UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

Trabajo Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

AUTOR/A: Kotei Zuriaga, Benson Alejandro

Tutor/a: Baraza Calvo, Juan Carlos

Cotutor/a: Gracia Morán, Joaquín

CURSO ACADÉMICO: 2021/2022

RESUMEN

El presente documento aborda el desarrollo completo de un robot humanoide destinado a un circuito de pruebas urbano. Mediante impresión 3D y aplicaciones CAD se diseñarán y producirán las piezas de dicho modelo. A su vez se añaden sensores y actuadores, que junto a un potente microcontrolador hacen posible una navegación autónoma en un entorno similar al urbano.

En este proyecto se usará como sensor una cámara capaz de tomar toda la información necesaria del exterior y una Raspberry Pi que hará de puente entre el sensor y los actuadores. Finalmente se conseguirá un robot autónomo capaz de comportarse como un peatón o viandante, sin necesidad de acciones externas.

Debido a la naturaleza del proyecto, se tocan ramas como son el diseño, la programación, la impresión 3D, el montaje y la validación

Palabras clave: Vehículo autónomo; Arduino; Impresión 3D.

RESUM

El present document aborda el desenvolupament complet d'un robot humanoide destinat a un circuit de proves urbà. Mitjançant impressió 3D i aplicacions CAD es dissenyaran i produiran les peces d'aquest model. Al seu torn s'afigen sensors i actuadors, que al costat d'un potent microcontrolador fan possible una navegació autònoma en un entorn similar a l'urbà.

En aquest projecte s'usarà com a sensor una càmera capaç de prendre tota la informació necessària de l'exterior i una Raspberry Pi que farà de pont entre el sensor i els actuadors. Finalment s'aconseguirà un robot autònom capaç de comportar-se com un vianant o vianant, sense necessitat d'accions externes.

A causa de la naturalesa del projecte, es toquen branques com són el disseny, la programació, la impressió 3D, el muntatge i la validació

Paraules clau: Vehicle autònom; Arduino; Impressió 3D.

ABSTRACT

This paper deals with the complete development of a humanoid robot for an urban test track. The parts of this model are designed and produced using 3D printing and CAD applications. At the same time, a powerful microcontroller, sensors and actuators are added, to make possible an autonomous navigation in an urban-like environment.

In this project, is used a camera capable of taking all the necessary information from the outside and a Raspberry Pi that will be the bridge between the sensor and the actuators. Finally, we will get an autonomous robot able to behave like a living person, without the need of external actions.

Because of the naturalness of the project, branches such as design, programming, 3D printing, assembly and validation are touched.

Keywords: Autonomous vehicle; Arduino; 3D printing.

ÍNDICE

DOCUMENTOS CONTENIDOS EN EL TFG

I. Memoria
II. Presupuesto
III. Planos
IV. Anexos

ÍNDICE GENERAL

ÌNDICE DE FIGURAS

ÍNDICE DE TABLAS

I.MEMORIA

CAPITULO. 1 INTROCUCIÓN

1.1. OBJETO DEL TRABAJO

A lo largo del presente proyecto se aborda el desarrollo de un robot humanoide, con la impresión 3D como herramienta pilar para su diseño. Al mismo tiempo se cuenta con el añadido de un microcontrolador así como de sensores y actuadores, para poder implantar funcionalidades de reconocimiento de imágenes que ayuden a conseguir un comportamiento autónomo en un entorno urbano.

Por tanto, se entiende como objetivo principal del presente proyecto el diseño e implementación de un robot humanoide con comportamientos autónomos semejantes a un peatón en la vía pública. Estas características se tratarán de lograr a través de una cámara incorporada, junto con programación basada en visón artificial.

Por último, cabe destacar el carácter global y profesional de este proyecto, debido a la gran variedad de disciplinas que están integradas en el mismo. Estas disciplinas han sido impartidas a lo largo de toda la titulación, haciendo hincapié en Informática, Impresión 3D y Fabricación, Tecnología Informática Industrial, Proyectos, Sistemas de Producción y Fabricación, Sistemas Automáticos, Tecnología Eléctrica, etc.

1.2. ANTECEDENTES

El auge de los sistemas autónomos es una realidad en la actualidad cada vez se da más atención a sistemas capaces de responder por sí solos a estímulos externos, tanto a nivel industrial como cotidiano. Un ejemplo de esto son los vehículos autónomos o las casas inteligentes. Sin embargo, estos sistemas no pueden saltar a su zona de acción sin un entrenamiento previo en ambientes similares a los reales. Esto es lo que se trata de resolver con este proyecto.

Los coches autónomos son una realidad y necesitan zonas de entrenamientos y pruebas similares a las reales para poder mejorar y validar los sistemas de conducción autónoma basados en inteligencia artificial. Para esto se hacen uso de sistemas de simulación basado en machine learning y entornos de pruebas.

Así, en el presente proyecto se desarrolla el diseño y montaje completo de un robot humanoide capaz de comportarse como viandante, pudiendo circular por la vía pública y cruzar por zonas controladas por semáforos. El objetivo final es que el robot forme parte de un banco de pruebas más complejo, en el que conviven tanto vehículos autónomos como otros peatones.

En un primer lugar, se realizará un planteamiento general del proyecto, teniendo en cuenta los materiales necesarios, las herramientas pertinentes y las funcionalidades que ha de cumplir. A continuación, se desarrolla el diseño para su posterior impresión en 3D y, tras el montaje, se le ha proporcionado la autonomía propia de un viandante a través de programación basada en visión artificial.

Por último, se llevan a cabo una serie de pruebas para verificar la correcta respuesta ante distintos agentes externos en un entorno de pruebas controlado, obteniendo de esta forma el prototipo deseado.

