UVIC
UNIVERSITAT DE VIC
UNIVERSITAT CENTRAL
DE CATALUNYA

UGRANOLLERS
UNIVERSITAT DE VIC
UNIVERSITAT CENTRAL
DE CATALUNYA

Proyecto final de carrera
Grado de ingeniería de automoción

Visión artificial para el control de calidad de un retrovisor

Yeray Merino Romero

Supervisores:
David Arcos Gutiérrez
Héctor Álvarez Sánchez

Resumen

Título: Visión artificial para el control de calidad de un retrovisor
Autor: Yeray Merino Romero
Tutores: David Arcos Gutiérrez y Héctor Álvarez Sánchez
Fecha: Junio de 2021
Palabras clave: Revolución industrial, industria 4.0, visión artificial, OpenCV

La continua revolución industrial que se ha producido desde el siglo XVIII es vital para el desarrollo del mundo. Estas transformaciones industriales son los factores desencadenantes de los procesos de transformación tecnológica y productiva que afectan directamente a la sociedad y a la economía.

En la actualidad, estamos siendo testigos de la Cuarta Revolución Industrial [1], denominada como la Industria 4.0, que arranca con la consolidación de las fábricas inteligentes, representando una evolución en los procesos de fabricación al incorporar las nuevas metodologías digitales disponibles en la actualidad. En consecuencia, es fundamental que cualquier empresa industrial se sume a esta evolución para no perder su nivel de competitividad y no quedarse en un lugar desfavorecido en el mercado.

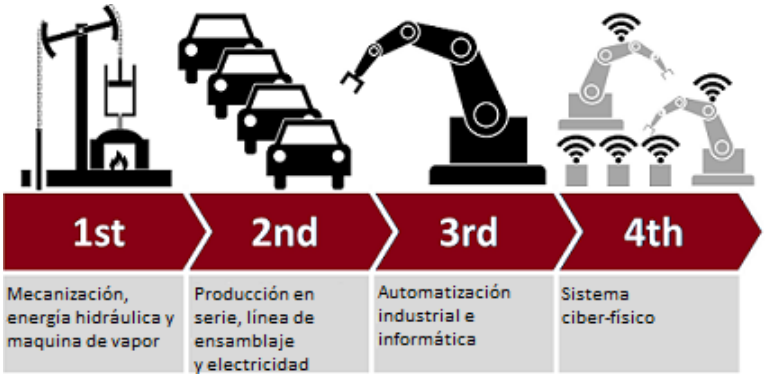


Figura 0. Diagrama de las primeras cuatro revoluciones industriales. Roser, C. 2015 [2]

La visión artificial representa una de las herramientas transversales más relevantes dentro de la industria 4.0 con la función de obtener, procesar y analizar imágenes del mundo físico, con el objetivo de generar información que pueda ser interpretada y empleada por una máquina, a través de procesos digitales. La trazabilidad, el control de calidad, el soporte a la producción, la seguridad industrial, el control de procesos, la logística así como la generación de enormes cantidades de datos, constituyen una parte integral de las acciones en las que los sistemas de visión artificial intervienen.

En este trabajo se presenta el desarrollo del sistema de control de calidad de un producto final guiado por visión artificial, utilizando una cámara web integrada al software libre Python y una cámara térmica que trabajan en conjunto con varias librerías siendo la principal OpenCV. El principio de funcionamiento del control de calidad, está basado en un sistema de posicionamiento y un pequeño programa capaz de mover las lunas de los retrovisores, así como encender su calefactado y luces, frente a la lente de una cámara, la cual se encargará de capturar la imagen y procesarla por medio de lenguaje software. El sistema permite analizar piezas previo envío a cliente, comprobando que tanto el montaje como las funcionalidades son correctas, en caso contrario, se notifica al usuario de forma detallada las incidencias o fallos que posee el producto. De esta forma, se consigue un sistema de control de calidad automático, meticuloso, eficaz y rápido pero el operario, ya que no requiere de su atención durante el proceso de verificación.

Abstract

Title: Artificial vision for quality control of a rear-view mirror
Author: Yeray Merino Romero
Tutors: David Arcos Gutiérrez and Héctor Álvarez Sánchez
Date: June 2021
Keywords: Industrial Revolution, Industry 4.0, Computer Vision, OpenCV

The continuing industrial revolution that has occurred since the 18th century is vital to the development of the world. These industrial transformations are triggers for processes of technological and productive transformation that directly affect society and the economy.

Today we are witnessing the fourth industrial revolution [1], Industry 4.0, which began with the integration of smart factories and represents the evolution of the manufacturing process by merging the new digital methods available today. Therefore, it is essential that any industrial company adheres to this development process so as not to lose its level of competitiveness and not remain in a disadvantageous position in the market.

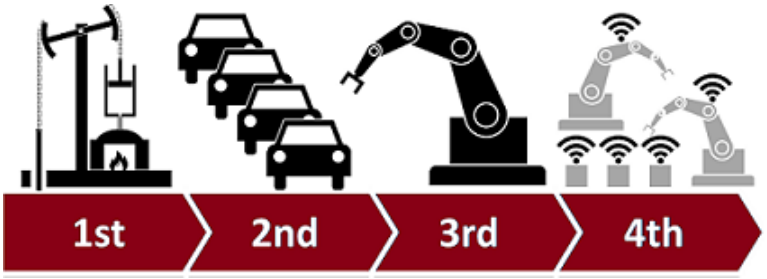


Figure 0. Diagram of the first four industrial revolutions. Roser, C. 2015 [2]

Artificial vision represents one of the most relevant transversal tools within Industry 4.0 with the function of obtaining, processing and analyzing images of the physical world, with the aim of generating information that can be interpreted and used by a machine, through digital processes. Traceability, quality control, production support, industrial safety, process control, logistics as well as the generation of large amounts of data, constitute an integral part of the actions in which artificial vision systems intervene.

This work presents the development of the quality control system of a final product guided by artificial vision, using a web camera integrated into the free Python software and a thermal camera that work in conjunction with several libraries, the main one being OpenCV. The operating principle of quality control is based on a positioning system and a small program capable of moving the mirror glass, as well as turning on its heater and lights, in front of the lens of a camera, which will take care of capture the image and process it through language software. The system allows parts to be analyzed prior to shipment to the customer, checking that both the assembly and the functionalities are correct, otherwise, the user is notified in detail of incidents or faults that the product has. In this way, an automatic, meticulous, efficient and fast quality control system is achieved, but the operator does not require his attention during the verification process.

Index

Capítulo 1: Introducción

11.1 Motivación

La cuarta revolución industrial, también conocida como industria 4.0, es la revolución actual donde los dispositivos inteligentes obtienen gran importancia. Cuando hablamos de procesos automáticos no podemos evitar hablar de la visión artificial, la cual es la disciplina a la que nos referimos cuando tratamos con métodos con el objetivo de analizar, procesar o comprender imágenes reales mediante ordenadores, lo que requiere una transformación de la información a un lenguaje que los ordenadores puedan manejar.

Hoy en día estos procesos están principalmente disponibles en el mercado como servicios, es decir, se puede contratar una entidad privada que estudie los procesos de fabricación de una empresa e implemente un sistema de visión artificial. El problema reside en los altos costes de dichos servicios, apartando del mercado a pequeñas empresas o líneas reducidas por no obtener rentabilidad económica de estos servicios.

Por esta razón, junto con las ganas de mejorar el conocimiento de dos de las tecnologías con mayor auge en los últimos años, como son el lenguaje de programación python y el hardware Raspberry Pi, se dispone a crear un sistema de visión artificial eficaz de bajo coste, capaz de implementarse en cualquier control de calidad.

La empresa Arcol S.A [3], donde actualmente se están cursando las prácticas del grado, me ha motivado para emprender el proyecto junto a ellos, creando un sistema de calidad enfocado a una familia de productos, con el objetivo de mejorar el sistema de calidad y reducir el tiempo necesario de la correcta verificación de los productos finales.

1.2 Objetivos

El objetivo principal del proyecto es el diseño, desarrollo e implementación de un software basado en visión artificial de bajo costo, capaz de realizar un control de calidad de un retrovisor. Concretamente el control consiste en la verificación del correcto montaje de un retrovisor, su calefactado, los grados de movimiento de las lunas, así como otros posibles componentes. A continuación se enumeran los objetivos desglosados:

- Adquirir conocimientos y bases sobre la visión artificial.
- Gestionar todas las fases del desarrollo de un nuevo producto, desde la toma de requerimientos o necesidades o la creación e implementación.
- Producir un sistema de adquisición de imagen eficaz y asequible (de coste reducido).
- Desarrollar un módulo de tratamiento de imagen con un rendimiento adecuado.
- Realizar un sistema capaz de leer las temperaturas de calefactado así como interpretarlas.
- Diseñar un sistema de centrado del retrovisor comprobando el funcionamiento de los motores.
- Planificar y crear el entorno de trabajo para el control de calidad y el sistema de visión artificial.
- Desarrollar un prototipo e implementar el sistema en la empresa ARCOL S.A.

1.3 Organización Gantt-Chart

Con el objetivo de exponer el tiempo de dedicación previsto en las diferentes etapas a lo largo del tiempo total del proyecto, se ha utilizado la herramienta gráfica diagrama de Gantt-Chart[1] [4], compuesto por un eje vertical donde se establecen las actividades que constituyen el trabajo que se va a ejecutar y en un eje horizontal se muestra un calendario de duración de cada una de ellas.

Gracias a este diagrama se ha planificado el desarrollo del proyecto, relacionando las actividades de manera que se pueda visualizar el camino crítico (es decir, la actividad más compleja o la que demandará más tiempo) y permitiendo reflejar la escala de tiempos para facilitar la asignación de recursos y determinar incluso el presupuesto del proyecto (en cuanto a tiempo se refiere). También ha sido útil para la detección de posibles riesgos y cuellos de botella que pudieran comprometer la ejecución en tiempo del proyecto.

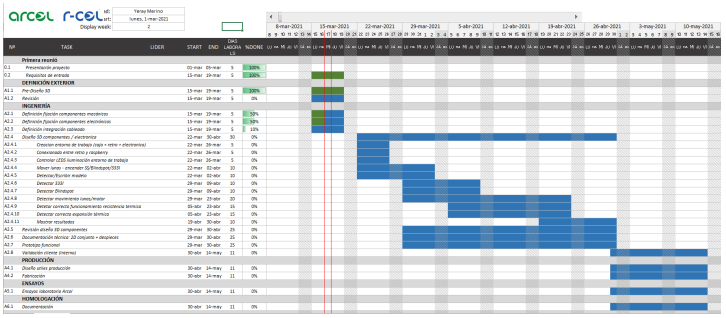


Figura 1.3.1 Gantt-chart del proyecto.

1.4 Descripción de los capítulos

A continuación, se introduce en detalle la estructura del informe y un breve resumen del contenido de cada capítulo.

- Capítulo 1: Introducción al proyecto explicando los motivos que llevaron a la elección y realización del proyecto, así como los objetivos marcados inicialmente.
- Capítulo 2: Breve introducción a la visión artificial, sus aplicaciones actuales, diferentes métodos de uso, sus componentes y finalmente el principio de funcionamiento de un sistema.
- Capítulo 3: Se nombran los recursos utilizados durante el desarrollo del proyecto, desde los recursos humanos al hardware y software. A su vez se detalla el presupuesto realizado.
- Capítulo 4: Descripción de la metodología usada y la explicación de todas las acciones realizadas cronológicamente.
- Capítulo 5: Exposición e ilustración de las pruebas realizadas y resultados obtenidos
- Capítulo 6: Conclusiones basadas en el desarrollo del proyecto y las pruebas realizadas así como los objetivos logrados.
- Capítulo 7: Referencias.
- Capítulo 8: Apéndice.

[1] Gantt-chart → pág 38, punto 8.1 del apéndice.

Capítulo 2: Estado del arte

2.1 Introducción a la Visión Artificial

Uno de los sentidos más importantes de los seres humanos es la visión [5], esta se utiliza para obtener información visual del entorno físico y se calcula que más del 70% de las tareas cerebrales se utilizan para analizar información visual. Casi todas las disciplinas científicas utilizan herramientas gráficas para transmitir conocimiento. Por ejemplo, en Ingeniería Electrónica se emplean esquemas de circuitos, a modo gráfico, para describirlos. Se podría hacerlo mediante texto, pero para la especie humana resulta mucho más eficiente procesar imágenes que procesar texto. La visión humana es el sentido más desarrollado y el que menos se conoce debido a su gran complejidad. Es una actividad inconsciente y difícil de saber cómo se produce. De hecho, hoy en día, se carece de una teoría que explique cómo los humanos perciben el exterior a través de la vista.

La Visión Artificial o también llamada Visión por Computador, pretende capturar la información visual del entorno físico para extraer características relevantes visuales, utilizando procedimientos automáticos. Según Marr, "*Visión es un proceso que produce a partir de imágenes del mundo exterior, una descripción que es útil para el observador y que no tiene información irrelevante*".

Para tratar sobre la visión, lo primero es estudiar la naturaleza de la luz, un fenómeno que se puede interpretar a partir de la teoría ondulatoria de Huygens [6], considerando que las fuentes luminosas emanan de un frente de ondas, las cuales pueden ser representadas, imaginariamente, por líneas rectas en la dirección de la propagación del tren de ondas que llamaremos rayo. Si tratamos la luz como una onda electromagnética, se puede apreciar que es un tipo de energía que no requiere de medio material. Las radiaciones electromagnéticas se propagan en forma de dos ondas vectoriales mutuamente acopladas y perpendiculares entre sí; una onda para el campo eléctrico y otra para el campo magnético (figura 2.1). Según la teoría ondulatoria, la luz se propaga en forma de onda que viaja en el espacio libre con una velocidad constante $c \approx 3 \cdot 10^8 m/s$.

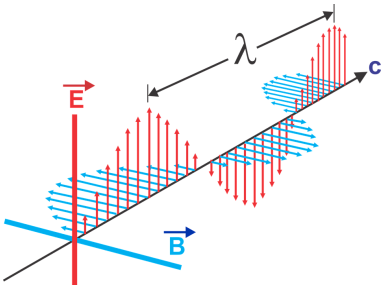


Figura 2.1.1 Campo electromagnético

El espectro visible es una porción muy pequeña del conjunto de ondas electromagnéticas que tiene la peculiaridad de ser captada por los ojos y procesada en el cerebro. El ojo humano es capaz de distinguir radiaciones de longitudes de onda comprendidas entre los 380 nm a los 780 nm, cuyas frecuencias oscilan entre los $3.2 \cdot 10^{14} Hz$ y $7.7 \cdot 10^{14} Hz$. El sentido de la vista transforma las diferentes amplitudes y frecuencias del espectro visible en sensaciones conocidas como brillo y color respectivamente.

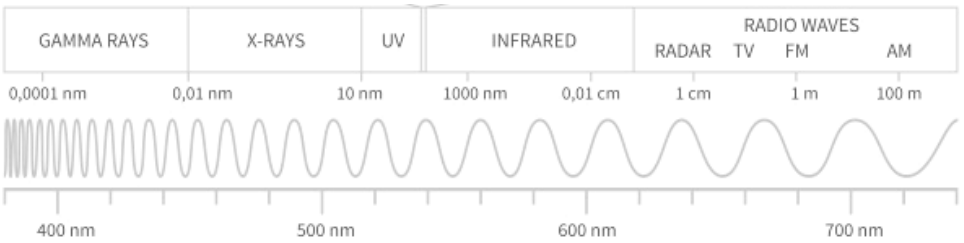


Figura 2.1.2 Espectro de luz

2.2 Configuración básica de un sistema de Visión Artificial

Los dos pilares del sistema físico de visión artificial son: el sistema de formación de las imágenes y el sistema de procesamiento de estas. El primer apartado está constituido por el subsistema de iluminación, de captación de la imagen y de adquisición de la señal en el computador. Una vez introducida la señal en el computador, se procesa mediante los algoritmos para transformarla en información de alto nivel. Esta información puede ser utilizada para su representación visual, para actuar en el planificador de un robot o puede ser fuente de datos para un autómata programable. En definitiva, múltiples periféricos pueden ser receptores de esta información y vincularse con el sistema de procesamiento de las imágenes. A continuación se detallan los subsistemas:

- Subsistema de iluminación: conjunto de artefactos que producen radiación electromagnética que inciden sobre los objetos a visualizar, por ejemplo una lámpara, un filtro de luz o un láser.

- Subsistema de captación: son los transductores que convierten la radiación reflejada luminosa en señales eléctricas. Fundamentalmente se habla de las cámaras CCD, no sólo en el espectro visible, sino que van desde la radiación gamma hasta la radiofrecuencia o microondas, dando paso a sensores de ultrasonidos, sonar, radar, telescópica.

- Subsistema de adquisición: la señal eléctrica procedente de las cámaras forman la señal de vídeo. Hay una tendencia creciente a que su naturaleza sea de tipo digital, pero todavía existen muchas señales de vídeo de carácter analógico [7] (CCIR, PAL, RS170, NTSC,…). Para ser tratadas hay que muestrearlas y cuantificarlas. Ambas tareas son realizadas por las tarjetas de adquisición. También se las llama frame grabbers. Se depositan en el bus de expansión del computador. Disponibles para buses desde PCI hasta VMP. Recientemente, también se están empleando las tecnologías de USB o FireWire.

- Subsistema de procesamiento: Suele ser un computador o un cluster de computadores, dependiendo de las necesidades de los algoritmos de Visión Artificial. Parten de una representación digital de las imágenes y procesan esta información hasta alcanzar otro tipo de información de más alto nivel. La transformación dependerá de la algoritmia.

- Subsistemas periféricos: conjunto de elementos receptores de la información de alto nivel. Puede ser un monitor de altas prestaciones gráficas, un automatismo, una impresora sacando las características…

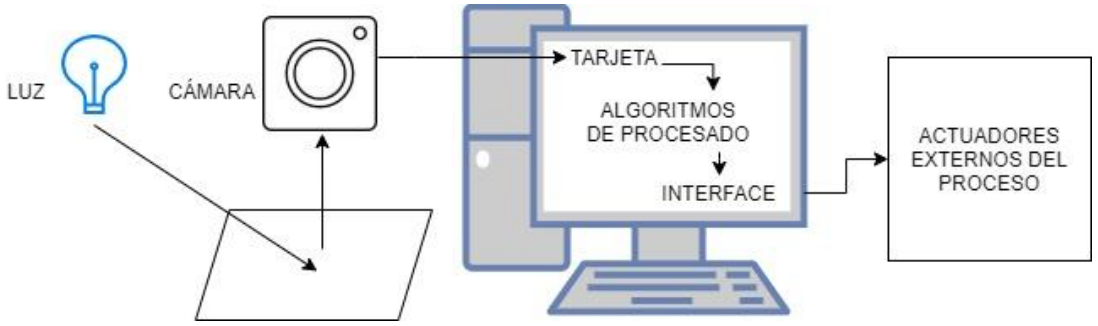


Figura 2.2 Configuración básica Visión Artificial.

2.2.1 Iluminación

La iluminación es una de las claves del éxito en los resultados de la visión artificial. Los sistemas de visión artificial crean imágenes a través del análisis de la luz reflejada por un objeto, no del análisis del propio objeto. Una técnica de iluminación implica una fuente de luz y su ubicación con respecto a la pieza y la cámara. Una técnica de iluminación particular puede mejorar una imagen de forma que se anulen algunas características y se mejoran otras, silueteando una pieza que oscurece los detalles superficiales para permitir la medición de sus bordes, por ejemplo.

1. **Retroiluminación**: Para aplicaciones que solo requieren mediciones externas o de borde, la retroiluminación mejora los contornos del objeto. La luz de fondo ayuda a detectar formas y hace que las medidas dimensionales sean más fiables.

2. **Iluminación difusa axial**: bloquea la luz en la trayectoria óptica desde el lateral (coaxial). El espejo semitransparente ilumina desde el lateral arrojando la luz sobre la pieza de trabajo. La pieza refleja la luz hacia la cámara a través de un espejo semitransparente, produciendo así una imagen homogénea e iluminada uniformemente.

3. **Luz estructurada:** es la proyección de un patrón de luz (plano, rejilla o forma más compleja) sobre un objeto en un ángulo conocido. Es muy útil para ofrecer inspecciones superficiales independientes del contraste, obtener información dimensional y calcular volúmenes.

4. **Iluminación de campo oscuro:** La iluminación direccional revela con más facilidad los defectos superficiales e incluye la iluminación de campo oscuro y campo brillante. La iluminación de campo oscuro se prefiere generalmente para aplicaciones de bajo contraste. En la iluminación de campo oscuro, la luz especular se refleja alejándose de la cámara, y la luz difusa procedente de la textura superficial y los cambios de elevación se refleja en la cámara.

5. **Iluminación de campo brillante:** es ideal para aplicaciones de alto contraste. Sin embargo, las fuentes de luz muy direccionales como el sodio a alta presión y los halógenos de cuarzo pueden producir sombras nítidas y, por lo general, no aportan una iluminación consistente en todo el campo de visión. Por consiguiente, las reflexiones especulares y los efectos “hot spot” en superficies brillantes o reflectantes pueden requerir una fuente de luz más difusa que proporcione incluso iluminación en el campo brillante.

6. **Iluminación cenital difusa:** Ilumina uniformemente las características de interés y puede enmascarar irregularidades que no sean de interés, confundiendo la escena.

7. **Iluminación estroboscópica:** Usada en aplicaciones de alta velocidad para examinar objetos inmovilizando su movimiento. Para evitar la borrosidad se puede usar una luz estroboscópica

8. **Filtros de color**: crean contraste para aclara u oscureecer las caracteristicas del objeto. Los filtros de color similar aclaran y los filtros de color opuesto oscurecen. También pueden ser puntos de referencia.

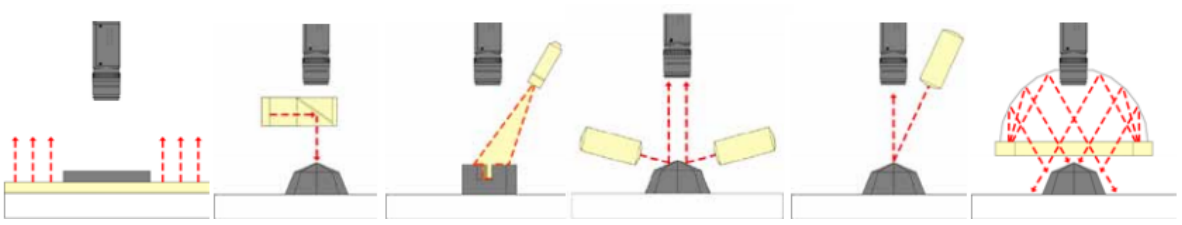


Figura 2.2.1 Tipos de iluminación

2.3 Etapas básicas de una aplicación en Visión Artificial

Se puede decir que todos los sistemas de visión artificial, independientemente de la aplicación a la que estén orientados, tienen unas etapas de funcionamiento común. La primera fase está constituida por el sistema de iluminación, la captación de la imagen y la adquisición de la señal en el ordenador, mientras que en la segunda fase se procesa la imagen mediante algoritmos para alcanzar otro tipo de información. Las etapas básicas de un sistema de visión artificial son las siguientes:

Figura 2.3.1 Etapas básicas

1. Adquisición de la imagen: Se trata de la primera etapa para la construcción del sistema de formación de imágenes. Mediante técnicas fotográficas se realza las características visuales del objeto, como podrían ser: formas, texturas, colores...

2. Procesado: En esta etapa se incluyen operaciones para atenuar imperfecciones, mejorar la relación señal-ruido (SNR), mejorar el contraste, regularizar la imagen, optimizar la distribución de intensidades, realzar bordes, etc.

3. Segmentación: Se particiona la imagen decidiendo qué áreas necesitan interpretación o análisis y cuáles no. Existen un abanico de técnicas para realizar estas áreas con significado: crecimiento de regiones, umbralizaciones, uso de color o movimiento, etc.

4. Representación y descripción: Es la etapa en la que se extrae las características morfológicas de las distintas zonas, tales como el área, perímetro, excentricidad así como características basades en textura o color. Mediante esta representación obtenemos una información visual más elaborada.

5. Reconocimiento e interpretación: Es la etapa de realizar una serie de evaluaciones para determinar el resultado de la función del sistema, en el caso de la calidad de un objeto, determinar si es correcto o no. Esta interpretación se puede realizar de varias formas, midiendo objetos, encontrando similitudes o diferencias entre imágenes, analizando formas y áreas, etc. También se pueden añadir procesos externos según los resultados, por ejemplo que se enciendan avisos visuales, avisos acústicos, acciones en la línea de producción, etc.

Otra representación para las distintas etapas son expuestas por los autores González y Woods [8], que mencionan tres niveles de información: bajo (adquisición de imágenes y procesamiento), medio (segmentación y representación) y alto (reconocimiento e interpretación) . El valor añadido es que todas están conectadas a través del conocimiento.

