).

Una vez obtenidas las ecuaciones lineales, se despejan hacia la concentración, obteniendo de esa forma seis ecuaciones que permiten obtener concentraciones a partir de coordenadas RGB. Éstas se encuentran en la tabla 20. R1, G1 y B1 se refieren al primer cuadrado de la tira, R2, G2 y B2 al segundo.

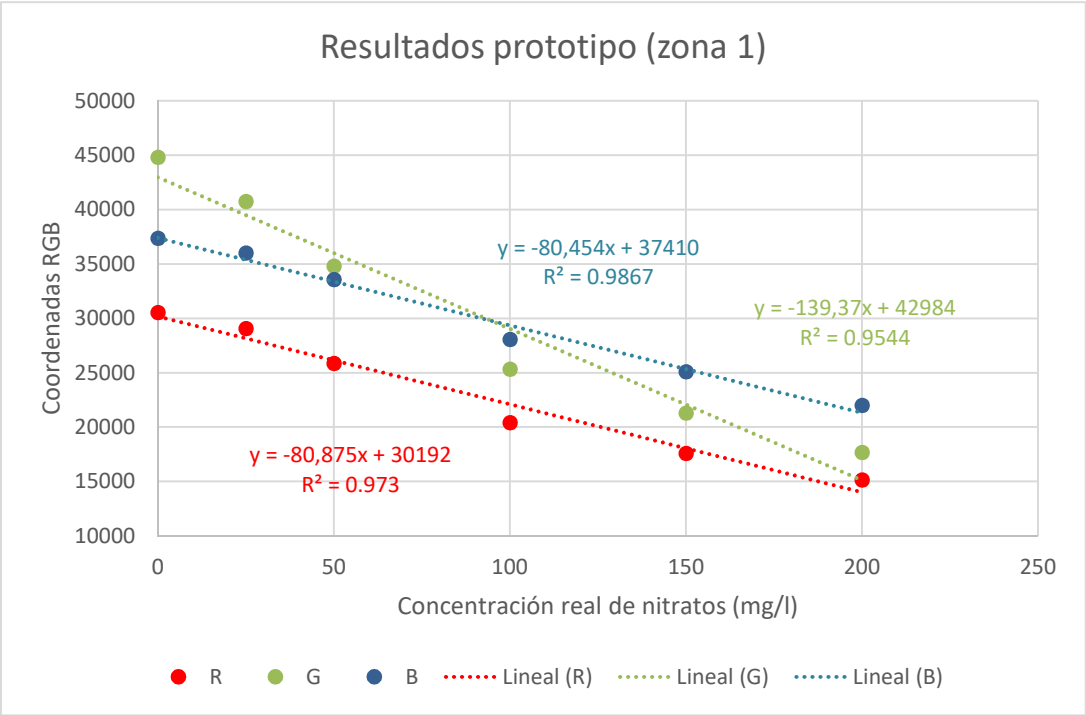


Ilustración 186: Regresiones lineales para la primera zona reactiva.

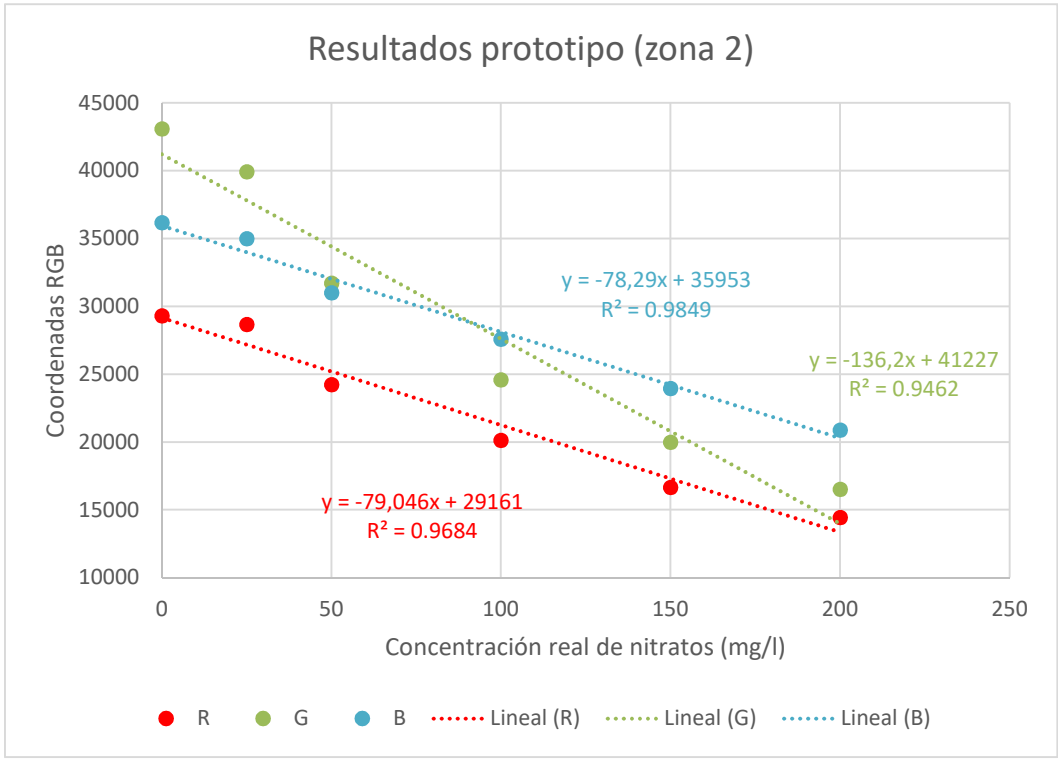


Ilustración 187: Regresiones lineales para la segunda zona reactiva.

	$y(x) = R(c)$ ó $G(c)$ ó $B(c)$	$x(y) = c(R)$ ó $c(G)$ó $c(B)$
R1	$y = -80.875x + 30192$	$x = \frac{30192 - y}{80.875}$
G1	$y = -139.37x + 42984$	$x = \frac{42984 - y}{139.37}$
B1	$y = -80.454x + 37410$	$x = \frac{37410 - y}{80.454}$
R2	$y = -79.046x + 29161$	$x = \frac{29161 - y}{79.046}$
G2	$y = -136.2x + 41227$	$x = \frac{41227 - y}{136.2}$
B2	$y = -78.29x + 35953$	$x = \frac{35953 - y}{78.29}$

Tabla 20: Ecuaciones de la regresión lineal (izquierda) y para el cálculo de la concentración a partir de coordenadas RGB (derecha).

El siguiente paso consiste en insertar los valores RGB obtenidos en sus ecuaciones correspondientes y en comparar la concentración resultante con la real. Para ello, se calculará el coeficiente de determinación (R^2) y de correlación (R) de los valores predichos por el modelo frente a los reales. En las siguientes gráficas se muestra el valor de lo observado frente a lo predicho para la coordenada B1 (mejor resultado del prototipo), para la coordenada G2 (peor resultado del prototipo) y para el sistema comercial.

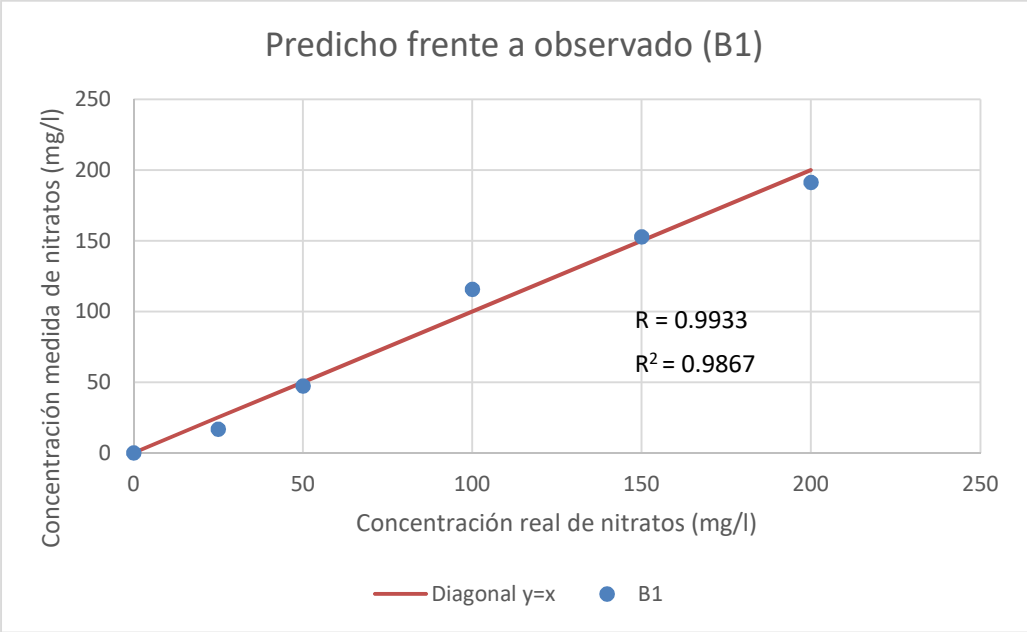


Ilustración 188: Mejor resultado del prototipo (predicho frente a observado de B1).

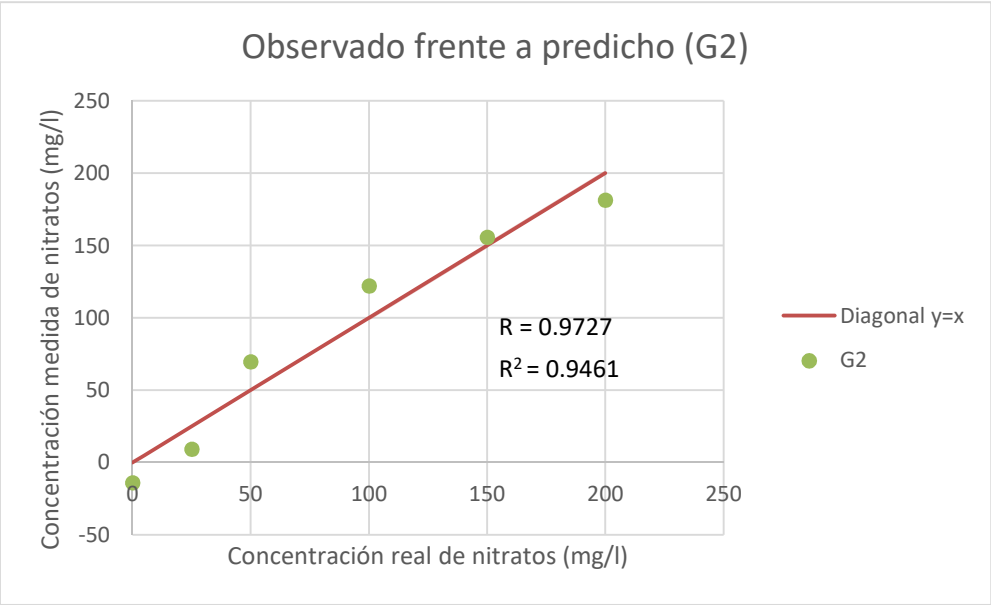


Ilustración 189: Peor resultado del prototipo (predicho frente a observado de G2).

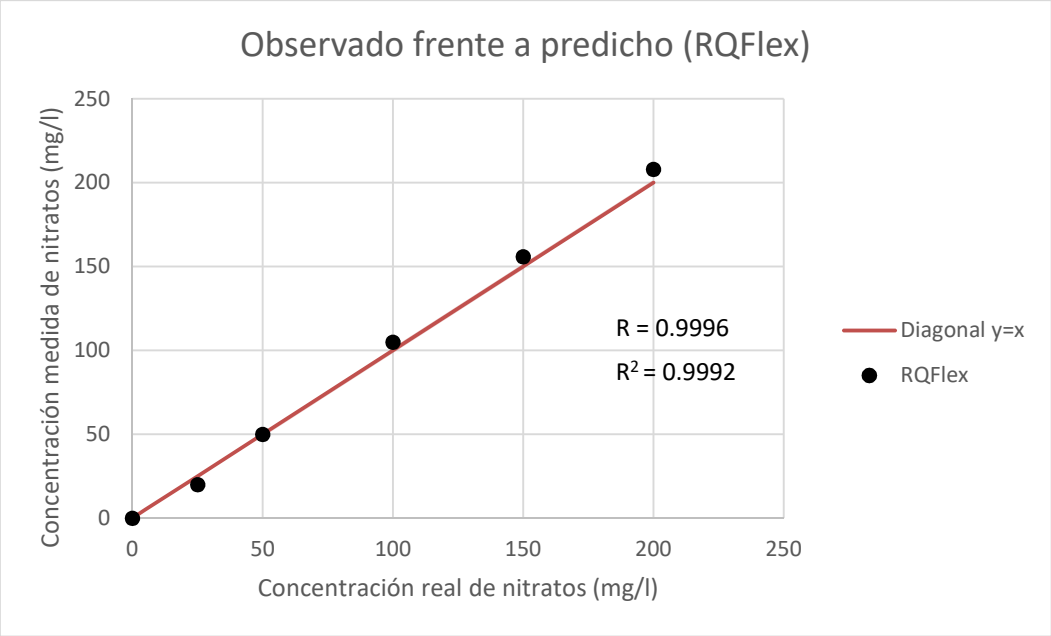


Ilustración 190: Predicho frente a observado para el sistema comercial RQFlex.

En la siguiente tabla se encuentran el coeficiente de correlación y el coeficiente de determinación de todas las coordenadas. Para que los resultados sean buenos, ambos parámetros deberían acercarse lo máximo posible a 1.

	RQFlex	R1	G1	B1	R2	G2	B2
R	0.9992	0.9730	0.9544	0.9867	0.9684	0.9462	0.9849
R^2	0.9996	0.9864	0.9769	0.9933	0.9841	0.9727	0.9924

Tabla 21: Coeficientes de determinación y de correlación del modelo obtenido para el sistema comercial y las coordenadas de ambas zonas reactivas de las tiras.

Se puede observar que, aunque no superan la precisión del sistema comercial, ambos coeficientes son superiores a 0.94, lo que indica que el modelo es aceptable.

El siguiente paso para evaluar los resultados obtenidos con el prototipo consiste en averiguar el error medio en cada mínimo, es decir, la desviación estándar y la desviación relativa por medida.

Para averiguar ambas variables, se calculan las diferencias entre los resultados del modelo y los valores reales de concentración, se elevan al cuadrado, se saca la media de los cuadrados y se calcula la raíz de dicha media.

Ese valor se divide entre el máximo de concentración que se puede detectar mediante las tiras reactivas, es decir, 225 mg/l, obteniendo así la desviación relativa media por medida.

	RQFlex	G2 (peor)	B1(mejor)
Desviación estándar media (mg/l)	5,00	16,78	8,16
Desviación relativa (%)	2,22	7,46	3,63

Tabla 22: Comparación de las desviaciones del sistema comercial y el prototipo.

Se puede observar que la desviación relativa media del prototipo oscila entre el 3.8 y el 5.12% por medida, mientras que la del sistema comercial se encuentra en torno al 2.2%.

El último paso para la evaluación de los resultados consiste en comprobar si lo obtenido con el equipo comercial concuerda con lo proporcionado por el prototipo. Para ello, se calcularon los coeficientes de correlación y de determinación de los resultados del prototipo respecto a los del RQFlex. A continuación, se encuentran las gráficas con el mejor (B1, ilustración 191) y el peor (G2, ilustración 192) resultado obtenido.

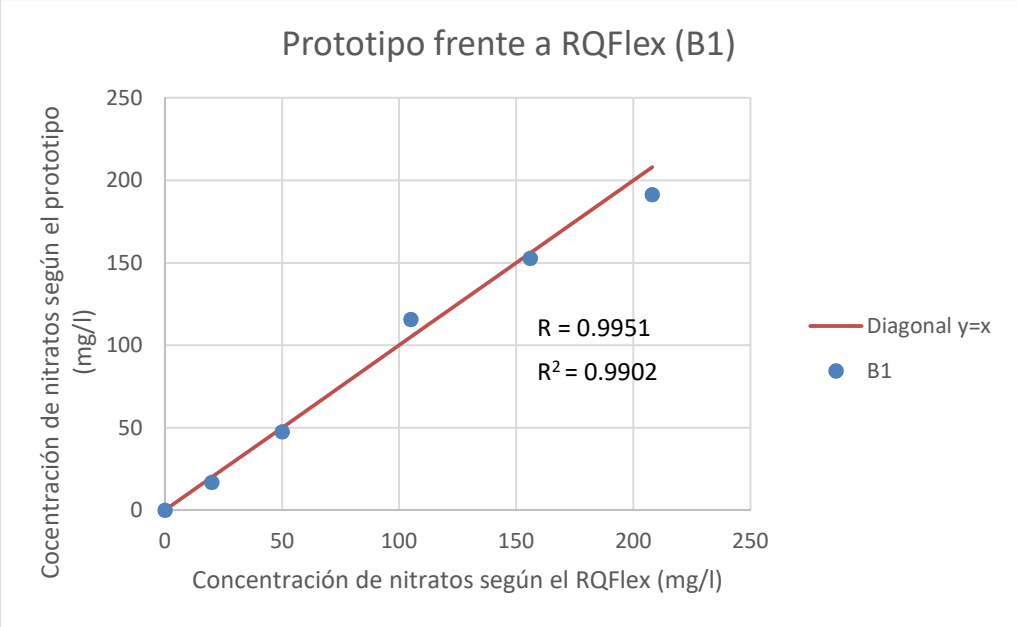


Ilustración 191: Mejor resultado del prototipo (concordancia entre los datos del dispositivo comercial y el prototipo).

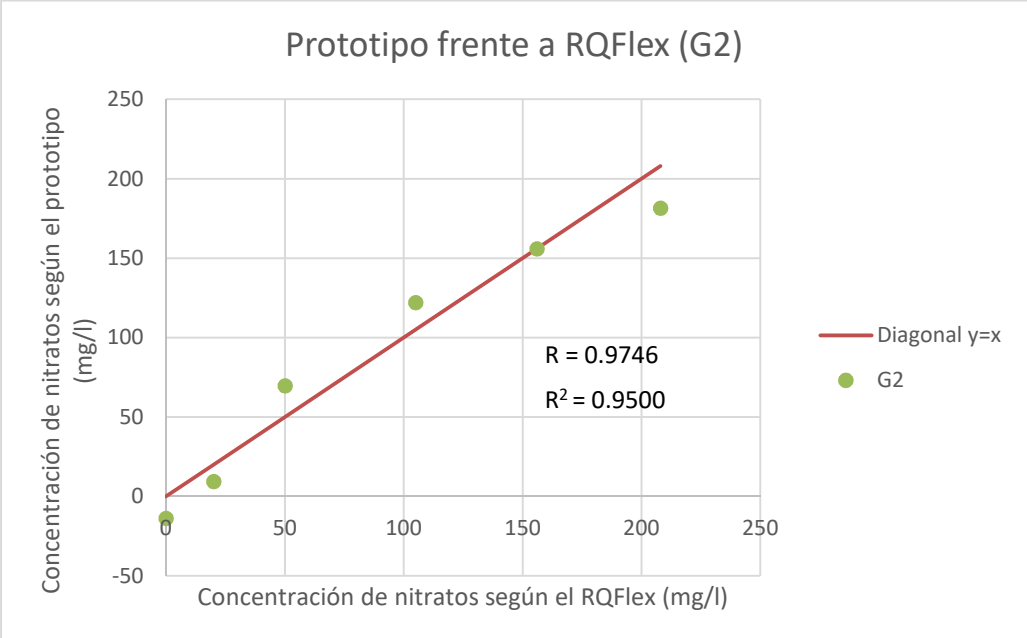


Ilustración 192: Peor resultado del prototipo (concordancia entre los datos del dispositivo comercial y el prototipo).

6.4 Conclusiones

Con el apartado anterior queda demostrado que es posible emplear el prototipo resultado del presente trabajo para la determinación del nivel de nitratos en agua con un error equiparable al de un sistema comercial. Si se toma la coordenada azul, es posible obtener datos con una desviación media por medida menor al 1.6 % respecto al sistema comercial. Los mejores resultados del prototipo se obtienen para la primera zona reactiva, puesto que los de la segunda se ven contaminados por el fondo (se termina la tira).

Sin embargo, hay que tener en cuenta varios aspectos sobre los equipos con los que se efectuaron las pruebas.

En primer lugar, el RQFlex 10 es un reflectómetro comercial que proporciona datos con un coeficiente de determinación altísimo. En el caso de los nitratos, la desviación estándar por medida que se determinó es de 5 mg/l en una escala que como máximo llega a 225 mg/l, es decir, se obtienen medidas muy fiables. Además, los resultados son proporcionados en aproximadamente 1 minuto, mientras que el prototipo requiere un mínimo de 4 minutos.