1.3. JUSTIFICACION

Este proyecto presenta la oportunidad de mejorar las zonas de pruebas de los sistemas autónomos. Parte de la premisa de ser capaces de crear ambientes realistas controlados para poner a prueba estos mismos sistemas.

Se pretende traspasar los problemas que conlleva la captación de información desde tan solo una cámara sin ningún otro sensor externo. Se ha desarrollado así un proyecto global, capaz de resolver este reto y avanzar así en este campo.

Además, el trabajo requiere manejar herramientas muy importantes para la ingeniería, como es el diseño CAD y la impresión 3D, que nos permiten crear prototipos y piezas funcionales con relativa rapidez. También se hará uso de la visón artificial, empleada actualmente en multitud de aplicaciones como detección de objetos, navegación autónoma, clasificación de objetos, detección de movimientos, reconocimiento facial, etc.

1.4. MOTIVACION

Tras cursar un grado tan completo como es el de Ingeniería en Tecnologías Industriales, se obtienen conocimientos de muchas ramas de la ingeniería, pero con ganas de profundizar en muchas otras. Por ello resulta atractivo realizar un proyecto en un ámbito más específico, pero que a su vez integra varias disciplinas que tienden a trabajar juntas.

De esta forma, el presente proyecto integra tanto una potente componente de Diseño y Fabricación de Productos (creación del prototipo) como uno de Automatización y Robótica (programación de algoritmos). Así, junto con otras ramas presentes en menor medida, el proyecto aborda una situación que se puede dar a lo largo de la vida laboral de cualquier ingeniero, poniendo en valor la importancia de realizar un trabajo académico de esta índole.

Se ha aceptado este proyecto como un reto que engloba una amplia variedad de disciplinas, y obliga a realizar un fuerte trabajo de autoaprendizaje e investigación. En efecto, a pesar de tener conocimientos de muchos aspectos de este proyecto, se presentas otros muchos como enigmas de los que aprender.

Finalmente, se podría destacar la complejidad del proyecto a nivel académico, debido al nivel de conocimientos que aplicar y obtener. Además, tiene el aliciente de presentar una aplicación actual en auge, de gran interés para el sector profesional, como es el desarrollo de sistemas autónomos.

1.5. ESTRUCTURA DEL PROYECTO

La memoria se plantea según se ha realizado el trabajo. Con una estructura lineal, cada capítulo trata una serie de retos que son explicados y resueltos concluyendo en un resultado final.

A continuación, se desglosan las distintas etapas que componen el proyecto:

- Estado del Arte. Se aporta cierto contexto teórico acerca de las materias que componen este TFG (diseño 3D, impresión 3D, microcontroladores Raspberry Pi y su programación).
- Materiales y entorno de trabajo. Se describen los componentes que forman parte del proyecto, tanto el hardware como el software usado, además de los entornos de trabajo con los que se ha desarrollado el proyecto
- Diseño, modelado y ensamblado. Se explicará el proceso seguido para obtener el cuerpo del robot y el montaje del prototipo completo.
- Visión artificial. En este capítulo se hace hincapié en los fundamentos de la visión artificial, así como de las diferentes funciones que se utilizan en este proyecto para llevar a cabo el tratamiento de la información captada por la cámara.

- Algoritmo de circulación. Aquí se explica cómo se ha diseñado la programación y el flujo de estados que se lleva a cabo para automatizar la circulación del robot como un viandante.
- Conclusión. En este capítulo se concluye con los objetivos alcanzados y se resume tanto el proceso realizado para obtener los resultados como los problemas que se han encontrado.

CAPITULO. 2 ESTADO DEL ARTE

En este capítulo se aporta un breve contexto teórico acerca de las materias que comprometen este proyecto. Se pretende colocar al lector en una posición favorable para la compresión de los siguientes apartados.

2.1. EL DISEÑO 3D

El diseño 3D de las piezas se ha llevado a cabo usando aplicaciones CAD ("Computer-Aided Design", diseño asistido por computadora). Las aplicaciones CAD se han vuelto una herramienta indispensable a la hora de diseñar objetos o piezas y, por ello, existen una gran variedad de estas.

Debido a su extenso uso en el mundo de la ingeniería, las más populares son:

- Autodesk Fusion 360: es una aplicación CAD que permite no solo diseñar objetos si no también realizar simulaciones de esfuerzos.
- AutoCad: fue de las primeras aplicaciones CAD del mercado. A pesar de su amplio uso, no es el más fácil de usar.
- Creo: líder del mercado de diseño, es una herramienta completa capaz diseñar y realizar cálculos de diseño, análisis térmicos, estructurales y de movimientos.
- OpenSCAD: es un software gratuito de código abierto, diseñado para modelar solidos en 3D. Es una aplicación intuitiva para programadores, basado en el lenguaje descriptivo.
- SketchUp: es un programa especializado en el modelado de entornos urbanos, arquitectura, ingeniería civil, diseño industrial, diseño escénico, videojuego o películas. Permite el modelado en tres dimensiones basado en caras, lo que lo vuelve factible para el uso académico.
- Solidworks: de los más conocidos. Utiliza el diseño paramétrico, generando tres archivos: pieza, ensamble y dibujo. Además, presenta buenas prestaciones en la validación de diseño e ingeniería inversa.
- Catia: permite la simulación y el modelado 3D. Es utilizado especialmente en el sector aeroespacial.

El modelado 3D, en la mayoría de las aplicaciones, empieza por el dibujo de un boceto cerrado en un plano. A este boceto se le aplica una operación que transforma la superficie encerada en un volumen. Este proceso puede repetirse en cualquier boceto, este puede estar en las caras del nuevo volumen o en un plano auxiliar. De esta forma, repitiendo este proceso se crear un objeto final en tres dimensiones.