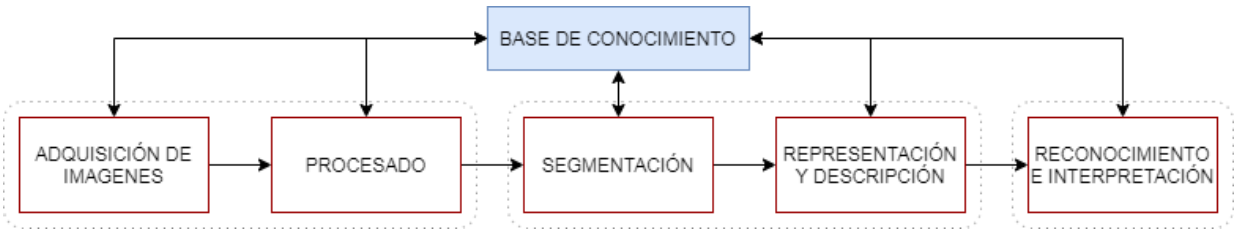


Figura 2.3.2 Etapas básicas por González y Woods

Capítulo 3: Recursos utilizados

3.1 Recursos Humanos

El principal recurso humano de este proyecto ha sido el autor de este. Gracias a las prácticas de trabajo final de carrera, he podido invertir gran parte de mi tiempo diario en la consolidación del proyecto en las instalaciones de la empresa ARCOL S.A, obteniendo apoyo y asesoramiento de los compañeros de trabajo. También se ha dispuesto de un enorme tiempo en el domicilio particular, donde se dispone de varios ordenadores y una impresora 3D, que ha facilitado el desarrollo del prototipo así como utillaje para su correcto funcionamiento (bisagras, carcasas protectoras de componentes, puntos de luz ..). Durante el primer mes se ha realizado un curso para aprender vision artificial con openCV y YOLO [9], estableciendo las bases para el proyecto.

También se ha contado con el apoyo y asesoramiento constante por parte de sus tutores: Héctor Álvarez, responsable técnico de I+D en la empresa ARCOL S.A y David Arcos, candidato a Doctorado en la UPC y desarrollador móvil autónomo. Gracias a sus directrices se ha conseguido gran cantidad de conocimientos transversales, organización de trabajo, gestión del tiempo y experiencia en el desarrollo de proyectos, sabiendo indicar correctamente el rumbo a tomar y las diferentes etapas a cumplir, asesorando los pequeños fallos detectados así como una constante revisión del trabajo realizado.

Figura 3.1.1 Logotipo empresa ARCOL S.A

Otra gran ayuda adicional para el cumplimiento de los objetivos del proyecto ha sido José María Modamio, técnico analista en la línea de inserción automática en R-COL, compañero y amigo en el trabajo. José María me ha guiado desde el principio del proyecto hasta el final hasta el punto de compartir su espacio de trabajo conmigo, guiandome en todas las etapas del proyecto, asesorando en el área electrónica y realizando innumerables pruebas para que el proyecto funcionara correctamente. Sin su ayuda habría sido imposible alcanzar los objetivos de desarrollo del software así como la implementación del sistema en la empresa ARCOL.

Figura 3.1.2 Logotipo empresa R-COL S.A

3.2 Software

El sistema operativo utilizado ha sido Raspberry Pi OS [10] (anteriormente llamado Raspbian) es una distribución del sistema operativo GNU/Linux basado en Debian, y por lo tanto libre para la SBC Raspberry Pi, orientado a la enseñanza de informática. El lanzamiento inicial fue en junio de 2012. Hay varias versiones de Raspbian, siendo la actual Raspbian Buster.

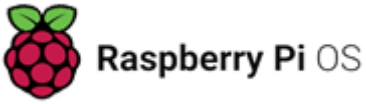


Figura 3.2.1 Logotipo de Raspberry Pi OS.

Dentro del sistema operativo Raspbian se trabaja con el código de lenguaje Phyton [11], un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma. Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License.

Figura 3.2.2 Logotipo de python

Dentro de python se utiliza OpenCV [12] (Open Source Computer Vision), una librería de programación de código abierto dirigida principalmente a la visión por computador en tiempo real, desarrollada por la división rusa de Intel en el centro de Nizhni Nóvgorod. Actualmente también cuenta con el apoyo de Willow Garage y la compañera de visión Itseez. El uso es gratuito bajo la licencia open source BSD. La librería OpenCV es multiplataforma. OpenCV permite desarrollar en C, C ++ o Python y es compatible con el IDE QT Creator y sus correspondientes librerías QT.

Las áreas de aplicación de OpenCV incluye entre muchas otras:

- Odometría visual
- Sistema de reconocimiento facial
- Reconocimiento de gestos
- Interacción hombre-ordenador (HCI)
- Robótica móvil
- Comprensión del movimiento
- Identificación del objeto
- Segmentación y reconocimiento
- Visión estereoscópica
- Estructura de movimiento (SFM)
- Rastreo de movimiento
- Realidad aumentada
- Aprendizaje del árbol de decisiones
- Algoritmo maximización expectativas
- Algoritmo de vecino más cercano k
- Clasificador Bayes
- Redes neuronales
- Soporte de máquinas vectoriales

Figura 3.2.3 Logotipo OpenCV

3.3 Hardware

Como hardware principal se ha utilizado Raspberry Pi [13], es un ordenador de bajo coste (desde 6,17 hasta 94,90€ según modelo) y tamaño reducido, tanto es así que cabe en la palma de la mano, pero puedes conectarle un televisor y un teclado para interactuar con ella exactamente igual que cualquier otra computadora. Es la placa de un ordenador simple compuesto por un SoC, CPU, memoria RAM, puertos de entrada y salida de audio y vídeo, conectividad de red, ranura SD para almacenamiento, reloj, una toma para la alimentación, conexiones para periféricos de bajo nivel, reloj... vamos, prácticamente lo mismo que si miras la parte de atrás de la torre de un ordenador, porque la Raspberry es un ordenador. Eso sí, no tiene interruptor para encenderlo o apagarlo.

Se ha escogido este hardware por su versatilidad y su bajo coste comparado con un ordenador convencional. La última versión, la Raspberry Pi 4B+ cuenta con una potencia de sobra para ser un buen ordenador con Linux, con hasta 5 lenguajes de programación y un tamaño reducido para no tener problemas con la ubicación del dispositivo.

Figura 3.3.1 Raspberry Pi 4B+

Para la obtención de imágenes se utiliza “Raspberry Pi module V2” [14]. Los módulos de cámara Raspberry Pi son productos oficiales de la Fundación Raspberry Pi. El módulo de cámara es un sensor de imagen Sony IMX219 de 8 megapíxeles de alta calidad diseñado a medida para la Raspberry Pi, con una lente de enfoque fijo. Es capaz de imágenes estáticas de 3280 x 2464 píxeles y también admite vídeo de 1080p30, 720p60 y 640x480p90.

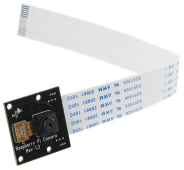


Figura 3.3.2 Cámara de Raspberry Pi

El AMG8833 [15] es un sensor de temperatura de 64 píxeles desarrollado por Panasonic bajo su línea de productos Grid-EYE®. El sensor contiene una matriz de termopilas infrarrojas de 8x8, que se aproximan a la temperatura midiendo la radiación infrarroja emitida por los cuerpos emisores. Gracias a que se puede comunicar a través del bus I2C es compatible con las Raspberry Pi. El sensor contiene una lente integrada que limita el ángulo de visión a 60 grados, lo que da como resultado una región de detección útil para objetos en el campo medio. Opera a 3.3V y 5V, a una frecuencia de muestreo de 1Hz-10Hz, con una resolución de temperatura aproximada de 0.25ºC en un rango de 0ºC a 80ºC. En este proyecto se utiliza este sensor para realizar un análisis de transferencia de calor entre la resistencia de la luna y la propia luna.

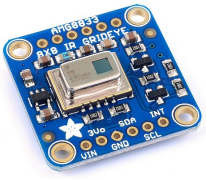


Figura 3.3.3 Camara Adafruit AMG8833

Como producto a realizar el control de calidad, y por tanto indispensable en el proyecto, se ha escogido el retrovisor 375-2 Diagonal[2], un retrovisor de la serie diagonal. El diseño del producto, el concepto del molde y el proceso de manufacturación son acordes con los estándares de automoción. Dicha serie, es la serie más reciente de la empresa ARCOL diseñada especialmente para las autocaravanas incluyendo las últimas tecnologías como cámaras para mejorar la seguridad. También cuenta con la posibilidad de implementarse en varias industrias como podrían ser: coach, buses, minibuses y vehículos especiales (camiones de bomberos, camiones de recogida de residuos, etc).

Figura 3.3.4 Retrovisor diagonal 375-2.

Cuenta con diferentes opciones para ayudar al usuario e incrementar su confort: la posibilidad de cuatro lunas, el nuevo sistema BSD (Blind spot detection), cámaras, motores eléctricos para el movimiento de las lunas, calefactado de las propias lunas, antena de radio, memoria para programar las diferentes configuraciones, posibilidad bicolor y luz de cortesía. Para el proyecto se ha utilizado un retrovisor con todos los extras:

- Voltaje: 12 V
- Dos lunas: superior II categoría y inferior IV categoría.
- Sistema BDS: control del ángulo muerto (incrementando la seguridad)
- Luz de cortesía: Luz alrededor del retrovisor mientras acampas
- Intensidad nominal motor DC (movimiento lunas): 80mA
- Intensidad de arranque motor DC: 150mA
- Calefactado de lunas mediante resistencias: II categoría 20W, IV categoría 14W
- Rango operacional de temperaturas: Entre -30ºC y 70ºC
- Peso unidad: 4 kg

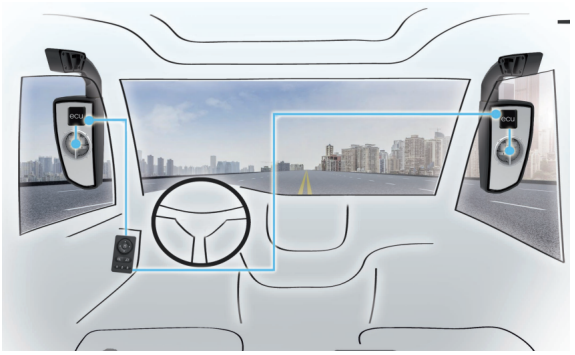


Figura 3.3.4 Representación retrovisores diagonal

[2] Especificaciones retrovisor 375-2 → pág 39-41, punto 8.2 del apéndice.

3.4 Presupuesto

A continuación se detalla el presupuesto utilizado para la elaboración del proyecto, con la consideración de los tres apartados anteriormente mencionados.

Los cálculos del coste de materiales se han realizado basándose en la amortización de elementos de patrimonio: Equipos para tratamiento de la información y sistemas y programas informáticos, según la tabla de amortización simplificada del Ministerio de Hacienda [15] para el cálculo de dicha amortización. Según esos datos, para los bienes informáticos, el coeficiente máximo de amortización es del 26% y el mínimo del 10% (puesto que el periodo máximo son 10 años). Se ha optado por aplicar el coeficiente máximo.

- Duración del proyecto: 5 meses. Enero 2021 - Junio 2022.
- Coeficiente de amortización del 26%
- Coste de amortización = Precio del bien * coeficiente de amortización * meses / 12

Unidad	Descripción	Precio	Coste amortización	Coste proyecto
PC	Raspberry Pi 4B - 8gb	97,27€	97,27*0,26*5/12	10,53€
Cámara visual	Raspberry Pi Camera v2	23,95€	23,95*0,26*5/12	2,59€
Cámara térmica	Camara térmica Adafruit AMG8833	60,71€	60,71*0,26*5/12	6,58€
Caja esmelux	Caja 800x600x425 ref 50223	55,16€	55,16*0,26*5/12	5,97€
Tarjeta expansion	Módulo de relé RPi de 6 Canales Tarjeta de expansión	21,49€	21,49*0,26*5/12	2,32€
Cable Camera	3 x Cable de cinta flexible de repuesto Flex Cable 100 cm	6,59€	6,59*0,26*5/12	0,713€

Figura 3.4.1 Presupuesto del proyecto

Para calcular el sueldo se ha tenido en cuenta el convenio de cooperación entre la universidad de Vic y industrial arcol S.A para la colaboración en la formación de estudiantes, concretamente la propia del autor. En dicho convenio se establece un periodo comprendido entre el 01/02/2021 y 31/05/2021, con un total de 300 h y un salario de 8€ la hora.

- Duración contrato: 4 meses
- Sueldo estimado: horas * sueldo/hora

Tipo de contrato	Horas totales	Sueldo / hora	Cálculo Sueldo	Coste proyecto
Prácticas	300h	8€/h	300*8	2400€

Figura 3.4.1 Presupuesto en sueldo del proyecto

Capítulo 4: Diseño e implementación del problema

Para la correcta absolución del proyecto se ha procedido a distinguir el proyecto en 3 fases:

- Diseño del espacio de trabajo: Realización de un prototipo 3D con el programa Siemens NX [16], creación del utillaje necesario para el correcto funcionamiento así como el alojamiento del retrovisor (piezas impresas con impresora 3D).

- Apartado electrónico: conectar el retrovisor a la raspberry y controlar el movimiento del motor y las lunas, el calefactado, el blindspot y la luz de cortesía. También mediante la raspberry controlar la iluminación del espacio de trabajo y mostrarse en un display.

- Apartado de visión artificial (software openCV): mediante la cámara visual detectar el correcto funcionamiento y movimiento de las lunas, verificar la luz de cortesía y del blindspot (ángulo de punto ciego), y mediante la cámara térmica revisar el funcionamiento del calefactado de las lunas y su correcta expansión térmica.

4.1 Espacio de trabajo

Para la creación del espacio de trabajo se ha diseñado un modelo CAD con el software Siemens NX 1953, optimizando el diseño, la distribución del espacio y la utilización de accesorios, así como la posibilidad de crear piezas en 3D. De esta forma se obtiene una primera vista general de todos los elementos necesarios para conformar el espacio. También se ha realizado una lista de materiales[3] (BOM) junto todos los 2D[4] de las piezas realizadas con impresora 3D y PLA como material. A continuación una vista explosionada del 3D realizado.

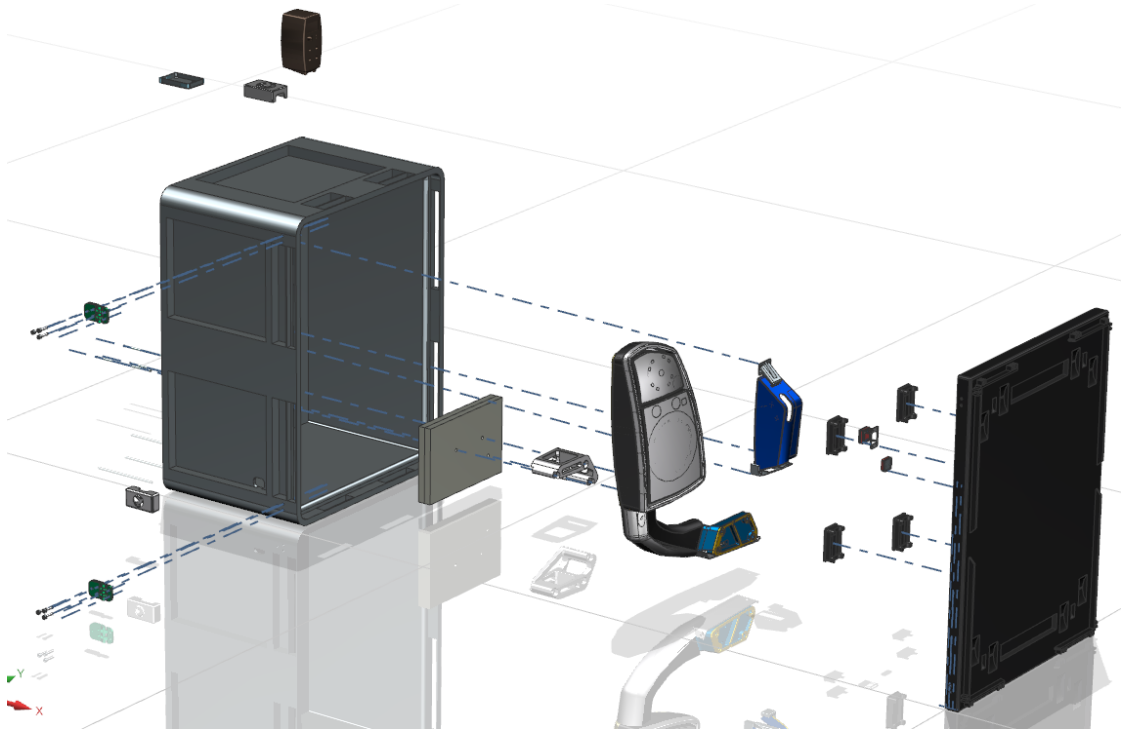


Figura 4.1.1 Vista explosionada CAD.

[3] Lista materiales → pág 42, punto 8.3 del apéndice.

[4] Planos 2D → pág 43- 50, punto 8.4 - 8.11 del apéndice.

El elemento principal es una caja de dimensiones 800x600x424 mm capaz de alojar por completo un retrovisor 375-2 eléctrico, calefactado, con luz blindspot y luz de cortesía. Para la correcta sujeción del retrovisor en la caja se ha utilizado 3 varillas de M6 con sus respectivas tuercas, 2 rectángulos de madera para aumentar el superficie de contacto entre varillas - caja y una geometría de impresión 3D para colocar el retrovisor de forma paralela a la caja, ya que la cara por la que se sujeta tiene una pequeña pendiente. También se han diseñado 3 soportes para alojar la luz de cortesía.

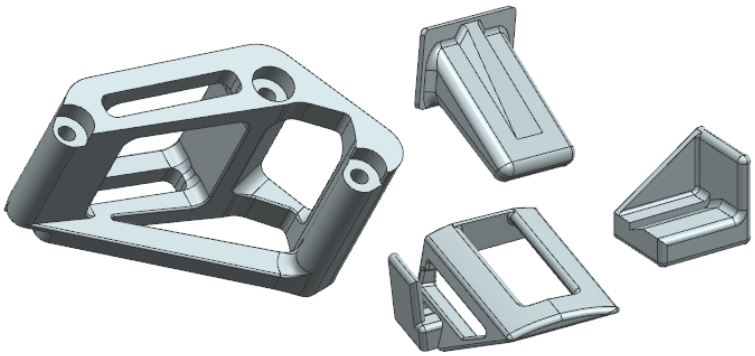


Figura 4.1.2 Soporte retrovisor y soportes luz de cortesía.

En la tapa de la caja se encuentra la cámara térmica y la cámara web con sus respectivas carcasas de impresión 3D. A la vez, se sitúan 4 luces leds con sus respectivas carcasas y un filtro de celofán transformando las luces blancas en 4 colores distintos; rojo, verde, azul y amarillo. Para la unión entre caja y tapa se han diseñado 2 bisagras de impresión 3D capaces de soportar el peso de la tapa y sus componentes, las cuales están sujetas mediante tornillos M6 allen y tuercas hexagonales M6.

En la parte superior de la caja se sitúan dos fuentes de alimentación, una a 5V para alimentar la Raspberry y otra a 12V para alimentar el retrovisor. También se sitúa la Raspberry Pi 4B+ con una tarjeta de expansión de 6 relés permitiendo controlar el retrovisor de 12V con la Raspberry de 5V.

Figura 4.1.3 Vista general prototipo[5]

[5] Imágenes del prototipo → pág 51-53, punto 8.12 del apéndice.

4.2 Electrónica

Como se comenta anteriormente, el cerebro del proyecto es una Raspberry Pi 4B+ la cual ha de interactuar con el retrovisor. Como el retrovisor y sus funciones funcionan a 12V y la Raspberry a 5V se ha introducido una tarjeta de expansión con 6 relés, de esta forma la corriente débil de salida Raspberry PI IO puede controlar el encendido y apagado de la corriente fuerte, controlando el calefactado, las luces y los motores del retrovisor.

Para la conexión de la cámara web se utiliza un cable de cinta flexible de 100 cm de longitud enfundado con cinta negra conectado directamente a la Raspberry. Lo mismo pasa con la cámara térmica que en vez de utilizar un cable flex, utiliza I^2C[17], un bus de comunicación serie síncrono , multimaestro, multiesclavo, conmutado por paquetes , de un solo extremo compuesto por 4 cables; corriente, massa, SDA y SCL.

En el apartado 8.12 del apéndice se pueden encontrar imágenes de los circuitos eléctricos[6]. A continuación un esquema de toda la conexión eléctrica del sistema.

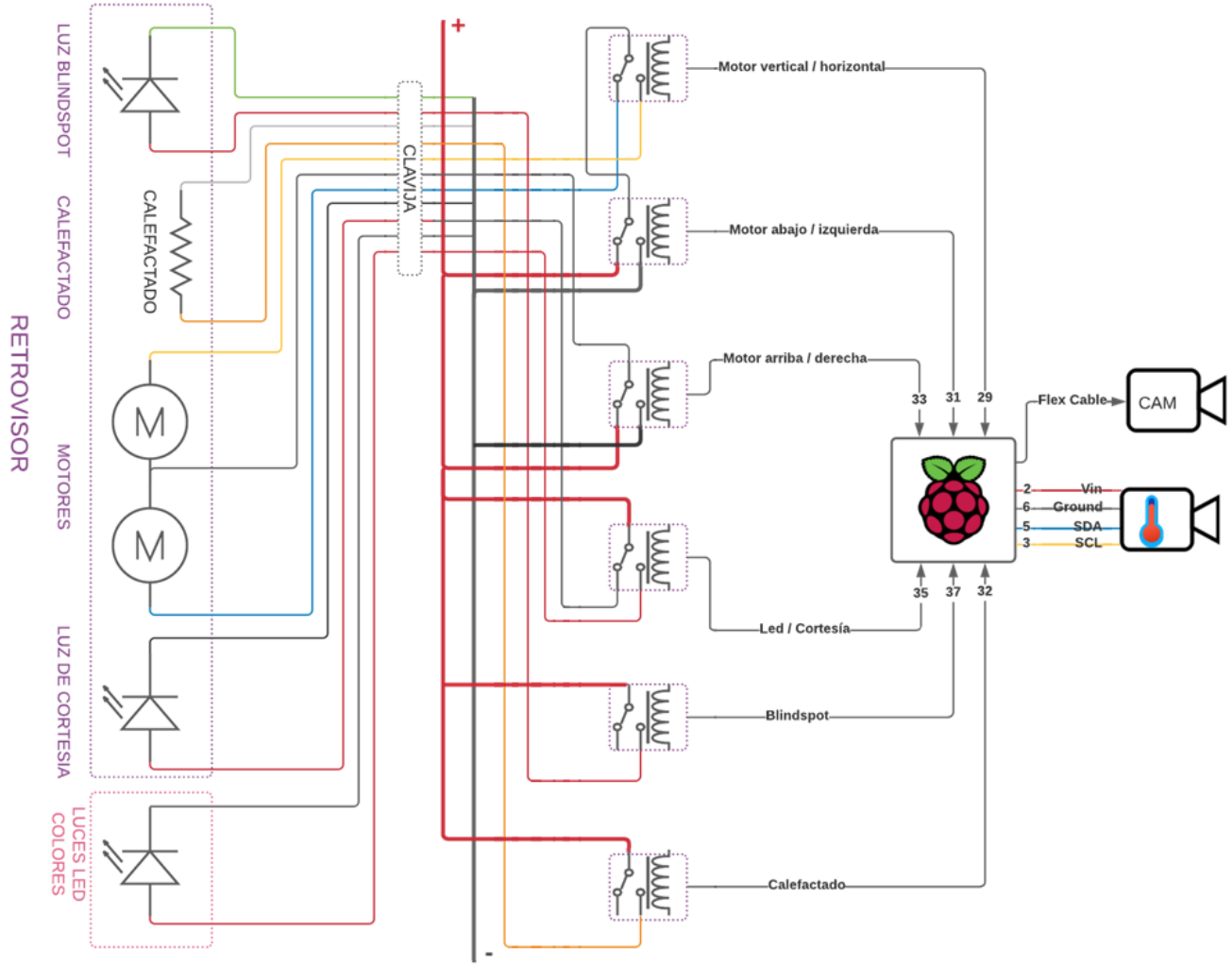


Figura 4.2.1 Esquema eléctrico

[6] Imágenes del circuito eléctrico → pág 54-55, punto 8.13 del apéndice.