No obstante, el trabajo con el RQFlex 10 puso de manifiesto algunas de sus desventajas. La primera de ellas es la colocación de las muestras, que se efectúa mediante una pinza de plástico no muy ergonómica y que falla en ocasiones. Si a eso se le añade que para una optimización de los resultados se ha de realizar la inserción de la muestra en el equipo en un plazo de entre 5 y 10 segundos, el resultado es que la colocación de las tiras reactivas no es tan fácil como cabría esperar en un equipo comercial.

Otro aspecto negativo del equipo es que sólo admite las tiras reactivas de la empresa que lo produce. Se trata de una estrategia comercial muy común que, sin embargo, limita las opciones del usuario a la hora de realizar tests de control, puesto que sólo tendrá acceso a los que comercialice la empresa. Las tiras de nitratos de Merck tienen un coste aproximado de 1 € por tira, que habría que añadir al del reflectómetro. Además, limita enormemente las opciones de desarrollo de tiras reactivas para nuevos analitos, dificultando la investigación en este campo.

Por otra parte, el precio del equipo es muy elevado. El personal del laboratorio en el que se realizaron las pruebas reveló un coste del RQFlex 10 de aproximadamente 3000 €. Si bien es cierto que este equipo permite acceso un gran número de tests de control a través

de las tiras reactivas que vende la empresa, cuenta con ciertas limitaciones que llevan a pensar que su precio podría ser exagerado para un equipo tan poco versátil.

Una vez vistas algunas de las características del sistema comercial se puede pasar a tratar el prototipo desarrollado en el presente trabajo.

Lo primero que hay que tener en cuenta es que se trata de un equipo de bajo coste no optimizado, es decir, es la primera aproximación para intentar encontrar una alternativa a los sistemas comerciales. Por ello, presenta algunos fallos estructurales, como una base no lo suficientemente horizontal o un cristal para la colocación de muestras que se encuentra ligeramente inclinado. A pesar de estos defectos, que sería posible corregir en una simple mejora del prototipo aquí presentado, los resultados obtenidos son equiparables a los del equipo comercial.

El tiempo de lectura de muestra es de 4 minutos, bastante mayor que en el RQFlex 10. No obstante, hay que tener en cuenta que con un motor más potente sería posible igualar e incluso mejorar la velocidad de lectura.

Además, en el momento de la redacción de este capítulo no se contaba con la función de lectura única en el software programado, pero sería interesante comprobar su eficacia y compararla con la del equipo comercial. Para ello, habría que colocar las muestras manualmente sobre el sensor, pero se obtendrían resultados a la misma velocidad que en el RQFlex 10.

Otro punto a tener en cuenta es que el escaneo con el prototipo proporciona una mayor cantidad de información, puesto que el resultado es una gráfica que representa toda la tira. Así, es posible determinar variaciones locales en la concentración.

Sin embargo, una de las características más remarcables del prototipo es su versatilidad. Su construcción permite analizar sin problemas una gran variedad de tipos de muestras, desde tiras reactivas de cualquier tipo o empresa hasta impresiones en hojas de papel. Todo ello con una precisión equiparable a la de una equipo comercial muy caro y que podría ser mejorada en futuros desarrollos.

Finalmente, hay que mencionar el bajo coste del equipo, que ronda los 100 €, como se mostrará en el siguiente capítulo. Por estas razones, es idóneo para el análisis químico de tiras reactivas, en concreto para la detección de nitratos en agua potable. El procedimiento que se ha seguido para la detección de nitratos puede ser llevado a cabo para cualquier otra tira reactiva, lo que demuestra nuevamente el gran potencial del prototipo una vez desarrollado y optimizado.

7 Presupuesto

Material	**Cantidad**	**Precio total (€)**
Sensor RGB TCS34725	1	11.99
Arduino UNO con cable USB	1	12.25
Driver L298N	1	1.32
Cables jumper	40	0.60
Conector de chasis para alimentación	1	0.66
Tornillos	1 caja (100 uds.)	2.95
Arandelas	1 caja (100 uds.)	0.83
Z-ULTRAT	200 gr	11.62
Barra roscada M3	1	0.7
Barra de latón de 4 mm	1	2.95
Tuercas	1 caja (20 uds.)	2.75
Caja	1	15.95
Rodamientos	2	9.42
Fuente de alimentación	1	14.95
SM-42BYG011-25	1	16.5
Cartón pluma	1 lámina	0.3
TOTAL		**105.74**

Tabla 23: Coste de los materiales empleados para la construcción del prototipo y precio total de éste.

8 Recapitulación, conclusiones y futuros desarrollos

La meta principal del presente trabajo era la elaboración de un prototipo mecánico destinado al análisis de muestras químicas y el software necesario para manejarlo. Concretamente, el sistema se ha de basar en al sensor RGB TCS34725.

Con tal fin, en primer lugar se investigaron los productos relacionados que se puede hallar en el mercado, tales como reflectómetros, escáners o aplicaciones que utilicen un sensor RGB.

Seguidamente se revisó el marco teórico requerido para la completa compresión del problema a solucionar. Para ello se buscó información sobre los conceptos básicos físicos, electrónicos, químicos, mecánicos e informáticos requeridos.

Además, se caracterizó el sensor RGB del que se dispone, con la finalidad de que éste pueda trabajar en condiciones óptimas en el prototipo mecánico.

A partir de los conocimientos adquiridos fue posible construir un sistema mecánico y escribir un software adecuado para su manejo.

Una vez acabado el prototipo, se comprobó su respuesta y su capacidad de detección a distintos parámetros.

Dado que los resultados fueron satisfactorios, se procedió a emplear el prototipo en una aplicación real: la determinación de nitratos en agua. Para evaluar la calidad de las medidas realizadas, se contó con el reflectómetro comercial RQFlex.

Se descubrió que los valores de concentración medidos con el prototipo son algo peores que los del sistema comercial, pero no demasiado. Concretamente, se observó un coeficiente de determinación de 0.999 en el RQFlex y de 0.9858 en el prototipo. Además, la desviación relativa media por medida es de 2.22 % en el RQFlex y de 3.8 % en el sistema desarrollado. Si se compara el precio de ambos productos (el RQFlex cuesta 3000 € y el escáner RGB apenas 100 €) se pone de manifiesto que los resultados obtenidos son muy buenos.

Si además se tiene en cuenta la altísima versatilidad del equipo diseñado, que permite el análisis de un gran número y variedad de muestras y el detalle de los resultados, se acaba con un sistema que podría ser interesante desarrollar más a fondo.

Un punto muy importante es que dichos resultados son mejorables. El prototipo obtenido en el presente trabajo se trata únicamente de una primera aproximación a la idea de aplicar un sensor RGB de bajo coste para el análisis de muestras químicas. Por lo tanto, para futuros desarrollos, habría que tener en cuenta algunos puntos que no se pudieron tratar con suficiente detalle en el presente trabajo.

En primer lugar, dada la altísima sensibilidad del TCS34725 a la distancia de la muestra, es de vital importancia que la base del sistema sea completamente plana. Asimismo, el cristal para la colocación de las muestras debería encontrarse en paralelo al sensor. Este no ha sido el caso en el presente trabajo, puesto que la base algo combada de la caja comprada y los fallos en la impresión de la tapa para el cristal no han permitido optimizar la colocación de la muestra. No obstante, los resultados han sido muy buenos. Si se solucionase el problema aquí descrito, es casi seguro que se podría llegar a igualar o incluso a superar los valores obtenidos con un sistema comercial.

En segundo lugar, sería de ayuda contar con un motor más potente para futuros desarrollos. Esto permitiría realizar lecturas a una velocidad comparable a la existente en sistemas comerciales.

En tercer lugar, hay que mencionar que el software programado podría ser mejorado. La calibración podría ser realizada aparte del escaneo, se podría incluir una función para guardar y aplicar rectas de calibrado directamente a los resultados y, en general, se podría hacer el código más robusto.

Para acabar, se puede afirmar que como primera aproximación a la aplicación de un sensor RGB de bajo coste al análisis químico, los resultados obtenidos son satisfactorios. Además, se ha demostrado que la investigación y el desarrollo en esta dirección podrían ser interesantes en futuras aplicaciones.

Bibliografía

[1] A.H.Fauzi et al: *Automated machine for sorting Sarawak pepper berries*, August 2015, DOI: 10.1109/CITA.2015.7349840.

[2] L.Bhen khelifa et al: *Ant-Cams Network: a cooperative network model for silly cameras*, 2016, DOI: 10.1145/2967413.2967437.

[3] Stanka Baycheva et al: *Investigating the possibilities of document cameras for quality assessment of foodstuffs by measuring colors*, 2016, extracted from *Proceedings of the 11th International Conference On Virtual Learning*, ISSN: 1844-8933.

[4] Xinxhi Zhang et al: *Illumination adaptation with rapid-response color sensors*, 2014, DOI: 10.1117/12.2062105.

[5] János Schanda: *Tristimulus Color Measurement of Self-Luminous Sources. Colorimetry: Understanding the CIE System*, 2007, Wiley Interscience, DOI: 10.1002/9780470175637, ISBN 978-0-470-04904-4.

[6] Javier Corzo, https://bioquibi.webs.ull.es/practicas/2.pdf, version del 11/11/2007

[7] Susan K Strasinger et al*: Análisis de orina y de los líquidos corporales (5ª ed.),* 2008, Editorial Panamericana, ISBN 978-950-06-1938-7.

[8] Dr. Thomas Bitter et al: *Elemente Chemie II*, 2013, Ernst Klett Verlag, ISBN 3-12-756700-6

[9] https://www.amazon.com, versión de febrero de 2018

[10] Michael Bass: *Handbook of Optics Volume II - Devices, Measurements and Properties, 2nd Ed.*, McGraw-Hill 1995, ISBN 978-0-07-047974-6

[11] John James: *Spectrograph Design Fundamentals*, 2007 (Cambridge University Press), ISBN 0-521-86463-1

[12] Robert L. Pecsok y L. Donal Shields: *Métodos modernos de análisis químicos*, 1983, Editorial limusa

[13] Kodak Alaris Information: *i5850 Scanner information and accessories – Management*, www.kodakalaris.com, retrieved 24/09/2017.

[14] J. Sachs: *Digital Image Basics, Digital Light & Color*, 2015, Archived from the original.

[15] James R. Janesick: *Scientific charge-coupled devices*, 2001, SPIE Press. p. 4. ISBN 978-0-8194-3698-6.

[16] Canon: *Canon CIS technology overview*, http://compo.canon/en/, versión de enero del 2012

[17] Ams datasheet: *TCS34725 datasheet*, versión 08/02/2016

[18] Resultados de la búsqueda del término "sensor RGB" en https://es.aliexpress.com, versión del 12/12/2017

[19] Gary Waldman: *Introduction to light : the physics of light, vision, and color*, Mineola: Dover Publications, ISBN 978-0-486-42118-6.

[20] Stockman, MacLeod & Johnson: *Journal of the Optical Society of America A*, 1993, http://psy.ucsd.edu/~dmacleod/publications/61StockmanMacLeodJohnson1993.pdf, versión de marzo del 2007

[21] www.encyclopedia.com*: Electromagnetic Spectrum facts, information, pictures, articles about Electromagnetic Spectrum*, versión de septiembre del 2017

[22] Mary Jo Nye (editor): *The Cambridge History of Science: The Modern Physical and Mathematical Sciences*, Cambridge University Press, ISBN 978-0-521-57199-9.

[23] John C. D. Brand: *Lines of light: the sources of dispersive spectroscopy*, 1995, ISBN 978-2-88449-163-1.

[24] A.D. Logvinenko: *The geometric structure of color*, 2015, Journal of Vision, 6. DOI: 10.1167/15.1.16

[25] H. Grassmann: Zur Theorie der Farbenmischung, 1853, Poggendorffs Annalen der Physik

[26] David Falk, Dieter Brill, David Stork: *Seeing the Light*, New York 1986, ISBN 0-471-60385-6

[27] Commission Internationale De L'Eclairage: *CIE 15:2004 – Colorimetry*, ISBN 3-901906-33-9

[28] Harald Küppers: *Die Farbenlehre der Fernseh-, Foto- und Drucktechnik*, 1985, DuMont

[29] R. W. G. Hunt: *The Reproduction of Colour (6th ed.)*, 2004, Chichester UK: Wiley–IS&T Series in Imaging Science and Technology, ISBN 0-470-02425-9

[30] Mark D. Fairchild: *Color Appearance Models, 2013*, Wiley-IS&T Series in Imaging Science and Technology (3 ed.), Hoboken: John Wiley & Sons, ISBN 978-1-119-96703-3.

[31] Charles Poynton: *Digital Video and HDTV Algorithms and Interfaces*. Morgan Kaufmann Publishers, San Francisco 2003, ISBN 1-55860-792-7

[32] Keith Jack: *Video Demystified*, ISBN 1-878707-09-4

[33] Jander/Blasius: *Anorganische Chemie I - Einführung & Qualitative Analyse*, 17. Auflage, Hirzel, 2012, ISBN 978-3-7776-2134-0

[34] hyperphysics.phy-astr.gsu.edu, *Photodiode slide*, consultada el 20/12/2017

[35] Rudolf F. Graf: *Modern Dictionary of Electronics (7 ed.)*, 1999, Newnes, ISBN 0080511988.

[36] Datasheet del Arduino UNO, http://datasheet.octopart.com/A000066-Arduino-datasheet-38879526.pdf, consultado el 11/11/2017

[37] Atmel: *Atmega328P datasheet*, versión de noviembre de 2016

[38] Bela G. Liptak*: Instrument Engineers' Handbook: Process Control and Optimization*, 2005, ISBN 978-0-8493-1081-2

[39] Luis Llamas: *https://www.luisllamas.es/motor-paso-paso-28byj-48-arduino-driver-uln2003/*, consultado el 2/09/2017

[40] Prometec: *https://www.prometec.net/motores-paso-a-paso/*, consultado el 2/09/2017

[41] Kiatronics: *28BYJ-48 5V Stepper Motor*, versión de 2017

[42] Sparkfun*: ST-PM35-15-11C Mercury Motor Datasheet*, versión de 2017

[43] Sparkfun: *SM-42BYG011-25 Mercury Motor Datasheet*, versión del 27/03/2009

[44] Stephen Herman: *Industrial Motor Control*, Cengage Learning, 2009 chapter 11 "Limit Switches", ISBN 1435442393

[45] Aprendiendo Arduino: https://aprendiendoarduino.wordpress.com/2016/03/30/instalacion-del-ide-arduino/, versión del 30/03/2016.

[46] Tim Lindholm, Frank Yellin: *The Java Virtual Machine Specification*, Addison-Wesley, 1996

[47] MathWorks: *https://es.mathworks.com/*, consultado el 13/11/2017

[48] Dassault Systemes SolidWorks Corporation: *Introducción a SolidWorks*, versión de 2015

[49] *J.M.Bland et al.: "Statistics notes: measurement error"*, 1996, DOI:10.1136/bmj.312.7047.1654

Anexo A: Programa de Arduino para la caracterización del TCS34725

```
#include <Wire.h>
#include "Adafruit_TCS34725.h"

/* Example code for the Adafruit TCS34725 breakout library */

/* Connect SCL    to analog 5
   Connect SDA    to analog 4
   Connect VDD    to 3.3V DC
   Connect GROUND to common ground */

/* Initialise with default values (int time = 2.4ms, gain = 1x) */
// Adafruit_TCS34725 tcs = Adafruit_TCS34725();

/* Initialise with specific int time and gain values */
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATION-
TIME_700MS, TCS34725_GAIN_1X);

void setup(void) {
  Serial.begin(9600);

  if (tcs.begin()) {
    Serial.println("Found sensor");
  } else {
    Serial.println("No TCS34725 found ... check your connections");
    while (1);
  }

  // Now we're ready to get readings!
}

void loop(void) {
  uint16_t r, g, b, c, colorTemp, lux;

  tcs.getRawData(&r, &g, &b, &c);
  colorTemp = tcs.calculateColorTemperature(r, g, b);
  lux = tcs.calculateLux(r, g, b);

  Serial.print("Color Temp: "); Serial.print(colorTemp, DEC); Se-
rial.print(" K - ");
  Serial.print("Lux: "); Serial.print(lux, DEC); Serial.print(" - ");
  Serial.print("R: "); Serial.print(r, DEC); Serial.print(" ");
  Serial.print("G: "); Serial.print(g, DEC); Serial.print(" ");
  Serial.print("B: "); Serial.print(b, DEC); Serial.print(" ");
  Serial.print("C: "); Serial.print(c, DEC); Serial.print(" ");
  Serial.println(" ");
}
```

Anexo B: Programas de la caracterización de los motores

Programa de Arduino para la caracterización del motor a velocidad constante.

```
// ConstantSpeed.pde
// -*- mode: C++ -*-
//
// Shows how to run AccelStepper in the simplest,
// fixed speed mode with no accelerations
/// \author  Mike McCauley (mikem@open.com.au)
// Copyright (C) 2009 Mike McCauley
// $Id: HRFMessage.h,v 1.1 2009/08/15 05:32:58 mikem Exp mikem $

#include <AccelStepper.h>

AccelStepper stepper(4,8,9,10,11); // nº conexiones, pines

void setup()
{
   stepper.setSpeed(50);       //se cambia la velocidad
}

void loop()
{
   stepper.runSpeed();
}
```

Programa de Arduino para la caracterización del motor simulando la toma de datos con el sensor.

```
//============== librerías ========================================
#include <AccelStepper.h>
#include <Wire.h>
//============== constantes del motor ==============================
#define  SPEED       1000       // motor speed (RPM) 11 con 9V
#define  STEPSREV    208

#define  COIL_1X     8
#define  COIL_2X     9
#define  COIL_3X     10
#define  COIL_4X     11

unsigned long previousPos;
unsigned long currentPos;
const long scanInterval = 208;  //intervalo de toma de datos en pasos

// create the instances of the stepper class.
AccelStepper stepperX(4, COIL_1X, COIL_2X, COIL_3X, COIL_4X);

void setup() {
```

```
  stepperX.setMaxSpeed(SPEED);          //inicialización del motor
  stepperX.setAcceleration(100000.0);

  stepperX.setCurrentPosition(0);
  previousPos = stepperX.currentPosition();
}

void loop() {
  currentPos = stepperX.currentPosition();
  stepperX.run();
  if (currentPos - previousPos >= scanInterval) { //cada cierto tiempo
        previousPos = currentPos;
        delay(1000);                              //se pausa durante 1s
      }

}
```