Entre los tipos de operaciones que se pueden realizar, existen unas operaciones que se pueden aplicar a los bocetos (operaciones de creación) y otras que se pueden aplicar sobre el volumen ya creado (operaciones de modificación).

Entre las operaciones de creación, las más comunes son:

- Extrusión: la superficie cerrada del boceto se proyecta ortogonalmente al plano del dibujo, generándose un volumen entre el boceto y la proyección (ver Figura 1).

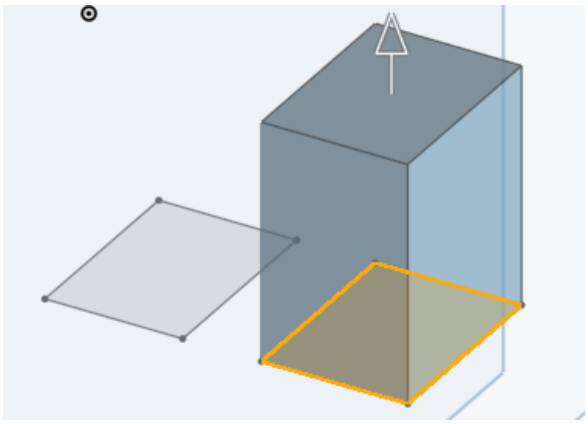


Figura 1. Operación de extrusión
Fuente: Elaboración Propia

- Revolución: en este caso se parte de un boceto y un eje. En torno a este eje se gira una proyección del boceto y se va generando un volumen (ver Figura 2).

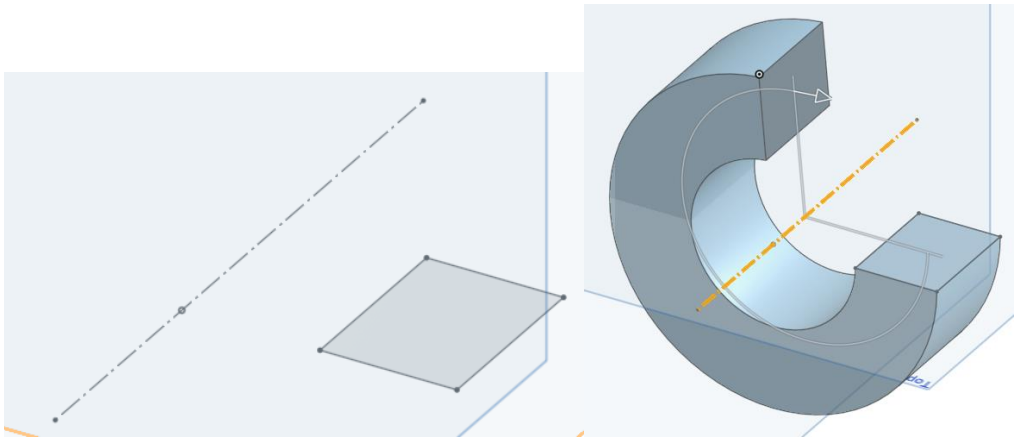


Figura 2. Operación de revolución
Fuente: Elaboración Propia

- Barrido: la proyección del boceto es arrastrada a lo largo de un recorrido previamente trazado (ver Figura 3).

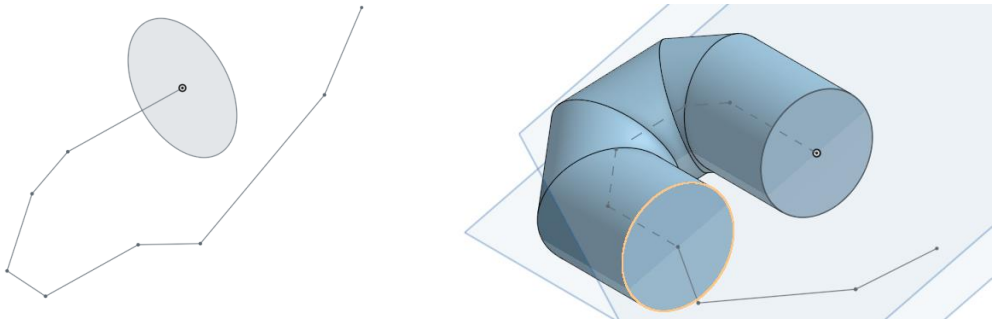


Figura 3. Operación de barrido
Fuente: Elaboración Propia

- Solevación: esta operación parte de varios bocetos, creando un volumen a partir de la unión de estos (ver Figura 4).

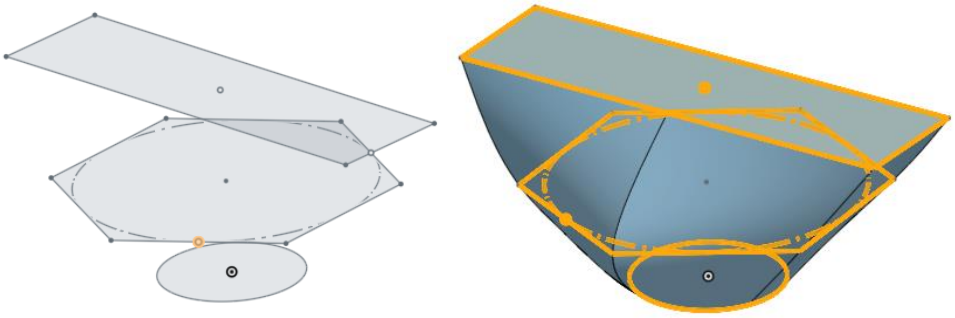


Figura 4. Operación de solevación
Fuente: Elaboración Propia

Una vez generados los volúmenes, se pueden aplicar operaciones que modifican su forma. Los más comunes son:

- Agujero: permite la creación de agujeros sin necesidad de bocetos. Además, agrupa varios parámetros para establecer las características del agujero, incluso otorgarles avellanado (ver Figura 5).