4.3 Software de visión artificial

El software tiene el objetivo de verificar el correcto funcionamiento del retrovisor, comprobando el calefactado de las lunas, el correcto movimiento de los motores y sus dos luces, la de blindspot y la luz de cortesía. A continuación se muestra un diagrama UML [18] del proceso del código utilizado.

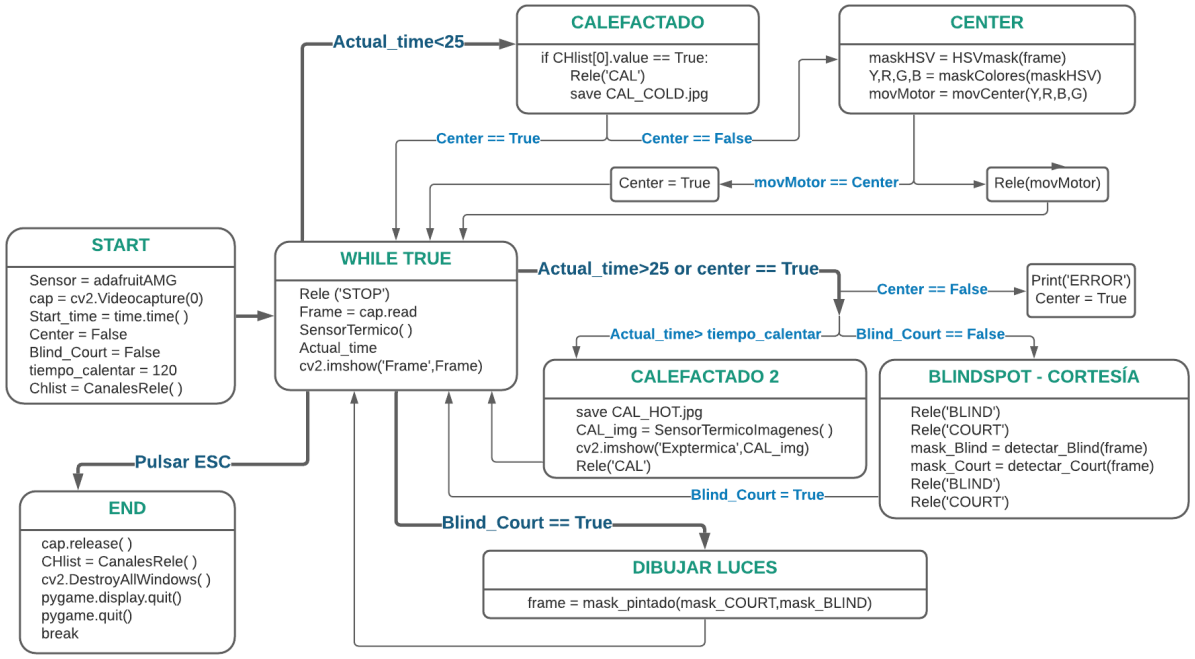


Figura 4.3.0.1 Diagrama UML del codigo

Como se puede apreciar en el diagrama el código realiza un START donde pone en marcha el sensor térmico, la cámara y la programación de los relés, los cuales son los encargados de manejar el retrovisor a la vez que determina parámetros básicos como temporizadores, contadores e importación de librerías. El siguiente paso es entrar en un bucle donde constantemente paramos los motores, obtenemos la imagen en tiempo real tanto de la cámara web como la cámara térmica y mostramos esas imágenes al operario.

En los primeros 25 segundos el programa procede a encender el calefactado y encontrar el centro del retrovisor (mediante el reflejo de las luces de colores posicionadas en la tapa de la caja). Si encuentra el centro del retrovisor en menos de 25 segundos verificamos el correcto funcionamiento de los motores, en caso contrario, saldrá un aviso para comunicar el error al operario.

Una vez han pasado los 25 segundos o se haya encontrado el centro, se procede a verificar el correcto funcionamiento de las luces mediante máscaras, si las encuentra, dibujara unos círculos en la imagen de la cámara web para mostrar al operario donde ha encontrado esas luces, evitando falsos positivos.

Cuando hayan pasado 120 segundos, el programa mostrará una nueva ventana donde se podrá apreciar una imagen del retrovisor sin calefactar junto con la imagen del retrovisor calefactado 120 segundos. También mostrará la diferencia de imágenes y la temperatura máxima alcanzada.

Finalmente para salir del programa, el operario puede pulsar ESC (acción que se puede realizar en cualquier momento). A continuación se procede a explicar cada parte del programa con más detalle, aunque el código[7] entero se encuentra en el punto 8.13 del apéndice.

[7] Código python completo → pág 56-61, punto 8.14 del apéndice.

4.3.1 Calibración de la cámara

Las cámaras estenopeicas económicas de hoy en día introducen mucha distorsión en las imágenes siendo las principales la distorsión radial y la distorsión tangencial. Debido a la distorsión radial, las líneas rectas aparecerán curvas. Su efecto es mayor a medida que nos alejamos del centro de la imagen. A continuación una imagen donde se puede apreciar los tipos de distorsión radial y como las líneas rectas están abultadas.

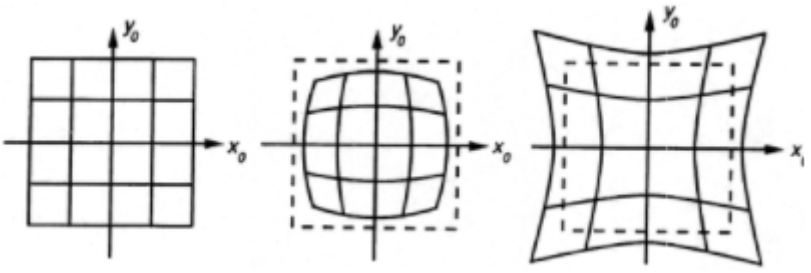


Figura 4.3.1.1 a) Imagen b) Distorsión de barril c) Distorsión de acerico

Para corregir estas distorsiones existe el proceso de calibración de imágenes, que trata de determinar los parámetros intrínsecos de las cámaras, tales como la distancia focal, el centro del plano sensor y la distorsión de la lente para poder posteriormente determinar la geometría de los objetos observados por la cámara. Las distorsiones se resuelven de la siguiente manera:

Distorsión radial: $X_{corregida} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$ $\quad Y_{corregida} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$

Distorsión tangencial: $X_{corregida} = x + [2p_1 xy + p_2(r^2 + 2x^2)$ $\quad Y_{corregida} = y + [p_1(r^2 + 2y^2) + 2p_2 xy)$

La forma de actuar consiste en tomar un conjunto de imágenes con la cámara permaneciendo invariante los parámetros intrínsecos de ésta, es decir, no se puede mover el enfoque de la óptica de la cámara ya que afectaría a la distancia focal. Tampoco se puede variar el diagrama, pues el modelo de distorsión se modificaría. A parámetros intrínsecos constantes se adquieren varias imágenes tomando fotos de un objeto patrón. El objeto patrón suele ser un tablero de ajedrez para facilitar la búsqueda de puntos significativos. Estos puntos significativos del espacio, son las esquinas de los cuadrados de ajedrez.

La idea consiste en localizar los parámetros intrínsecos a través de n-conjunto de ecuaciones que se establece entre los puntos del espacio, la geometría conocida (en este caso, el tamaño de la cuadrícula del ajedrez) y la proyección de estos puntos sobre el elemento sensor que son determinados con precisión subpixel, obteniendo la distancia focal f_x, f_y y los centros ópticos c_x, c_y también llamada matriz de cámara .

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Una vez fijado los parámetros intrínsecos de la cámara es posible determinar a qué distancia se encuentran los objetos respecto a la cámara. Al vector de traslación junto a la matriz de rotación que liga el punto del espacio con su proyección en el elemento sensor se le llaman parámetros extrínsecos, es decir, los parámetros extrínsecos corresponden a vectores de rotación y traslación que traducen las coordenadas de un punto 3D a un sistema de coordenadas.

Haciendo un breve resumen, para encontrar todos los parámetros mencionados en este punto, tenemos que proporcionar al sistema al menos 10 imágenes de muestra de un patrón (tablero de ajedrez) encontrado algunos puntos específicos (esquinas del tablero). De esta forma conoceremos sus coordenadas en el mundo real y sus coordenadas en la imagen obtenida. Con estos datos obtenemos los coeficientes de distorsión corrigiendo la imagen obtenida.

Se necesita tomar al menos 10 patrones de prueba para la calibración de la cámara, en este proyecto se ha utilizado un patrón de tablero de ajedrez[8] impreso en una hoja DIN A4. El primer paso es preparar las variables del código: el número de esquinas en el eje X (9) , el número de esquinas en el eje Y (6) y la matriz de los puntos físicos en 3D usando mgrid.

El siguiente paso es una definición que al ser llamada realiza el número deseado de capturas cuando el usuario apreta “enter”, guardando las imágenes con el nombre “calibrate” seguido del número de imagen (1,2,3..). De esta forma podemos asegurarnos de guardar imágenes que capturen bien el patrón.

La definición calibrate() se basa en las imágenes guardadas (es decir, que no se puede utilizar si previamente no se han capturado las imágenes). Este código busca de nuevo las esquinas del patrón en cada imagen y mediante “cv2.drawChessboardCorners()” las dibuja en cada imagen. La función “cv2.imshow” muestra estas imágenes con las esquinas encontradas y dibujadas para que el usuario pueda verificar su correcto funcionamiento.

Se usa la función “cv2.calibrateCamera()”, la cual, con todos los datos obtenidos anteriormente, calibra la cámara y nos devuelve “mtx”, que son los parámetros intrínsecos, y “dist”, que describe los parámetros de distorsión.

Por último se usa la función “cv2.undistort()” para anular la distorsión de las imágenes. A partir de este punto, cuando se precisa realizar un algoritmo de visión artificial nos aseguramos que con anterioridad hayamos utilizado “cv2.undistort()” con los parámetros de la cámara.

En la página 56 del apéndice se encuentra el código[9] usado para calibrar la cámara.

Figura 4.3.1.2 a) Esquinas del patrón encontradas

[8] Patrón - Tablero de ajedrez → pág 62, punto 8.15 del apéndice.

[9] Código python calibración → pág 63-64, punto 8.16 del apéndice.

4.3.2 Control del retrovisor

Como se ha podido observar en la figura 4.2.1, se utilizan 6 relés para controlar las acciones del retrovisor y para controlar el encendido y apagado de las luces leds de colores situadas en la tapa frontal de la caja. Seis pines de la Raspberry son los encargados de controlar cada relé, siendo el pin 29 (CH1) para el calefactado, el 31 (CH2) para la luz blindspot, el 33 (CH3) para la luz de cortesía y leds, el 35 (CH4) para la dirección izquierda/arriba del motor, el 37 (CH5) derecha abajo del motor y el 32 (CH6) para elegir motor vertical o horizontal. Para poder configurar cualquier pin de la Raspberry se ha utilizado la librería "board".

```
import board                                #Configuración pines Raspberry
```

Con la definición "CanalesRele( )" configuramos los 6 pines anteriormente comentados

```
def CanalesRele():
    CH1=digitalio.DigitalInOut(board.D5)  #Pin29
    CH2=digitalio.DigitalInOut(board.D6)  #Pin31
    CH3=digitalio.DigitalInOut(board.D13) #Pin33
    CH4=digitalio.DigitalInOut(board.D19) #Pin35
    CH5=digitalio.DigitalInOut(board.D26) #Pin37
    CH6=digitalio.DigitalInOut(board.D12) #Pin32
    CHlist=(CH1,CH2,CH3,CH4,CH5,CH6)      #Creamos lista Reles
    for ch in CHlist:                     #para cada rele
        ch.direction = digitalio.Direction.OUTPUT #Pines de salida
        ch.value=True                    #Rele estado normal (no activado)
    return CHlist                        #Devolvemos lista reles
```

Con la definición "Rele(acción)" se simplifica el control dando nombre a las diferentes opciones de control; calefactado, luz blindspot, luz cortesía, leds de colores, motor izquierda, motor derecha, motor arriba, motor abajo, parada de motores.

```
def Rele(accion):
    if accion == 'CAL':                  #CALEFACTADO
        CHlist[0].value= 1- CHlist[0].value #cambia estado calefactado
        if CHlist[0].value == False:     #Si se enciende avisa
            print('1   - Calefacción encendida (proceso calentamiento luna')
    elif accion == 'BLIND':              #BLINDSPOT
        CHlist[1].value= 1- CHlist[1].value #cambia estado blindspot
    elif accion == 'COURT':              #CORTESIA
        CHlist[2].value= False           #enciende court, apaga led
    elif accion == 'LED':                #LEDS COLORES
        CHlist[2].value= True            #enciende leds, apaga court
    elif accion == 'ML':                 #Motor left (izq)
        CHlist[3].value= False
        CHlist[4].value= True
        CHlist[5].value= True
    elif accion == 'MR':                #Motor right(derecha)
        CHlist[3].value= True
        CHlist[4].value= False
        CHlist[5].value= True
    elif accion == 'MU':                #Motor up (arriba)
        CHlist[3].value= False
        CHlist[4].value= True
        CHlist[5].value= False
    elif accion == 'MD':                #Motor down (abajo)
        CHlist[3].value= True
        CHlist[4].value= False
        CHlist[5].value= False
    elif accion == 'STOP':              #Motor parados
        CHlist[3].value= True
        CHlist[4].value= True
        CHlist[5].value= True
```

4.3.3 Luz de cortesía

Para la correcta detección de la luz de cortesía se utiliza la cámara web importando la librería "Opencv" e iniciándose con la variable "cap". El frame se lee constantemente dentro del bucle

```
import cv2                              #computer vision library
cap = cv2.VideoCapture(0)               #Iniciamos la camara numero 0
while (True):
    ret, frame = cap.read()             #Leemos frame camara
```

Se realiza un ROI ("Region of interest") del frame para evitar información innecesaria. La imagen se pasa a gris y se desenfoca con un filtro gaussiano de kernel 15,15. Gracias a un umbral convierte la imagen en negro excepto los pixeles que esten entre los valores 233-255 (es decir blanco). La función label etiqueta las regiones de la matriz (los blancos detectados) mientras se crea una máscara vacía con el mismo tamaño llena de ceros (totalmente negra).

```
roi_COURT=actual_frame[100:300,400:620]                 #Region de interes
grayCOURT = cv2.cvtColor(roi_COURT,cv2.COLOR_BGR2GRAY) #Img en gris
blurred = cv2.GaussianBlur(grayCOURT,(15,15),0)         #desenfocamos
ret,thresh_COURT = cv2.threshold(blurred, 235,255, cv2.THRESH_BINARY)
Etiqueta_COURT= measure.label(thresh_COURT, background=0) #etiquetamos
mask_COURT = np.zeros(thresh_COURT.shape, dtype="uint8") #mascara vacia
```

El siguiente paso es verificar si cada etiqueta de región detectada es mayor que 2500 pixeles, en caso afirmativo, añadir esa región a la máscara vacía. De esta forma nos aseguramos de solo detectar la luz de cortesía en pleno funcionamiento y no reflejos o falsos positivos. Si al final de la función la máscara está vacía, significa que no hemos encontrado la luz de cortesía por tanto avisamos al operario de un posible error.

```
for label in np.unique(Etiqueta_COURT): #Para cada figura de la imagen.
    if label == 0: #si detecta que es fondo (0) no hagas nada.
        continue
    labelMask = np.zeros(thresh_COURT.shape, dtype="uint8")
    labelMask[Etiqueta_COURT == label] = 255 #Totalmente blanco
    numPixels = cv2.countNonZero(labelMask) #Contamos el numero de pixels
    if numPixels > 2500: #Si es mayor que 2500 añadelo
        mask_COURT = cv2.add(mask_COURT, labelMask) #añadimos region
if mask_COURT.any() == False:
    print('4   -  <<ERROR>> Luz de cortesia no encontrada ')
else:
    print('4   - Luz Cortesia encontrada.')
```

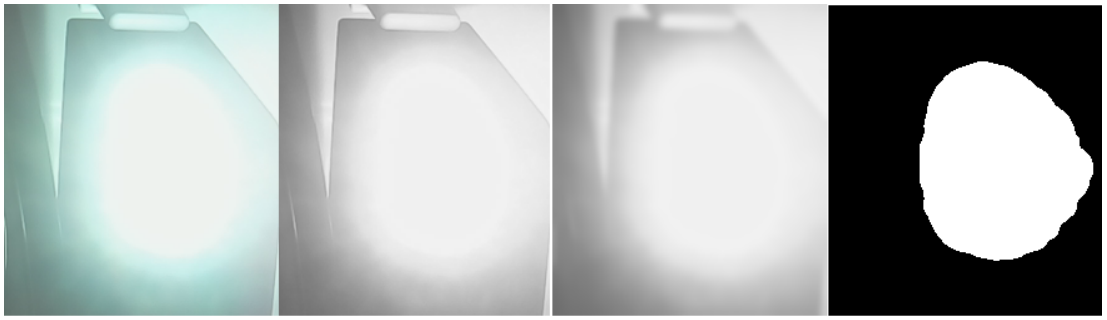


Figura 4.3.3.1 a) Imagen ROI b) Imagen en gris c) Imagen desenfocada d) Máscara cortesía

4.3.4 Leds del blindspot

Para la detección de los leds del blindspot se utiliza una metodología parecida a la de la luz de cortesía pero con la variación que detectamos únicamente las luces rojas. Una vez obtenido el frame de la cámara se realiza un ROI pero esta vez transformando la imagen a escala HSV [20] una alternativa de representación del modelo de color BGR (blue, green, red) [21].

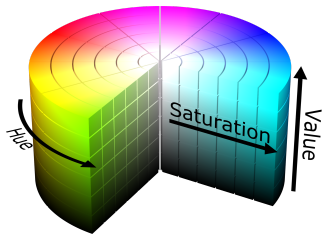


Figura 4.3.4.1 Modelo de color HSV

Estableciendo los rangos del color rojo se crea una máscara donde los píxeles dentro del rango serán de color negro mientras el fondo sera blanco. Aplicando transformaciones morfológicas con un kernel de 3x3 se consigue limpiar la imagen, creando una máscara más uniforme. Finalmente se invierte la mascara para que la zona de interes sea blanca y el fondo negro. Como en la detección de la luz de cortesía, si esta máscara está vacía informa al operario de un posible error.

```
def Detectar_BLIND(actual_frame):
    roi_BLIND = actual_frame[0:100,300:400]  #Region de interes
    hsv_BLIND = cv2.cvtColor(roi_BLIND, cv2.COLOR_BGR2HSV) #RGB to HSV
    color_bajos = np.array([0,0,25]) #Establecemos rango mín y máxi de HSV:
    color_altos = np.array([179,145,164])
    mask_BLIND = cv2.inRange(hsv_BLIND, color_bajos, color_altos) #Mascara
    kernel = np.ones((3,3), np.uint8)  #Creamos kernel y eliminamos el ruido:
    mask_BLIND = cv2.morphologyEx(mask_BLIND, cv2.MORPH_CLOSE, kernel)
    mask_BLIND = cv2.morphologyEx(mask_BLIND, cv2.MORPH_OPEN, kernel)
    mask_BLIND = 255 - mask_BLIND #invierto la mascara
    if mask_BLIND.any() == True:
        print('3   - Luz BlindSpot encontrada.')
    else:
        print('3   -  <<ERROR>> Luz Blindspot no encontrada ')
    return mask_BLIND
```

A continuación una muestra del proceso de detección de los leds blindspot:

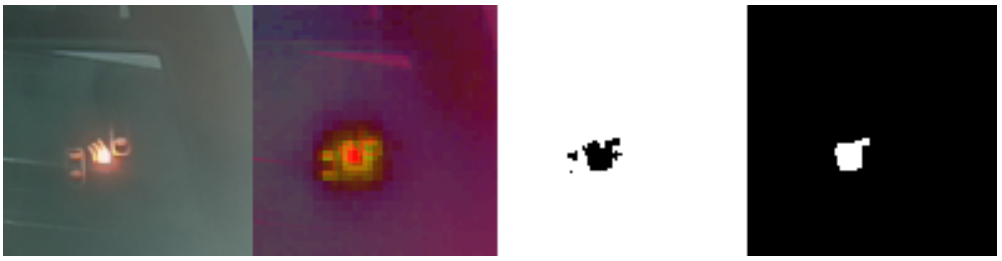


Figura 4.3.4.2 a) Imagen ROI b) Imagen HSV c) Máscara color rojo d) Máscara blindspot

4.3.5 Lunas / motores

Para verificar el correcto funcionamiento de los motores el programa intenta centrar correctamente la luna, esta función la realiza mediante el reflejo de las 4 luces leds en la tapa frontal de la caja. Las luces y la cámara están situadas de tal forma que cuando se reflejan en la luna forman un cuadrado en el centro del retrovisor, siendo este reflejo en el centro, el objeto del programa.

Figura 4.3.5.1 a) Imagen leds tapa b) Imagen reflejo retrovisor c) Imagen frame cámara

El primer paso es recortar la imagen en el centro de la luz (ROI), una vez recortada se pasa a escala de grises y mediante el algoritmo de Canny [22] se detecta las esquinas de la imagen (figura 4.3.5.2 c). Con la función "getStructuringElement" se crea un kernel de 8x8 con la forma elipse que se utiliza para dilatar las esquinas anteriormente encontradas. La imagen dilata se utiliza como máscara encima del frame previamente pasado a escala HSV. De esta forma el programa muestra en la imagen solo los colores encontrados y esconde cualquier otro objeto/forma (por ejemplo las cámaras).

```
def HSVmask(frame):
    frame = frame [187:463,137:393]                          #Recortamos imagen
    frame_gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #Imagen a gris
    frame_HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) #Imagen escala HSV
    Canny = cv2.Canny(frame_gris, 200,500)                   #Detectamos esquinas
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(8,8)) #kernel 8x8
    dilated = cv2.dilate(Canny,kernel)                       #Dilatamos las esquinas
    frame_HSVmask = cv2.bitwise_and(frame_HSV,frame_HSV, mask=dilated)#mascara
    return frame_HSVmask
```

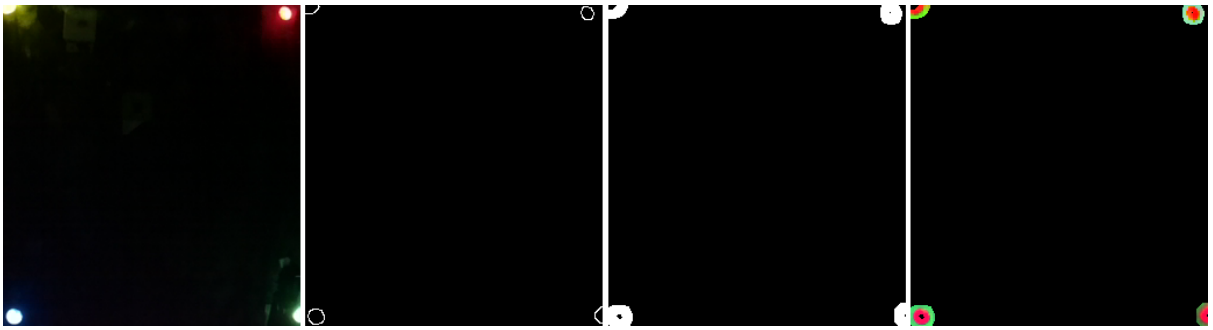


Figura 4.3.5.2 a) ROI b) Algoritmo Canny c) Esquinas dilatadas d) Máscara frame HSV

Varios vectores marcan los límites de los colores en la escala HSV, de esta manera se crean 4 mascaras detectando el color rojo, amarillo, verde y azul. De tal manera detectamos si dentro de nuestra zona de interés (ROI) está situada una luz o no.

```
def maskColores(frame):
    rojoBajo1 = np.array([0, 100, 20], np.uint8)     #Vector rojo 1
    rojoAlto1 = np.array([10, 255, 255], np.uint8)
    rojoBajo2 = np.array([175, 100, 20], np.uint8)   #Vector rojo 2
    rojoAlto2 = np.array([180, 255, 255], np.uint8)
    amarilloBajo = np.array([20, 100, 20], np.uint8)  #Vector amarillo
    amarilloAlto = np.array([32, 255, 255], np.uint8)
    verdeBajo = np.array([45, 100, 20], np.uint8)    #Vector verde
    verdeAlto = np.array([80, 255, 255], np.uint8)
    azulBajo = np.array([90, 100, 20], np.uint8)       #vector azul
    azulAlto = np.array([140, 255, 255], np.uint8)
    maskRed1 = cv2.inRange(frame, rojoBajo1, rojoAlto1)  #mascara rojo1
    maskRed2 = cv2.inRange(frame, rojoBajo2, rojoAlto2)  #mascara rojo2
    maskRed = cv2.add(maskRed1, maskRed2)                #mascara rojo
    maskYellow = cv2.inRange(frame, amarilloBajo, amarilloAlto) #masc amarilla
    maskGreen = cv2.inRange(frame, verdeBajo, verdeAlto) #mascara verde
    maskBlue = cv2.inRange(frame, azulBajo, azulAlto) #mascara azul
    return (maskYellow.any(),maskRed.any(),maskBlue.any(),maskGreen.any())
```

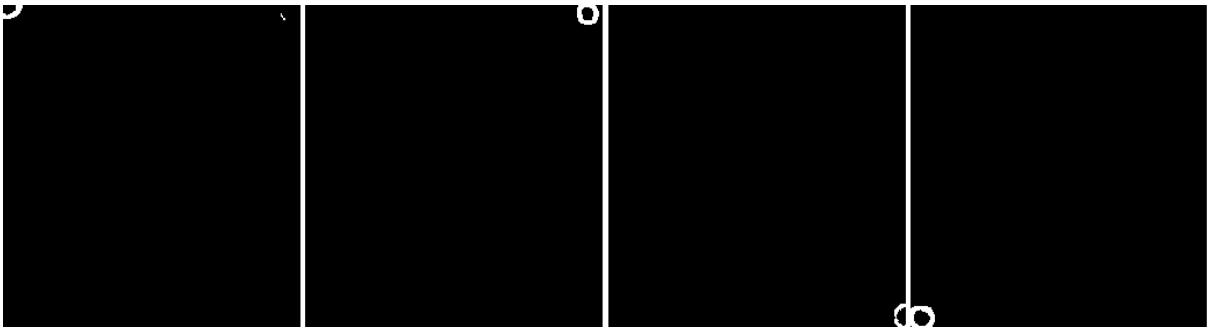


Figura 4.3.5.3 Diferentes mascaras de colores (Amarilla, roja, verde, azul)

Mediante la siguiente tabla de la verdad, dependiendo de los colores que encuentra dentro de la zona de interés, se crea una función capaz de determinar el siguiente movimiento del retrovisor con la finalidad de encontrar los 4 puntos de luces, centrando el retrovisor y confirmando el correcto funcionamiento de los motores. Si en 25 segundos es incapaz de visualizar las 4 luces y por tanto, incapaz de centrar el retrovisor, el programa avisa al operario que deberá revisar los motores (puede ser que esten montados a la inversa), la luna o el funcionamiento de los leds (rojo, verde,amarillo, azul).