Anexo C: Programa de Arduino del prototipo

```
//============== librerías ========================================
#include <AccelStepper.h>
#include <Wire.h>
#include <Adafruit_TCS34725.h>

//============== constantes del motor ==============================
#define  SPEED       1000       // motor speed (RPM) 11 con 9V
#define  STEPSREV    208

#define  COIL_1X     8
#define  COIL_2X     9
#define  COIL_3X     10
#define  COIL_4X     11

//============== definición sensor y motor =========================
//Motor (nº conexiones y pines)
AccelStepper stepperX(4, COIL_1X, COIL_2X, COIL_3X, COIL_4X);

//sensor
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATION-
TIME_700MS, TCS34725_GAIN_1X);

//============== variables =========================================
int newXpos;          // Nueva posición del motor

//variables para el escaneo
unsigned long previousPos;
unsigned long currentPos;
const long scanInterval = 208;             //208 - 0,5 mm
int s = 1;
int lectura = 0;

long Xmax;
long XoCal;

long XmaxRead;

long errorRead;


int calStatus = 0;
int valStatus = 0;
int matlabData;
int calibracion = 0;

long startTime;
long endTime;
long scanTime;

//Variables para comprobar el estado de conexión del motor
long hi;
long bye;
long tim1;
long tom1;
```

```
long tim2;
long tom2;
long timtom = 2000;
int count;
long t1; long t2;

//============== functions ====================================
// Función de escaneo
void scanX(int newX) {
  uint16_t r, g, b, c, colorTemp, lux;
  delay(500);
  if (calStatus == 1 & digitalRead(13) == HIGH) {      //si se ha ca-
librado y el sensor se encuentra en FC1
    s = 1;                                             //estado de es-
caneo
    previousPos = stepperX.currentPosition();
    delay(500);
    while (digitalRead(12) == LOW) {                    //mientras no
llegues a FC2
      currentPos = stepperX.currentPosi-
tion();                                 //guarda posiciÃ³n actual
      stepperX.moveTo(30000);
      stepperX.run();                    //avanza

      if (currentPos - previousPos >= scanInterval) {   //toma de da-
tos en un intervalo determinado
        previousPos = currentPos;
        tcs.getRawData(&r, &g, &b, &c);                 //toma de da-
tos
        colorTemp = tcs.calculateColorTemperature(r, g, b);
        lux = tcs.calculateLux(r, g, b);
        Serial.print(s, DEC); Serial.print("\n");       //envío de da-
tos a matlab
        Serial.print(r, DEC); Serial.print("\n");
        Serial.print(g, DEC); Serial.print("\n");
        Serial.print(b, DEC); Serial.print("\n");
        Serial.print(lux, DEC); Serial.print("\n");
        Serial.print(colorTemp, DEC); Serial.print("\n");
        Serial.print(c, DEC); Serial.print("\n");
        lectura = lectura + 1;                          //contador de nú-
mero de lecturas
      }
      if (digitalRead(12) == HIGH) {
        XmaxRead = stepperX.currentPosition();       //posición actual
        s = 0;                                       //escaneo finali-
zado
        Serial.print(s, DEC); Serial.print("\n");    //se comunica a
matlab
      }
    }
    t1=millis();
    while (digitalRead(13) == LOW) {                 //se vuelve a la
posición de inicio
      stepperX.moveTo(-30000);
      stepperX.run();
      t2 = millis();
      if (t2 - t1 >= 2000) {
        Serial.println(1);
        t1 = millis();
      }
    }
    if (digitalRead(13) == HIGH) {
```

```
      Serial.println(0);
      stepperX.setCurrentPosition(0);
    }
    endTime = millis();
  }
}

//función de calibrado RGB
void calX() {
  uint16_t r, g, b, c, colorTemp, lux;
  tcs.getRawData(&r, &g, &b, &c);
  Serial.print(r, DEC); Serial.print("\n");
  Serial.print(g, DEC); Serial.print("\n");
  Serial.print(b, DEC); Serial.print("\n");
}
//función lectura única
void singleRead() {
  uint16_t r, g, b, c, colorTemp, lux;
  tcs.getRawData(&r, &g, &b, &c);
  colorTemp = tcs.calculateColorTemperature(r, g, b);
  lux = tcs.calculateLux(r, g, b);
  Serial.print(r, DEC); Serial.print("\n");
  Serial.print(g, DEC); Serial.print("\n");
  Serial.print(b, DEC); Serial.print("\n");
  Serial.print(lux, DEC); Serial.print("\n");
  Serial.print(colorTemp, DEC); Serial.print("\n");
  Serial.print(c, DEC); Serial.print("\n");
}

//función de calibrado del recorrido
void streckeCal() {
  if (calStatus == 0) {   //si no se ha calibrado/se ha calibrado mal
    t1 = millis();
    while (digitalRead(13) == LOW) {    //retroceder, comunicar con
Matlab
      stepperX.moveTo(-30000);
      stepperX.run();
      t2 = millis();
      if (t2 - t1 >= 2000) {
        Serial.println(1);
        t1 = millis();
      }
    }
    delay(500);
    if (digitalRead(13) == HIGH) {       //si has llegado a FC1
      stepperX.setCurrentPosition(0);    //definición del 0

     t1 = millis;
    while (digitalRead(12) == LOW) {   //mientras no llegues al FC fi-
nal (FC2)
      stepperX.moveTo(30000);         //avanza y comunica con matlab
      stepperX.run();
      t2 = millis();
      if (t2 - t1 >= 2000) {
        Serial.println(1);
        t1 = millis();
      }
    }
    }
    if (digitalRead(12) == HIGH) {         //si has llegado a FC2
      Xmax = stepperX.currentPosition();
```

```
      delay(500);                                   //pequeño descanso
    t1 = millis();
    while (digitalRead(13) == LOW) {      //mientras no llegues a FC1
      stepperX.moveTo(-30000);             //retrocede y comunica
      stepperX.run();
      t2 = millis();
      if (t2 - t1 >= 2000) {
        Serial.println(1);
        t1 = millis();
      }
    }
    }
    if (digitalRead(13) == HIGH) {    //si has llegado a FC1
      calStatus = 1;
      XoCal = stepperX.currentPosition();
      calibracion = 0;
      Serial.println(calibracion);
      Serial.print(calStatus); Serial.print("\n"); //calibración exi-
tosa, se comunica
    }else {
      calibracion = 0;
      Serial.println(calibracion);
      Serial.print(calStatus); Serial.print("\n"); //calibración fa-
llida, se comunica
    }
  }
  delay(1000);
}
//función de validación
void validation() {
  if (calStatus == 1 & digitalRead(13) == HIGH) { //si se ha calibrado
y estamos en el "0"
    if (Xmax >= XmaxRead) {                       //cálculo del error
      errorRead = (Xmax - XmaxRead) + XoCal;
    }
    if (XmaxRead >= Xmax) {
      errorRead = (XmaxRead - Xmax) + XoCal;
    }
    if (errorRead <= 208 & errorRead >= -208) {
      valStatus = 1;
      Serial.print(valStatus); Serial.print("\n");    //validación
exitosa, se comunica
    }
  }
  else {
    valStatus = 0;
    Serial.print(valStatus); Serial.print("\n");    //validation fa-
llida, se comunica
  }

  stepperX.disableOutputs();
  lectura = 0;                    //reset lecturas
  calStatus = 0;                  //reset calStatus
  valStatus = 0;                  //reset valStatus
}

void handshakeMotor() {   //comprobación de si el motor está conectado
  count = 0;
  if (digitalRead(13) == LOW && count == 0) {
    tom1 = millis();        //start
    tim1 = millis();        //current time
```

```
    while (digitalRead(13) == LOW && (tim1 - tom1) <= timtom)
{  //mientras no llegues al FC final (FC2)
      stepperX.moveTo(-30000);          //avanza
      stepperX.run();
      tim1 = millis();
    }
    if (digitalRead(13) == HIGH) {
      Serial.println(1);                //conexión correcta, comunica
    }
    else {
      Serial.println(0);              //conexión incorrecta, comunica
    }
    count = 1;
  }

  if (digitalRead(13) == HIGH && count == 0) {
    tom2 = millis();
    tim2 = millis();

    while (digitalRead(13) == HIGH && (tim2 - tom2) <= timtom)
{      //mientras no llegues al FC final (FC2)
      stepperX.moveTo(30000);          //avanza
      stepperX.run();
      tim2 = millis();
    }
    if (digitalRead(13) == LOW) {
      Serial.println(1);            //conexión correcta
    }
    else {
      Serial.println(0);     //conexión incorrecta
    }
    count = 1;
  }
}
void resetMotorPosition() {  //función para colocar el motor en el
punto de inicio
  t1 = millis();
  while (digitalRead(13) == LOW) {
    stepperX.moveTo(-30000);          //retrocede
    stepperX.run();
    t2 = millis();
    if (t2 - t1 >= 2000) {
      Serial.println(0);
      t1 = millis();
    }
  }
  if (digitalRead(13) == HIGH) {
    Serial.println(1);                //motor ha llegado al inicio
  }
  else{
    Serial.println(0);                //motor no ha llegado
  }
}
//============== SETUP ============================================
void setup() {
  Serial.begin(9600);

  pinMode(13, INPUT);  //FC1
  pinMode(12, INPUT);  //FC2

  digitalWrite(13, HIGH); //activa la resistencia pullup
  digitalWrite(12, HIGH); //activa la resistencia pullup
```

```
  tcs.begin();              //inicialización sensor
  stepperX.setMaxSpeed(SPEED);     //inicialización motor
  stepperX.setAcceleration(100000.0);
}

//============== LOOP ================================================
void loop() {
  if (Serial.available() > 0)     // si hay datos que leer
  {
    matlabData = Serial.read(); // lee los datos
    if (matlabData == '1') {     //calibrado, escaneo, validación
      startTime = millis();
      streckeCal();
      scanX(newXpos);
      validation();
    }
    if (matlabData == '2') { // Calibración RGB
      calX();
    }
    if (matlabData == '3') {//Reset de la posicón del motor
      resetMotorPosition();
    }
    if (matlabData == '4') { //Comprobación del estado de la conexión
con motor
      handshakeMotor();
    }
    if (matlabData == '5') { //Comunicación con matlab
      Serial.println(1);  //respuesta
    }
    if (matlabData == '6') { //Lectura única
      singleRead();
    }
  }
}
```