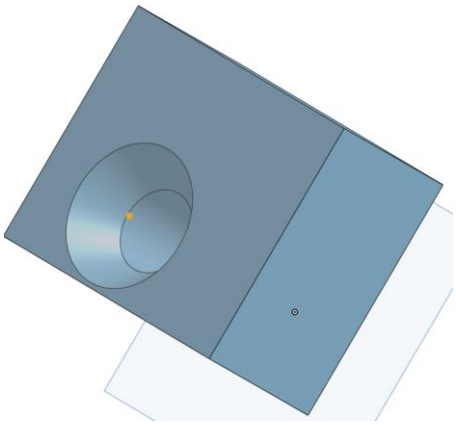


Figura 5. Operación de agujero
Fuente: Elaboración Propia

- Redondeo: crea una transición redondeada entre dos caras adyacentes del volumen (ver Figura 6)

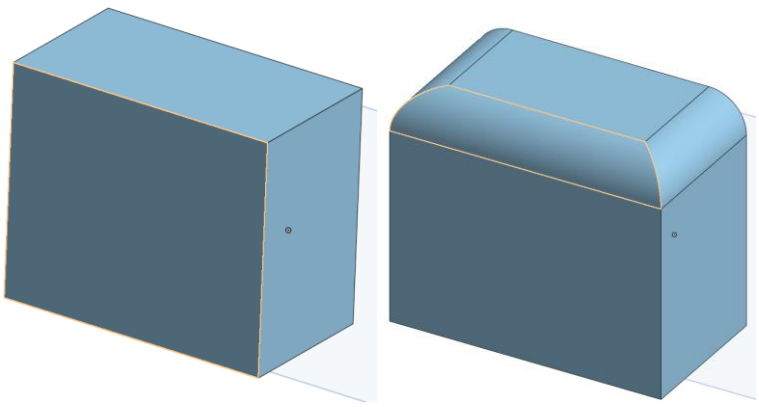


Figura 6. Operación de redondeo
Fuente: Elaboración Propia

- Chaflán: sustituye la arista que une dos caras por un plano oblicuo (ver Figura 7).

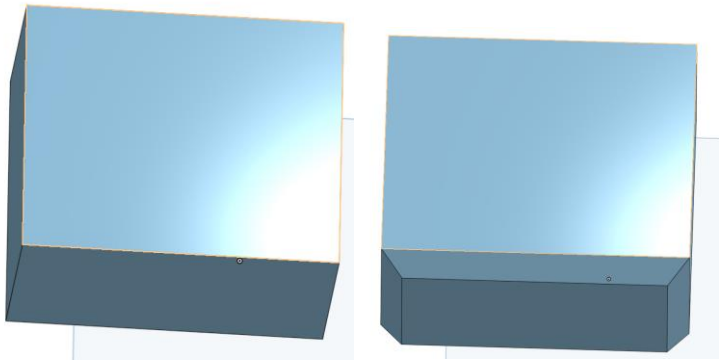


Figura 7. Operación de chaflán
Fuente: Elaboración Propia

- Vaciado: como su nombre indica, realiza un vaciado del volumen dejando un espesor modificable (ver Figura 8).

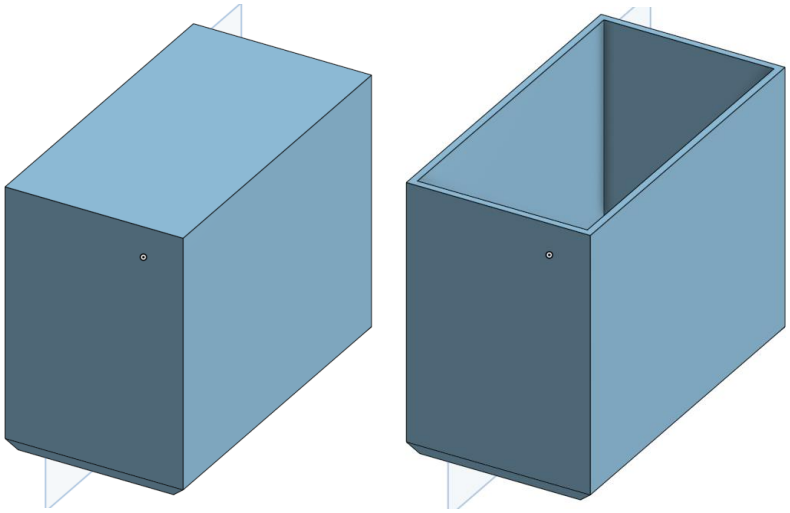


Figura 8. Operación de vaciado
Fuente: Elaboración Propia

Una de las grandes ventajas que aporta el diseño CAD la dependencia entre operaciones y piezas. De esta forma, estas mismas operaciones sirven para a delimitación de los planos técnicos de las piezas creadas, planos conjuntos y planos de explosión. Incluso se pueden crear videos que muestran el despiece secuencial.

2.2. LA IMPRESIÓN 3D

La materialización de las piezas diseñadas para este proyecto se ha hecho utilizando impresoras 3D. La impresión 3D es un extenso mundo que integra una multitud de técnicas de impresión con una gran variedad de materiales, como plásticos, metales, cementos, cerámica o incluso tejidos vivos.

Esta tecnología tiene como principio base la tecnología aditiva, que consiste en la deposición del material de fabricación en planos virtuales horizontales realizados por diseño CAD o software de rebanado. Dependiendo de la técnica que se utilice, el material o materiales de unión se depositan sobre el lecho de construcción, de tal forma que se van generando capas sucesivas de material hasta que el objeto se completa.