	Error	⏪	⏩	⏫	⏪	⏪	Error	⏪	⏩	Error	⏩	⏩	⏬	⏪	⏩	OK
Y	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
R	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
B	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
G	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figura 4.3.5.4 Tabla direcciones del motor

A continuación la definición encargada de mover los motores según las luces visualizadas..

```
def movCenter(Y,R,B,G):
    if (Y and R and B and G) == True:  #Centro encontrado
        mov = ('CENTER')
    elif ((Y == G == False) and (R == B == True)) or ((Y == G == True) and
        mov = ('Error')
    elif not (Y or R or G or B):  #Error, combinacion imposible
        mov = ('Error')
    elif (Y == R == True) and (B == G == False): #Motor down (abajo)
        mov = ('MD')
    elif  (Y == R == False) and (B == G == True): #Motor up (arriba)
        mov = ('MU')
    elif (Y==R==B== False) and (G == True): #motor left (izquierda)
        mov = ('ML')
    elif (Y==R==B== True) and (G == False): #motor right (derecha)
        mov = ('MR')
    elif R == True:   #motor left (izquierda)
        mov = ('ML')
    else:             #motor right( derecha)
        mov = ('MR')
    return mov
```

Cuando las 4 luces se encuentran dentro de la imagen un total de 4 veces o más, los motores se paran y se avisa al operario de que el centro de la luna se ha encontrado (el retrovisor está centrado) y por tanto los motores funcionan correctamente.

```
    if Center == False:
        maskHSV = HSVmask(frame)       #definicion crear mask colores
        Y,R,B,G = maskColores(maskHSV) #Que colores detecta
        movMotor = movCenter(Y,R,B,G)  #movimientos motor
        cv2.rectangle(frame,(137,187),(393,463),(255,255,255),1) #cuadrado
        cv2.imshow('Frame Camara',frame) #muestra imagen
        if movMotor == 'CENTER':
            countCenter+=1
            if countCenter>3: #cuando encuentres 3 veces el centro
                Center = True
                Centertime = time.time()
                print('2   - Centro luna encontrado (Luna correctamente mo
        elif movMotor == 'Error': #si encuentras error 30 veces:
            countErrorColores+=1
            if countErrorColores > 30:
                print('2   -  <<ERROR>> No se ha podido encontrar centro -
                Center = True
                Centertime = time.time()
        else:
            Rele(movMotor) # mueve los motores dirrecciones mandadas.
```

4.3.6 Expansión térmica

Para la detección de la expansión térmica se utiliza el sensor de cámara térmica adafruit AMG con comunicación I2C mientras que para su visualización utilizamos pygame [23], un conjunto de módulos del lenguaje Python que permite la creación de videojuegos. El primer paso es importar las librerías para utilizar tanto el sensor como el módulo pygame, así como librerías suplementarias para realizar cálculos.

```
import adafruit_amg88xx   #Sensor AMG8833 termico
with contextlib.redirect_stdout(None):
    import pygame   #Modulos de lenguaje creacion videojuegos (visual thermal)
import os            #acceder funcionalidades del sistema operativo
import contextlib    #Utilidades comunes involucradas con with
import math         #funciones matematicas
import time         #funciones de tiempo
import numpy as np  #paquete para informatica cientifica
from scipy.interpolate import griddata   #interpolar datos dd no estructurados
from colour import Color  #convierte y manipula representacion colores
import adafruit_amg88xx   #Sensor AMG8833 termico
from imutils import contours #generadores de contorno
import imutils          #libreria procesamiento de imagenes
from skimage import measure   #enviar metricas de aplicacioens usando UDP
```

Una vez importadas las librerías se define el canal I2C, se inicia los módulos pygame y el sensor adafruit AMG. El programa también define el alto y ancho de la pantalla en pixels para crear el display donde visualizar la información térmica.

```
i2c_bus = busio.I2C(board.SCL, board.SDA) #Definimos bus I2C
os.putenv("SDL_FBDEV", "/dev/fb1")
pygame.init() # Inicializamos todos los modulos importados pygame
sensor = adafruit_amg88xx.AMG88XX(i2c_bus) #Inicializamos sensor

displayPixelWidth = width / 30 #ancho pantalla pixels
displayPixelHeight = height / 30 #alto pantalla pixels
lcd = pygame.display.set_mode((width, height)) #creamos display
lcd.fill((255,0,0))

pygame.display.update() #actualizo display
pygame.mouse.set_visible(False) # desaparezca el cursor por la pantalla
lcd.fill((0, 0, 0))
pygame.display.update() #actualizo las partes de la pantalla
```

El sensor proporciona la información de la temperatura con una matriz de 8x8 que habrá que transformar a color en una imagen. Para ello primero se define el rango de colores, la temperatura mínima y máxima que queremos visualizar, así como dos matrices de 8x8 con longitud de paso 32 número complejo.

```
#Definimos mintemp(azul),maxtemp(rojo) y cuantos valore tendremos.
MINTEMP,MAXTEMP,COLORDEPTH,blue = 20.0,28.0,1024,Color("indigo")
#escogemos el rango colores de azul a rojo
colors = list(blue.range_to(Color("red"), COLORDEPTH))
#creamos un array de colores
colors = [(int(c.red * 255), int(c.green * 255), int(c.blue * 255)) for c in colors]

points = [(math.floor(ix/8),(ix%8)) for ix in range(0,64)] # puntos matriz 8x8
#Creamos 2 matrices 8x8 con longitud de paso 32 numero complejo
grid_x, grid_y = np.mgrid[0:7:32j, 0:7:32j]
height,width = 240,240 #Creamos cuadrado ya que matriz es 8x8
```

Para la transformación de lectura del sensor a una imagen ilustrativa de colores se utiliza la definición SensorTermico(). De esta forma se actualiza la pantalla en cada bucle. En esa definición se lee cada pixel detectado por el sensor y se añade a una lista previamente creada. Antes de pintar cada pixel de su color correspondiente, hacemos una interpolación bicúbica obteniendo una superficie de color mas suave. Finalmente se actualiza la pantalla para mostrar los datos en tiempo real.

```
def SensorTermico():
    pixels = [] #lista pixels vacia
    for row in sensor.pixels: #para cada pixel detectado del sensor:
        pixels = pixels + row #creamos lista de los pixels
    pixels = [map_value(p, MINTEMP, MAXTEMP, 0, COLORDEPTH - 1) for p in pixels]
    # Hacemos interpolacion bicubica
    bicubic = griddata(points, pixels, (grid_x, grid_y), method="cubic")  # perform
    for ix, row in enumerate(bicubic): #pintamos todo
        for jx, pixel in enumerate(row):
            pygame.draw.rect(lcd, colors[constrain(int(pixel), 0, COLORDEPTH - 1)],
                             (displayPixelHeight *
                              ix,displayPixelWidth *
                              jx,displayPixelHeight,
                              displayPixelWidth,),)
    pygame.display.update()
```

Para mostrar al usuario la expansión térmica se toma una captura al principio del programa (segundo 0) y otra al final (segundo 120) y, mediante una definición, estas imágenes se unen lateralmente, se anota el porcentaje de diferencia, la temperatura máxima alcanzada y la imagen resultante de restar una con otra (es decir, la diferencia visual). De esta forma se comprueba la expansión térmica.

```
def SensorTermicoImagenes():
    CAL_cold= cv2.imread('CAL_COLD.jpeg')
    CAL_hot= cv2.imread('CAL_HOT.jpeg')
    res = cv2.absdiff(CAL_hot,CAL_cold)
    res = res.astype(np.uint8)
    percentage = (np.count_nonzero(res) * 100)/ res.size
    difference = cv2.subtract (CAL_cold,CAL_hot)
    cv2.putText(difference,"DIFF",(10,230),cv2.FONT_ITALIC,0.7,(256,256,256),2)
    cv2.putText(difference,"Temp max:"+str(np.amax(sensor.pixels)),(100,230),
                cv2.FONT_ITALIC,0.3,(256,256,256),1)
    cv2.putText(CAL_cold,"COLD",(10,230),cv2.FONT_ITALIC,0.7,(256,256,256),2)
    cv2.putText(CAL_hot,"HOT",(10,230),cv2.FONT_ITALIC,0.7,(256,256,256),2)
    Image_cal = np.concatenate((CAL_cold,CAL_hot), axis=1)
    Image_cal = np.concatenate((Image_cal,difference), axis=1)
    print('5   - Expansión térmica en 120 segundos.')
    return (Image_cal)
```

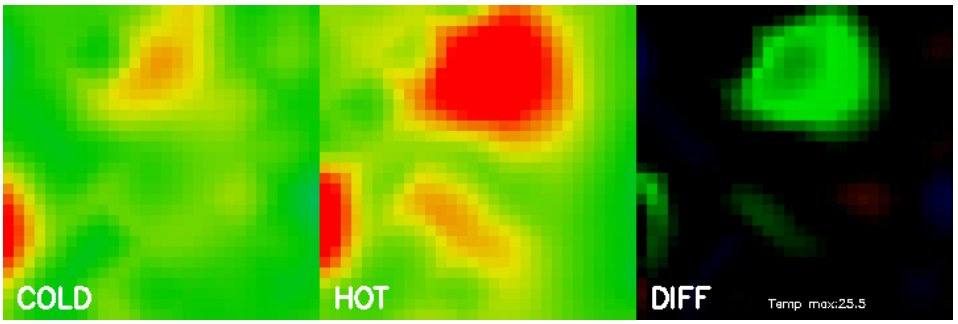


Figura 4.3.6.1 Imagen calefactado retrovisor

Capítulo 5: Pruebas y resultados

Para la correcta segmentación de las pruebas se han realizado varios códigos con el objetivo de comprobar diferentes zonas de manera individual así como los componentes que conforman el control de calidad. De esta forma se puede y se ha podido comprobar el correcto funcionamiento de los componentes y de los algoritmos usados de forma individual, siendo el resultado satisfactorio. Hay que remarcar que el usuario tiene a su disposición estos códigos para su uso. A continuación un listado de los diferentes test individuales.

- Test cámara web
- Test cámara térmica
- Test de los relés
- Test para encontrar correctos parámetros HSV
- Test detección luces
- Test detección luces de colores
- Test detección centro de la luna

En este capítulo se mostrarán todas las pruebas realizadas y los resultados finales obtenidos en cada una de las partes del proceso.

En el proceso de calibración de la cámara, se ha utilizado un tablero de ajedrez impreso en una hoja DINA4 con 10 cuadrados horizontales y 7 verticales. Utilizando el algoritmo del código escrito ha corregido correctamente la distorsión de la cámara siendo satisfactoria la calibración.

Figura 5.1 a) Patrón b)Imagen calibrada c) imagen calibrada y recortada

En la detección de luces se ha utilizado el retrovisor Diagonal 375-2 que posee luz de blindspot y luz de cortesía. Se han realizado varias pruebas, con las dos luces, con solo una luz, sin luz, y en todas las pruebas realizadas se ha detectado las luces y su posicionamiento correctamente.

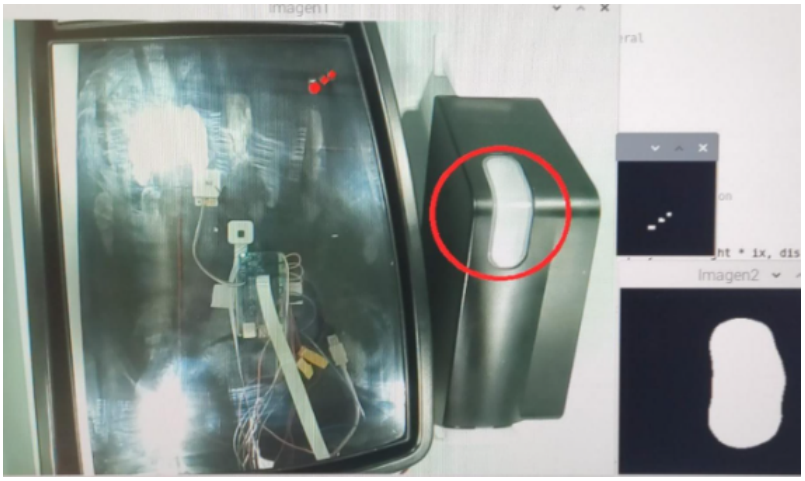


Figura 5.2 Luces encontradas

La expansión térmica, es decir, el sensor adafruit AMG8833 y el algoritmo utilizado para visualizar las temperaturas con colores, se ha comprobado en el retrovisor y su calefactado como con otras entidades, siendo estas: personas humanas, manos, ordenadores, luces del estudio, etc. Aunque en algunos casos para visualizar correctamente la matriz de colores se ha tenido que cambiar los valores mínimos y máximos de temperatura, en todos los casos ha resultado ser intuitivo. Es verdad que en el proyecto el tiempo de calentamiento de la luna térmica es de 120 segundos, siendo suficiente pero no su tiempo óptimo, ya que cuanto mayor sea el tiempo de calentamiento, mayores son las diferencias de temperaturas así como más fácil es ver la dirección y los puntos calientes de la expansión térmica. Aun así, con 120 segundos es suficiente para detectar su funcionamiento y es un tiempo correcto para comprobar un retrovisor.

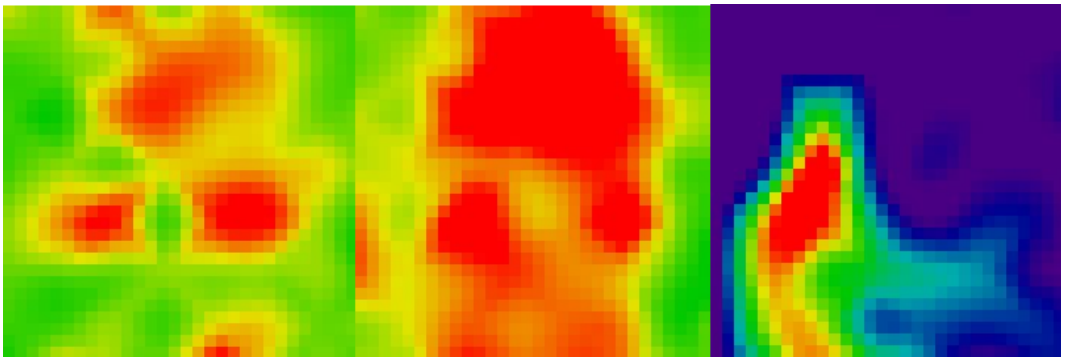


Figura 5.3 a) Imagen térmica 1 minuto b) Imagen térmica 5 minutos c) Imagen térmica humano

Las pruebas para comprobar el correcto funcionamiento del motor se han realizado tanto con el motor correcto, como con el motor del revés (direcciones contrarias), siendo imposible alcanzar el centro y por tanto avisando al usuario que los motores se han de revisar. Estas comprobaciones también se han realizado partiendo de todas las posiciones posibles que permite la luna, es decir, iniciando el programa con la luna totalmente a la izquierda, o totalmente inclinada hacia arriba, etc.
Para comprobar la respuesta del programa y si avisa cuando hay fallos de luz, se han falseado los leds de colores, siendo el programa incapaz de encontrar el centro y avisando nuevamente de comprobar el motor o las luces.

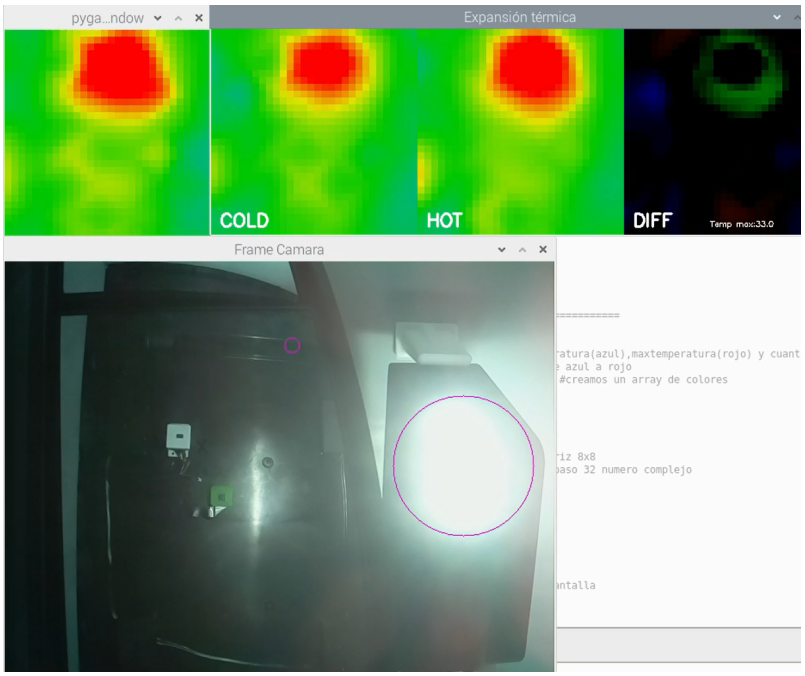


Figura 5.3 Imagen general control de calidad

Capítulo 6: Conclusiones y desarrollos futuros

6.1 Conclusiones

El objetivo principal de este proyecto ha sido diseñar y desarrollar un software basado en la visión artificial de bajo coste, capaz de realizar eficazmente el control de calidad de un retrovisor. El control se centra principalmente en revisar las funciones de calefactado de las lunas, los ángulos de movimiento del motor y las luces disponibles en el dispositivo, permitiendo así verificar de forma rápida y fiable el correcto funcionamiento y acabado del retrovisor.

Tras la finalización del proyecto se puede concluir que los objetivos propuestos han sido cumplidos por completo con un alto grado de satisfacción, siendo modificado únicamente el objetivo de comprobar los motores de las lunas. En un inicio se había propuesto verificar los grados de movimiento de las lunas, verificando así, el correcto funcionamiento de los motores y sus topes, sin embargo, durante el transcurso del proyecto se ha detectado que verificar únicamente sus movimientos era una solución más rápida y eficaz (ya que si una luna del retrovisor no gira, será comprobada durante el montaje del propio retrovisor). También se ha aprendido que durante el proceso de montaje del retrovisor existen muchas tolerancias que dependen del propio operario, variando la posición exacta de la luna y por tanto haciendo más difícil calcular con eficacia los ángulos y grados de movimiento. Comprobando si el retrovisor es capaz de llegar al centro, se cumple el objetivo interno de la empresa, de verificar que ningún retrovisor llegue al cliente con los motores cortocircuitados.

En cuanto al coste del proyecto considero que ha sido satisfactorio, teniendo en cuenta que el precio de todos los productos no amortizados es de 265,17€, una cifra inferior al coste de un lote del producto y una cifra muy asequible para cualquier tipo de empresa. En cuanto al coste del personal, se ha estimado en un valor de 2400€, un importe reducido si se compara con el coste de realizar este tipo de proyectos en empresas externas. También es importante tener presente que todos los conocimientos adquiridos en la realización de este proyecto contribuirán a disminuir el tiempo destinado a futuros proyectos puesto que no será necesario la etapa de formación, o si la hay, será menor, reduciendo así el coste medio por proyecto. Con un precio final de 2665,17€ se confirma que el proyecto ha sido asequible y de bajo coste, cumpliendo de esta manera con uno de los objetivos del trabajo.

Hay que tener en cuenta que también es un proyecto abierto, el cual se podría seguir modificando y ampliando para poder comprobar varios productos con formas y utilidades diferentes, es decir, creando diferentes soportes en la caja, se puede colocar varios objetos diferentes, por ejemplo pilotos de autobuses, esqueletos de retrovisores, etc. Al ser la Raspberry Pi un ordenador con escritorio, el usuario únicamente deberá encender el programa adecuado para comprobar el objeto, pudiendo tener decenas de programas y con el mismo sistema (cámara web y cámara térmica) hacer diferentes controles de calidad.

Otro objetivo cumplido ha sido el de adquirir conocimientos y bases sobre la visión artificial, cumpliendo con las fechas especificadas en el gantt-chart y despertando en mí una curiosidad voraz sobre el mundo de la visión por computadora, hasta el punto de buscar postgrados relacionados.

Finalmente se ha logrado desarrollar un prototipo e implementar el sistema en la planta de Arcol S.A, estableciéndose como un control de calidad eficaz en el departamento de calidad y en los puestos de trabajo, siendo accesible para todos los operarios. También se ha convertido en una puerta abierta hacia el control de calidad mediante visión artificial, con la posibilidad de crear futuros proyectos similares.

Para concluir, me gustaría citar una frase de Benjamin Franklin la cual define todo el esfuerzo y recompensas obtenidas: *“Cuéntame y olvido. Enséñame y recuerdo. Involúcrame y aprendo*”.

6.2 Desarrollos futuros

A continuación se detallan aspectos a mejorar en posibles versiones futuras, tanto del código, del espacio de trabajo o del proyecto en general.

Con el código actual, el operario ha de seleccionar el producto al que desea hacer un control de calidad. Unas futuras líneas podrían ser el desarrollo de un algoritmo capaz de detectar el objeto que tiene enfrente para posteriormente realizarle un control de calidad, es decir, que el usuario no tenga que determinar que programa hay que iniciar. Hay varios algoritmos con los que detectar un objeto o un patrón, siendo de fácil acceso y realización los clasificadores en cascada basados en funciones Haar[10]. Un método basado en aprendizaje automático a partir de muchas imágenes positivas y negativas que identifica objetos en una imagen y un video, de esta forma el usuario solo tendría que iniciar un programa, y el propio programa según qué objeto detecta, iniciar un algoritmo u otro.

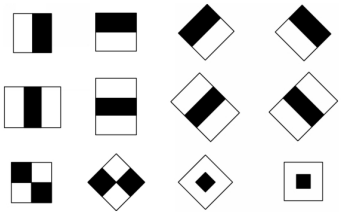


Figura 6.2.1 Clasificadores Haar.

Hablando del código, también sería muy interesante realizar algunas líneas nuevas para realizar un control de las funciones del prototipo, es decir, verificar el correcto funcionamiento de la cámara térmica, las luces leds así como de los relés que controlan las acciones del retrovisor. Actualmente si un relé no funciona correctamente, se podría catalogar el problema en el retrovisor. Otro aspecto de control debería ser el cierre obligatorio de la caja para iniciar el programa, detectando este cierre con un sensor de final de carrera por ejemplo. Actualmente puedes correr el control de calidad con la puerta abierta y eso no tiene ningún sentido, ya que no está ni correctamente enfocado ni en el sitio correcto, pudiendo entrar luz en el ensayo y creando falsos positivos.