Anexo D: Programa de Matlab del prototipo

```
function varargout = gui1(varargin)
% GUI1 MATLAB code for gui1.fig
%      GUI1, by itself, creates a new GUI1 or raises the existing
%      singleton*.
%
%      H = GUI1 returns the handle to a new GUI1 or the handle to
%      the existing singleton*.
%
%      GUI1('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI1.M with the given input argu-
ments.
%
%      GUI1('Property','Value',...) creates a new GUI1 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before gui1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property ap-
plication
%      stop.  All inputs are passed to gui1_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui1

% Last Modified by GUIDE v2.5 04-Jan-2018 16:25:24

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui1_OpeningFcn, ...
                   'gui_OutputFcn',  @gui1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui1 is made visible.  //inicialización
function gui1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui1 (see VARARGIN)


clear scan_data  trans_data cal_data trans error;

global ard;
global scan_data;
global raw_data;
global trans_data;
global cal_data;
global trans;
global error;
global s;
global add_scan_data;
global add_plot_data;
global index;
global lux_data;
global colorTemp_data;
global c_data currentFolder dataFolder scanFolder rawFolder calscan-
Folder calFolder minFolder

%add_scan_data=1;

currentFolder=pwd;
dataFolder='\data\';
dataFolder=strcat(currentFolder,dataFolder);
cal=exist(dataFolder,'dir');
if cal ~=7
mkdir(dataFolder);
end

scanFolder='\data\scan_data\';
scanFolder=strcat(currentFolder,scanFolder);
cal=exist(scanFolder,'dir');
if cal ~=7
mkdir(scanFolder);
end

rawFolder='\data\scan_data\raw_data\';
rawFolder=strcat(currentFolder,rawFolder);
cal=exist(rawFolder,'dir');
if cal ~=7
mkdir(rawFolder);
end

calscanFolder='\data\scan_data\calibrated_data\';
calscanFolder=strcat(currentFolder,calscanFolder);
cal=exist(calscanFolder,'dir');
if cal ~=7
mkdir(calscanFolder);
end

calFolder='\data\cal_data\';
calFolder=strcat(currentFolder,calFolder);
cal=exist(calFolder,'dir');
if cal ~=7
mkdir(calFolder);
```

```
end

minFolder='\data\min_data\';
minFolder=strcat(currentFolder,minFolder);
cal=exist(minFolder,'dir');
if cal ~=7
mkdir(minFolder);
end


%Inicialización puerto
Puertos_Activos=instrfind; % Lee los puertos activos
if isempty(Puertos_Activos)==0 % Comprueba si hay puertos activos
    fclose(Puertos_Activos); % Cierra los puertos activos
    delete(Puertos_Activos) % Borra la variable Puertos_Activos
    clear Puertos_Activos % Destruye la variable Puertos_Activos
end

set(handles.text7,'BackgroundColor',[1,0,0]);
set(handles.text8,'BackgroundColor',[1,0,0]);
set(handles.text9,'BackgroundColor',[1,0,0]);

%Botones grupo Single Read
set(handles.togglebutton5,'enable','off');
set(handles.pushbutton42,'enable','off');
set(handles.pushbutton29,'enable','off');
set(handles.pushbutton33,'enable','off');
set(handles.radiobutton11,'enable','off');

%Botones grupo Plot
set(handles.togglebutton2,'enable','off');
set(handles.pushbutton20,'enable','off');
set(handles.pushbutton16,'enable','off');
set(handles.pushbutton43,'enable','off');
set(handles.pushbutton23,'enable','off');
set(handles.pushbutton14,'enable','off');
set(handles.pushbutton15,'enable','off');

set(handles.radiobutton5,'value',1);
set(handles.radiobutton6,'value',1);
set(handles.radiobutton7,'value',1);
set(handles.radiobutton8,'value',1);
set(handles.radiobutton9,'value',1);
set(handles.radiobutton10,'value',1);

set(handles.radiobutton5,'enable','off');
set(handles.radiobutton6,'enable','off');
set(handles.radiobutton7,'enable','off');
set(handles.radiobutton8,'enable','off');
set(handles.radiobutton9,'enable','off');
set(handles.radiobutton10,'enable','off');

%Botones grupo RGB calibration
set(handles.togglebutton3,'enable','off');
set(handles.pushbutton18,'enable','off');
set(handles.pushbutton1,'enable','off');
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','off');
set(handles.pushbutton4,'enable','off');
set(handles.pushbutton6,'enable','off');
```

```
%Botones grupo scan
set(handles.pushbutton21,'enable','off');
set(handles.togglebutton1,'enable','off');
set(handles.pushbutton10,'enable','off');
set(handles.pushbutton12,'enable','off');
set(handles.pushbutton31,'enable','off');
set(handles.pushbutton35,'enable','off');

%Botones grupo mínimos
set(handles.togglebutton4,'enable','off');
set(handles.pushbutton41,'enable','off');
set(handles.pushbutton30,'enable','off');
set(handles.pushbutton25,'enable','off');
set(handles.pushbutton26,'enable','off');
set(handles.pushbutton28,'enable','off');
set(handles.pushbutton34,'enable','off');
set(handles.radiobutton2,'enable','off');
set(handles.radiobutton3,'enable','off');
set(handles.radiobutton4,'enable','off');
set(handles.radiobutton13,'enable','off');

set(handles.pushbutton22,'enable','off');
%set(handles.pushbutton24,'enable','off');
%set(handles.pushbutton19,'enable','off');
%set(handles.pushbutton44,'enable','off');
clc;

% Choose default command line output for gui1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);




% UIWAIT makes gui1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
global ard;
Puerto=getAvailableComPort(); %Busca los puertos disponibles
ard=serial(Puerto,'BaudRate',9600);
set(ard,'BaudRate',9600);
set(ard,'TimeOut',2);

try
    fopen(ard);           %handshake con puerto
    pause(2);
```

```
    set(handles.text7,'BackgroundColor',[0.5,1,0.5]); %conexión con
puerto correcta
    set(handles.text7, 'Value',1);
    set(handles.text7,'String','ONLINE');


    fprintf(ard,'%s',53); % activar handshake con arduino (5)
    pause(1);

    res=fscanf(ard,'%d\n');%leer respuesta de arduino

    if res==1
        set(handles.text8,'BackgroundColor',[0.5,1,0.5]); %conexión
con arduino correcta
        set(handles.text8, 'Value',1);
        set(handles.text8,'String','ONLINE');

        fprintf(ard,'%s',52); % activar handshake con motor
        pause(1);

        res2=fscanf(ard,'%d\n');%leer respuesta de arduino
        if res2==1
            set(handles.text9,'BackgroundColor',[0.5,1,0.5]); %conex-
ión con motor correcta
            set(handles.text9, 'Value',1);
            set(handles.text9,'String','ONLINE');
        else
            set(handles.text9,'BackgroundColor',[1,0,0]); %conexión
con motor incorrecta
            set(handles.text9, 'Value',0);
            h=errordlg('Communication with motor failed. Please check
whether the power supply is connected and try to reconnect.','ERROR');
        end
    else
        set(handles.text8,'BackgroundColor',[1,0,0]); %conexión con
arduino incorrecta
        set(handles.text8, 'Value',0);
        h=errordlg('Communication with arduino failed. Please check if
the correct programm is running on the Arduino board.','ERROR');
    end

catch
    set(handles.text7,'BackgroundColor',[1,0,0]); %conexión con puerto
incorrecta
    set(handles.text7, 'Value',0);
    set(handles.text8,'BackgroundColor',[1,0,0]); %conexión con puerto
incorrecta
    set(handles.text8, 'Value',0);
    set(handles.text9,'BackgroundColor',[1,0,0]); %conexión con puerto
incorrecta
    set(handles.text9, 'Value',0);
    h=errordlg('Serial port connection failed. Check if the USB cable
is connected properly or if there are any available serial
ports.','ERROR');
end
try fclose(ard);
catch
end

set(handles.togglebutton4,'enable','on');
set(handles.togglebutton4,'Value',0);
```

```
set(handles.pushbutton41,'enable','on');
set(handles.togglebutton2,'enable','on');
set(handles.togglebutton2,'Value',0);
set(handles.pushbutton20,'enable','on');

if get(handles.text7,'Value')==1 && get(handles.text8,'Value')==1
    set(handles.togglebutton3,'enable','on');
    set(handles.togglebutton3,'Value',0);
    set(handles.pushbutton18,'enable','on');
    set(handles.togglebutton5,'enable','on');
    set(handles.togglebutton5,'Value',0);
    set(handles.pushbutton42,'enable','on');

    set(handles.pushbutton22,'enable','on');
end

if get(handles.text7,'Value')==1 && get(handles.text8,'Value')==1 &&
get(handles.text9,'Value')==1
    set(handles.togglebutton1,'enable','on');
    set(handles.togglebutton1,'Value',0);
    set(handles.pushbutton21,'enable','on');
end


% --- Red calibration
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ard;
global rr gr br;

set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');
set(handles.pushbutton22,'enable','off');

set(handles.togglebutton3,'enable','off');
set(handles.pushbutton18,'enable','off');
set(handles.pushbutton1,'enable','off');
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','off');
set(handles.pushbutton4,'enable','off');
set(handles.pushbutton6,'enable','off');
drawnow

fopen(ard); % initiate arduino communication
pause(2);  %wait for communication to stabilize

fprintf(ard,'%s',50); % send answer variable content to arduino
pause(1);

rr=fscanf(ard,'%d\n');
pause(.1);

gr=fscanf(ard,'%d\n');
pause(.1);

br=fscanf(ard,'%d\n');
pause(.1);
```

```
pause(2);
fclose(ard);

set(handles.text3, 'Value',1);
set(handles.text3,'BackgroundColor',[0.5,1,0.5]);
set(handles.text3,'String','DONE');

set(handles.pushbutton24,'enable','on');
set(handles.pushbutton19,'enable','on');
set(handles.pushbutton44,'enable','on');
set(handles.pushbutton22,'enable','on');

set(handles.togglebutton3,'enable','on');
set(handles.pushbutton18,'enable','on');
set(handles.pushbutton1,'enable','on');
set(handles.pushbutton2,'enable','on');
set(handles.pushbutton3,'enable','on');
set(handles.pushbutton4,'enable','on');
set(handles.pushbutton6,'enable','on');
drawnow


% --- Green calibration
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ard;
global rg gg bg;

set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');
set(handles.pushbutton22,'enable','off');

set(handles.togglebutton3,'enable','off');
set(handles.pushbutton18,'enable','off');
set(handles.pushbutton1,'enable','off');
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','off');
set(handles.pushbutton4,'enable','off');
set(handles.pushbutton6,'enable','off');
drawnow

fopen(ard); % initiate arduino communication
pause(2);  %wait for communication to stabilize

fprintf(ard,'%s',50); % send answer variable content to arduino
pause(1);

rg=fscanf(ard,'%d\n');
pause(.1);
% handles.rg=rg;
% guidata(hObject,handles);
gg=fscanf(ard,'%d\n');
pause(.1);
% handles.gg=gg;
bg=fscanf(ard,'%d\n');
pause(.1);
```

```
% handles.bg;
% guidata(hObject,handles);
pause(2);
fclose(ard);

set(handles.text4, 'Value',1);
set(handles.text4,'BackgroundColor',[0.5,1,0.5]);
set(handles.text4,'String','DONE');

set(handles.pushbutton24,'enable','on');
set(handles.pushbutton19,'enable','on');
set(handles.pushbutton44,'enable','on');
set(handles.pushbutton22,'enable','on');

set(handles.togglebutton3,'enable','on');
set(handles.pushbutton18,'enable','on');
set(handles.pushbutton1,'enable','on');
set(handles.pushbutton2,'enable','on');
set(handles.pushbutton3,'enable','on');
set(handles.pushbutton4,'enable','on');
set(handles.pushbutton6,'enable','on');
drawnow


% --- Blue calibration
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global rb gb bb;
global ard;

set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');
set(handles.pushbutton22,'enable','off');

set(handles.togglebutton3,'enable','off');
set(handles.pushbutton18,'enable','off');
set(handles.pushbutton1,'enable','off');
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','off');
set(handles.pushbutton4,'enable','off');
set(handles.pushbutton6,'enable','off');
drawnow

fopen(ard); % initiate arduino communication
pause(2);  %wait for communication to stabilize
fprintf(ard,'%s',50); % send answer variable content to arduino
pause(1);
rb=fscanf(ard,'%d\n');
pause(.1);
gb=fscanf(ard,'%d\n');
pause(.1);
handles.gb=gb;
bb=fscanf(ard,'%d\n');

pause(2);
fclose(ard);
```

```
set(handles.text5, 'Value',1);
set(handles.text5,'BackgroundColor',[0.5,1,0.5]);
set(handles.text5,'String','DONE');

set(handles.pushbutton24,'enable','on');
set(handles.pushbutton19,'enable','on');
set(handles.pushbutton44,'enable','on');
set(handles.pushbutton22,'enable','on');

set(handles.togglebutton3,'enable','on');
set(handles.pushbutton18,'enable','on');
set(handles.pushbutton1,'enable','on');
set(handles.pushbutton2,'enable','on');
set(handles.pushbutton3,'enable','on');
set(handles.pushbutton4,'enable','on');
set(handles.pushbutton6,'enable','on');
drawnow


% --- Save calibration
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global rr gr br rg gg bg rb gb bb rn gn bn;
global trans;
global error;
global cal_data scanFolder rawFolder calscanFolder calFolder min-
Folder;

set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');
set(handles.pushbutton22,'enable','off');

set(handles.togglebutton3,'enable','off');
set(handles.pushbutton18,'enable','off');
set(handles.pushbutton1,'enable','off');
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','off');
set(handles.pushbutton4,'enable','off');
set(handles.pushbutton6,'enable','off');
drawnow

check_red=get(handles.text3, 'Value');
check_green=get(handles.text4, 'Value');
check_blue=get(handles.text5, 'Value');
check_black=get(handles.text6, 'Value');

if check_red==1 && check_green==1 && check_blue==1 && check_black==1

cal_data=zeros(3,3);
black=zeros(3,1);

cal_data(1,1)=rr;
cal_data(2,1)=gr;
cal_data(3,1)=br;

cal_data(1,2)=rg;
cal_data(2,2)=gg;
```

```
cal_data(3,2)=bg;

cal_data(1,3)=rb;
cal_data(2,3)=gb;
cal_data(3,3)=bb;

black(1)=rn;
black(2)=gn;
black(3)=bn;

alpha=[65535 0 0; 0 65535 0;0 0 65535];    %sistema cartesiano
beta=cal_data;                             %sistema RGB
beta1=inv(cal_data);                       %inversa del sistema RGB
trans=beta1*alpha;                         %matriz de transformación RGB a
cartesiano
error=trans*black;

filename=sprintf('Calibration_data_%s.mat',datestr(now,'dd-mm-yyyy HH-
MM'));
%save(filename,'trans','error')
save([calFolder filename],'trans','error')

set(handles.text3,'BackgroundColor',[0.5,0.5,0.5]);
set(handles.text3,'String','NO DATA');
set(handles.text3, 'Value',0);

set(handles.text4,'BackgroundColor',[0.5,0.5,0.5]);
set(handles.text4,'String','NO DATA');
set(handles.text4, 'Value',0);

set(handles.text5,'BackgroundColor',[0.5,0.5,0.5]);
set(handles.text5,'String','NO DATA');
set(handles.text5, 'Value',0);

set(handles.text6,'BackgroundColor',[0.5,0.5,0.5]);
set(handles.text6,'String','NO DATA');
set(handles.text3, 'Value',0);

h=msgbox('Calibration saved','Gui1');
else
warndlg('The is not enough data to generate a calibration file. Scan
the remaining color samplings before saving.', 'WARNING');
end

set(handles.pushbutton24,'enable','on');
set(handles.pushbutton19,'enable','on');
set(handles.pushbutton44,'enable','on');
set(handles.pushbutton22,'enable','on');

set(handles.togglebutton3,'enable','on');
set(handles.pushbutton18,'enable','on');
set(handles.pushbutton1,'enable','on');
set(handles.pushbutton2,'enable','on');
set(handles.pushbutton3,'enable','on');
set(handles.pushbutton4,'enable','on');
set(handles.pushbutton6,'enable','on');
drawnow


% --- Black calibration
```

```
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ard;
global rn gn bn;

set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');
set(handles.pushbutton22,'enable','off');

set(handles.togglebutton3,'enable','off');
set(handles.pushbutton18,'enable','off');
set(handles.pushbutton1,'enable','off');
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','off');
set(handles.pushbutton4,'enable','off');
set(handles.pushbutton6,'enable','off');
drawnow

fopen(ard); % initiate arduino communication
pause(2);  %wait for communication to stabilize

fprintf(ard,'%s',50); % send answer variable content to arduino
pause(1);
rn=fscanf(ard,'%d\n');
pause(.1);
gn=fscanf(ard,'%d\n');
pause(.1);
bn=fscanf(ard,'%d\n');
pause(.1);
pause(2);
fclose(ard);

set(handles.text6, 'Value',1);
set(handles.text6,'BackgroundColor',[0.5,1,0.5]);
set(handles.text6,'String','DONE');

set(handles.pushbutton24,'enable','on');
set(handles.pushbutton19,'enable','on');
set(handles.pushbutton44,'enable','on');
set(handles.pushbutton22,'enable','on');

set(handles.togglebutton3,'enable','on');
set(handles.pushbutton18,'enable','on');
set(handles.pushbutton1,'enable','on');
set(handles.pushbutton2,'enable','on');
set(handles.pushbutton3,'enable','on');
set(handles.pushbutton4,'enable','on');
set(handles.pushbutton6,'enable','on');
drawnow


% --- Close button
function pushbutton19_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
opc=questdlg('¿Do you really want to exit?','EXIT','Yes','No','No');
if strcmp(opc,'No')
    return;
end

%fclose(ard);

% Cierra los puertos abiertos
Puertos_Activos=instrfind; % Lee los puertos activos
if isempty(Puertos_Activos)==0 % Comprueba si hay puertos activos
    fclose(Puertos_Activos); % Cierra los puertos activos
    delete(Puertos_Activos) % Borra la variable Puertos_Activos
    clear Puertos_Activos % Destruye la variable Puertos_Activos
end

clc
close all
clear all


% --- PLOT NUEVO
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global raw_data;
global add_scan_data;
global lux_data;
global colorTemp_data;
global c_data;

set(handles.togglebutton2,'enable','off');
set(handles.pushbutton20,'enable','off');
set(handles.pushbutton16,'enable','off');
set(handles.pushbutton23,'enable','off');
set(handles.pushbutton14,'enable','off');
set(handles.pushbutton15,'enable','off');
set(handles.radiobutton5,'enable','off');
set(handles.radiobutton6,'enable','off');
set(handles.radiobutton7,'enable','off');
set(handles.radiobutton8,'enable','off');
set(handles.radiobutton9,'enable','off');
set(handles.radiobutton10,'enable','off');
drawnow

clear title xlabel ylabel;


[r,k]=size(raw_data);
u=52*add_scan_data;                 %GUARDAR ADD_SCAN_DATA EN ARCHIVO
DE RAW DATA!!!
%n=(linspace(0,52,r))';             %cambiar!!
n=(linspace(0,u,r))';

figure()
hold on
%plot(n(:,1),raw_data(:,1),'r-',n(:,1), raw_data(:,2),'g-
',n(:,1),raw_data(:,3),'b-',n(:,1),lux_data(:,1),'c-',n(:,1),color-
Temp_data(:,1),'m-',n(:,1),c_data(:,1),'k-')
%legend('Raw R','Raw G','Raw B','Lux data','Color temperature')
```

```
if get(handles.radiobutton10,'Value')==1
pcc=plot(n(:,1),c_data(:,1),'k-');
else
    pcc=plot(0,35000,'w');
end

if get(handles.radiobutton5,'Value')==1
    pr=plot(n(:,1),raw_data(:,1),'r-');
else
    pr=plot(0,35000,'w');
end

if get(handles.radiobutton6,'Value')==1
pg=plot(n(:,1),raw_data(:,2),'g-')
else
    pg=plot(0,35000,'w');
end

if get(handles.radiobutton7,'Value')==1
pb=plot(n(:,1),raw_data(:,3),'b-');
else
    pb=plot(0,35000,'w');
end

if get(handles.radiobutton8,'Value')==1
pl=plot(n(:,1),lux_data(:,1),'c-')
else
    pl=plot(0,35000,'w');
end

if get(handles.radiobutton9,'Value')==1
pct=plot(n(:,1),colorTemp_data(:,1),'m-');
else
    pct=plot(0,35000,'w');
end

legend('C coordinate','Raw R','Raw G','Raw B','Lux data [lux]','Color
temperature [K]')


grid on
grid minor
xlabel('Scanning distance [mm]')
ylabel('RGB coordinates [-],[K],[lux]')
title('RGB coordinates without calibration')
hold off

set(handles.togglebutton2,'enable','on');
set(handles.pushbutton20,'enable','on');
set(handles.pushbutton16,'enable','on');
set(handles.pushbutton23,'enable','on');
set(handles.pushbutton14,'enable','on');
set(handles.pushbutton15,'enable','on');
set(handles.radiobutton5,'enable','on');
set(handles.radiobutton6,'enable','on');
set(handles.radiobutton7,'enable','on');
set(handles.radiobutton8,'enable','on');
set(handles.radiobutton9,'enable','on');
set(handles.radiobutton10,'enable','on');
```

```
drawnow



% --- CALIBRATE AND PLOT
function pushbutton15_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global trans_data;
global raw_data;
global cal_data;
global trans;
global error;
global add_scan_data;
global lux_data;
global colorTemp_data;
global c_data calFolder calscanFolder;

set(handles.togglebutton2,'enable','off');
set(handles.pushbutton20,'enable','off');
set(handles.pushbutton16,'enable','off');
set(handles.pushbutton23,'enable','off');
set(handles.pushbutton14,'enable','off');
set(handles.pushbutton43,'enable','off');
set(handles.pushbutton15,'enable','off');
set(handles.radiobutton5,'enable','off');
set(handles.radiobutton6,'enable','off');
set(handles.radiobutton7,'enable','off');
set(handles.radiobutton8,'enable','off');
set(handles.radiobutton9,'enable','off');
set(handles.radiobutton10,'enable','off');
drawnow


opc=questdlg('Calibration will be applied only to the RGB coordinates.
Do you wish to proceed?','EXIT','Yes','No','Yes');
if strcmp(opc,'No')
    set(handles.togglebutton2,'enable','on');
    set(handles.pushbutton20,'enable','on');
    set(handles.pushbutton16,'enable','on');
    set(handles.pushbutton23,'enable','on');
    set(handles.pushbutton14,'enable','on');
    set(handles.pushbutton15,'enable','on');
    set(handles.radiobutton5,'enable','on');
    set(handles.radiobutton6,'enable','on');
    set(handles.radiobutton7,'enable','on');
    set(handles.radiobutton8,'enable','on');
    set(handles.radiobutton9,'enable','on');
    set(handles.radiobutton10,'enable','on');
    drawnow
    return;
end
if strcmp(opc,'Yes')
%Choose a calibration file and load its variables into the workspace
[FileName PathName] = uigetfile(calFolder,'MATLAB Files');
f = fullfile(PathName,FileName);

if PathName~=0
load(f);
```

```
[r,k]=size(raw_data);
trans_data=zeros(r,k);
sd=zeros(3,1);
td=zeros(3,1);
index=0;
for c=1:r
    index=index+1;
    sd(1)=raw_data(index,1);
    sd(2)=raw_data(index,2);
    sd(3)=raw_data(index,3);

    td=trans*sd-error;

    trans_data(index,1)=td(1);
    trans_data(index,2)=td(2);
    trans_data(index,3)=td(3);
end

%question: do u wish to save the calibrated data? + matfile
clear raw_data;
raw_data=trans_data;
filename=sprintf('Calibrated_data_%s.mat',datestr(now,'dd-mm-yyyy HH-
MM'));
%save(filename,'trans','error')
save([calscanFolder file-
name],'raw_data','add_scan_data','lux_data','color-
Temp_data','c_data');

%filename2=sprintf('trans_data_%s.xls',datestr(now,'dd-mm-yyyy HH-
MM'));
%xlswrite(filename2,trans_data);

clear title xlabel ylabel;

%[u z]=size(trans_data);

a=52*add_scan_data;

n=(linspace(0,a,r))';

figure()
hold on;
%plot(n(:,1),trans_data(:,1),'r-',n(:,1), trans_data(:,2),'g-
',n(:,1),trans_data(:,3),'b-',n(:,1),lux_data(:,1),'c-',n(:,1),color-
Temp_data(:,1),'m-',n(:,1),c_data(:,1),'k-')
%legend('Raw R','Raw G','Raw B','Lux data','Color temperature','C co-
ordinate')
if get(handles.radiobutton10,'Value')==1
pcc=plot(n(:,1),c_data(:,1),'k-')
else
    pcc=plot(0,0,'w');
end

if get(handles.radiobutton5,'Value')==1
    pr=plot(n(:,1),raw_data(:,1),'r-')
else
    pr=plot(0,0,'w');
end

if get(handles.radiobutton6,'Value')==1
```

```
pg=plot(n(:,1),raw_data(:,2),'g-')
else
    pg=plot(0,0,'w');
end

if get(handles.radiobutton7,'Value')==1
pb=plot(n(:,1),raw_data(:,3),'b-')
else
    pb=plot(0,0,'w');
end

if get(handles.radiobutton8,'Value')==1
pl=plot(n(:,1),lux_data(:,1),'c-')
else
    pl=plot(0,0,'w');
end

if get(handles.radiobutton9,'Value')==1
pct=plot(n(:,1),colorTemp_data(:,1),'m-')
else
    pct=plot(0,0,'w');
end

legend('C coordinate','Raw R','Raw G','Raw B','Lux data [lux]','Color
temperature [K]')


grid on
grid minor
xlabel('Scanning distance [mm]')
ylabel('RGB coordinates [-],[K],[lux]')
title('RGB coordinates (calibrated)')
hold off

set(handles.togglebutton2,'enable','on');
set(handles.pushbutton20,'enable','on');
set(handles.pushbutton16,'enable','on');
set(handles.pushbutton23,'enable','on');
set(handles.pushbutton14,'enable','on');
set(handles.pushbutton43,'enable','on');
set(handles.pushbutton15,'enable','on');
set(handles.radiobutton5,'enable','on');
set(handles.radiobutton6,'enable','on');
set(handles.radiobutton7,'enable','on');
set(handles.radiobutton8,'enable','on');
set(handles.radiobutton9,'enable','on');
set(handles.radiobutton10,'enable','on');
drawnow
else
    set(handles.togglebutton2,'enable','on');
    set(handles.pushbutton20,'enable','on');
    set(handles.pushbutton16,'enable','on');
    set(handles.pushbutton23,'enable','on');
    set(handles.pushbutton14,'enable','on');
    set(handles.pushbutton15,'enable','on');
    set(handles.radiobutton5,'enable','on');
    set(handles.radiobutton6,'enable','on');
    set(handles.radiobutton7,'enable','on');
    set(handles.radiobutton8,'enable','on');
    set(handles.radiobutton9,'enable','on');
    set(handles.radiobutton10,'enable','on');
```

```
    drawnow
end

end
```

```
% --- LOAD PLOT DATA
function pushbutton16_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global add_scan_data;
global scan_data;
global raw_data;
global lux_data;
global colorTemp_data scanFolder

set(handles.togglebutton2,'enable','off');
set(handles.pushbutton20,'enable','off');
set(handles.pushbutton16,'enable','off');
set(handles.pushbutton23,'enable','off');
set(handles.pushbutton14,'enable','off');
set(handles.pushbutton15,'enable','off');
set(handles.radiobutton5,'enable','off');
set(handles.radiobutton6,'enable','off');
set(handles.radiobutton7,'enable','off');
set(handles.radiobutton8,'enable','off');
set(handles.radiobutton9,'enable','off');
set(handles.radiobutton10,'enable','off');
drawnow

add_scan_data=1;

[FileName PathName] = uigetfile(scanFolder,'MATLAB files');
f = fullfile(PathName,FileName);

if PathName~=0
load(f);
h=msgbox('Done','GUI1');

set(handles.togglebutton2,'enable','on');
set(handles.pushbutton20,'enable','on');
set(handles.pushbutton16,'enable','on');
set(handles.pushbutton23,'enable','on');
set(handles.pushbutton14,'enable','on');
set(handles.pushbutton15,'enable','on');
set(handles.radiobutton5,'enable','on');
set(handles.radiobutton6,'enable','on');
set(handles.radiobutton7,'enable','on');
set(handles.radiobutton8,'enable','on');
set(handles.radiobutton9,'enable','on');
set(handles.radiobutton10,'enable','on');
drawnow
else

set(handles.togglebutton2,'enable','on');
set(handles.pushbutton16,'enable','on');
```

```
drawnow
end


% --- Help on plot panel
function pushbutton20_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
helpdlg('Load data: Allows user to choose a data file to load into the
workspace. Plot raw data: Plots the chosen data without calibration.
Plot calibrated data: Allows the user to choose a calibration file,
calibrates the data in the workspace, saves it and then plots it.')


% --- ESCANEO VERSIÓN FINAL
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ard;
global scan_data;
global s;
global index;
global raw_data;
global lux_data;
global colorTemp_data;
global c_data;
global add_scan_data rawFolder;

set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');
set(handles.pushbutton22,'enable','off');

set(handles.pushbutton21,'enable','off');
set(handles.togglebutton1,'enable','off');
set(handles.pushbutton10,'enable','off');
set(handles.pushbutton12,'enable','off');
set(handles.pushbutton31,'enable','off');
set(handles.pushbutton35,'enable','off');
drawnow

% delete(instrfind)
% arduino=serial('COM5','BaudRate',9600);
add_scan_data=1;
set(ard,'TimeOut',4);
fopen(ard); % initiate arduino communication
pause(2);  %wait for communication to stabilize

fprintf(ard,'%s',49); % envía '1' para activar programa de escaneo

%escribir en el static text calibrating...

calibration='';
while isempty(calibration)==1 || calibration==1
    calibration=fscanf(ard,'%d\n');
end
calStatus=fscanf(ard,'%d\n'); %variable de estado de calibrado
```

```
if calStatus==1
    s=1;
    %calibración ha salido bien (mensaje en static text)
end

if calStatus~=1
    s=2; %calibración ha salido mal(mensaje de error)
end

pause(0.1);
N=100;
scan_data  = zeros(N,3);     %array para guardar los datos//se resetea
en cada escaneo!!!
lux_scan = zeros(N,1);
colorTemp_scan=zeros(N,1);
c_scan=zeros(N,1);
index = 0;
while s == 1
    index = index + 1;
    s=fscanf(ard,'%d\n');  %para que se siga cumpliendo s=1  ,'%c'
    if s==1
        pause(.1);
        r=fscanf(ard,'%d\n');
        pause(.1);
        g=fscanf(ard,'%d\n');
        pause(.1);
        b=fscanf(ard,'%d\n');
        pause(.1);
        lux=fscanf(ard,'%d\n');
        pause(.1);
        colorTemp=fscanf(ard,'%d\n');
        pause(.1);
        c=fscanf(ard,'%d\n');
        pause(.1);

        scan_data(index,1) = r;  % Your values to be stored
        scan_data(index,2) = g;  % Your values to be stored
        scan_data(index,3) = b;  % Your values to be stored

        lux_scan(index) = lux;
        colorTemp_scan(index) = colorTemp;
        c_scan(index)=c;
    end
end

scan_data = scan_data(1:index-1, :);  % Crop unused elements
lux_scan = lux_scan(1:index-1, :);  % Crop unused elements
colorTemp_scan = colorTemp_scan(1:index-1, :);  % Crop unused elements
c_scan = c_scan(1:index-1, :);  % Crop unused elements
raw_data=scan_data;                    %Convertir en raw_data
lux_data=lux_scan;
colorTemp_data=colorTemp_scan;
c_data=c_scan;

%fin de escaneo, toca validar!
validating='';

while isempty(validating)==1 || validating==1
validating=fscanf(ard,'%d\n');
end
```

```
valStatus=fscanf(ard,'%d\n');
if valStatus~=1
    s=0;
    %mensaje de error
    pause(4);
    fclose(ard);
    errordlg('Scan failed. Validation failed. Please retry the scan.
u_u','ERROR');
    drawnow
end
if valStatus==1
    s=0;
    %validación exitosa
    pause(4);
    fclose(ard);
    %h=msgbox('Scan completed. Data saved successfully.  \(^_^)/');

    %SAVE DATA

    if s==0

        filename=sprintf('raw_data_%s.mat',datestr(now,'dd-mm-yyyy HH-
MM'));
        save([rawFolder file-
name],'raw_data','add_scan_data','lux_data','colorTemp_data','c_data')
%guardar como txt
        %filename2=sprintf('raw_data_%s.xls',datestr(now,'dd-mm-yyyy
HH-MM'));
        %dlmwrite('scan_data.txt', scan_data, 'delimiter','\t')
        %xlswrite(filename2,raw_data)                    %guardar como
xls
        h=msgbox('Scan completed. Data saved successfully.  \(^_^)/');
        %index=0;
    elseif s~=0
        h=errordlg('Scanning. Do not dare to save (yet).','Error');
    end
else
    h=errordlg('Validation failed.','Error');
end
set(handles.pushbutton24,'enable','on');
set(handles.pushbutton19,'enable','on');
set(handles.pushbutton44,'enable','on');
set(handles.pushbutton22,'enable','on');

set(handles.pushbutton21,'enable','on');
set(handles.togglebutton1,'enable','on');
set(handles.pushbutton10,'enable','on');
set(handles.pushbutton12,'enable','on');
set(handles.pushbutton31,'enable','on');
set(handles.pushbutton35,'enable','on');
drawnow
```

```
% --- Add scan data
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global add_scan_data;
global raw_data;
global s;
global index;
global lux_data;
global colorTemp_data;
global c_data;

global ard;
global scan_data rawFolder;

set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');
set(handles.pushbutton22,'enable','off');

set(handles.pushbutton21,'enable','off');
set(handles.togglebutton1,'enable','off');
set(handles.pushbutton10,'enable','off');
set(handles.pushbutton12,'enable','off');
set(handles.pushbutton31,'enable','off');
set(handles.pushbutton35,'enable','off');
drawnow

add_scan_data=add_scan_data+1;
clear scan_data;
[r,k]=size(raw_data);

data1=raw_data;

%scan----------------------------------------------------------------
set(ard,'TimeOut',4);
fopen(ard); % initiate arduino communication
pause(2);  %wait for communication to stabilize

fprintf(ard,'%s',49); % envía '1' para activar programa de escaneo

%escribir en el static text calibrating...

calibration='';
while isempty(calibration)==1 || calibration==1
    calibration=fscanf(ard,'%d\n');
end
calStatus=fscanf(ard,'%d\n'); %variable de estado de calibrado

if calStatus==1
    s=1;
    %calibración ha salido bien (mensaje en static text)
end

if calStatus~=1
    s=2; %calibración ha salido mal(mensaje de error)
end


pause(0.1);
N=100;
```

```
scan_data  = zeros(N,3);     %array para guardar los datos//se resetea
en cada escaneo!!!
lux_scan = zeros(N,1);
colorTemp_scan = zeros(N,1);
c_scan=zeros(N,1);
index = 0;
while s == 1
    index = index + 1;
    s=fscanf(ard,'%d\n');  %para que se siga cumpliendo s=1  ,'%c'
    if s==1
        pause(.1);
        r=fscanf(ard,'%d\n');
        pause(.1);
        g=fscanf(ard,'%d\n');
        pause(.1);
        b=fscanf(ard,'%d\n');
        pause(.1);
        lux=fscanf(ard,'%d\n');
        pause(.1);
        colorTemp=fscanf(ard,'%d\n');
        pause(.1);
        c=fscanf(ard,'%d\n');
        pause(.1);

        scan_data(index,1) = r;  % Your values to be stored
        scan_data(index,2) = g;  % Your values to be stored
        scan_data(index,3) = b;  % Your values to be stored

        lux_scan(index)=lux;
        colorTemp_scan(index)=colorTemp;
        c_scan(index)=c;
    end
end

scan_data = scan_data(1:index-1, :);  % Crop unused elements
lux_scan = lux_scan(1:index-1, :);  % Crop unused elements
colorTemp_scan = colorTemp_scan(1:index-1, :);  % Crop unused elements
c_scan = c_scan(1:index-1, :);  % Crop unused elements

%fin de escaneo, toca validar!
validating='';

while isempty(validating)==1 || validating==1
validating=fscanf(ard,'%d\n');
end
valStatus=fscanf(ard,'%d\n');
if valStatus~=1
    s=0;
    %mensaje de error
end
if valStatus==1
    s=0;
    %validación exitosa
end

pause(4);
fclose(ard);
%scan end

dataLux=[lux_data;lux_scan];
dataColorTemp=[colorTemp_data;colorTemp_scan];
```

```
dataC=[c_data;c_scan];
data2=[data1;scan_data];
raw_data=data2;
lux_data=dataLux;
colorTemp_data=dataColorTemp;
c_data=dataC;
clear data2 data1;
%h=msgbox('Scan completed. Data added successfully.','Gui1');


%SAVE DATA
if s==0
    %raw_data=scan_data;
    filename=sprintf('raw_data_%s.mat',datestr(now,'dd-mm-yyyy HH-
MM'));
    %save(filename,'scan_data')
    save([rawFolder filename],'raw_data','add_scan_data','lux_da-
ta','colorTemp_data','c_data')
    %load('data.mat')                                        %guardar
como txt
    filename2=sprintf('raw_data_%s.xls',datestr(now,'dd-mm-yyyy HH-
MM'));
    %dlmwrite('scan_data.txt', scan_data, 'delimiter','\t')
    %xlswrite(filename2,raw_data)                    %guardar como xls
    h=msgbox('Scan completed. Data added successfully.','Gui1');
    %index=0;
elseif s~=0
    h=errordlg('Scanning. Do not dare to save (yet).','Error');
end


set(handles.pushbutton24,'enable','on');
set(handles.pushbutton19,'enable','on');
set(handles.pushbutton44,'enable','on');
set(handles.pushbutton22,'enable','on');

set(handles.pushbutton21,'enable','on');
set(handles.togglebutton1,'enable','on');
set(handles.pushbutton10,'enable','on');
set(handles.pushbutton12,'enable','on');
set(handles.pushbutton31,'enable','on');
set(handles.pushbutton35,'enable','on');
drawnow



% --- Help on scan panel
function pushbutton21_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton21 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
helpdlg('Scan: Scans 52mm after a calibration and then validates the
data.         Save and load: Saves scan data and loads it into the
workspace.          Add scan data: Scans 52mm after a calibration, va-
lidates the data and adds the new data to the saved scan data.')


% --- Help on calibration panel
function pushbutton18_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton18 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
helpdlg('After putting a red sampling on the scanner, press the cali-
bration button for red. Repeat operation for the remaining colors.
Then, you can save the calibration data by pressing the SAVE button.')


% --- TEXT RED
function text3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- TEXT GREEN
function text4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- TEXT BLUE
function text5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- TEXT BLACK
function text6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Reset motor position
function pushbutton22_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton22 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');
set(handles.pushbutton22,'enable','off');

set(handles.togglebutton3,'enable','off');
set(handles.pushbutton18,'enable','off');
set(handles.pushbutton1,'enable','off');
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton3,'enable','off');
set(handles.pushbutton4,'enable','off');
set(handles.pushbutton6,'enable','off');
drawnow

global ard;
```

```
fopen(ard); % initiate arduino communication
%set(ard,'TimeOut',120);
pause(2);  %wait for communication to stabilize

fprintf(ard,'%s',51); % send answer variable content to arduino

waiting='';
while isempty(waiting)==1 || waiting==0;
    waiting=fscanf(ard,'%d\n');
end

if waiting==1
   h=msgbox('Motor at start position','GUI1');
else
    h=errordlg('Communication with motor failed','ERROR');
end
fclose(ard);

set(handles.pushbutton24,'enable','on');
set(handles.pushbutton19,'enable','on');
set(handles.pushbutton44,'enable','on');
set(handles.pushbutton22,'enable','on');

set(handles.togglebutton3,'enable','on');
set(handles.pushbutton18,'enable','on');
set(handles.pushbutton1,'enable','on');
set(handles.pushbutton2,'enable','on');
set(handles.pushbutton3,'enable','on');
set(handles.pushbutton4,'enable','on');
set(handles.pushbutton6,'enable','on');
drawnow




% --- Add plot data
function pushbutton23_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global add_scan_data;
global raw_data;
global lux_data;
global colorTemp_data;
global c_data dataFolder;

set(handles.togglebutton2,'enable','off');
set(handles.pushbutton20,'enable','off');
set(handles.pushbutton16,'enable','off');
set(handles.pushbutton23,'enable','off');
set(handles.pushbutton14,'enable','off');
set(handles.pushbutton15,'enable','off');
set(handles.radiobutton5,'enable','off');
set(handles.radiobutton6,'enable','off');
set(handles.radiobutton7,'enable','off');
set(handles.radiobutton8,'enable','off');
set(handles.radiobutton9,'enable','off');
set(handles.radiobutton10,'enable','off');
drawnow
```

```
scan_1=raw_data;
lux_1=lux_data;
temp_1=colorTemp_data;
c_1=c_data;
add_temp=add_scan_data;

%Choose a file
[FileName PathName] = uigetfile(dataFolder,'MATLAB Files');
f = fullfile(PathName,FileName);

if PathName~=0
load(f);

add_temp=add_scan_data + add_temp;

dataLux=[lux_1;lux_data];
dataColorTemp=[temp_1;colorTemp_data];
dataC=[c_1;c_data];
data2=[scan_1;raw_data];

raw_data=data2;
lux_data=dataLux;
colorTemp_data=dataColorTemp;
c_data=dataC;
add_scan_data=add_temp;

h=msgbox('Data added successfully','GUI1');

set(handles.togglebutton2,'enable','on');
set(handles.pushbutton20,'enable','on');
set(handles.pushbutton16,'enable','on');
set(handles.pushbutton23,'enable','on');
set(handles.pushbutton14,'enable','on');
set(handles.pushbutton15,'enable','on');
set(handles.radiobutton5,'enable','on');
set(handles.radiobutton6,'enable','on');
set(handles.radiobutton7,'enable','on');
set(handles.radiobutton8,'enable','on');
set(handles.radiobutton9,'enable','on');
set(handles.radiobutton10,'enable','on');
drawnow
else
    set(handles.togglebutton2,'enable','on');
    set(handles.pushbutton20,'enable','on');
    set(handles.pushbutton16,'enable','on');
    set(handles.pushbutton23,'enable','on');
    set(handles.pushbutton14,'enable','on');
    set(handles.pushbutton15,'enable','on');
    set(handles.radiobutton5,'enable','on');
    set(handles.radiobutton6,'enable','on');
    set(handles.radiobutton7,'enable','on');
    set(handles.radiobutton8,'enable','on');
    set(handles.radiobutton9,'enable','on');
    set(handles.radiobutton10,'enable','on');
    drawnow
end


% ---Texto conexión puerto
function text7_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to text7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Texto conexión arduino
function text8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Texto conexión motor
function text9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- Retry connection
function pushbutton24_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton24 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


global ard;
Puerto=getAvailableComPort(); %Busca los puertos disponibles
ard=serial(Puerto,'BaudRate',9600);
set(ard,'BaudRate',9600);
set(ard,'TimeOut',2);

try
    fopen(ard);           %handshake con puerto
    pause(2);
    set(handles.text7,'BackgroundColor',[0.5,1,0.5]); %conexión con
puerto correcta
    set(handles.text7, 'Value',1);
    set(handles.text7,'String','ONLINE');


    fprintf(ard,'%s',53); % activar handshake con arduino (5)
    pause(1);

    res=fscanf(ard,'%d\n');%leer respuesta de arduino

    if res==1
        set(handles.text8,'BackgroundColor',[0.5,1,0.5]); %conexión
con arduino correcta
        set(handles.text8, 'Value',1);
        set(handles.text8,'String','ONLINE');

        fprintf(ard,'%s',52); % activar handshake con motor
        pause(1);

        res2=fscanf(ard,'%d\n');%leer respuesta de arduino
```

```
        if res2==1
            set(handles.text9,'BackgroundColor',[0.5,1,0.5]); %cone-
xión con motor correcta
            set(handles.text9, 'Value',1);
            set(handles.text9,'String','ONLINE');
        else
            set(handles.text9,'BackgroundColor',[1,0,0]); %conexión
con motor incorrecta
            set(handles.text9, 'Value',0);
            h=errordlg('Communication with motor failed. Please check
whether the power supply is connected and try to reconnect.','ERROR');
        end
    else
        set(handles.text8,'BackgroundColor',[1,0,0]); %conexión con
arduino incorrecta
        set(handles.text8, 'Value',0);
        h=errordlg('Communication with arduino failed. Please check if
the correct programm is running on the Arduino board.','ERROR');
    end

catch
    set(handles.text7,'BackgroundColor',[1,0,0]); %conexión con puerto
incorrecta
    set(handles.text7, 'Value',0);
    set(handles.text8,'BackgroundColor',[1,0,0]); %conexión con puerto
incorrecta
    set(handles.text8, 'Value',0);
    set(handles.text9,'BackgroundColor',[1,0,0]); %conexión con puerto
incorrecta
    set(handles.text9, 'Value',0);
    h=errordlg('Serial port connection failed. Check if the USB cable
is connected properly or if there are any available serial
ports.','ERROR');
end
try fclose(ard);
catch
end

set(handles.togglebutton4,'enable','on');
%set(handles.togglebutton4,'Value',0);
set(handles.pushbutton41,'enable','on');
set(handles.togglebutton2,'enable','on');
%set(handles.togglebutton2,'Value',0);
set(handles.pushbutton20,'enable','on');

if get(handles.text7,'Value')==1 && get(handles.text8,'Value')==1
    set(handles.togglebutton3,'enable','on');
    %set(handles.togglebutton3,'Value',0);
    set(handles.pushbutton18,'enable','on');
    set(handles.togglebutton5,'enable','on');
    %set(handles.togglebutton5,'Value',0);
    set(handles.pushbutton42,'enable','on');
    set(handles.pushbutton22,'enable','on');
end

if get(handles.text7,'Value')==1 && get(handles.text8,'Value')==1 &&
get(handles.text9,'Value')==1
    set(handles.togglebutton1,'enable','on');
    %set(handles.togglebutton1,'Value',0);
    set(handles.pushbutton21,'enable','on');
```

```
    h=msgbox('Connections working fine','GUI1');
end


% --- Find minima
function pushbutton25_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton25 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global raw_data add_scan_data minR minG minB rm gm bm xm idxvalR
idxvalG idxvalB mvalR mvalG mvalB

%set(handles.text13,'String','Looking for minima...');
set(handles.togglebutton4,'enable','off');
set(handles.pushbutton30,'enable','off');
set(handles.pushbutton25,'enable','off');
set(handles.pushbutton26,'enable','off');
set(handles.pushbutton28,'enable','off');
set(handles.pushbutton34,'enable','off');
set(handles.radiobutton2,'enable','off');
set(handles.radiobutton3,'enable','off');
set(handles.radiobutton4,'enable','off');
set(handles.radiobutton13,'enable','off');
drawnow

if get(handles.radiobutton2,'Value')==1
    prominence=1;
end
if get(handles.radiobutton3,'Value')==1
    prominence=100;
end
if get(handles.radiobutton4,'Value')==1
    prominence=2000;
end

rm=raw_data(:,1);
gm=raw_data(:,2);
bm=raw_data(:,3);

[u,z]=size(raw_data);
l=52*add_scan_data;
xm=linspace(0,l,u)';

[valR,idxvalR] = findpeaks(-rm,'MinPeakProminence',prominence,'Min-
PeakWidth',1);   %red minima
[valG,idxvalG] = findpeaks(-gm,'MinPeakProminence',prominence,'Min-
PeakWidth',1);
[valB,idxvalB] = findpeaks(-bm,'MinPeakProminence',prominence,'Min-
PeakWidth',1);

mvalB=-valB;
mvalG=-valG;
mvalR=-valR;

xR=xm(idxvalR);
xG=xm(idxvalG);
xB=xm(idxvalB);

minR=[xR,mvalR];
minG=[xG,mvalG];
```

```
minB=[xB,mvalB];
H=msgbox('Finished minima search.','GUI1');

set(handles.togglebutton4,'enable','on');
set(handles.pushbutton30,'enable','on');
set(handles.pushbutton25,'enable','on');
set(handles.pushbutton26,'enable','on');
set(handles.pushbutton28,'enable','on');
set(handles.pushbutton34,'enable','on');
set(handles.radiobutton2,'enable','on');
set(handles.radiobutton3,'enable','on');
set(handles.radiobutton4,'enable','on');
set(handles.radiobutton13,'enable','on');
drawnow




% --- Plot minima
function pushbutton26_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton26 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global rm gm bm xm idxvalR idxvalG idxvalB

set(handles.togglebutton4,'enable','off');
set(handles.pushbutton30,'enable','off');
set(handles.pushbutton25,'enable','off');
set(handles.pushbutton26,'enable','off');
set(handles.pushbutton28,'enable','off');
set(handles.pushbutton34,'enable','off');
set(handles.radiobutton2,'enable','off');
set(handles.radiobutton3,'enable','off');
set(handles.radiobutton4,'enable','off');
set(handles.radiobutton13,'enable','off');
drawnow

figure
hold on
plot(xm,rm,'r',xm(idxvalR),rm(idxvalR),'*r')
plot(xm,gm,'g',xm(idxvalG),gm(idxvalG),'*g')
plot(xm,bm,'b',xm(idxvalB),bm(idxvalB),'*b')
legend('R coordinates','R minima','G coordinates','G minima','B coor-
dinates','B minima')
xlabel('Distance (mm)')
ylabel('RGB coordinates')
hold off

set(handles.togglebutton4,'enable','on');
set(handles.pushbutton30,'enable','on');
set(handles.pushbutton25,'enable','on');
set(handles.pushbutton26,'enable','on');
set(handles.pushbutton28,'enable','on');
set(handles.pushbutton34,'enable','on');
set(handles.radiobutton2,'enable','on');
set(handles.radiobutton3,'enable','on');
set(handles.radiobutton4,'enable','on');
set(handles.radiobutton13,'enable','on');
drawnow
```

```
% --- Save minima as Excel
function pushbutton28_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton28 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global minFolder rm gm bm xm idxvalR idxvalG idxvalB raw_data
add_scan_data minR minG minB mvalR mvalG mvalB

set(handles.togglebutton4,'enable','off');
set(handles.pushbutton30,'enable','off');
set(handles.pushbutton25,'enable','off');
set(handles.pushbutton26,'enable','off');
set(handles.pushbutton28,'enable','off');
set(handles.pushbutton34,'enable','off');
set(handles.radiobutton2,'enable','off');
set(handles.radiobutton3,'enable','off');
set(handles.radiobutton4,'enable','off');
set(handles.radiobutton13,'enable','off');
drawnow

minRGB=[mvalR,mvalG,mvalB];
filename=sprintf('Minima_%s.xls',datestr(now,'dd-mm-yyyy HH-MM'));
fullname=fullfile(minFolder,filename);
xlswrite(fullname,minRGB);
h=msgbox('Minima saved.','GUI1');

set(handles.togglebutton4,'enable','on');
set(handles.pushbutton30,'enable','on');
set(handles.pushbutton25,'enable','on');
set(handles.pushbutton26,'enable','on');
set(handles.pushbutton28,'enable','on');
set(handles.pushbutton34,'enable','on');
set(handles.radiobutton2,'enable','on');
set(handles.radiobutton3,'enable','on');
set(handles.radiobutton4,'enable','on');
set(handles.radiobutton13,'enable','on');
drawnow


% --- Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton4


% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton3


% --- Executes on button press in radiobutton2.
```

```
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2


% --- Opción R
function radiobutton5_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton5


% --- Executes on button press in radiobutton6.
function radiobutton6_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton6


% --- Executes on button press in radiobutton7.
function radiobutton7_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton7


% --- Executes on button press in radiobutton8.
function radiobutton8_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton8


% --- Executes on button press in radiobutton9.
function radiobutton9_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton9


% --- Executes on button press in radiobutton10.
function radiobutton10_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of radiobutton10


% --- SINGLE READ
function pushbutton29_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton29 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ard;
global rsingle gsingle bsingle luxsingle ctsingle csingle;

set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');

set(handles.togglebutton5,'enable','off');
set(handles.pushbutton29,'enable','off');
set(handles.pushbutton33,'enable','off');
set(handles.radiobutton11,'enable','off');
drawnow

fopen(ard); % initiate arduino communication
pause(2);  %wait for communication to stabilize

fprintf(ard,'%s',54); % activar single read
pause(1);

rsingle=fscanf(ard,'%d\n');
pause(.1);

gsingle=fscanf(ard,'%d\n');
pause(.1);

bsingle=fscanf(ard,'%d\n');
pause(.1);

luxsingle=fscanf(ard,'%d\n');
pause(.1);

ctsingle=fscanf(ard,'%d\n');
pause(.1);

csingle=fscanf(ard,'%d\n');
pause(.1);
fclose(ard);



set(handles.text23,'String',rsingle);
set(handles.text21,'String',gsingle);
set(handles.text19,'String',bsingle);
set(handles.text20,'String',luxsingle);
set(handles.text22,'String',ctsingle);
set(handles.text25,'String',csingle);

set(handles.pushbutton24,'enable','on');
set(handles.pushbutton19,'enable','on');
set(handles.pushbutton44,'enable','on');
```

```
set(handles.togglebutton5,'enable','on');
set(handles.pushbutton29,'enable','on');
set(handles.pushbutton33,'enable','on');
set(handles.radiobutton11,'enable','on');

drawnow


% --- MINIMA: load data
function pushbutton30_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton30 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global add_scan_data;
global scan_data;
global raw_data;
global lux_data;
global colorTemp_data scanFolder;

set(handles.togglebutton4,'enable','off');
set(handles.pushbutton30,'enable','off');
set(handles.pushbutton25,'enable','off');
set(handles.pushbutton26,'enable','off');
set(handles.pushbutton28,'enable','off');
set(handles.pushbutton34,'enable','off');
set(handles.radiobutton2,'enable','off');
set(handles.radiobutton3,'enable','off');
set(handles.radiobutton4,'enable','off');
set(handles.radiobutton13,'enable','off');
drawnow

add_scan_data=1;

[FileName PathName] = uigetfile(scanFolder,'MATLAB files');
f = fullfile(PathName,FileName);


if PathName~=0
    load(f);
    h=msgbox('Done','GUI1');
    set(handles.togglebutton4,'enable','on');
    set(handles.pushbutton30,'enable','on');
    set(handles.pushbutton25,'enable','on');
    set(handles.radiobutton2,'enable','on');
    set(handles.radiobutton3,'enable','on');
    set(handles.radiobutton4,'enable','on');
else
    set(handles.togglebutton4,'enable','on');
    set(handles.pushbutton30,'enable','on');
end
```

```
% --- Save SCAN_DATA as excel
function pushbutton31_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton31 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global scan_data;
global raw_data;
global lux_data;
global colorTemp_data;
global c_data rawFolder;

set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');

set(handles.pushbutton21,'enable','off');
set(handles.togglebutton1,'enable','off');
set(handles.pushbutton10,'enable','off');
set(handles.pushbutton12,'enable','off');
set(handles.pushbutton31,'enable','off');
set(handles.pushbutton35,'enable','off');
drawnow

title={'R','G','B','Lux','Color Temperature','C'};
dsave=[raw_data,lux_data,colorTemp_data,c_data];
dsave=num2cell(dsave);
dsave2=[title;dsave];

filename=sprintf('Raw_data_%s.xls',datestr(now,'dd-mm-yyyy HH-MM'));
fullname=fullfile(rawFolder,filename);
xlswrite(fullname,dsave2);
h=msgbox('Raw data saved as Excel file.','GUI1');

set(handles.pushbutton24,'enable','on');
set(handles.pushbutton19,'enable','on');
set(handles.pushbutton44,'enable','on');

set(handles.pushbutton21,'enable','on');
set(handles.togglebutton1,'enable','on');
set(handles.pushbutton10,'enable','on');
set(handles.pushbutton12,'enable','on');
set(handles.pushbutton31,'enable','on');
set(handles.pushbutton35,'enable','on');
drawnow



% --- Copy to clipboard
function pushbutton33_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton33 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global rsingle gsingle bsingle luxsingle ctsingle csingle;

set(handles.togglebutton5,'enable','off');
set(handles.pushbutton29,'enable','off');
set(handles.pushbutton33,'enable','off');
set(handles.radiobutton11,'enable','off');
drawnow
```

```
title={'R','G','B','Lux','Color Temperature','C'};
dsave=[rsingle,gsingle,bsingle,luxsingle,ctsingle,csingle];
dsave2=num2cell(dsave);
dsave3=[title;dsave2];

tag=get(handles.radiobutton11,'Value');

if tag==1
    format short g
    mat2clip(dsave3);
end

if tag==0
    format short g
    mat2clip(dsave2);
end

h=msgbox('Data added to clipboard','GUI1');

set(handles.togglebutton5,'enable','on');
set(handles.pushbutton29,'enable','on');
set(handles.pushbutton33,'enable','on');
set(handles.radiobutton11,'enable','on');
drawnow


% --- Executes on button press in radiobutton11.
function radiobutton11_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton11


% --- Copy to clipboard MINIMA
function pushbutton34_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton34 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global mvalR mvalG mvalB;

set(handles.togglebutton4,'enable','off');
set(handles.pushbutton30,'enable','off');
set(handles.pushbutton25,'enable','off');
set(handles.pushbutton26,'enable','off');
set(handles.pushbutton28,'enable','off');
set(handles.pushbutton34,'enable','off');
set(handles.radiobutton2,'enable','off');
set(handles.radiobutton3,'enable','off');
set(handles.radiobutton4,'enable','off');
set(handles.radiobutton13,'enable','off');
drawnow

data=[mvalR,mvalG,mvalB];
datacell=num2cell(data);
title={'R minima','G minima','B minima'};
```

```
tag=get(handles.radiobutton13,'Value');
if tag==1
    data2=[title;datacell];
    format short g
    mat2clip(data2);
end
if tag==0
    format short g
    mat2clip(data);
end

h=msgbox('Data added to clipboard','GUI1');

set(handles.togglebutton4,'enable','on');
set(handles.pushbutton30,'enable','on');
set(handles.pushbutton25,'enable','on');
set(handles.pushbutton26,'enable','on');
set(handles.pushbutton28,'enable','on');
set(handles.pushbutton34,'enable','on');
set(handles.radiobutton2,'enable','on');
set(handles.radiobutton3,'enable','on');
set(handles.radiobutton4,'enable','on');
set(handles.radiobutton13,'enable','on');
drawnow


% --- Executes on button press in radiobutton13.
function radiobutton13_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton13


% --- SCAN:Load data
function pushbutton35_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton35 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global add_scan_data;
global scan_data;
global raw_data;
global lux_data;
global colorTemp_data scanFolder

set(handles.pushbutton24,'enable','off');
set(handles.pushbutton19,'enable','off');
set(handles.pushbutton44,'enable','off');

set(handles.pushbutton21,'enable','off');
set(handles.togglebutton1,'enable','off');
set(handles.pushbutton10,'enable','off');
set(handles.pushbutton12,'enable','off');
set(handles.pushbutton31,'enable','off');
set(handles.pushbutton35,'enable','off');
drawnow

add_scan_data=1;
```

```
[FileName PathName] = uigetfile(scanFolder,'MATLAB files');
f = fullfile(PathName,FileName);
if PathName~=0
    load(f);
    h=msgbox('Done','GUI1');

    set(handles.pushbutton24,'enable','on');
    set(handles.pushbutton19,'enable','on');
    set(handles.pushbutton44,'enable','on');

    set(handles.pushbutton21,'enable','on');
    set(handles.togglebutton1,'enable','on');
    set(handles.pushbutton10,'enable','on');
    set(handles.pushbutton12,'enable','on');
    set(handles.pushbutton31,'enable','on');
    set(handles.pushbutton35,'enable','on');
    drawnow
else

    set(handles.pushbutton24,'enable','on');
    set(handles.pushbutton19,'enable','on');
    set(handles.pushbutton44,'enable','on');

    set(handles.pushbutton21,'enable','on');
    set(handles.togglebutton1,'enable','on');
    set(handles.pushbutton10,'enable','on');
    set(handles.pushbutton35,'enable','on');
    drawnow
end




% --- Executes on button press in pushbutton42.
function pushbutton42_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton42 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
helpdlg('Press the Read Button in order to read data. The adquired
data can be copied into the clipboard.');

% --- Executes on button press in pushbutton41.
function pushbutton41_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton41 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
helpdlg('Load data into the workspace with the Load Data Button. Se-
lect a resolution and look for minima in the data. The minima can be
plotted, saved or copied into the clipboard.');


% --- Plot and calibrate SAVE AS EXCEL
function pushbutton43_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton43 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global trans_data;
global lux_data;
global colorTemp_data;
global c_data calscanFolder;
```

```
set(handles.togglebutton2,'enable','off');
set(handles.pushbutton20,'enable','off');
set(handles.pushbutton16,'enable','off');
set(handles.pushbutton23,'enable','off');
set(handles.pushbutton14,'enable','off');
set(handles.pushbutton43,'enable','off');
set(handles.pushbutton15,'enable','off');
set(handles.radiobutton5,'enable','off');
set(handles.radiobutton6,'enable','off');
set(handles.radiobutton7,'enable','off');
set(handles.radiobutton8,'enable','off');
set(handles.radiobutton9,'enable','off');
set(handles.radiobutton10,'enable','off');
drawnow

title={'R','G','B','Lux','Color Temperature','C'};
dsave=[trans_data,lux_data,colorTemp_data,c_data];
dsave=num2cell(dsave);
dsave2=[title;dsave];

filename=sprintf('Calibrated_data_%s.xls',datestr(now,'dd-mm-yyyy HH-
MM'));
fullname=fullfile(calscanFolder,filename);
xlswrite(fullname,dsave2);
h=msgbox('Calibrated data saved as Excel file.','GUI1');

set(handles.togglebutton2,'enable','on');
set(handles.pushbutton20,'enable','on');
set(handles.pushbutton16,'enable','on');
set(handles.pushbutton23,'enable','on');
set(handles.pushbutton14,'enable','on');
set(handles.pushbutton43,'enable','on');
set(handles.pushbutton15,'enable','on');
set(handles.radiobutton5,'enable','on');
set(handles.radiobutton6,'enable','on');
set(handles.radiobutton7,'enable','on');
set(handles.radiobutton8,'enable','on');
set(handles.radiobutton9,'enable','on');
set(handles.radiobutton10,'enable','on');
drawnow


% --- BUTTON Scan panel
function togglebutton1_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton1
global raw_data;
if get(handles.togglebutton4,'Value')==0 && get(handles.togglebut-
ton3,'Value')==0 && get(handles.togglebutton2,'Value')==0 && get(hand-
les.togglebutton5,'Value')==0
    sc=get(handles.togglebutton1,'Value');
    if sc==1
        set(handles.togglebutton1,'String','OFF');
        set(handles.uipanel4,'BackgroundColor',[1,1,1]);
        hay=isempty(raw_data);
        if hay==0
            set(handles.pushbutton10,'enable','on');
```