Existen múltiples categorías de tecnologías, con diferentes beneficios y limitaciones, que pueden ser utilizadas de forma complementaria. Sus principales diferencias se encuentran en los materiales que

son capaces de manejar y el proceso con el que materializan las capas de material que forma el objeto final. A continuación, se exponen brevemente estas categorías:

- Extrusión de material *(material extrusion*)
  Filamento termoplástico sólido a través de una boquilla caliente es depositado por capas en un estado semifluido, seguido de su enfriamiento.

- Fotopolimerización (*vat photopolymerization*)
  Resina fotosensible en un depósito sufre un endurecimiento selectivo mediante una fuente de luz.

- Proyección de material (material jetting)
  De forma selectiva, se depositan gotas de material líquido, que posteriormente solidifica. Permite imprimir diferentes materiales en un mismo objeto.

- Proyección de adhesivo (*binder jetting*)
  Similar a la proyección de material, en este caso se depositan de forma selectiva gotas de adhesivo sobre capas sucesivas de material en polvo.

- Fusión de material en polvo (*powder bed fusion*)
  Una fuente de calor funde de forma selectiva material en polvo en capas sucesivas. Se consigue la fusión entre partículas en polvo dentro de un área de construcción para generar el objeto final.

- Deposición dirigida de energía (*directed energy deposition*)
  Similar a la fusión de material en polvo, con la diferencia de que la fuente de calor funde el material en polvo a medida que la deposita. Utiliza un rayo láser o haz de electrones alimentado con polvo o hilo de metal para formar las capas de material del objeto final.

- Laminación de hojas (*sheet lamination*)
  Una tecnología en desuso debido a su desperdicio de material. Es una tecnología aditiva que coloca finas láminas de material (papel, plástico o metal) que ha sido recortado con la forma de fresador o un láser de CO_2.

Es necesario comentar algunas de las limitaciones técnicas presentes generalmente en la impresión 3D.

En primer lugar, no se puede dejar de lado la acción de la gravedad. En algunas técnicas es necesario aportar material de soporte para sustentar voladizos, puentes y partes huecas. De lo contrario, en el proceso de impresión la pieza se deformaría (ver Figura 9). Aunque este material de soporte es creado de forma automática por el rebanador (como PrusaSlicer, el empleado en el presente TFG, y del que se habla en la sección de *software*), es recomendable posicionar o diseñar la pieza evitando la creación de voladizo o puentes.

Figura 9. Ejemplo de problemas de la impresión 3D
Fuente: Imágenes extraídas de B. Redwood et al., "The 3D printing handbook", 3D Hubs, 2018

Otro problema es la resolución, o acabado de la pieza. Debido a la técnica aditiva de capas, se genera un aspecto escalonado (superficie de los objetos impresos rayados). Además, las piezas deben presentar unas medidas mínimas para que sea imprimible. Estos problemas dependen mucho de la técnica que se utilice.

Por otro lado, las piezas acabadas presentan una importante anisotropía que depende de la posición en la que se imprima el objeto. Además, pueden darse deformaciones de combado y encogimiento debido a un enfriamiento desigual. Estos problemas son de gran interés en este proyecto, debido al tamaño de las piezas y los esfuerzos que sufren.

Por último, algunas técnicas requieren en mayor o menor medida un postprocesado tras la impresión, para eliminar material de soporte o material sobrante. Además, puede ser necesario aplicar tratamientos superficiales como el lijado, pulido, sellado, pintado etc. para mejorar o cambiar el acabado o las propiedades físicas.

A continuación, la Tabla 1 muestra un resumen de las aplicaciones de cada categoría de técnicas, además de los materiales que se pueden utilizar.

TÉCNICA	**APLICACIONES**	**MATERIALES**
Extrusión de material	Prototipado rápido, plantillas y accesorios, uso final, tiradas cortas	Termoplásticos, fibra de carbono, compuestos termoplásticos más madera, compuestos termoplásticos más metal, compuestos termoplásticos más cerámica, cemento, arcillas, comida, metales
Fotopolimerización	Herramientas, núcleos/moldes para fundición, moldes para inyección, uso final, productos de consumo (uso médico, joyería)	Materiales elásticos, termoplásticos, fotopolímeros, polímeros basados en cera, compuestos fotopolímero más cerámica, resinas transparentes
Proyección de material	Prototipado, núcleos/moldes para fundición, moldes para inyección, modelado (uso médico, educacional, ...), uso final, arte	Acero inoxidable, materiales cerámicos
Proyección de adhesivo	Prototipado, núcleos/moldes para fundición, moldes para inyección, modelado (uso médico, educacional, ...), uso final	Yeso, plásticos, aleaciones, acero inoxidable, bronce, aleaciones vidrio, arena, materiales cerámicos

Fusión de material en polvo	Herramientas, uso final (piezas funcionales fuertes, conductos, tuberías, ...), tiradas cortas (uso médico, aeroespacial, automoción)	Plásticos (poliamidas), cerámicos, arena, cera, metales básicos, aleaciones, aluminio, cobre, hierro, plata, oro, titanio, acero, compuestos de nylon y vidrio, aluminio
Deposición dirigida de energía	Herramientas, uso final (piezas funcionales fuertes, conductos, tuberías, ...), tiradas cortas (uso médico, aeroespacial, automoción), reparación de piezas dañada	Metales en polvo, aleaciones, materiales cerámicos, compuestos, titanio, acero inoxidable, cobre, níquel, aceros
Laminación de hojas	Modelado (educacional), arte	Plástico, papel, cartón, corcho

Tabla 1. Resumen de las categorías de impresión 3D
Fuente: Apuntes de la asignatura de impresión 3D

Se ha decidido realizar este proyecto usando la técnica FFF/FDM (del inglés fused *filamente fabrication* / *fused deposition modeling*), perteneciente a la categoría de extrusión de material mediante una impresora 3D (Original Prusa i3 MK3S, Figura 10). Se pretende aprovechar el reducido tiempo de impresión y la robustez del objeto final, a pesar de no tener demasiada precisión o una resolución muy alta. Estas características son idóneas para realizar prototipos rápidos y de forma económica.