Como se ha comentado anteriormente, este proyecto es un proyecto abierto, es decir, que se puede explotar la cantidad de objetos a los que realizar un control de calidad. Un desarrollo futuro sería poder comparar diferentes productos con el mismo espacio de trabajo, es decir, la caja de plástico Esmelux. De esta forma, con un solo prototipo podrías comprobar diferentes productos según la producción de los mismos, ahorrando espacio de almacenaje de utillajes y optimizando el actual.

Otro aspecto a mejorar sería la interfaz con el usuario (UI), es decir, aquella parte visual con la que los usuarios interaccionan con el sistema. Actualmente no existe ninguna interfaz de usuario más allá de la que ofrece el propio sistema operativo de Raspberry, siendo casi inexistente. Una gran mejora sería diseñar una interfaz donde el resultado del control de calidad se visualice fácilmente y de forma intuitiva, ya sea por indicadores luminosos, acústicos o visuales.

Figura 6.2.2 Interfaz de usuario.

[10] Clasificadores en cascada basados en funciones Haar → pág 65-66, punto 8.17 del apéndice.

Capítulo 7: Referencias

[1] ¿Qué es la Cuarta Revolución Industrial? (2018, 10 abril). Blog de Salesforce. https://www.salesforce.com/mx/blog/2018/4/Que-es-la-Cuarta-Revolucion-Industrial.html

[2] Industria 4.0. (2021, 24 enero). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Industria_4.0

[3] Arcol – Fabricante de retrovisores y accesorios para autobúses, autocares.. (2021). https://arcol.es/.

[4] Colaboradores de Wikipedia. (2020, 13 octubre). Diagrama de Gantt. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Diagrama_de_Gantt

[5] Carlos Platero Dueñas. (2009). Introducción a la Visión Artificial. http://www.ieef.upm.es/webantigua/spain/Asignaturas/Robotica/ApuntesVA/cap1IntroVA.pdf

[6] Espinosa, J. (2009). Tema 6 - Teoría ondulatoria de la luz [Diapositivas]. Repositorio Institucional de la Universidad de Alicante. https://rua.ua.es/dspace/bitstream/10045/16578/1/6.%20Introducci%C3%B3n%20a%20la%20teor%C3%ADa%20ondulatoria%20de%20la%20luz.pdf

[7] López Tomás, R. (s. f.). Estándars de vídeo y formato entrelazado - Infaimon. Infaimon. https://www.infaimon.com/enciclopedia-de-la-vision/estandars-de-video-y-formato-entrelazado/

[8] Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2003). Digital Image Processing Using Matlab. Prentice Hall.

[9] Fernandez, A. (2021, 5 febrero). Aprende Visión Artificial con OpenCV y YOLO. ▷ Cursos de Programación de 0 a Experto © Garantizados. https://unipython.com/cursos/aprende-vision-artificial-con-opencv/

[10] Raspberry Pi OS. (2021, 16 febrero). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Raspberry_Pi_OS

[11] Python. (2021, 14 febrero). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Python

[12] OpenCV. (2021, 10 mayo). OpenCV. https://opencv.org

[13] Luis, E. R. (2018, 18 septiembre). De cero a maker: todo lo necesario para empezar con Raspberry Pi. Xataka. https://www.xataka.com/makers/cero-maker-todo-necesario-para-empezar-raspberry-pi

[14] The Raspberry Pi Foundation. (2021). Buy a Camera Module V2 –. Raspberry Pi. https://www.raspberrypi.org/products/camera-module-v2/

[15] Industries, A. (2021, 3 marzo). Adafruit AMG8833 IR Thermal Camera Breakout. Adafruit. https://www.adafruit.com/product/3538

[16] 3.5.4 Tabla de amortización simplificada - Agencia Tributaria. (s. f.). https://www.agenciatributaria.es/. Recuperado 17 de febrero de 2021, de https://www.agenciatributaria.es/AEAT.internet/Inicio/Ayuda/Manuales__Folletos_y_Videos/Manuales_practicos/_Ayuda_Folleto_Actividades_economicas/3__Impuesto_sobre_la_Renta_de_las_Personas_Fisicas/3_5_Estimacion_directa_simplificada/3_5_4__Tabla_de_amortizacion_simplificada/3_5_4__Tabla_de_amortizacion_simplificada.html

[17] NX. (2021). Siemens Digital Industries Software. https://www.plm.automation.siemens.com/global/es/products/nx/

[18] colaboradores de Wikipedia. (2021, 15 febrero). I2C. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/I%C2%B2C

[19] Colaboradores de Wikipedia. (2021b, abril 22). Lenguaje unificado de modelado. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado

[20] colaboradores de Wikipedia. (2021, 5 mayo). Modelo de color HSV. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Modelo_de_color_HSV

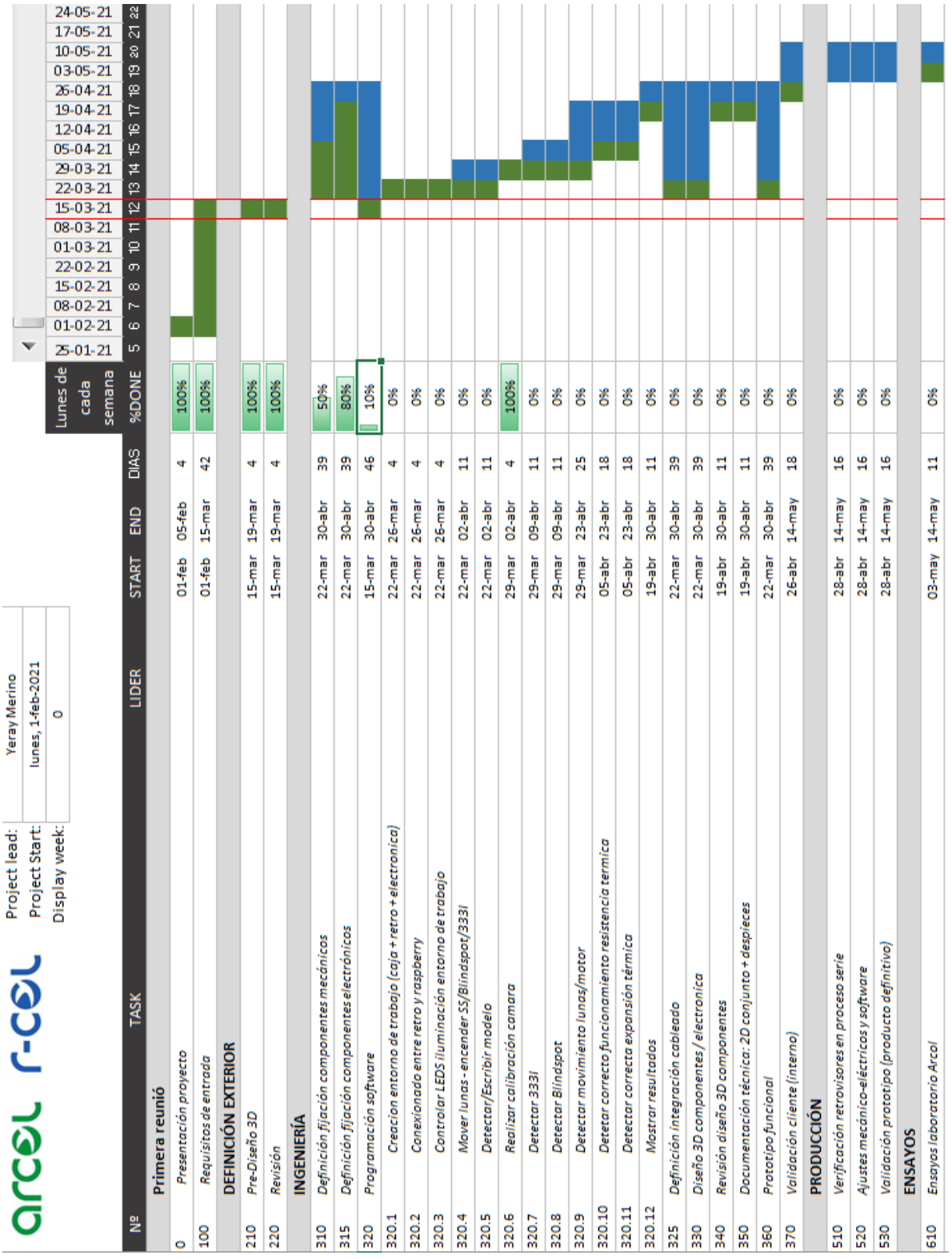
[21] A. (2020, 11 febrero). RGB en OpenCV - Python » omes-va.com. OMES. https://omes-va.com/rgb/

[22] Sosa-Costa, A. (2019, 30 agosto). Algoritmo de Canny. ▷ Cursos de Programación de 0 a Experto © Garantizados. https://unipython.com/algoritmo-de-canny/

[23] Sum, P. E. (2017, 26 diciembre). ¿Qué es PyGame? - Curso de programación de videojuegos con PyGame. https://www.programoergosum.com. https://www.programoergosum.com/cursos-online/raspberry-pi/246-videojuegos-en-python-con-pygame/que-es-pygame

Capítulo 8: Apéndice

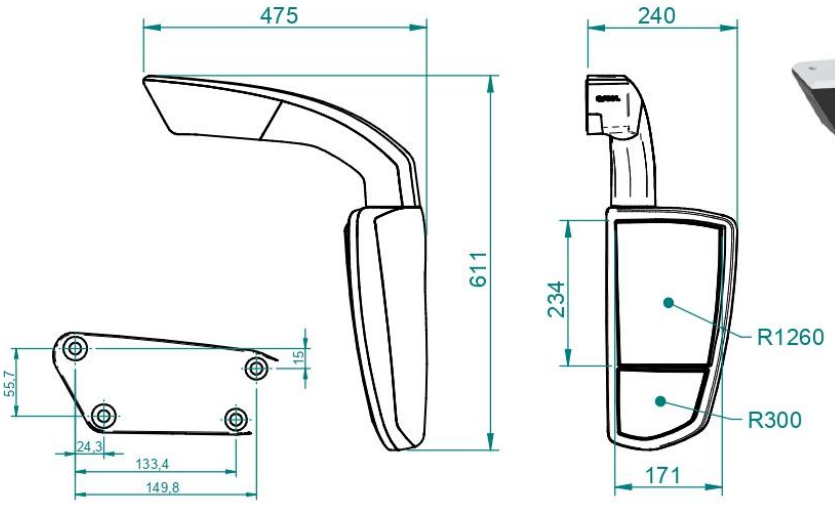
8.1 Gantt Chart



Nº	TASK	LIDER	START	END	DIAS	%DONE
	Primera reunió					
0	*Presentación proyecto*		01-feb	05-feb	4	100%
100	*Requisitos de entrada*		01-feb	15-mar	42	100%
	DEFINICIÓN EXTERIOR					
210	*Pre-Diseño 3D*		15-mar	19-mar	4	100%
220	*Revisión*		15-mar	19-mar	4	100%
	INGENIERÍA					
310	*Definición fijación componentes mecánicos*		22-mar	30-abr	39	50%
315	*Definición fijación componentes electrónicos*		22-mar	30-abr	39	80%
320	*Programación software*		15-mar	30-abr	46	10%
320.1	*Creacion entorno de trabajo (caja + retro + electronica)*		22-mar	26-mar	4	0%
320.2	*Conexionado entre retro y raspberry*		22-mar	26-mar	4	0%
320.3	*Controlar LEDS iluminación entorno de trabajo*		22-mar	26-mar	4	0%
320.4	*Mover lunas - encender SS/Blindspot/333i*		22-mar	02-abr	11	0%
320.5	*Detectar/Escribir modelo*		22-mar	02-abr	11	0%
320.6	*Realizar calibración camara*		29-mar	02-abr	4	100%
320.7	*Detectar 333i*		29-mar	09-abr	11	0%
320.8	*Detectar Blindspot*		29-mar	09-abr	11	0%
320.9	*Detectar movimiento lunas/motor*		29-mar	23-abr	25	0%
320.10	*Detetar correcto funcionamiento resistencia termica*		05-abr	23-abr	18	0%
320.11	*Detectar correcta expansión térmica*		05-abr	23-abr	18	0%
320.12	*Mostrar resultados*		19-abr	30-abr	11	0%
325	*Definición integración cableado*		22-mar	30-abr	39	0%
330	*Diseño 3D componentes/ electronica*		22-mar	30-abr	39	0%
340	*Revisión diseño 3D componentes*		19-abr	30-abr	11	0%
350	*Documentación técnica: 2D conjunto + despieces*		19-abr	30-abr	11	0%
360	*Prototipo funcional*		22-mar	30-abr	39	0%
370	*Validación cliente (interno)*		26-abr	14-may	18	0%
	PRODUCCIÓN					
510	*Verificación retrovisores en proceso serie*		28-abr	14-may	16	0%
520	*Ajustes mecánico-eléctricos y software*		28-abr	14-may	16	0%
530	*Validación prototipo (producto definitivo)*		28-abr	14-may	16	0%
	ENSAYOS					
610	*Ensayos laboratorio Arcol*		03-may	14-may	11	0%

8.2 Especificaciones retrovisor 375-2 diagonal

RVM 375-2 Diagonal / *Retrovisor 375-2 Diagonal*



Fixing points / Puntos de fijación

Specifications / *Especificaciones*

Description *Descripción*	External wing mirror for Motorhome, Mini and Midibus *Retrovisor para Autocaravanas, Mini o Midibús*
General features *Características generales*	Fields of view: II and IV Category Manual set up, optional heated glasses* and electrical set up** Optional camera, courtesy light (CL), blind spot detection and memory settings *Campos de visión: II y IV Categoría* *Actuador manual, opcional calefactado* y accionamiento eléctrico*** *Opcional cámara, luz de cortesía (LC), detección de ángulo muerto y posicionamiento automático*
Color	Black, optional Bicolor *Negro, opcional Bicolor*
Paint specifications *Especificaciones de pintado*	Clean the surface to paint. Dry recommended, not more than 40º *Limpiar la superficie a pintar. Recomendado en seco no más de 40º*
Maintenance / Cleaning *Mantenimiento / Limpieza*	Use water and neutral soap to wash the mirror *Use agua y jabón neutro para lavar el retrovisor*
Voltage *Voltaje*	12V or 24V
Mirror heater rated power at 12V / 24V* *Potencia nominal calefactado a 12V / 24V**	II Category 20W ±5%, IV Category 14W ±5% / II Category 30W ±5%, IV Category 14W ±5%
DC Motor nominal current** *Intensidad nominal motor DC***	80mA ±5%
DC Motor starting current** *Intensidad de arranque motor DC***	150mA ±5%
Operating temperature range *Rango operacional de temperaturas*	Between -30ºC and 70ºC *Entre -30ºC y 70ºC*
Unit Weight *Peso Unidad*	4 Kg Aprox.
Product Warranty Period *Periodo de Garantía del Producto*	2 years limited warranty *2 años de garantía limitada*
Approval number *Número de homologación*	II / IV-E9-04.11985
RoHS compliance *RoHS conformidad*	Yes *Sí*
REACH compliance *REACH conformidad*	Yes *Sí*

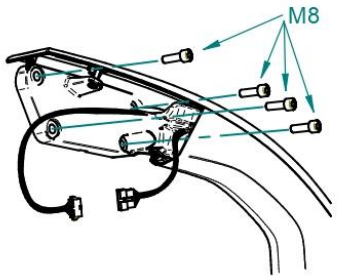
11/2019

Assembly Mod. 375 / *Montaje Mod. 375*

Step 1 / *Paso 1*

- Tighten the screws (Screws not included)

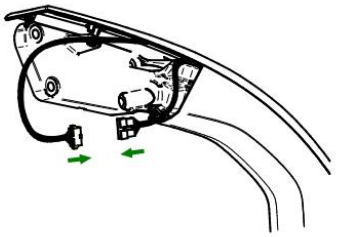
- *Apretar los tornillos (Tornillos no incluidos)*



Step 2 / *Paso 2*

- Connect the wiring

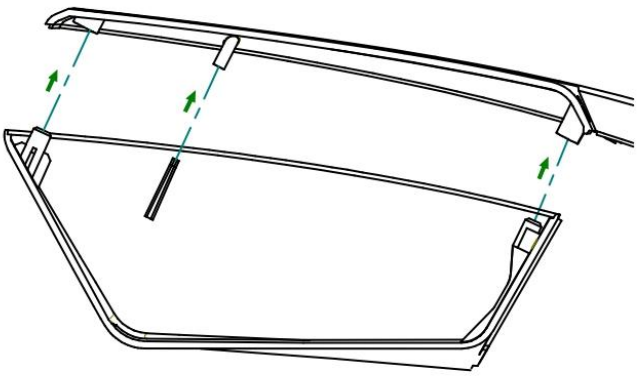
- *Conectar el cableado*



Step 3 / *Paso 3*

- Put the cover positioning the centering

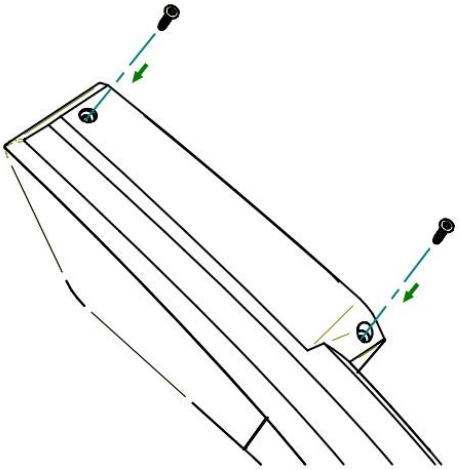
- *Colocar la tapa posicionando los centradores*



Step 4 / *Paso 4*

- Tighten the screws (STP41 screws included)

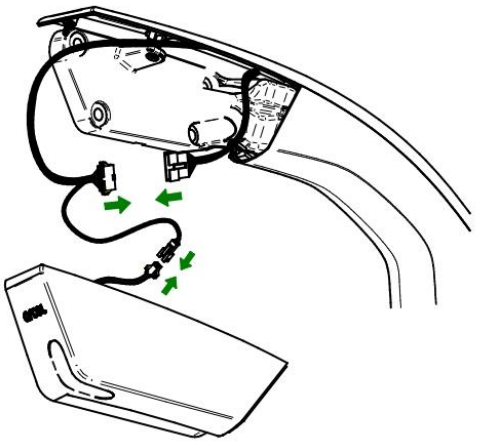
- *Apretar los tornillos (Tornillos STP41 incluidos)*



Courtesy light / Luz de cortesía

- In case the mirror would go with courtesy light, connect the connectors as indicated below

- *En el caso de que el retrovisor lleve luz de cortesía, conectar los conectores según indicado a continuación*



Wiring Scheme Mod. 375 / *Esquema de Cableado Mod. 375*

Mirror 375 / 375-2 / *Espejo 375 / 375-2*

Color	Function / *Función*	Cable section / *Sección del cable*
Black *Negro*	Ground Negativo	0,75 mm2
Blue *Azul*	Heated *Calefactado*	0,75 mm2
Yellow *Amarillo*	+/-	0,5 mm2
Grey *Gris*	⇑ ⇓	0,5 mm2
Brown *Marrón*	⇒ ⇐	0,5 mm2

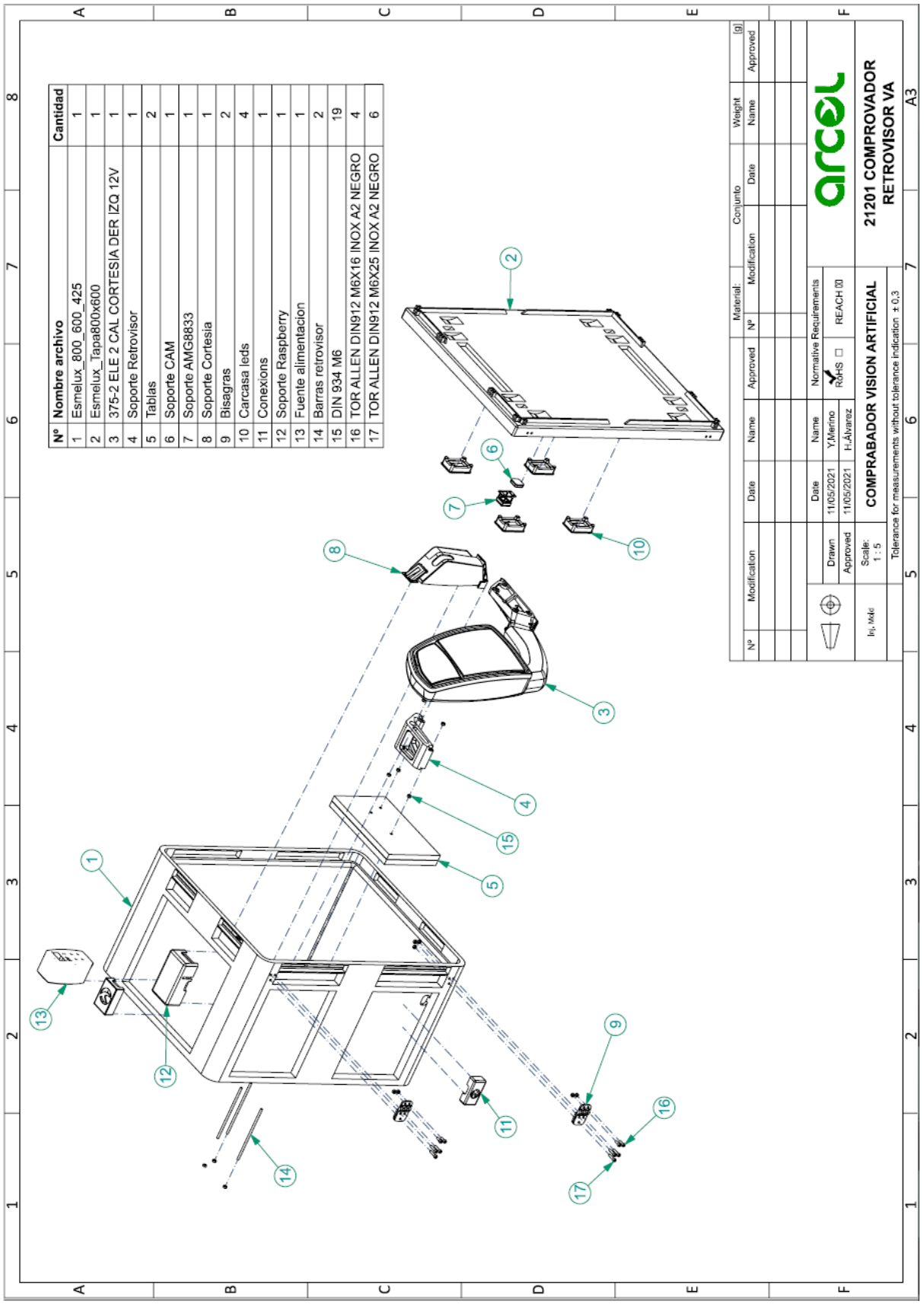
Mirror 375 CL / 375-2 CL / *Espejo 375 LC / 375-2 LC*

Color	Function / *Función*	Cable section / *Sección del cable*
Black *Negro*	Ground Negativo	0,75 mm2
Blue *Azul*	Heated *Calefactado*	0,75 mm2
Yellow *Amarillo*	+/-	0,5 mm2
Grey *Gris*	⇑ ⇓	0,5 mm2
Brown *Marrón*	⇒ ⇐	0,5 mm2
Red *Rojo*	Courtesy Light *Luz de Cortesía*	0,5 mm2
Black Negro	Courtesy Light *Luz de Cortesía*	0,5 mm2

Mirror 375 CL Camera / 375-2 CL Camera / *Espejo 375 LC Cámara / 375-2 LC Cámara*