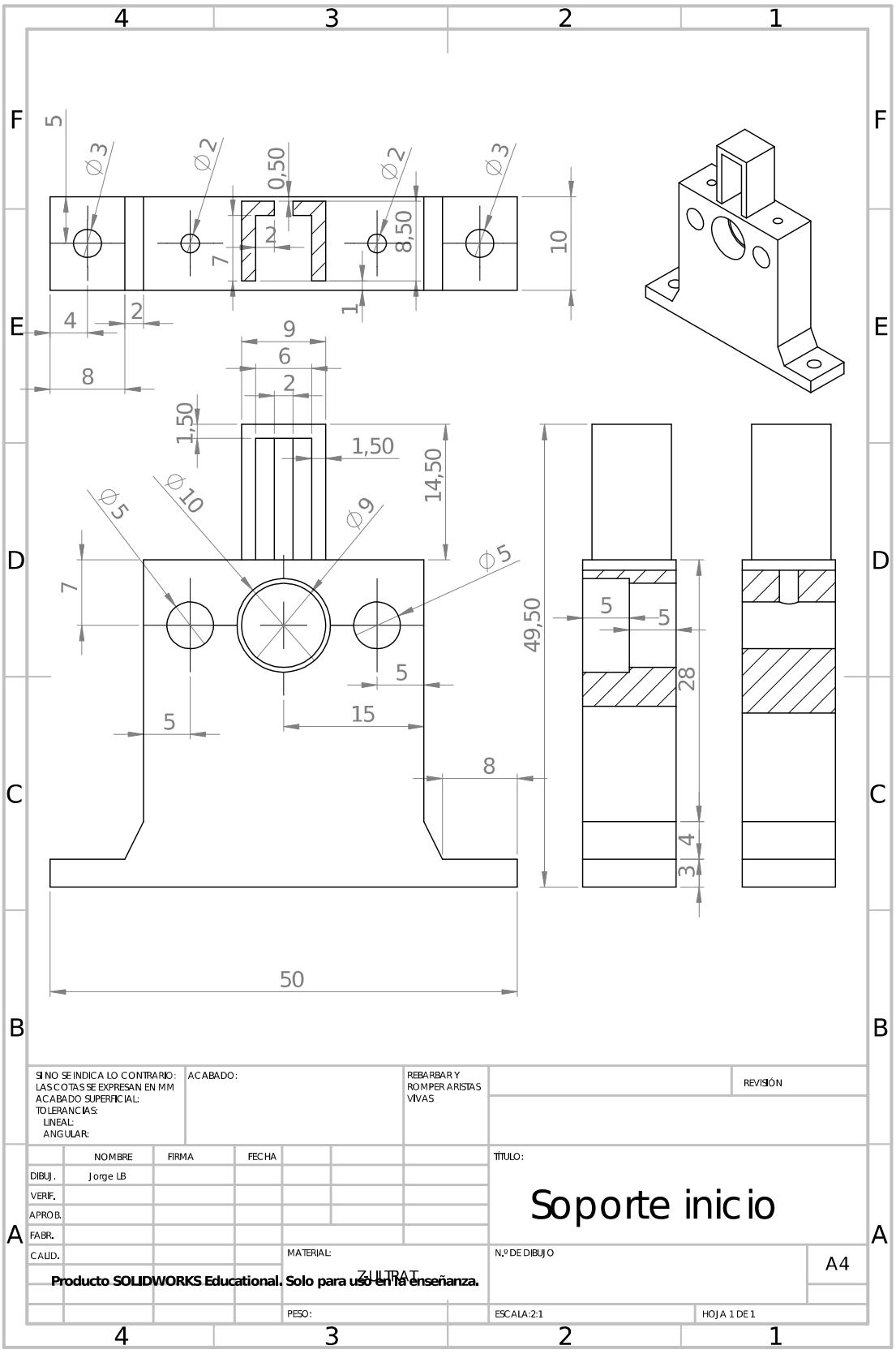
```
            set(handles.pushbutton35,'enable','on');
            set(handles.pushbutton12,'enable','on');
            set(handles.pushbutton31,'enable','on');
        elseif hay==1
            set(handles.pushbutton10,'enable','on');
            set(handles.pushbutton35,'enable','on');
        end
    elseif sc==0
        set(handles.uipanel4,'BackgroundColor',[0.94,0.94,0.94]);

        set(handles.pushbutton10,'enable','off');
        set(handles.pushbutton12,'enable','off');
        set(handles.pushbutton31,'enable','off');
        set(handles.pushbutton35,'enable','off');

        set(handles.togglebutton1,'String','ON');
    end
else
    set(handles.togglebutton1,'Value',0);
    warndlg('Please close the active panels before activating a new
one','GUI1');
end


% --- BUTTON: Single read
function togglebutton5_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton5
if get(handles.togglebutton4,'Value')==0 && get(handles.togglebut-
ton3,'Value')==0 && get(handles.togglebutton2,'Value')==0 && get(hand-
les.togglebutton1,'Value')==0
    sc=get(handles.togglebutton5,'Value');
    if sc==1
        set(handles.togglebutton5,'String','OFF');
        set(handles.uipanel13,'BackgroundColor',[1,1,1]);
        set(handles.text14,'BackgroundColor',[1,1,1]);
        set(handles.text15,'BackgroundColor',[1,1,1]);
        set(handles.text16,'BackgroundColor',[1,1,1]);
        set(handles.text17,'BackgroundColor',[1,1,1]);
        set(handles.text18,'BackgroundColor',[1,1,1]);
        set(handles.text24,'BackgroundColor',[1,1,1]);


        set(handles.togglebutton5,'enable','on');
        set(handles.pushbutton29,'enable','on');

    elseif sc==0
        set(handles.uipanel13,'BackgroundColor',[0.94,0.94,0.94]);
        set(handles.text14,'BackgroundColor',[0.94,0.94,0.94]);
        set(handles.text15,'BackgroundColor',[0.94,0.94,0.94]);
        set(handles.text16,'BackgroundColor',[0.94,0.94,0.94]);
        set(handles.text17,'BackgroundColor',[0.94,0.94,0.94]);
        set(handles.text18,'BackgroundColor',[0.94,0.94,0.94]);
        set(handles.text24,'BackgroundColor',[0.94,0.94,0.94]);

        set(handles.pushbutton29,'enable','off');
        set(handles.pushbutton33,'enable','off');
```