Figura 10. Impresora Original Prusa i3 MK3S
Fuente: https://www.impresoras3d.com/producto/impresora-3d-prusa-i3-mk3s/?network=tradetracker&utm_source=affiliate&utm_medium=360756

Las impresoras que llevan a cabo todo este proceso de impresión suelen ser máquinas de tipo CNC (Control Numérico Computarizado). Estas máquinas operan con una secuencia comandos previamente programados en un medio de almacenamiento. Esta secuencia de acciones esta almacenada en un archivo estandarizado llamado GCODE.

El GCODE es generado a partir de un archivo STL. El formato STL es uno de los formatos estándar que se utilizan en el diseño CAD, para facilitar la transferencia de archivos entre programas. De esta forma, a partir de un programa de rebanador, el archivo STL es cortado por planos para obtener los puntos del objeto que representa, y así general el GCODE con las acciones necesarias para imprimir.

2.3. COMPONENTES Y MEDIOS DE TRABAJO

En primer lugar, se describe el material usado para este proyecto: hardware, software y el entorno de trabajo, formado por las herramientas principales utilizadas para conseguir el objetivo.

2.3.1. HARDWARE

Se ha partido de un kit de un robot bípedo programable de la marca "SunFounder", del cual se pretende utilizar la mayor parte posible de los componentes. De esta forma, se han aprovechado la estructura de las piernas y la totalidad de los servos del kit.

De este kit se ha hecho uso de:

- 4 Servomotores "SunFounder"(ver Figura 11)
  Los servomotores, o servos, son motores eléctricos que poseen un circuito de control que permite controlar la velocidad y posición. Son controlados por modulación por ancho de pulsos (PWM) para establecer dirección o posición estáticas. Trabajan con un voltaje de entre 4.8 y 6V.

Figura 11.Servomotor SunFounder
Fuente: https://www.sunfounder.com/collections/servos/products/sg90-micro-digital-servo

- Estructura de las piernas
  Se ha aprovechado la estructura presente en el kit para el montaje de las piernas del robot. Así conseguimos sacar el máximo partido al kit del que se parte. Además, se han diseñado unos pies mayores que aguantan el peso del prototipo.

- Tornillos, tuercas y separadores
  Para poder realizar las uniones de los distintos elementos que componen el robot se dispone de los tronillos, tuercas y separadores presentes en el propio kit.

Debido a las limitaciones del kit, se hace mandatorio añadir equipo más potente capaz de alcanzar los objetivos del proyecto. Con esto en mente se añaden los siguientes componentes:

- Raspberry Pi 2 modelo B+ (ver Figura 12)
  Debido a que el microcontrolador que llevaba originalmente el kit (Arduino nano) no tenía potencia suficiente para poder llevar a cabo el procesamiento del video y la implementación de los algoritmos de circulación, se ha remplazado por otro más potente. Se elige este microcontrolador como alternativa más potente puesto que tiene mucha más memoria RAM (1 GB de RAM)

Figura 12.Raspberry Pi 2 modelo B+
Fuente: https://assets.mmsrg.com/isr/166325/c1/-/pixelboxx-mss-68056627/fee_325_225_png

- 6 Baterias recargables 18650 de Ión-Litio, 3.7V, 2.6Ah (ver Figura 13)
  Ni la Raspberry Pi ni los servomotores se pueden alimentar a través de pilas convencionales, ya que no aportan la corriente necesaria. Estas pilas presentan un voltaje de 3.7V y un amperaje de 2600mAh, de forma que combinando dos cubren las necesidades de los componentes. Además, se han elegido estas batería para alimentar al mismo tiempo los dos semáforos que componen la pista de pruebas.

Figura 13. Pila recargable 18650 de Ión-Litio, 3.7V, 2.6Ah
Fuente: https://es.rs-online.com/web/

- SAI UPS HAT (ver Figura 14)
  Es un sistema de alimentación ininterrumpida (SAI) que permite tanto cargar las baterias que alimentan la Raspberry Pi, como ajustar el voltaje y la corriente que necesita la Raspberry Pi (5V y 2A) para trabajar con las pilas 18650. Se monta superponiéndolo a los 40 pines del microcontrolador, y permite monitorizar el estado de las baterías y tomar decisiones cuando queda poca batería.

Figura 14.Sistema de alimentación ininterrumpida UPS HAT
Fuente: https://www.tiendatec.es/6399-large_default/hat-sai-ups-para-raspberry-pi.jpg

- Adaptador wifi (ver Figura 15)
  El TP-LINK TL-WN725N es necesario para establecer una conexión remota con la Raspberry, debido a que este modelo no dispone de modulo wifi integrado.

Figura 15. Adaptador Wifi TP-LINK
Fuente: https://www.powerplanetonline.com/cdnassets/tl_wn725n_01_l.jpg

- Controlador de servomotores PCA9685 (ver Figura 16)
  Se trata de un controlador por longitud de pulso, capaz de generar 16 señales PWM para controlar 16 servos. Se conecta a las Raspberry Pi por el canal I2C, y se alimenta a través de los pines de tierra y 3.3V. Los servomotores han de ser alimentados de forma independiente.