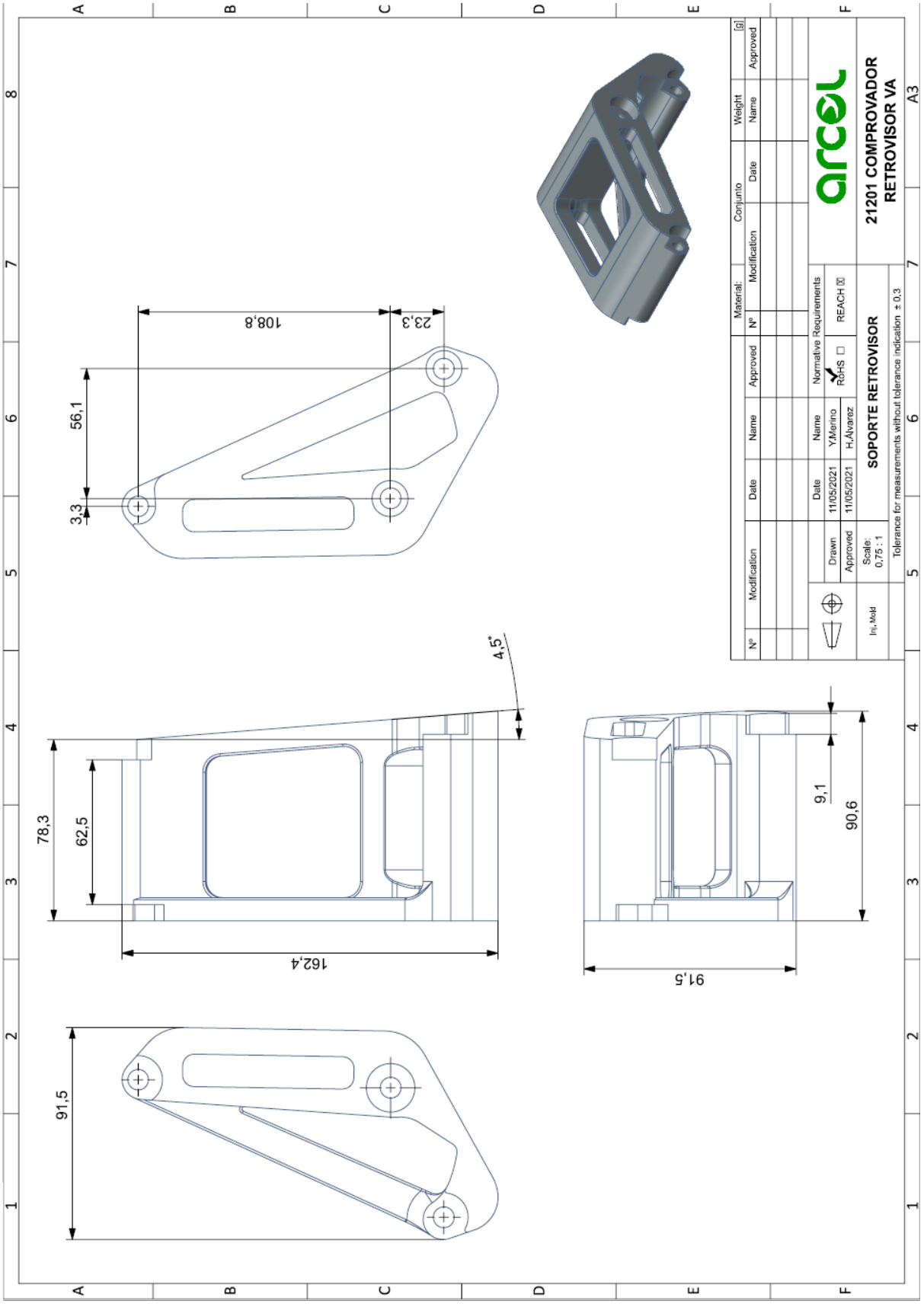
Color	Function / *Función*	Cable section / *Sección del cable*
Black *Negro*	Ground Negativo	0,75 mm2
Blue *Azul*	Heated *Calefactado*	0,75 mm2
Yellow *Amarillo*	+/-	0,5 mm2
Grey *Gris*	⇑ ⇓	0,5 mm2
Brown *Marrón*	⇒ ⇐	0,5 mm2
Green *Verde*	Courtesy Light *Luz de Cortesía*	0,5 mm2
White *Blanco*	Camera Power Alimentación Cámara	0,5 mm2
Black *Negro*	Ground Negativo	0,75 mm2
Black *Negro*	Ground Negativo	0,75 mm2
Transparent *Transparente*	Video *Vídeo*	Coaxial Core

8.3 Lista de materiales (BOM)

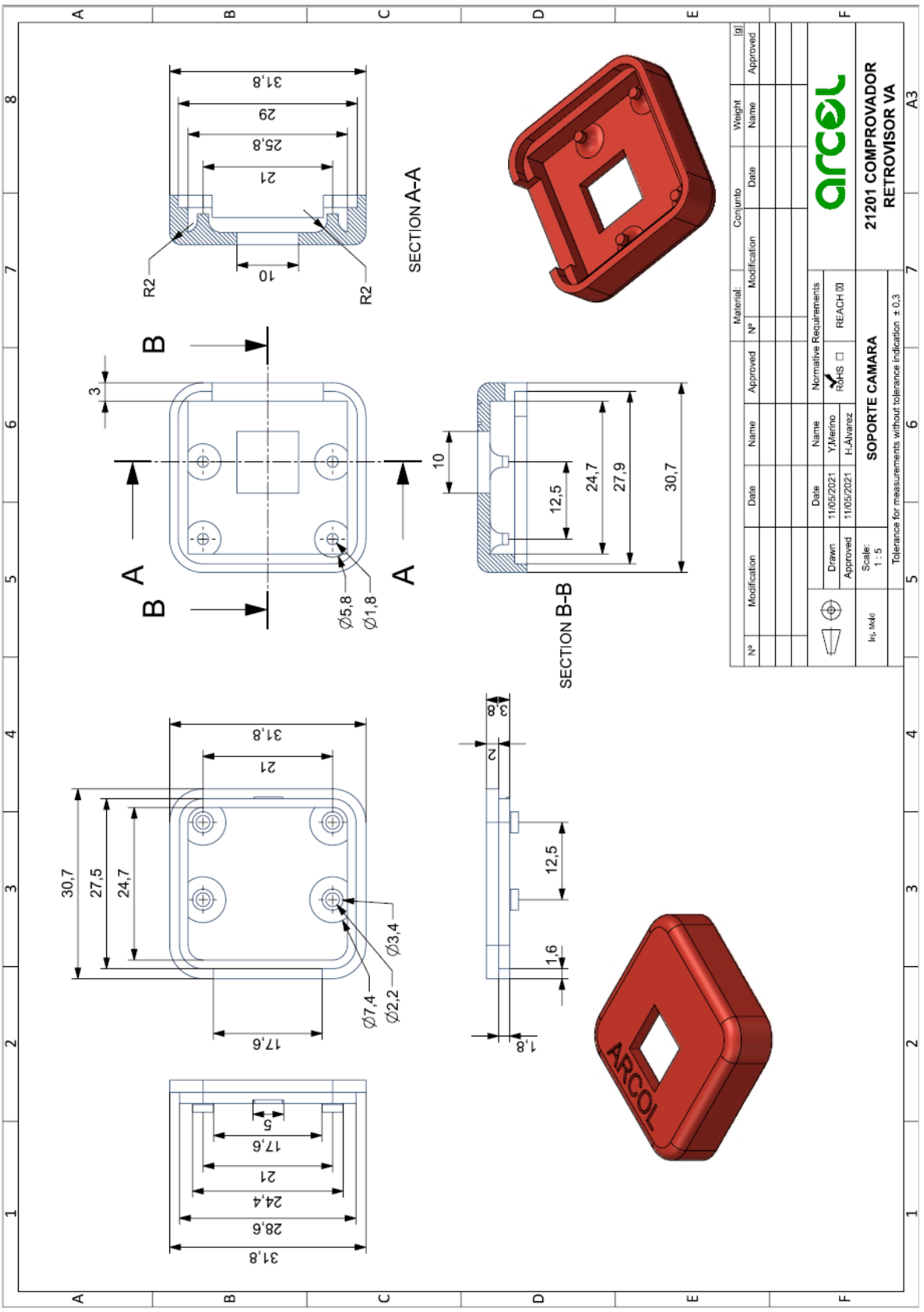


Nº	Nombre archivo	Cantidad
1	Esmelux_800_600_425	1
2	Esmelux_Tapa800x600	1
3	375-2 ELE 2 CAL CORTESIA DER IZQ 12V	1
4	Soporte Retrovisor	1
5	Tablas	2
6	Soporte CAM	1
7	Soporte AMG8833	1
8	Soporte Cortesia	1
9	Bisagras	2
10	Carcasa leds	4
11	Conexions	1
12	Soporte Raspberry	1
13	Fuente alimentacion	1
14	Barras retrovisor	2
15	DIN 934 M6	19
16	TOR ALLEN DIN912 M6X16 INOX A2 NEGRO	4
17	TOR ALLEN DIN912 M6X25 INOX A2 NEGRO	6