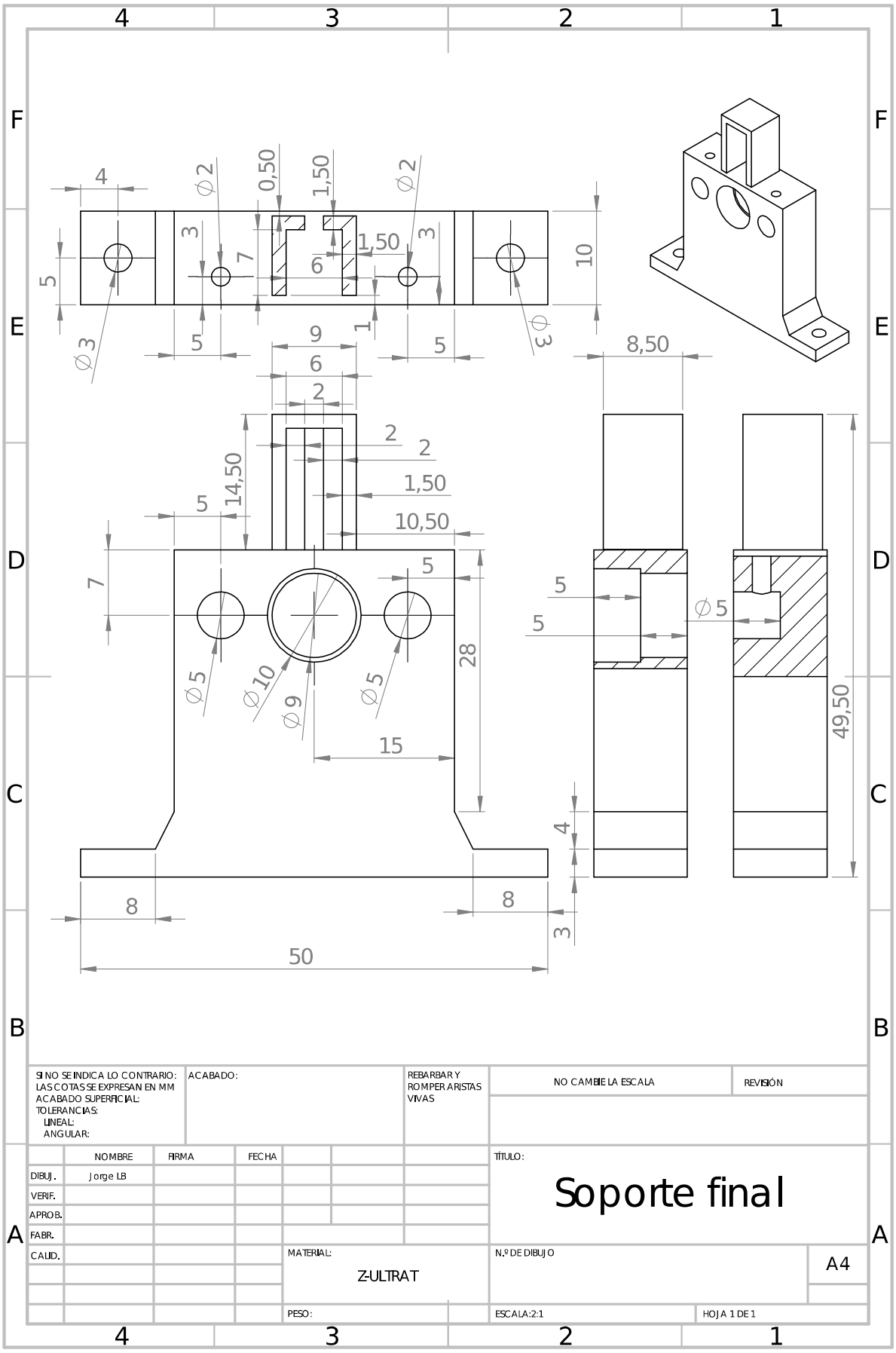
```
        set(handles.radiobutton11,'enable','off');


        set(handles.togglebutton5,'String','ON');
    end
else
    set(handles.togglebutton5,'Value',0);
    warndlg('Please close the active panels before activating a new
one','GUI1');
end


% --- Executes on button press in togglebutton4.
function togglebutton4_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton4
global raw_data;
if get(handles.togglebutton1,'Value')==0 && get(handles.togglebut-
ton3,'Value')==0 && get(handles.togglebutton2,'Value')==0 && get(hand-
les.togglebutton5,'Value')==0
    sc=get(handles.togglebutton4,'Value');
    if sc==1
        set(handles.togglebutton4,'String','OFF');
        set(handles.uipanel12,'BackgroundColor',[1,1,1]);

        hay=isempty(raw_data);
        if hay==0
            set(handles.radiobutton2,'enable','on');
            set(handles.radiobutton3,'enable','on');
            set(handles.radiobutton4,'enable','on');
            set(handles.pushbutton25,'enable','on');
            set(handles.pushbutton30,'enable','on');
        elseif hay==1
            set(handles.pushbutton30,'enable','on');
        end


    elseif sc==0
        set(handles.uipanel12,'BackgroundColor',[0.94,0.94,0.94]);
        set(handles.pushbutton30,'enable','off');
        set(handles.pushbutton25,'enable','off');
        set(handles.pushbutton26,'enable','off');
        set(handles.pushbutton28,'enable','off');
        set(handles.pushbutton34,'enable','off');
        set(handles.radiobutton2,'enable','off');
        set(handles.radiobutton3,'enable','off');
        set(handles.radiobutton4,'enable','off');
        set(handles.radiobutton13,'enable','off');

        set(handles.togglebutton4,'String','ON');
    end
else
    set(handles.togglebutton4,'Value',0);
    warndlg('Please close the active panels before activating a new
one','GUI1');
end
```