Figura 16.Controlador de servomotores PCA9685
Fuente: https://hifisac.com/web/image/product.template/2690/image_1024?unique=55e1006

- Módulo de cámara v2 8MP (ver Figura 17)
  Con este módulo se captura el entorno y se realizan acciones en consecuencia. Es capaz de lidiar con una resolución de foto de 3280 x 2464 y de 1080p30, 720p60 y 640X480p90 en video, siendo esta última la que se va a utilizar para la captura de video.

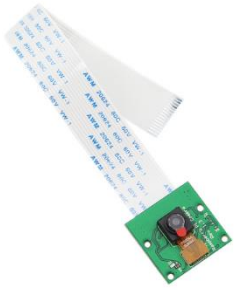


***Figura* 17.Modulo cámara Raspberry Pi**
Fuente: https://electronilab.co/wp-content/uploads/2013/07/Camara-para-Raspberry-Pi-5MP-Gen%C3%A9rica-Electronilab-1.jpg

- 2 Servomotores SG90 RC 9g (ver Figura 18)
  Son necesarios dos servomotores adicionales para controlar la posición de la cámara de forma independiente.

Figura 18. Servomotor SG90 RC 9g
Fuente: https://uelectronics.com/producto/servomotor-sg90-rc-9g/

- 2 Portapilas 18650 (ver Figura 19)
  Tanto los semáforos como los servomotores tienen como fuente de alimentación baterias recargables tipo 18650 de 3.7V (2600 mAh) que se colocan en este portapilas de dos pilas en específico.

Figura 19.Portapilas 18650
Fuente: http://d3ugyf2ht6aenh.cloudfront.net/stores/648/727/products/u0163-clear11-ca843d47a95df1659715783405429550-640-0.png

- Cables de conexión
  Se han usado cables con conectores hembra-hembra para la conexión eléctrica de los componentes que conforman el robot, y conectores hembra-macho para el circuito eléctrico de los semáforos.

Por último, se ha recreado una pequeña pista que imita un cruce urbano para poder realizar pruebas y validar el funcionamiento de la programación, para lo que se ha utilizado:

- Arduino nano (ver Figura 20)
  Proveniente del kit original, es un microcontrolador mucho menos potente que la Raspberry Pi. Sin embargo, es perfecto para el control de los dos semáforos que componen el cruce. De esta forma, alimentado por una pila y a través de un sencillo programa, controla cuatro ledes uno verde y otro rojo por cada semáforo estableciendo cuando ha de cruzar el robot.

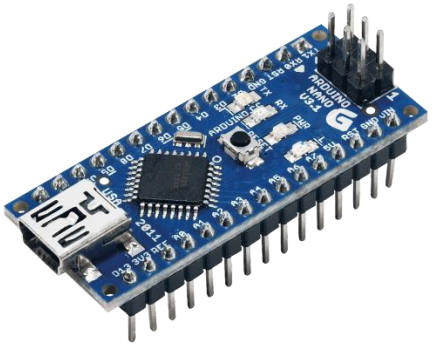


Figura 20.Arduino nano
Fuente: https://cdn.bodanius.com/media/1/73b102980_Arduino-Nano-R3_x.png

- Adaptador de fuente MB102 (ver Figura)21
  Se hace necesario, al igual que con la Raspberry Pi, el uso de una etapa de potencia para alimentar el Arduino nano a partir de las baterías 18650.

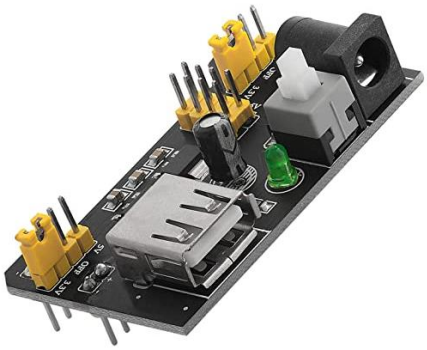


Figura 21.MB102 Placa de prototipado adaptador de fuente
Fuente: https://www.amazon.es/protoboard-AZDelivery-830-placa de prototipado-adaptador-corriente/dp/B06X962SPW/

- Placa de prototipado, ledes y resistencias
  El circuito eléctrico del semáforo está compuesto por dos ledes y dos resistencias; todo montado en una placa de prototipado con el fin de evitar soldar, y facilitar la modificación del circuito de control.

Cabe destacar que, la elección de las fuentes de alimentación ha conllevado problemas. En un principio, se intentó alimentar los servos a través de una pila de bloque de 9V, pero no aportaba la suficiente corriente, por lo que se pasó a las baterías de tipo 18650. Por otro lado, se pensó que con dos pilas de litio bastaría para alimentar el microcontrolador. Sin embargo, fue necesario utilizar un SAI.

2.3.2. SOFTWARE

El software utilizado para este proyecto se puede clasificar en dos tipos; el usado en el diseño y modelado de la estructura del robot, y el usado para la programación del robot, tanto para el sistema de visión artificial como para el control del movimiento.

Para la parte del diseño de la estructura del prototipo para la impresión 3D, se ha utilizado la herramienta online Onshape. Onshape es un software de diseño asistido por ordenador (CAD) que hace uso de tecnología “cloud”, lo que permite que toda la carga de procesamiento se realice en servidores externos, y no en la computadora local. A su vez, gracias a esta tecnología, permite compartir modelos con otros usuarios, de forma que pueden trabajar en ellos al mismo tiempo. El software acompaña al diseñador en todo el proceso de diseño, desde la creación del modelo, hasta la exportación para la impresión en 3D y la creación de los planos del modelo.