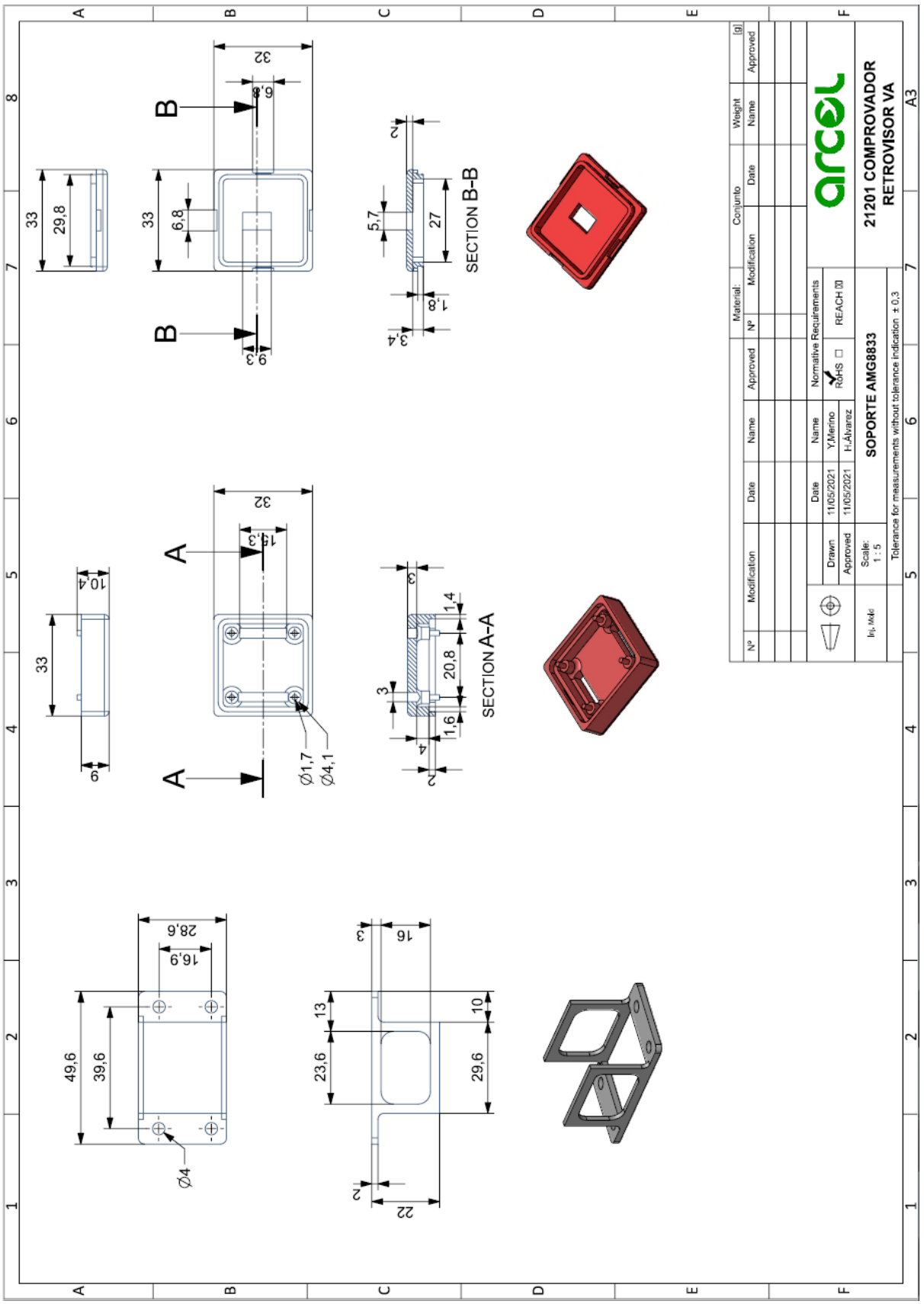
8.4 Plano soporte retrovisor



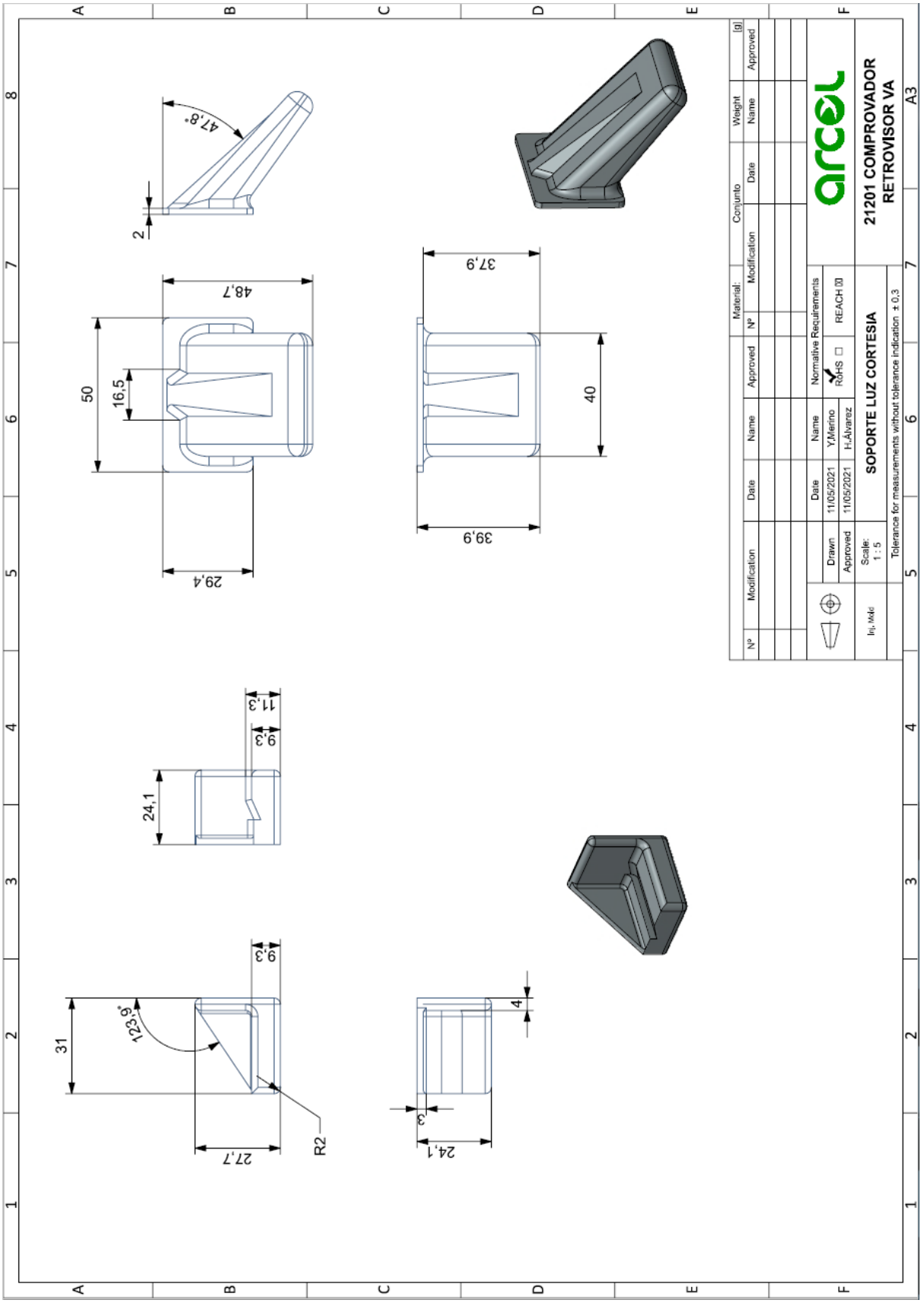
8.5 Plano soporte camara web



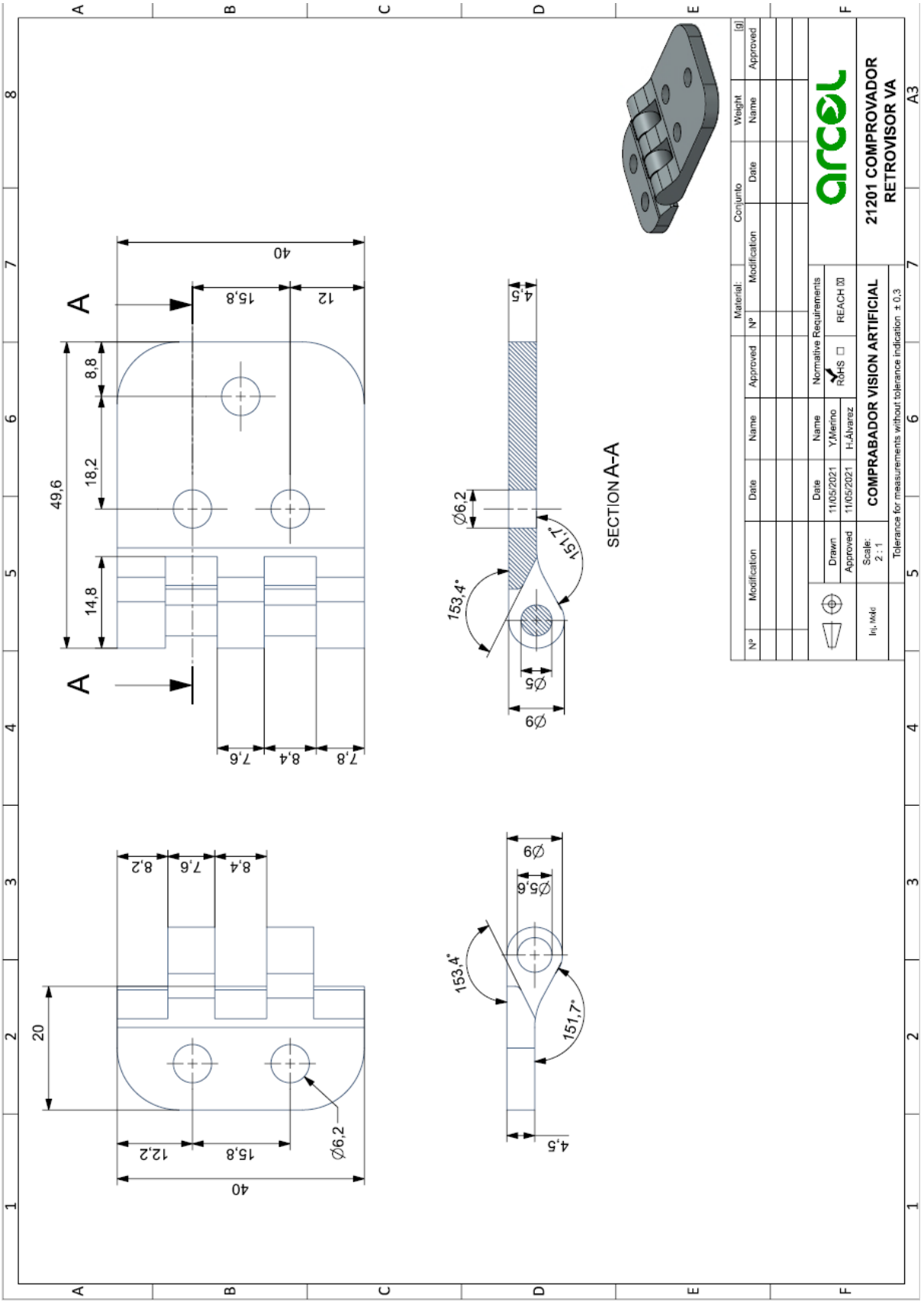
8.6 Plano soporte adafruit AMG8833



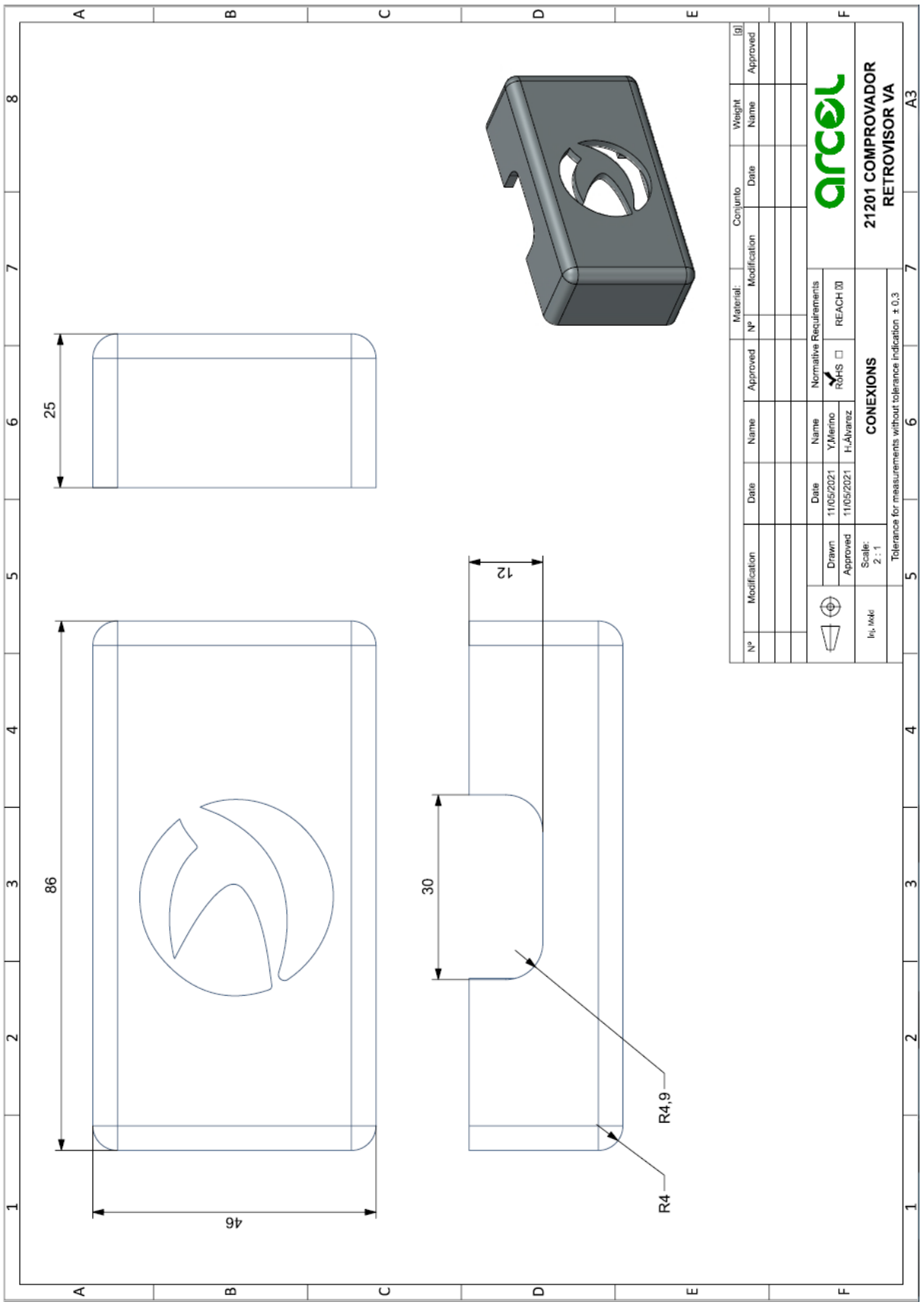
8.7 Plano soporte luz de cortesía



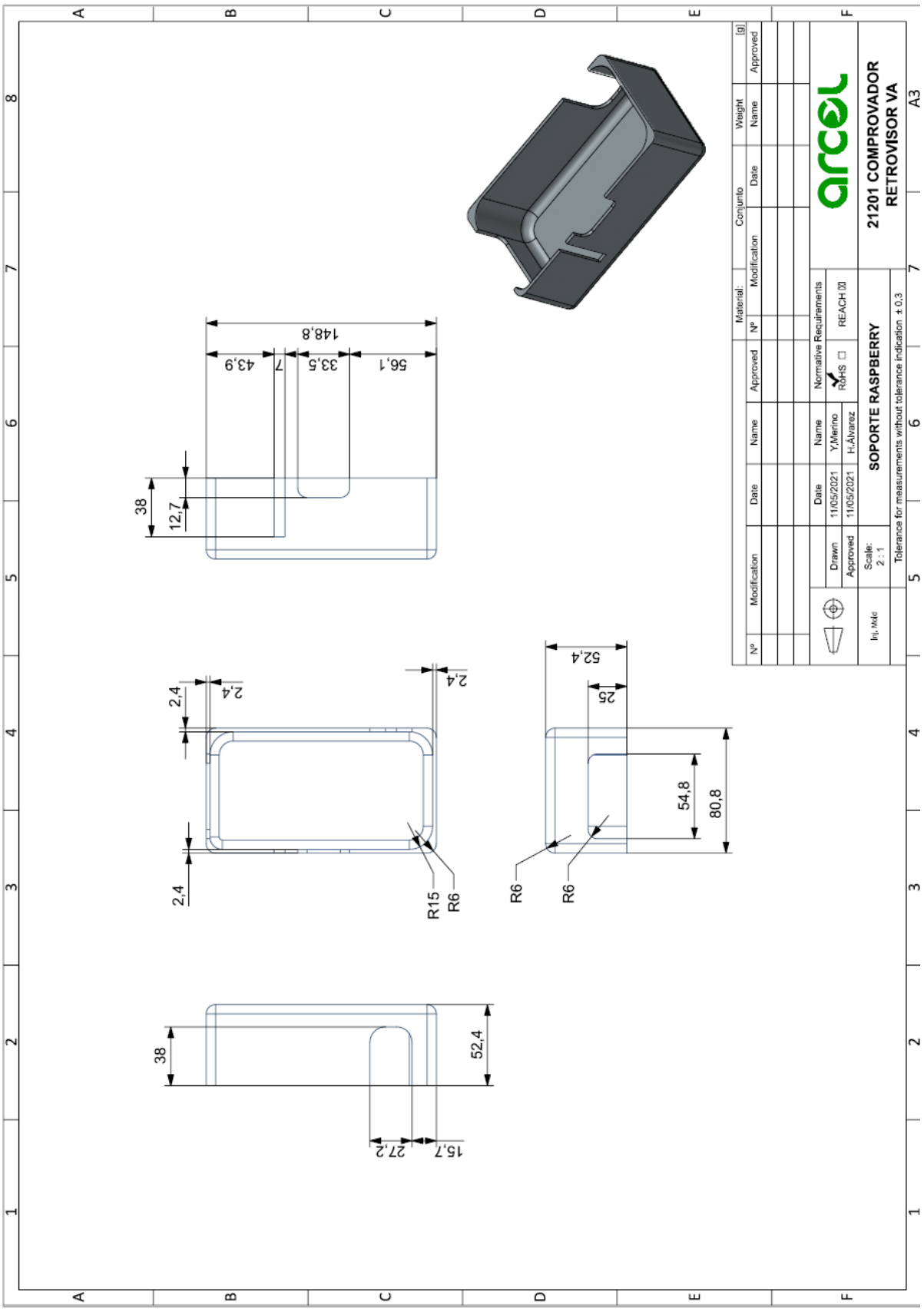
8.8 Plano bisagras



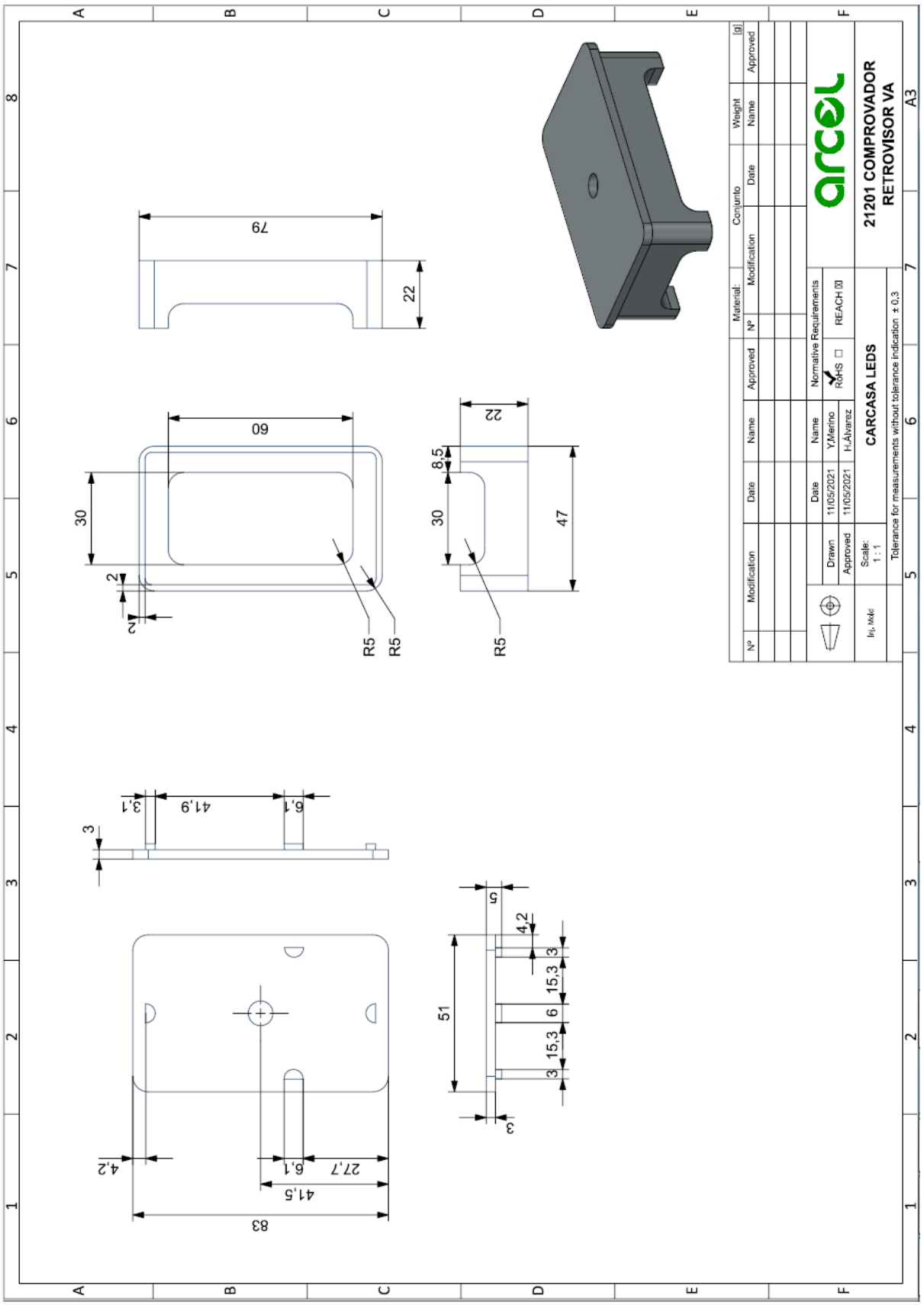
8.9 Plano conexiones



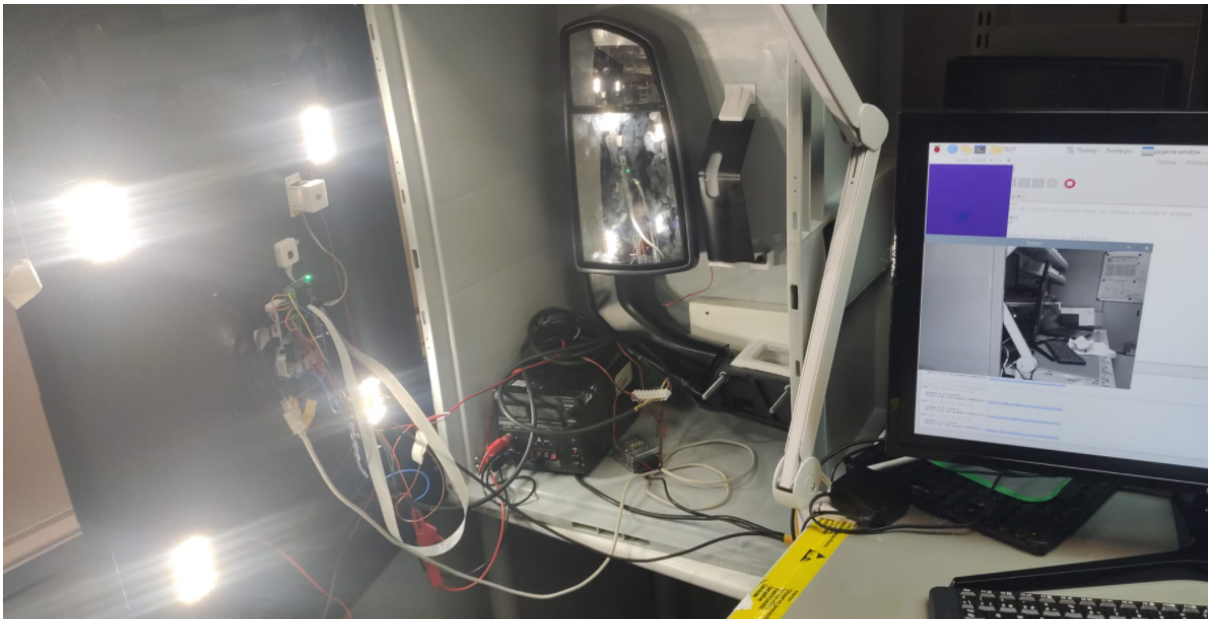
8.10 Plano soporte Raspberry Pi

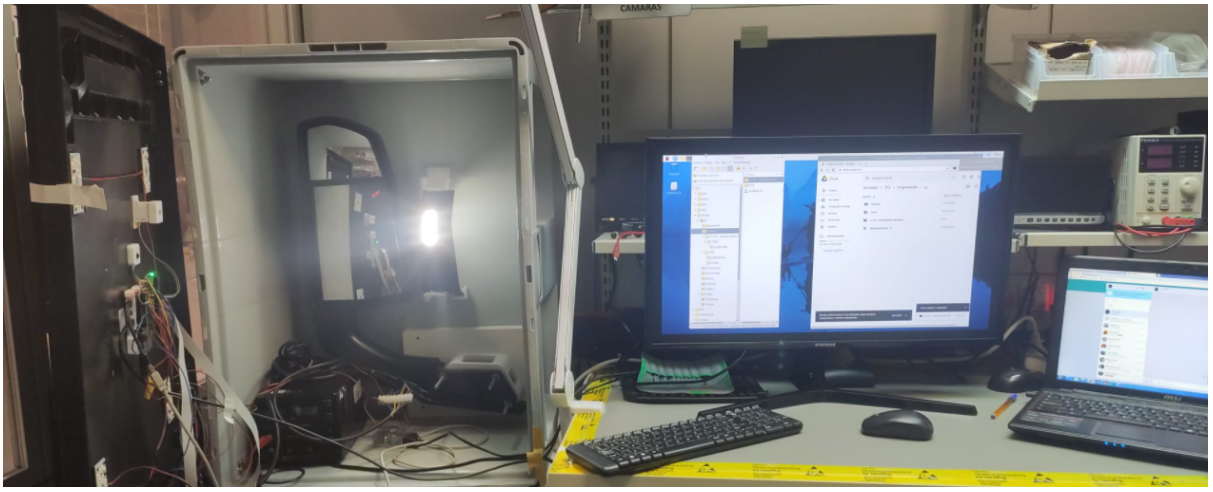


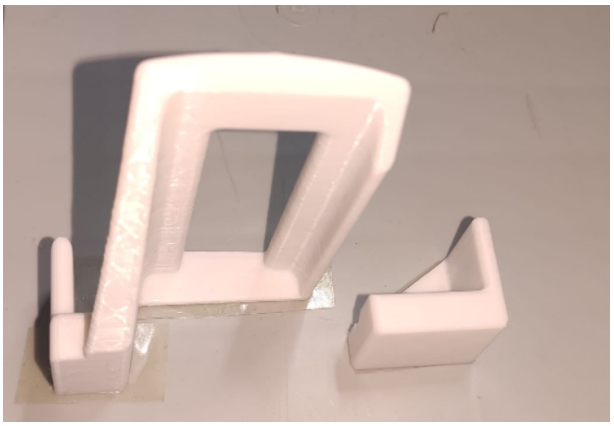
8.11 Plano carcasa leds



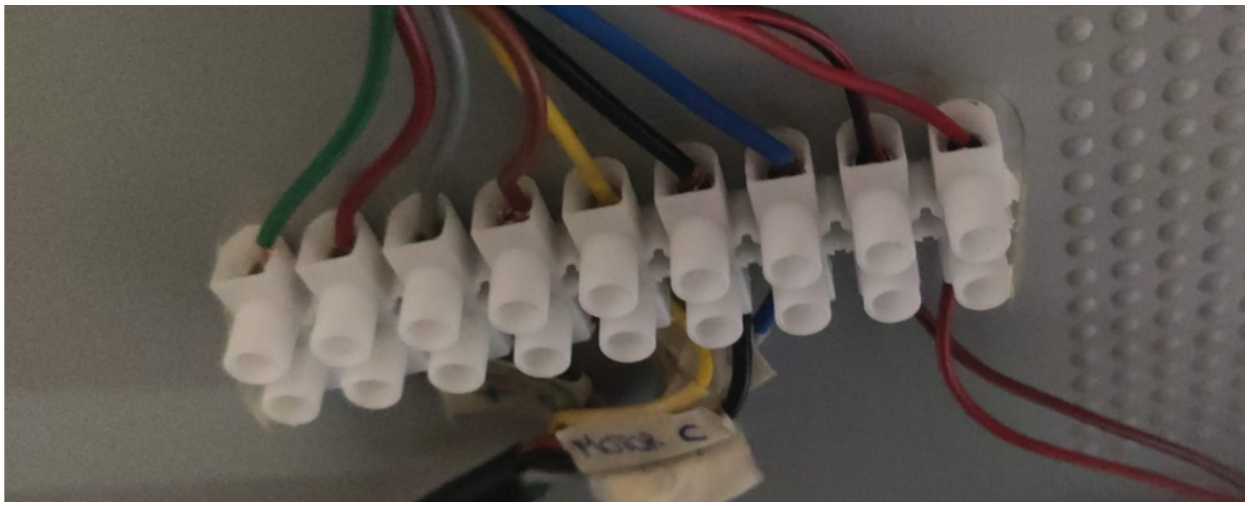
8.12 Imágenes del prototipo

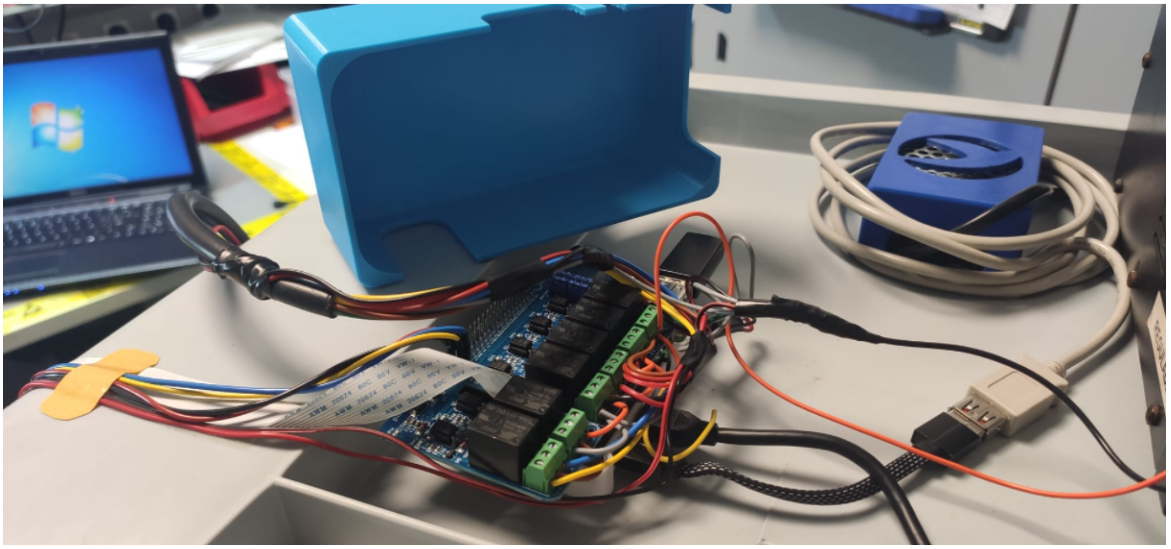


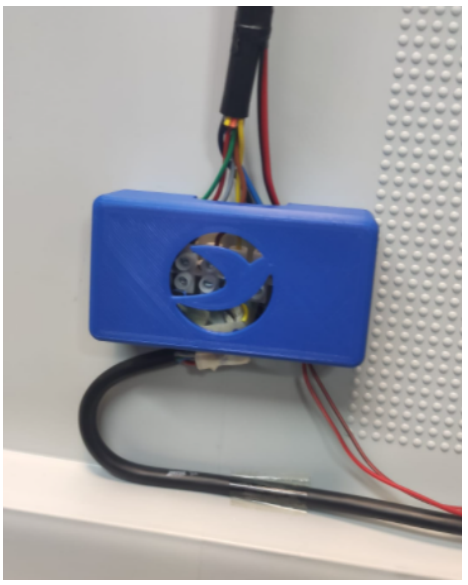




8.13 Imágenes del circuito eléctrico.








0-30V 2.5A DC
REGULABLE

8.14 Código python control de calidad

```
#==========================  PROJECT 21101 PROGRAM ==========================
#============================================================================
import os            #acceder funcionalidades del sistema operativo
import contextlib    #Utilidades comunes involucradas con with
with contextlib.redirect_stdout(None):
    import pygame    #Modulos de lenguaje creacion videojuegos (visual thermal)
import math          #funciones matematicas
import time          #funciones de tiempo
import numpy as np   #paquete para informatica cientifica
import busio         # modulo para variedad de protocolos seriales
import board          #Configuración pines Raspberry
from scipy.interpolate import griddata   #interpolar datos dd no estructurados
from colour import Color  #convierte y manipula representacion colores
import adafruit_amg88xx   #Sensor AMG8833 termico
import digitalio         #modulo contiene clases para proporcionar acceso E/S
import cv2              # Libreria computer vision
from imutils import contours #generadores de contorno
import imutils          #libreria procesamiento de imagenes
from skimage import measure   #enviar metricas de aplicacioens usando UDP
#============================     DEFINITIONS  ==============================
def CanalesRele():
    CH1=digitalio.DigitalInOut(board.D5)  #Pin29
    CH2=digitalio.DigitalInOut(board.D6)  #Pin31
    CH3=digitalio.DigitalInOut(board.D13) #Pin33
    CH4=digitalio.DigitalInOut(board.D19) #Pin35
    CH5=digitalio.DigitalInOut(board.D26) #Pin37
    CH6=digitalio.DigitalInOut(board.D12) #Pin32
    CHlist=(CH1,CH2,CH3,CH4,CH5,CH6)      #Creamos lista Reles
    for ch in CHlist:                     #para cada rele
        ch.direction = digitalio.Direction.OUTPUT #Pines de salida
        ch.value=True                     #Rele estado normal (no activado)
    return CHlist                         #Devolvemos lista reles

def Rele(accion):
    if accion == 'CAL':                  #CALEFACTADO
        CHlist[0].value= 1- CHlist[0].value #cambia estado calefactado
        if CHlist[0].value == False:    #Si se enciende avisa
            print('1   - Calefacción encendida (proceso calentamiento luna')
    elif accion == 'BLIND':             #BLINDSPOT
        CHlist[1].value= 1- CHlist[1].value #cambia estado blindspot
    elif accion == 'COURT':            #CORTESIA
        CHlist[2].value= False         #enciende court, apaga led
    elif accion == 'LED':              #LEDS COLORES
        CHlist[2].value= True          #enciende leds, apaga court
    elif accion == 'ML':               #Motor left (izq)
        CHlist[3].value= False
        CHlist[4].value= True
        CHlist[5].value= True
    elif accion == 'MR':              #Motor right(derecha)
        CHlist[3].value= True
        CHlist[4].value= False
        CHlist[5].value= True
    elif accion == 'MU':              #Motor up (arriba)
        CHlist[3].value= False
        CHlist[4].value= True
        CHlist[5].value= False
    elif accion == 'MD':              #Motor down (abajo)
        CHlist[3].value= True
        CHlist[4].value= False
        CHlist[5].value= False
    elif accion == 'STOP':            #Motor parados
        CHlist[3].value= True
        CHlist[4].value= True
        CHlist[5].value= True
```

```
66  def Detectar_CORTESIA(actual_frame):
67      roi_COURT=actual_frame[100:300,400:620] #Recortamos ROI
68      grayCOURT = cv2.cvtColor(roi_COURT,cv2.COLOR_BGR2GRAY) #convert gray
69      blurred = cv2.GaussianBlur(grayCOURT,(15,15),0) #nublamos img
70      ret,thresh_COURT = cv2.threshold(blurred, 235,255, cv2.THRESH_BINARY)
71      Etiqueta_COURT= measure.label(thresh_COURT, background=0) #Separamos la imagen
72      mask_COURT = np.zeros(thresh_COURT.shape, dtype="uint8")
73      for label in np.unique(Etiqueta_COURT): #Para cada figura de la imagen.
74          if label == 0: #si detecta que es fondo (0) no hagas nada.
75              continue
76          labelMask = np.zeros(thresh_COURT.shape, dtype="uint8") #Creamos una masca
77          labelMask[Etiqueta_COURT == label] = 255
78          numPixels = cv2.countNonZero(labelMask) #Contamos el numero de pixels
79          if numPixels > 2500: #Si es mayor que 2500 añadelo
80              mask_COURT = cv2.add(mask_COURT, labelMask)
81      if mask_COURT.any() == False:
82          print('4   -  <<ERROR>> Luz de cortesia no encontrada - Comprobar cableado
83      else:
84          print('4   - Luz Cortesia encontrada.')
85      return mask_COURT #devolvemos mascara de cortesia
86
87  def Detectar_BLIND(actual_frame):
88      roi_BLIND = actual_frame[0:100,300:400] #realizamos recorte
89      hsv_BLIND = cv2.cvtColor(roi_BLIND, cv2.COLOR_BGR2HSV) #convert HSV
90      color_bajos = np.array([0,0,25]) #Establecemos el rango mínimo y máximo de HSV
91      color_altos = np.array([179,145,164]) #limite color alto
92      mask_BLIND = cv2.inRange(hsv_BLIND, color_bajos, color_altos) #mascara
93      kernel = np.ones((3,3), np.uint8)  #Creamos el kernel y eliminamos el ruido:
94      mask_BLIND = cv2.morphologyEx(mask_BLIND, cv2.MORPH_CLOSE, kernel)#cerramos
95      mask_BLIND = cv2.morphologyEx(mask_BLIND, cv2.MORPH_OPEN, kernel)#abrimos
96      mask_BLIND = 255 - mask_BLIND #invierto la mascara
97      if mask_BLIND.any() == True:
98          print('3   - Luz BlindSpot encontrada.')
99      else:
100         print('3   -  <<ERROR>> Luz Blindspot no encontrada - Comprobar cableado y
101     return mask_BLIND
102
103 def mask_pintado(mask1,mask2):
104     mask_frame = np.zeros((480,640),dtype="uint8") #matrix de ceros
105     mask_frame [100:300,400:620] = mask1 #añadimos mask1 a la matriz
106     mask_frame [0:100,300:400] = mask2  #añadimos mask2 a la matriz
107     cnts = cv2.findContours(mask_frame.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
108     cnts = imutils.grab_contours(cnts) #cogemos contornos
109     if cnts:
110         cnts = contours.sort_contours(cnts)[0] #ordenamos contornos
111         for (i, c) in enumerate(cnts):
112             ((cX, cY), radius) = cv2.minEnclosingCircle(c) #encontramos radio
113             cv2.circle(frame, (int(cX), int(cY)), int(radius+4),(230, 20, 255), 1)
114     return frame    #devolvemos frame pintado
115
116 def constrain(val, min_val, max_val):
117     return min(max_val, max(min_val, val))
118
119 def map_value(x, in_min, in_max, out_min, out_max):
120     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
121
122 def SensorTermico():
123     pixels = [] #lista pixels vacia
124     for row in sensor.pixels: #para cada pixel detectado del sensor:
125         pixels = pixels + row #creamos lista de los pixels
126     pixels = [map_value(p, MINTEMP, MAXTEMP, 0, COLORDEPTH - 1) for p in pixels]
127     bicubic = griddata(points, pixels, (grid_x, grid_y), method="cubic")  # perfor
128     for ix, row in enumerate(bicubic): #pintamos todo
129         for jx, pixel in enumerate(row):
130            pygame.draw.rect(lcd, colors[constrain(int(pixel), 0, COLORDEPTH - 1)],
131                             (displayPixelHeight * ix,displayPixelWidth * jx,
132                              displayPixelHeight,displayPixelWidth,),)
```

```
    pygame.display.update()

def SensorTermicoImagenes():
    CAL_cold= cv2.imread('CAL_COLD.jpeg') #cargamos img fria
    CAL_hot= cv2.imread('CAL_HOT.jpeg') #cargamos img caliente
    res = cv2.absdiff(CAL_hot,CAL_cold) #diferencias absolutas
    res = res.astype(np.uint8)
    percentage = (np.count_nonzero(res) * 100)/ res.size #porcentaje cambio
    difference = cv2.subtract (CAL_cold,CAL_hot) #diferencia imagenes
    cv2.putText(difference,"DIFF",(10,230),cv2.FONT_ITALIC,0.7,(256,256,256),2)
    cv2.putText(difference,"Temp max:"+str(np.amax(sensor.pixels)),(100,230),
                cv2.FONT_ITALIC,0.3,(256,256,256),1)
    cv2.putText(CAL_cold,"COLD",(10,230),cv2.FONT_ITALIC,0.7,(256,256,256),2) #
    cv2.putText(CAL_hot,"HOT",(10,230),cv2.FONT_ITALIC,0.7,(256,256,256),2) #im
    Image_cal = np.concatenate((CAL_cold,CAL_hot), axis=1) #unimos imgenes
    Image_cal = np.concatenate((Image_cal,difference), axis=1) #unimos img
    print('5   - Expansión térmica en 100 segundos. \n\t Porcentage diferencia:
          percentage, '\n\t Tª max: ',np.amax(sensor.pixels))
    print(' >> Pulsar ESC para salir <<')
    return (Image_cal)

def HSVmask(frame):
    frame = frame [187:463,137:393]                          #Recortamos imagen
    frame_gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #Imagen a gris
    frame_HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) #Imagen escala HSV
    Canny = cv2.Canny(frame_gris, 200,500)              #Detectamos esquinas
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(8,8)) #kernel 8x8
    dilated = cv2.dilate(Canny,kernel)                  #Dilatamos las esquinas
    frame_HSVmask = cv2.bitwise_and(frame_HSV,frame_HSV, mask=dilated)#mascara
    return frame_HSVmask

def maskColores(frame):
    rojoBajo1 = np.array([0, 100, 20], np.uint8)     #Vector rojo 1
    rojoAlto1 = np.array([10, 255, 255], np.uint8)
    rojoBajo2 = np.array([175, 100, 20], np.uint8)   #Vector rojo 2
    rojoAlto2 = np.array([180, 255, 255], np.uint8)
    amarilloBajo = np.array([20, 100, 20], np.uint8)  #Vector amarillo
    amarilloAlto = np.array([32, 255, 255], np.uint8)
    verdeBajo = np.array([45, 100, 20], np.uint8)    #Vector verde
    verdeAlto = np.array([80, 255, 255], np.uint8)
    azulBajo = np.array([90, 100, 20], np.uint8)       #vector azul
    azulAlto = np.array([140, 255, 255], np.uint8)
    maskRed1 = cv2.inRange(frame, rojoBajo1, rojoAlto1)  #mascara rojo1
    maskRed2 = cv2.inRange(frame, rojoBajo2, rojoAlto2)  #mascara rojo2
    maskRed = cv2.add(maskRed1, maskRed2)                #mascara rojo
    maskYellow = cv2.inRange(frame, amarilloBajo, amarilloAlto) #masc amarilla
    maskGreen = cv2.inRange(frame, verdeBajo, verdeAlto) #mascara verde
    maskBlue = cv2.inRange(frame, azulBajo, azulAlto) #mascara azul
    return (maskYellow.any(),maskRed.any(),maskBlue.any(),maskGreen.any())


def calibracion(frame,mtx,dist,newcameramtx,roi):
    dst=cv2.undistort(frame,mtx,dist,None,newcameramtx)  #aplicamos cv2.undisto
    x,y,w,h=roi  #cogemos los datos de la region de interes
    x,y,w,h=int(x),int(y),int(w),int(h)  #nos aseguramos que sean numeros enter
    dst2=dst[y:y+h, x:x+w] #recortamos la imagen
    return dst2
```

```
def movCenter(Y,R,B,G):
    if (Y and R and B and G) == True:  #Centro encontrado
        mov = ('CENTER')
    elif ((Y == G == False) and (R == B == True)) or ((Y == G == True) and(R == B ==
        mov = ('Error')
    elif not (Y or R or G or B):  #Error, combinacion imposible
        mov = ('Error')
    elif (Y == R == True) and (B == G == False): #Motor down (abajo)
        mov = ('MD')
    elif  (Y == R == False) and (B == G == True): #Motor up (arriba)
        mov = ('MU')
    elif (Y==R==B== False) and (G == True): #motor left (izquierda)
        mov = ('ML')
    elif (Y==R==B== True) and (G == False): #motor right (derecha)
        mov = ('MR')
    elif R == True:   #motor left (izquierda)
        mov = ('ML')
    else:             #motor right( derecha)
        mov = ('MR')
    return mov
```