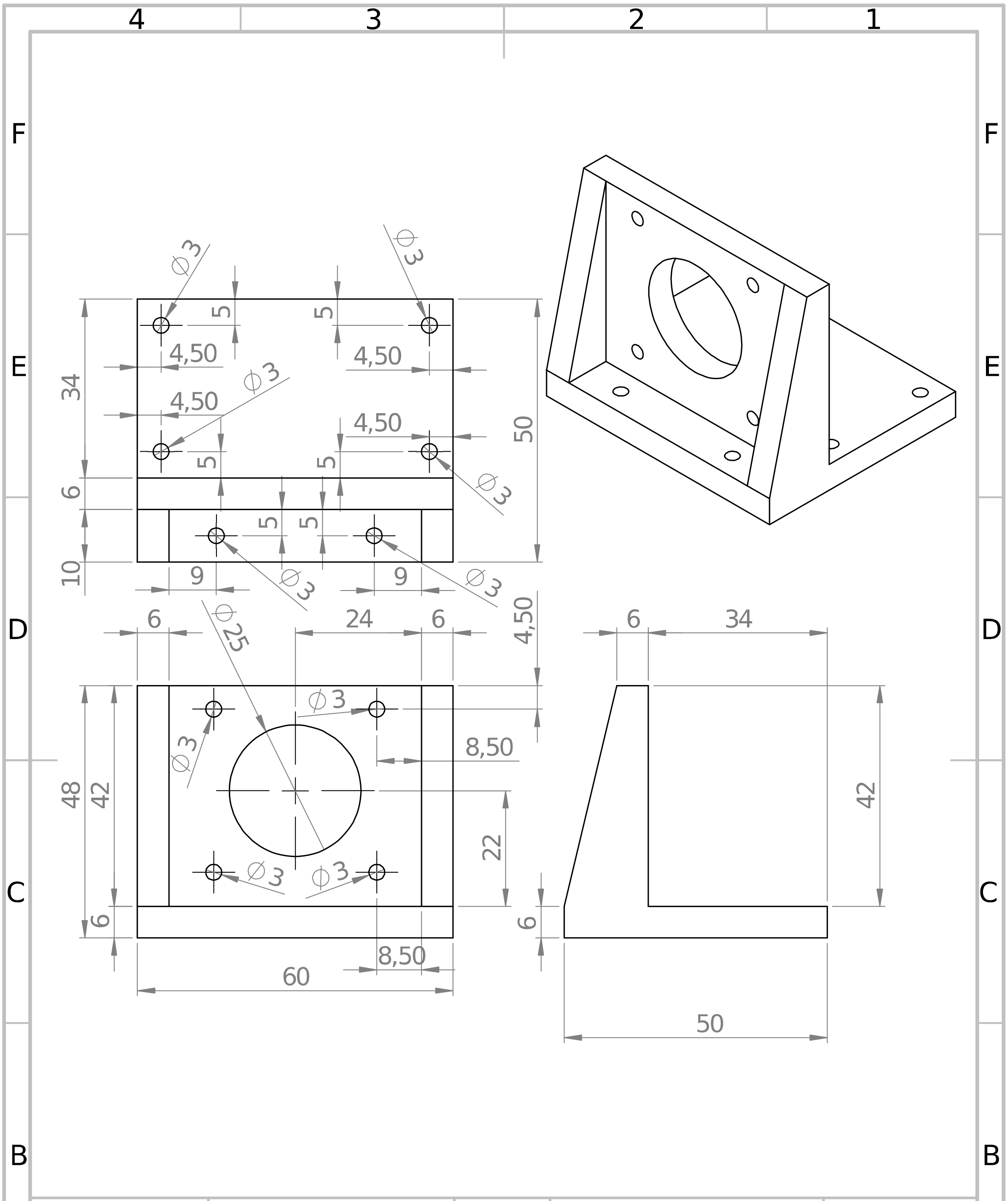
```
% --- BUTTON: Plot
function togglebutton2_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton2
global raw_data;
if get(handles.togglebutton4,'Value')==0 && get(handles.togglebut-
ton3,'Value')==0 && get(handles.togglebutton1,'Value')==0 && get(hand-
les.togglebutton5,'Value')==0
    sc=get(handles.togglebutton2,'Value');
    if sc==1
        set(handles.togglebutton2,'String','OFF');
        set(handles.uipanel5,'BackgroundColor',[1,1,1]);
        hay=isempty(raw_data);
        if hay==0
           set(handles.pushbutton20,'enable','on');
           set(handles.pushbutton16,'enable','on');
           set(handles.pushbutton23,'enable','on');
           set(handles.pushbutton14,'enable','on');
           set(handles.pushbutton15,'enable','on');
           set(handles.radiobutton5,'enable','on');
           set(handles.radiobutton6,'enable','on');
           set(handles.radiobutton7,'enable','on');
           set(handles.radiobutton8,'enable','on');
           set(handles.radiobutton9,'enable','on');
           set(handles.radiobutton10,'enable','on');
        elseif hay==1
            set(handles.pushbutton16,'enable','on');
        end
    elseif sc==0
        set(handles.uipanel5,'BackgroundColor',[0.94,0.94,0.94]);
        set(handles.pushbutton16,'enable','off');
        set(handles.pushbutton43,'enable','off');
        set(handles.pushbutton23,'enable','off');
        set(handles.pushbutton14,'enable','off');
        set(handles.pushbutton15,'enable','off');
        set(handles.radiobutton5,'enable','off');
        set(handles.radiobutton6,'enable','off');
        set(handles.radiobutton7,'enable','off');
        set(handles.radiobutton8,'enable','off');
        set(handles.radiobutton9,'enable','off');
        set(handles.radiobutton10,'enable','off');


        set(handles.togglebutton2,'String','ON');
    end
else
    set(handles.togglebutton2,'Value',0);
    warndlg('Please close the active panels before activating a new
one','GUI1');
end


% --- BUTTON: RGB Calibration
function togglebutton3_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of togglebutton3
if get(handles.togglebutton4,'Value')==0 && get(handles.togglebut-
ton1,'Value')==0 && get(handles.togglebutton2,'Value')==0 && get(hand-
les.togglebutton5,'Value')==0
    sc=get(handles.togglebutton3,'Value');
    if sc==1
        set(handles.uipanel1,'BackgroundColor',[1,1,1]);
        set(handles.togglebutton3,'String','OFF');
        set(handles.pushbutton18,'enable','on');
        set(handles.pushbutton1,'enable','on');
        set(handles.pushbutton2,'enable','on');
        set(handles.pushbutton3,'enable','on');
        set(handles.pushbutton4,'enable','on');
        set(handles.pushbutton6,'enable','on');
    elseif sc==0
        set(handles.uipanel1,'BackgroundColor',[0.94,0.94,0.94]);
        set(handles.pushbutton1,'enable','off');
        set(handles.pushbutton2,'enable','off');
        set(handles.pushbutton3,'enable','off');
        set(handles.pushbutton4,'enable','off');
        set(handles.pushbutton6,'enable','off');

        set(handles.togglebutton3,'String','ON');
    end
else
    set(handles.togglebutton3,'Value',0);
    warndlg('Please close the active panels before activating a new
one','GUI1');
end


% --- Executes on button press in pushbutton44.
function pushbutton44_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton44 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
helpdlg('Activate a panel to access its functions by pressing its "ON"
button. Depending on the connections you will be able to unlock cer-
tain panels. You can retry the connections or reset the motor position
by pressing the buttons intended.');
```

Anexo E: Planos del prototipo




4
3
2
1
F
E
D
C
B
A
4
∅2
0,50
1,50
∅2
3
7
1,50
6
3
10
5
∅3
5
9
1
5
∅3
8,50
6
2
2
2
14,50
1,50
5
10,50
7
5
28
∅5
∅10
∅9
∅5
15
5
5
∅5
49,50
4
8
8
3
50
SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MM
ACABADO SUPERFICIAL:
TOLERANCIAS:
LINEAL:
ANGULAR:
ACABADO:
REBARBAR Y ROMPER ARISTAS VIVAS
NO CAMBIE LA ESCALA
REVISIÓN
NOMBRE
FIRMA
FECHA
DIBUJ.
Jorge LB
VERIF.
APROB.
FABR.
CALID.
TÍTULO:
Soporte final
MATERIAL:
Z-ULTRAT
N.º DE DIBUJO
A4
PESO:
ESCALA:2:1
HOJA 1 DE 1



SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MM
ACABADO SUPERFICIAL:
TOLERANCIAS:
LINEAL:
ANGULAR:

ACABADO:

REBARBAR Y ROMPER ARISTAS VIVAS

NO CAMBIE LA ESCALA

REVISIÓN

	NOMBRE	FIRMA	FECHA
DIBUJ.	Jorge LB		
VERIF.			
APROB.			
FABR.			
CALID.			

TÍTULO:

Soporte motor

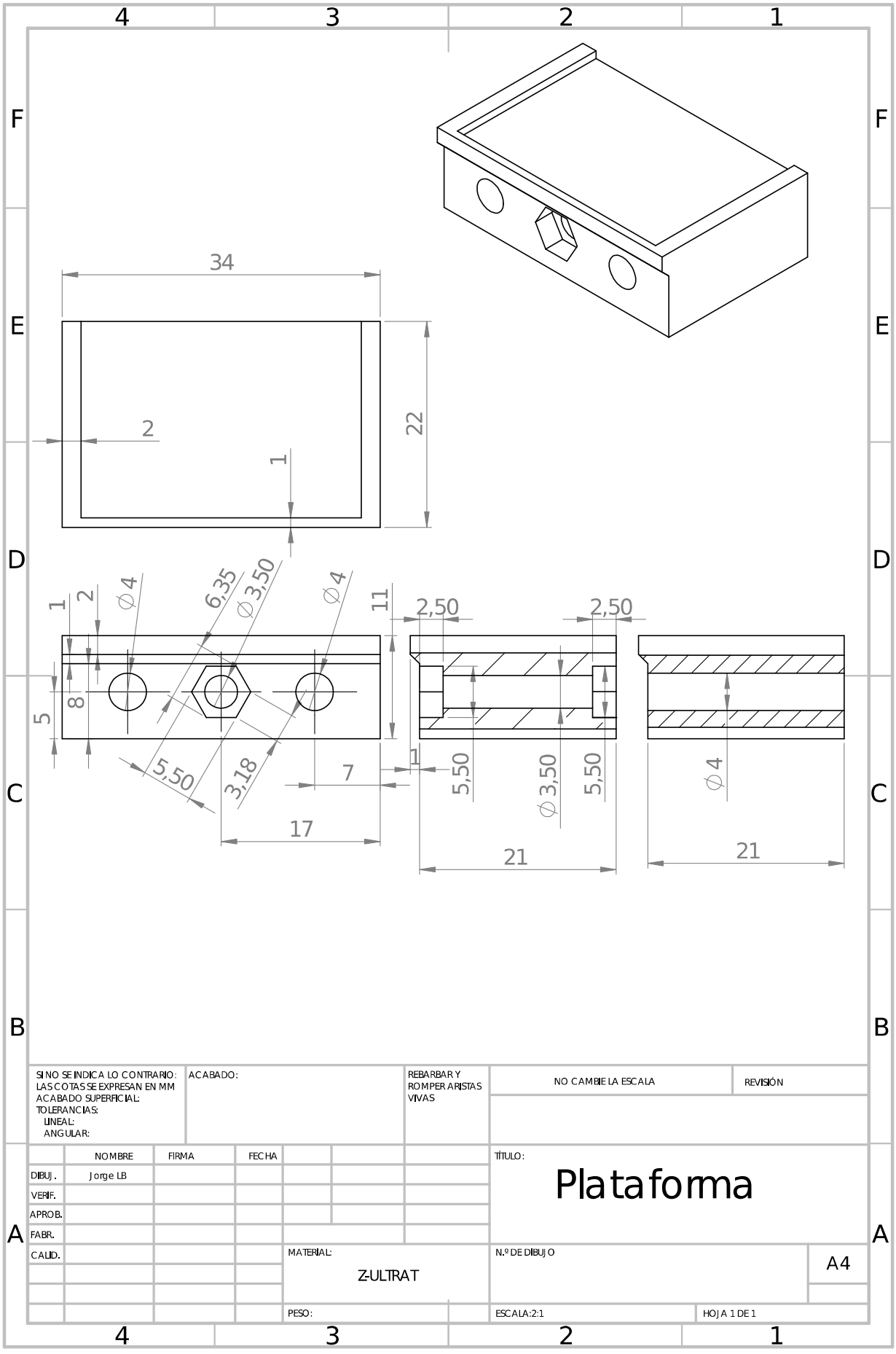
MATERIAL: Z-ULTRAT

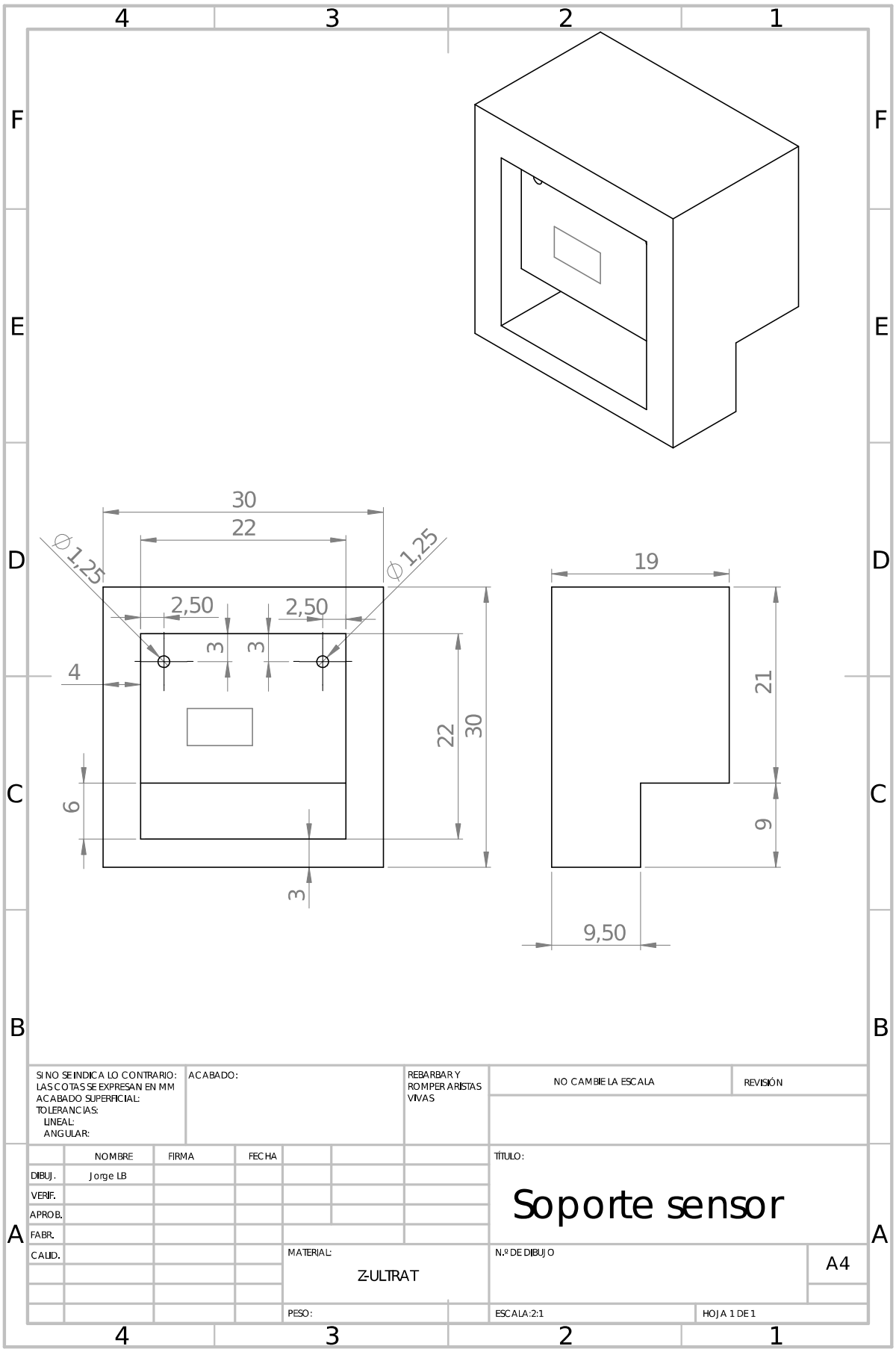
N.º DE DIBUJO

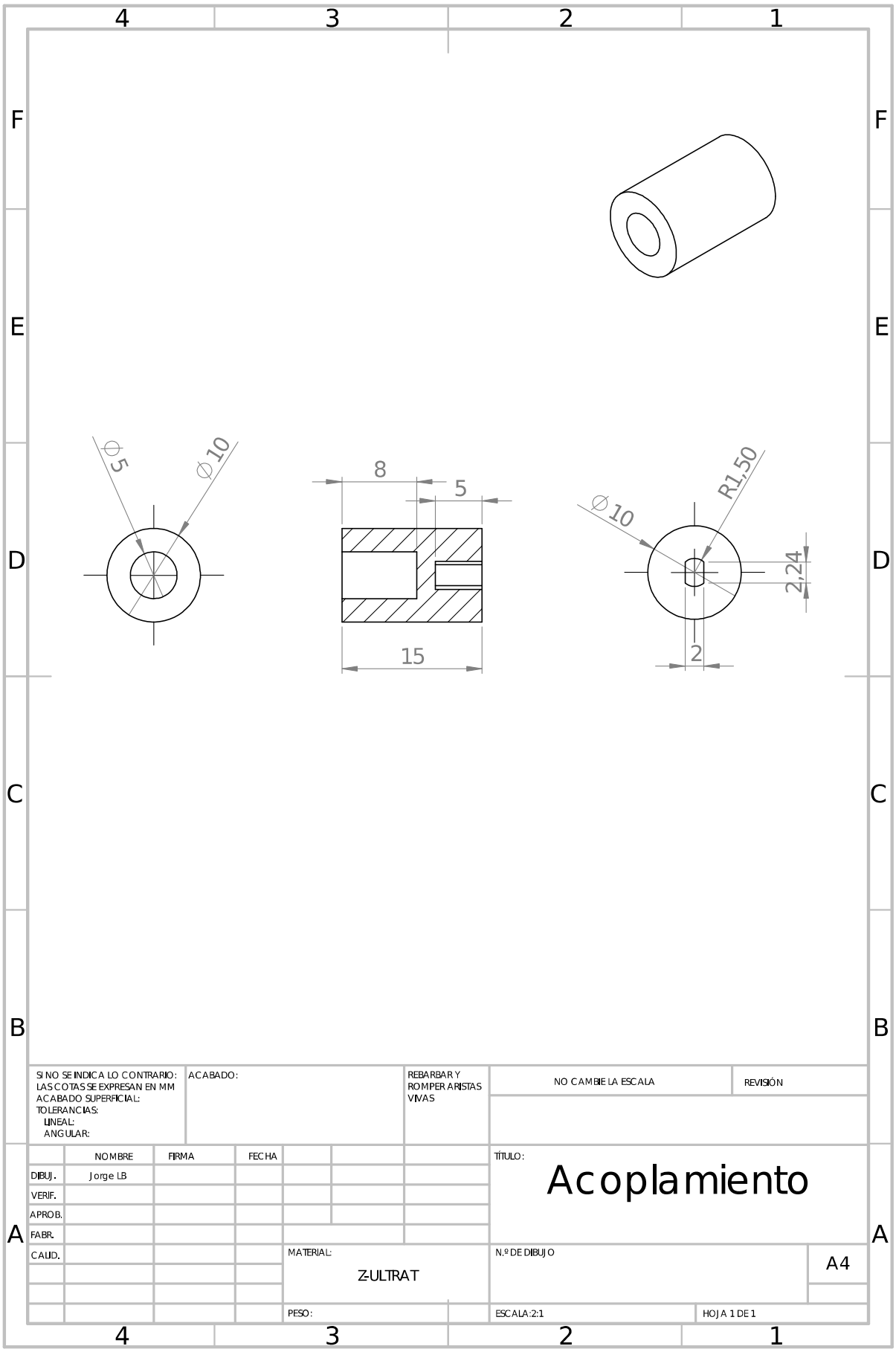
A4

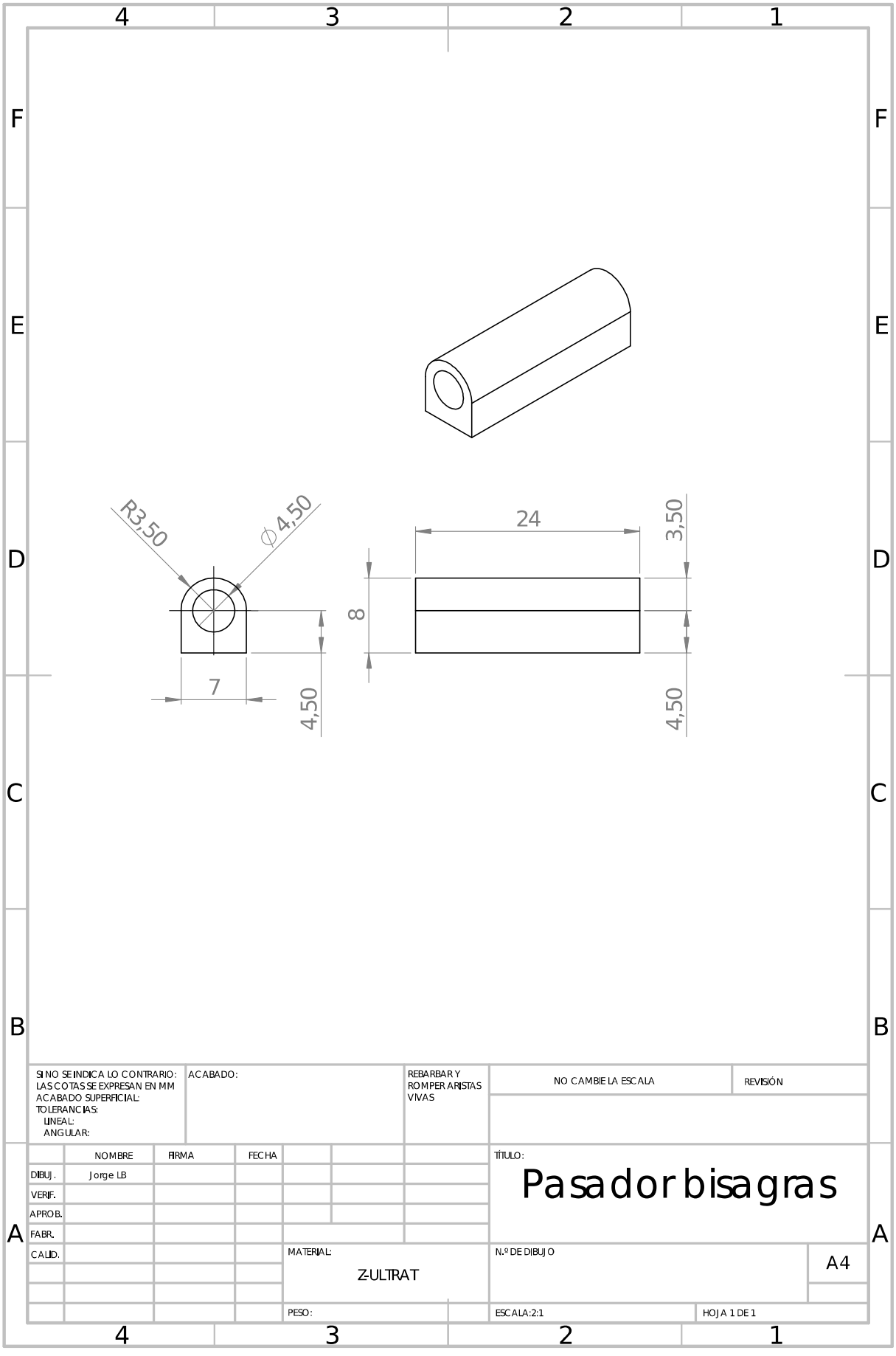
PESO:

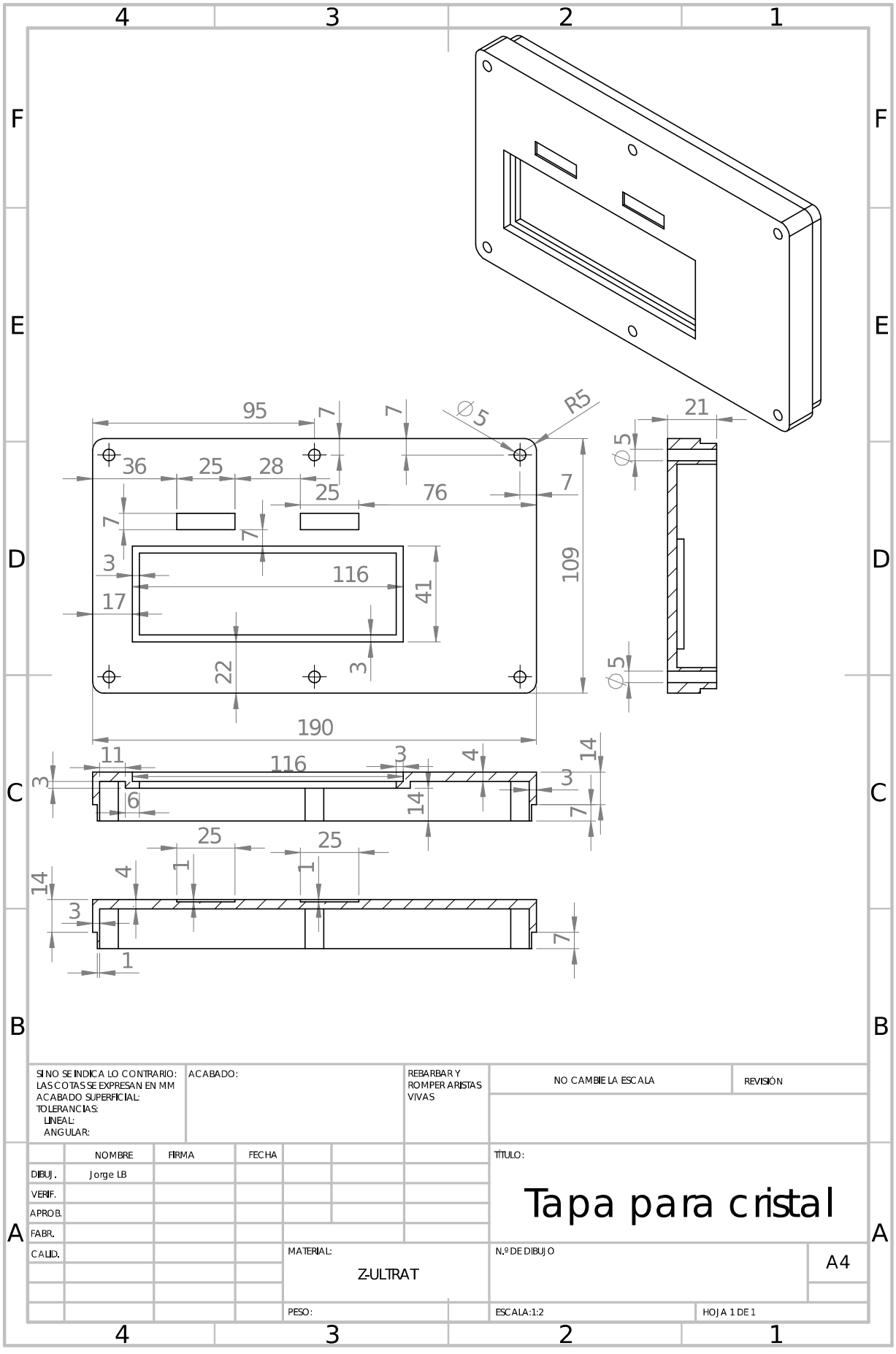
ESCALA:1:1

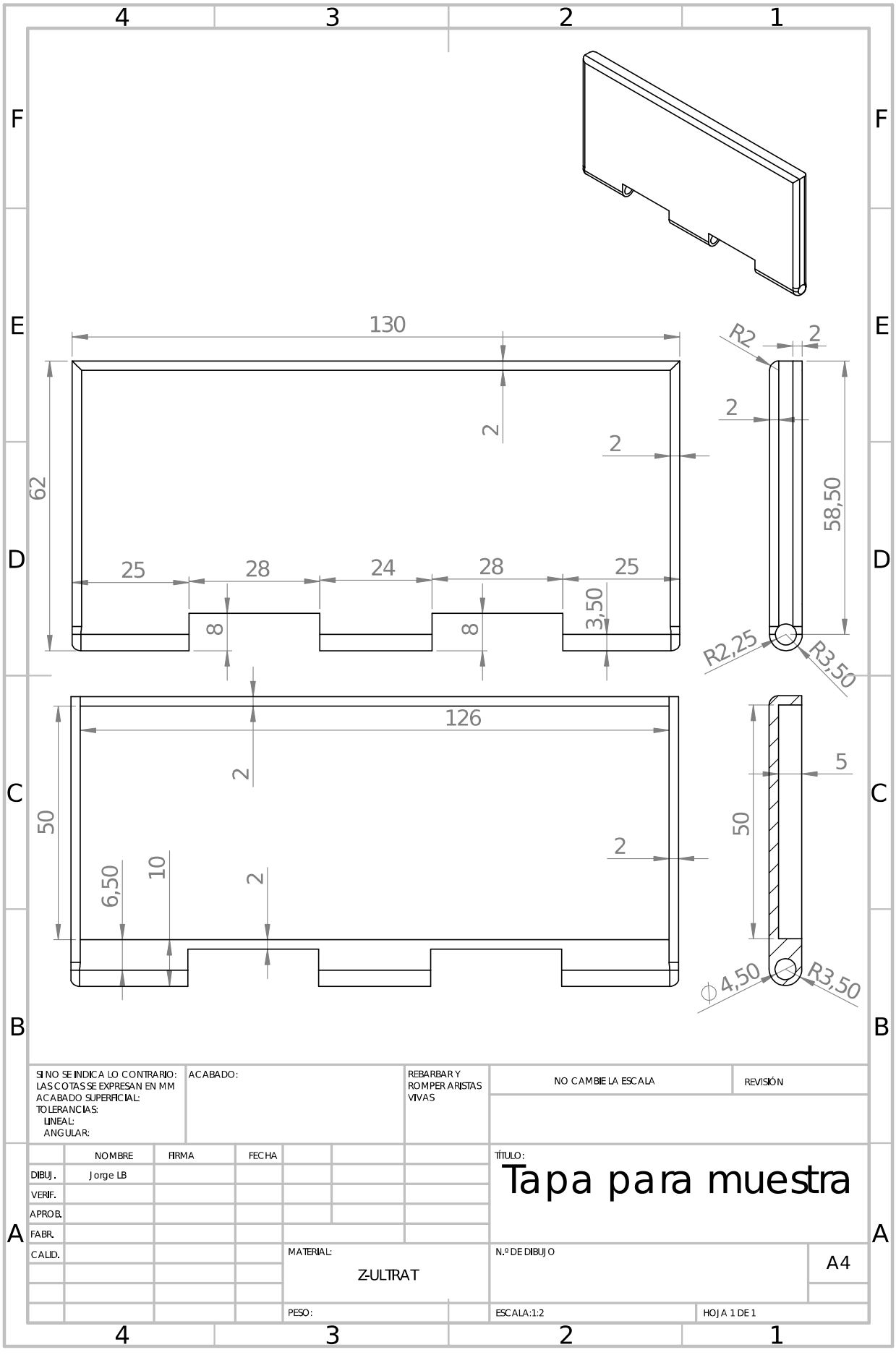

4
3
2
1
F
E
D
C
B
A
34
22
2
1
1
2
Ø4
6,35
Ø3,50
Ø4
11
2,50
2,50
8
5
5,50
3,18
7
1
5,50
Ø3,50
5,50
Ø4
17
21
21
SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MM
ACABADO SUPERFICIAL:
TOLERANCIAS:
LINEAL:
ANGULAR:
ACABADO:
REBARBAR Y ROMPER ARISTAS VIVAS
NO CAMBIE LA ESCALA
REVISIÓN
NOMBRE
FIRMA
FECHA
DIBUJ.
Jorge LB
VERIF.
APROB.
FABR.
CALID.
TÍTULO:
Plataforma
MATERIAL:
Z-ULTRAT
N.º DE DIBUJO
A4
PESO:
ESCALA:2:1
HOJA 1 DE 1

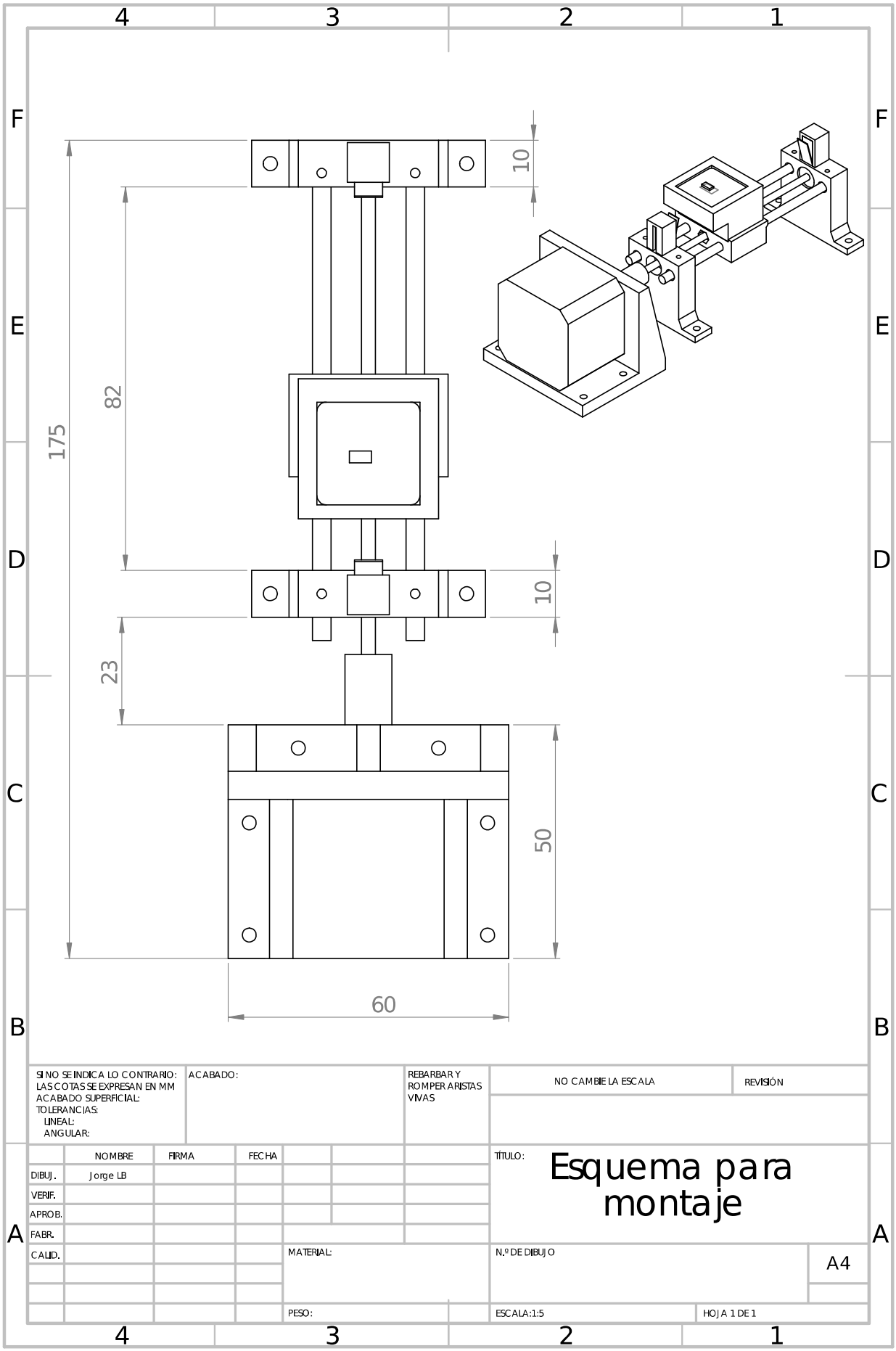

4
3
2
1
F
E
D
C
B
A
30
22
Ø1,25
Ø1,25
2,50
2,50
3
3
4
6
22
30
3
19
21
9
9,50
SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MM
ACABADO SUPERFICIAL:
TOLERANCIAS:
LINEAL:
ANGULAR:
ACABADO:
REBARBAR Y ROMPER ARISTAS VIVAS
NO CAMBIE LA ESCALA
REVISIÓN
NOMBRE
FIRMA
FECHA
DIBUJ.
Jorge LB
VERIF.
APROB.
FABR.
CALID.
TÍTULO:
Soporte sensor
MATERIAL:
Z-ULTRAT
N.º DE DIBUJO
A4
PESO:
ESCALA:2:1
HOJA 1 DE 1


4
3
2
1
F
E
D
C
B
A
∅5
∅10
8
5
15
∅10
R1,50
2,24
2
SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MM
ACABADO SUPERFICIAL:
TOLERANCIAS:
LINEAL:
ANGULAR:
ACABADO:
REBARBAR Y ROMPER ARISTAS VIVAS
NO CAMBIE LA ESCALA
REVISIÓN
NOMBRE
FIRMA
FECHA
DIBUJ.
Jorge LB
VERIF.
APROB.
FABR.
CALID.
TÍTULO:
Acoplamiento
MATERIAL:
Z-ULTRAT
N.º DE DIBUJO
A4
PESO:
ESCALA:2:1
HOJA 1 DE 1


4
3
2
1
F
E
D
C
B
A
R3,50
∅4,50
7
4,50
24
3,50
8
4,50
SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MM
ACABADO SUPERFICIAL:
TOLERANCIAS:
LINEAL:
ANGULAR:
ACABADO:
REBARBAR Y ROMPER ARISTAS VIVAS
NO CAMBIE LA ESCALA
REVISIÓN
NOMBRE
FIRMA
FECHA
DIBUJ.
Jorge LB
VERIF.
APROB.
FABR.
CALID.
TÍTULO:
Pasador bisagras
MATERIAL:
Z-ULTRAT
N.º DE DIBUJO
A4
PESO:
ESCALA:2:1
HOJA 1 DE 1


4
3
2
1
F
E
D
C
B
A
95
7
7
∅5
R5
21
36
25
28
25
76
7
109
116
41
17
3
22
190
11
6
14
25
1
4
1
SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MM
ACABADO SUPERFICIAL:
TOLERANCIAS:
LINEAL:
ANGULAR:
ACABADO:
REBARBAR Y ROMPER ARISTAS VIVAS
NO CAMBIE LA ESCALA
REVISIÓN
NOMBRE
FIRMA
FECHA
DIBUJ.
Jorge LB
VERIF.
APROB.
FABR.
CALID.
TÍTULO:
Tapa para cristal
MATERIAL:
Z-ULTRAT
N.º DE DIBUJO
A4
PESO:
ESCALA:1:2
HOJA 1 DE 1


4
3
2
1
F
E
D
C
B
A
130
2
2
62
25
28
24
28
25
8
8
3,50
R2
2
2
58,50
R2,25
R3,50
126
2
50
2
6,50
10
2
5
50
⌀4,50
R3,50
SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MM
ACABADO SUPERFICIAL:
TOLERANCIAS:
LINEAL:
ANGULAR:
ACABADO:
REBARBAR Y ROMPER ARISTAS VIVAS
NO CAMBIE LA ESCALA
REVISIÓN
NOMBRE
FIRMA
FECHA
DIBUJ.
Jorge LB
VERIF.
APROB.
FABR.
CALID.
TÍTULO:
Tapa para muestra
MATERIAL:
Z-ULTRAT
N.º DE DIBUJO
A4
PESO:
ESCALA:1:2
HOJA 1 DE 1


4
3
2
1
F
E
D
C
B
A
10
82
175
10
23
50
60
SI NO SE INDICA LO CONTRARIO:
LAS COTAS SE EXPRESAN EN MM
ACABADO SUPERFICIAL:
TOLERANCIAS:
LINEAL:
ANGULAR:
ACABADO:
REBARBAR Y ROMPER ARISTAS VIVAS
NO CAMBIE LA ESCALA
REVISIÓN
NOMBRE
FIRMA
FECHA
DIBUJ.
Jorge LB
VERIF.
APROB.
FABR.
CALID.
TÍTULO:
Esquema para montaje
MATERIAL:
N.º DE DIBUJO
A4
PESO:
ESCALA:1:5
HOJA 1 DE 1