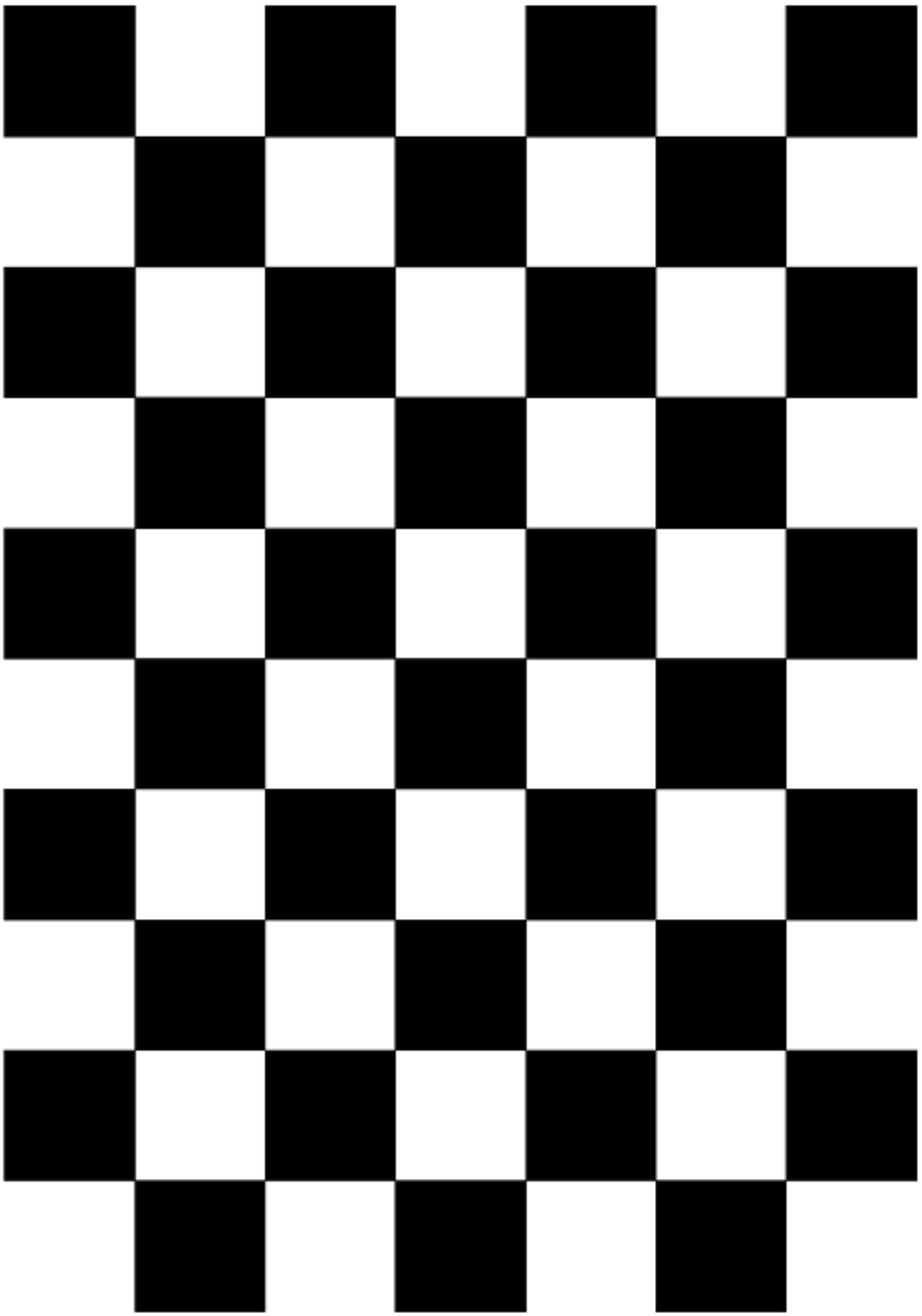
```

#==========================   PARAMETERS  ========================================

mtx=np.loadtxt('matrix.txt') #cargamos matriz camara
dist=np.loadtxt('distorsion.txt') #cargamos valores distorsion
newcameramtx=np.loadtxt('newcameramtx.txt') #nueva matriz
roi=np.loadtxt('roi.txt') #zona de interes

i2c_bus = busio.I2C(board.SCL, board.SDA) #Definimos bus I2C
MINTEMP,MAXTEMP,COLORDEPTH,blue = 20.0,28.0,1024,Color("indigo") #Definimos mintemper
colors = list(blue.range_to(Color("red"), COLORDEPTH)) #escogemos el rango colores de
colors = [(int(c.red * 255), int(c.green * 255), int(c.blue * 255)) for c in colors]

os.putenv("SDL_FBDEV", "/dev/fb1")
pygame.init() # Inicializamos todos los modulos importados pygame
sensor = adafruit_amg88xx.AMG88XX(i2c_bus) #Inicializamos sensor

points = [(math.floor(ix / 8), (ix % 8)) for ix in range(0, 64)] #Creamos puntos matr
grid_x, grid_y = np.mgrid[0:7:32j, 0:7:32j] #Creamos 2 matrices 8x8 con longitud de p
height,width = 240,240 #Creamos cuadrado ya que matriz es 8x8

displayPixelWidth = width / 30 #ancho pantalla pixels
displayPixelHeight = height / 30 #alto pantalla pixels
lcd = pygame.display.set_mode((width, height)) #creamos display
lcd.fill((255,0,0))

pygame.display.update() #actualizo display
pygame.mouse.set_visible(False) #Hago que desaparezca el cursor cuando pasa por la pa
lcd.fill((0, 0, 0))
pygame.display.update() #actualizo las partes de la pantalla

Center = False
Blind_Court = False
Start_time = time.time()
countCenter = 0
countErrorColores = 0
tiempo_calentar= 120
```

```
#==========================   PROGRAM  ========================================

CHlist=CanalesRele() #Configuramos todos los canales para controlar los reles
cap = cv2.VideoCapture(0) #Iniciamos la camara numero 0 (puede haber mas camaras en e
time.sleep(0.5) #tiempo para inicializar sensores y demas

while (True):
    ret, frame = cap.read()  #devuelve un bool (verdadero / falso). Si el cuadro se l
    Rele('STOP')
    frame = calibracion(frame,mtx,dist,newcameramtx,roi) #calibramos img
    Actual_time = time.time()- Start_time
    SensorTermico()
```

```
259
260         if Blind_Court == True:
261             frame = mask_pintado(mask_COURT,mask_BLIND)
262         if Center == False:
263             cv2.putText(frame,"Centrando luna: (" + str(int(Actual_time)) + '/' +
264                         str(tiempo_calentar)+')seg',(420,10),cv2.FONT_ITALIC,
265                         0.4,(256,256,256),1)
266         elif Blind_Court == False:
267             cv2.putText(frame,"Comprobando luces: ("+str(int(Actual_time))+'/'+
268                         str(tiempo_calentar)+')seg',(420,10),cv2.FONT_ITALIC,
269                         0.4,(256,256,256),1)
270         elif tiempo_calentar >Actual_time:
271             cv2.putText(frame,"Calentando luna: ("+str(int(Actual_time))+'/'+
272                         str(tiempo_calentar)+')seg',(420,10),cv2.FONT_ITALIC,
273                         0.4,(256,256,256),1)
274
275         cv2.imshow('Frame Camara',frame)
276
277         if Actual_time < 25:
278             if CHlist[0].value == True:
279                 Rele('CAL')
280                 pygame.image.save(lcd, "CAL_COLD.jpeg")
281
282             if Center == False:
283                 maskHSV = HSVmask(frame)       #definicion crear mask colores
284                 Y,R,B,G = maskColores(maskHSV) #Que colores detecta
285                 movMotor = movCenter(Y,R,B,G)  #movimientos motor
286                 cv2.rectangle(frame,(137,187),(393,463),(255,255,255),1) #cuadrado
287                 cv2.imshow('Frame Camara',frame) #muestra imagen
288                 if movMotor == 'CENTER':
289                     countCenter+=1
290                     if countCenter>3: #cuando encuentres 3 veces el centro
291                         Center = True
292                         Centertime = time.time()
293                         print('2   - Centro luna encontrado (Luna correctamente mon
294                 elif movMotor == 'Error': #si encuentras error 30 veces:
295                     countErrorColores+=1
296                     if countErrorColores > 30:
297                         print('2   -  <<ERROR>> No se ha podido encontrar centro -
298                         Center = True
299                         Centertime = time.time()
300                 else:
301                     Rele(movMotor) # mueve los motores dirrecciones mandadas.
302
303         if Actual_time > 25 or Center == True:
304             if Center == False:
305                 print('2   -  <<ERROR>> No se ha podido encontrar centro - Revisar
306                 Center = True
307             if Blind_Court == False:
308                 if (CHlist[1].value == CHlist[2].value == True):
309                     Rele('COURT')
310                     Rele('BLIND')
311                 elif (Actual_time > 28) or ((time.time()-Centertime)>2):
312                     mask_BLIND = Detectar_BLIND(frame)
313                     mask_COURT = Detectar_CORTESIA(frame)
314                     Rele('BLIND')
315                     Rele('LED')
316                     Blind_Court = True
317
318             if  (Actual_time > tiempo_calentar) and (CHlist[0].value == False):
319                 pygame.image.save(lcd, "CAL_HOT.jpeg")
320                 CAL_img = SensorTermicoImagenes()
321                 cv2.imshow('Expansión térmica',CAL_img)
322                 cv2.moveWindow('Expansión térmica',240,27)
323                 Rele('CAL')
```

```
324
325         k = cv2.waitKey(1) & 0xff
326         if k==27: #con ESC salimos del programa destruyendo todas las ventanas y cerrando
327             cap.release()
328             CHlist=CanalesRele()
329             cv2.destroyAllWindows()
330             pygame.display.quit()
331             pygame.quit()
332             break
333
334     cap.release()
335     CHlist=CanalesRele()
336     cv2.destroyAllWindows()
337     pygame.display.quit()
338     pygame.quit()
```

8.15 Patrón - Tablero de ajedrez



8.16 Código python calibración

```python
import numpy as np #importamos numpy para funciones matematicas
import cv2         #importamos openCV para computer vision
import glob        #expansión del patrón de nombres

tablero=(9,6)

def menutxt():  #Definicion del menu. Explica diferentes opcion para el usuario.
    print('_____________________________________________________________')
    print('A continuación escriba el número del menú que desea seleccionar:')
    print('\t 1- Guardar nuevas imagenes del patron de ajedrez. \n \t 2- Obtener nu

def leer():    #Definicion para asegurar que el usuario solo puede introducir numer
    while True:
        entrada = input()
        try:
            entrada=int(entrada)
            return entrada
        except ValueError:
            print('\n La entrada es incorrecta: Escribe un número entero')

def Tomarfotos(numero):        #Definicion para realizar X número de imagenes y gua
    contador=1                 #contador
    cap = cv2.VideoCapture(1)  #Iniciamos la camara numero 0 (puede haber mas camar
    print(' << Presiona enter cuando desees capturar la imagen >>')
    print(' <<         Presiona Esc para salir                >>')
    while contador<numero+1:   #Repite el proceso hasta llegar al numero indicado p
        ret, frame = cap.read() #Empieza a leer la camara
        cv2.imshow('frame',frame) #Muestra una ventana con la camara
        k = cv2.waitKey(50) & 0xff #Espera 50 milisegundos a una respuesta del tecl
        if k == 13:                #Si se presiona Enter guarda una imagen con el n
            nombrefoto= 'calibrate'+ str(contador)+ ".png"
            cv2.imwrite(nombrefoto,frame)
            print("Foto ",nombrefoto," tomada correctamente")
            contador+=1
        elif k == 27:              #Si se presiona escape se sale del menu.
            break
    cap.release()                 #una vez tomadas todas las fotos cierra la camara
    cv2.destroyAllWindows()       #Destruye todas las ventanas abiertas

def Calibracion():             #Definición para obtener parametros de calibración
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)  #obt
    puntos2D = []  #creamos una lista para los puntos 2D, los que obtenemos en la im
    puntos3D = []  #creamos una lista para los puntos 3D, los del objeto
    objectp3D = np.zeros((tablero[0]*tablero[1],3), np.float32)          #preparamo
    objectp3D[:,:2] =np.mgrid[0:tablero[0],0:tablero[1]].T.reshape(-1,2)  # (0,0,0),
    images = glob.glob('calibrate*.png')  #images es una lista de todos los archivos
    for filename in images: #Recorremos todas las imagenes tomadas
        img = cv2.imread(filename)   #leemos cada imagen
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) #pasamos cada imagen a gris
        ret, corners = cv2.findChessboardCorners(gray,tablero,None)   #Buscamos las
        if ret == True:   #Si se encuentran las esquinas:
            puntos3D.append(objectp3D)  #añadimos los puntos a la lista de puntos 3D
            corners2=cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria) #encont
            puntos2D.append(corners2) #añadimos esas esquinas del patron a la lista
            img= cv2.drawChessboardCorners(img,tablero,corners2,ret) #dibujamos en l
        cv2.imshow('img',img) #Mostramos al usuario la imagen, de esta forma podemos
        z=cv2.waitKey(200) #esperamos 0,2s entre foto y foto
    cv2.destroyAllWindows() #cerramos todas las imagenes abiertas
    h,w= img.shape[:2] #Encontramos el tamaño de la imagen, y guardamos la variable
```

```
    ret,mtx,dist,r_vecs,t_vecs= cv2.calibrateCamera(puntos3D,puntos2D,
                                                    gray.shape[::-1],None,None) #con la fu
    newcameramtx,roi=cv2.getOptimalNewCameraMatrix(mtx,dist, (w,h), 1,(w,h)) #Refinamos la
    tot_error=0 #Ahora realizamos una funcion para que nos salga el error de proyeccion, s
    for i in range(len(puntos3D)):
        imgpoints2, _ = cv2.projectPoints(puntos3D[i],r_vecs[i],t_vecs[i],
                                          mtx, dist)
        error= cv2.norm(puntos2D[i],imgpoints2,cv2.NORM_L2)/len(imgpoints2)
        tot_error+=error
    print('\t Error total: ', tot_error/len(puntos3D))

    np.savetxt('matrix.txt',mtx) #Guardamos la matriz de la camara en un documento de texto
    np.savetxt('distorsion.txt',dist)  #Guardamos los coeficientes de distorsion en un doc
    np.savetxt('newcameramtx.txt',newcameramtx)  #Guardamos la optimización de la matriz
    np.savetxt('roi.txt',roi)  #Guardamos la region de interes para saber donde recortar l
    return(mtx,dist,newcameramtx,roi)

def Deformacion():   #Matriz para quitar la distorsion de laimagen
    mtx=np.loadtxt('matrix.txt')     #cargamos todos los datos de los archivos de txt.
    dist=np.loadtxt('distorsion.txt')
    newcameramtx=np.loadtxt('newcameramtx.txt')
    roi=np.loadtxt('roi.txt')
    img=cv2.imread('calibrate1.png')

    dst=cv2.undistort(img,mtx,dist,None,newcameramtx)  #aplicamos cv2.undistort con la mat
    x,y,w,h=roi  #cogemos los datos de la region de interes
    x,y,w,h=int(x),int(y),int(w),int(h)  #nos aseguramos que sean numeros enteros
    dst2=dst[y:y+h, x:x+w] #recortamos la imagen
    cv2.imshow('Imagen sin calibrar',img) #Muestra imagen original
    cv2.imshow('Imagen calibrada',dst)   #Muestra imagen calibrada
    cv2.imshow('Imagen calibrada y recortada',dst2) #Muestra imagen calibrada y recortada
    k=cv2.waitKey(5000) #Esperamos 5 segundos y cerramos todo
    cv2.destroyAllWindows()
    cv2.imwrite('ImagenCalibrada.png',dst) #guardamoslas imagens obtenidas
    cv2.imwrite('ImagenCalibradaYRecortada.png',dst2)
```

```
#=============================================================================================
print('Bienvenido al programa de Vision Artificial para el control de calidad de retrovisores

while True:
    menutxt()
    menu=leer()
    if menu==1:
        print('¿Que cantidad de imagenes deseas guardar? (mayor que 10): ')
        num=leer()
        Tomarfotos(num)
    elif menu==2:
        mtx,dist,newcameramtx,roi=Calibracion()
        print('\t Se han guardado los nuevos coeficientes')
    elif menu==3:
        Deformacion()
        print('\t La deformación ha sido corregida. \n \t La imagen sin deformación y la imag
    elif menu==4:
        print('\t Has seleccionador salir \n \t Saliendo del programa.')
        break
    else:
        print(' << No existe el menu seleccionado, escriba de nuevo el número del menú >>')
```

8.17 Clasificadores en cascada basados en funciones de Haar

Un clasificador en cascada de Haar es un método eficaz de detección de objetos propuesto por Paul Viola y Michael Jones en su artículo "Detección rápida de objetos mediante una cascada mejorada de funciones simples" en 2001. Este es un método basado en aprendizaje automático a partir de muchas imágenes positivas y negativas que identifica objetos en una imagen y un video. El algoritmo se puede explicar en cuatro etapas:

1. **Calcular las características del cabello**: El primer paso es recopilar las características de Haar, que son cálculos que se realizan en regiones rectangulares adyacentes a una ubicación específica en una ventana de detección. El cálculo implica sumar las intensidades de píxeles en cada región y calcular las diferencias entre las suma.

2. **Creación de imágenes integrales**: con tal de facilitar el cálculo de características Haar en imágenes grandes se crean imágenes integrales. En una imagen integral en vez de calcular en cada píxel, se crean sub-rectángulos y referencias de matriz para cada sub-rectángulo.

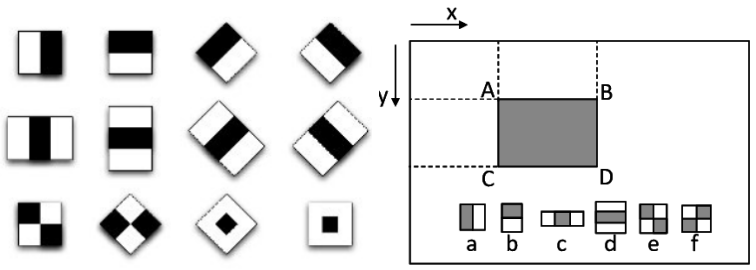


Figura 4.2.1 a) Clasificador Haar b) Ilustración imagen integral

3. **Entrenamiento Adaboost**: La gran mayoría de funciones Haar serán irrelevantes para realizar la detección del objeto. Adaboost elige las mejores características y entrena a los clasificadores para que las utilicen, utilizando una combinación de “clasificadores débiles” para crear un “clasificador fuerte” que el algoritmo pueda usar para detectar el objeto.

   Los clasificadores débiles se crean moviendo una ventana sobre la imagen de entrada y calculando las características de Haar para cada subsección. Esta diferencia se compara con un umbral aprendido que separa los no objetos de los objetos. Debido a que estos son clasificadores débiles, se necesita una gran cantidad de características Haar para que la precisión forme un clasificador fuerte. El último paso combina estos clasificadores débiles en un clasificador fuerte.

Figura 4.2.2 Representación modelo predictivo

4. **Implementación de clasificadores en cascada**: el clasificador en cascada se compone de una serie de etapas, donde cada etapa es una colección de clasificadores débiles. Cada clasificador se entrena mediante el refuerzo, lo que permite un clasificador de alta precisión a partir de la predicción media de todos los clasificadores débiles.

   Según esta predicción, el clasificador decide indicar que se encontró un objeto (positivo) o pasar a la siguiente región (negativo). Las etapas están diseñadas para rechazar muestras negativas lo más rápido posible, porque la mayoría de las ventanas no contienen nada de interés.

   Es importante maximizar una tasa baja de falsos negativos , porque clasificar un objeto como no objeto afectará gravemente al algoritmo de detección de objetos.

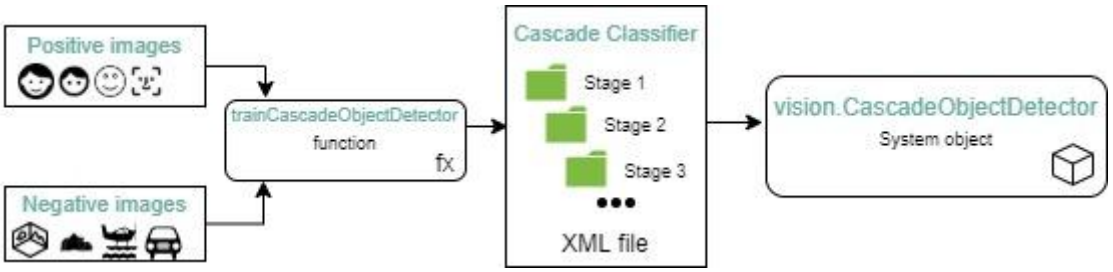


Figura 4.2.3 Representación modelo predictivo