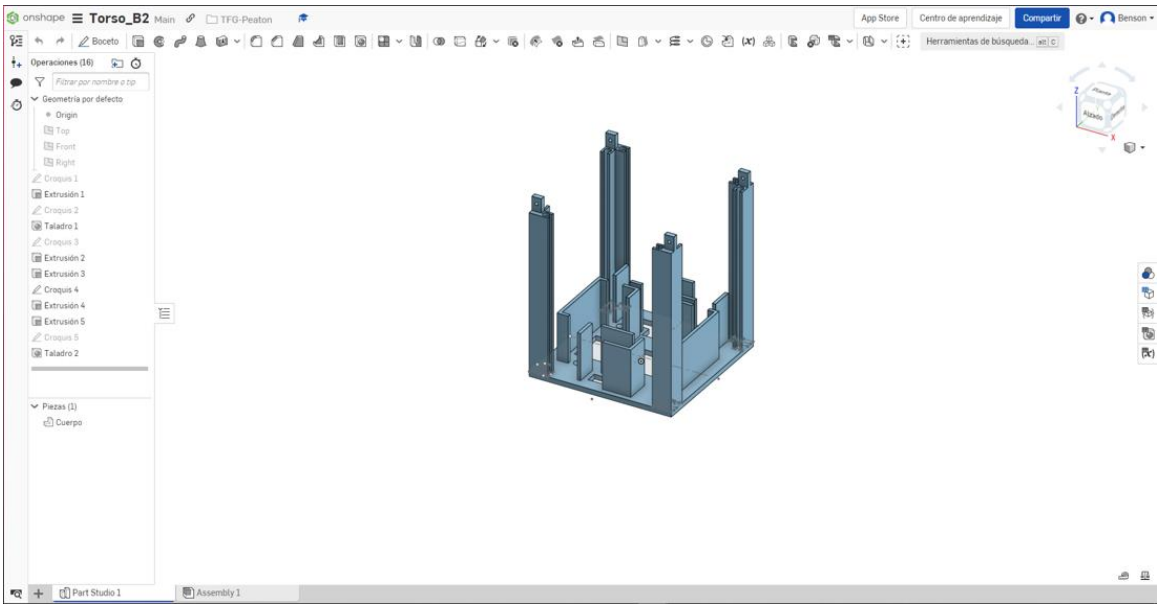


Figura 22.Espacio de trabajo Onshape
Fuente: Elaboración Propia

Junto con Onshape, se ha utilizado PrusaSlicer, un programa de rebanado proporcionado por Prusa, específico para las impresoras 3D de esta marca Prusa. Este programa es fundamental para hacer realidad los prototipos diseñados. A través de este programa se establecen los parámetros deseados para la impresión en 3D de las piezas modeladas.

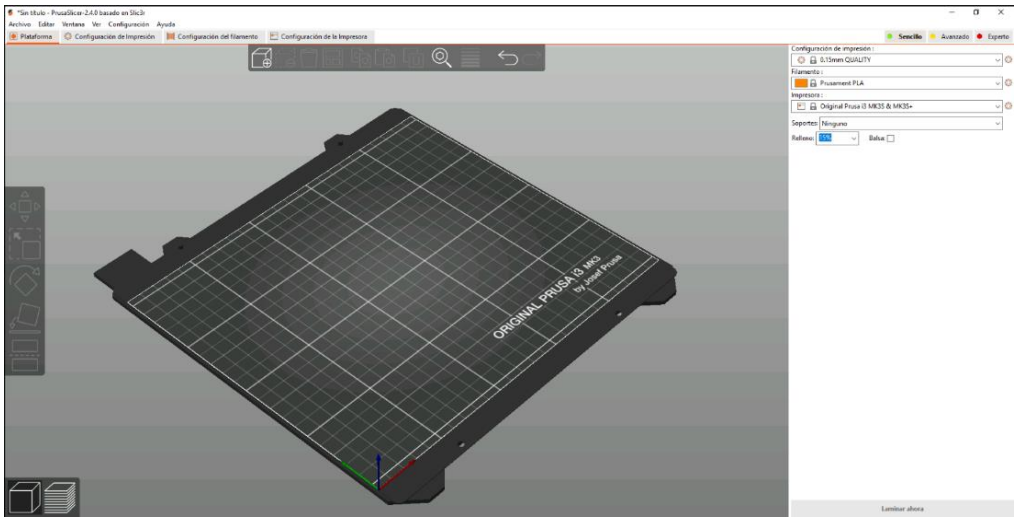


Figura 23.Espacio de trabajo PrusaSlicer
Fuente: Elaboración Propia

En cuanto a los medios de programación, se ha utilizado el sistema operativo propio de la Raspberry Pi (Raspbian SO). Este sistema operativo es gratuito y está proporcionado por la propia página oficial de la Raspberry, así como un sencillo método de instalación e implementación.

Inicialmente, a la hora de configurar la Raspberry Pi se han presentado complicaciones debidas a que el sistema operativo presente no era la última versión. Se decidió instalar el ultimo sistema operativo Rasbian, el más completo, en una nueva tarjeta SD con mayor capacidad. A través de un .exe de la página oficial se instala de forma automática en la tarjeta.

El propio sistema operativo trae consigo un entorno de desarrollo integrado (IDE) basado en Python, llamado Thonny, el cual se ha utilizar para crear los distintos algoritmos de control del robot. Thonny se centra en una programación de evaluación paso a paso en Python, permitiendo visualizar la evolución del programa y haciéndolo muy atractivo para programadores noveles.

Figura 24.Espacio de trabajo Thonny
Fuente: Elaboración Propia

2.3.3. ENTORNO DE TRABAJO

En un primer momento, se hace uso de un ordenador personal para la instalación del sistema operativo en la Raspberry Pi, y posteriormente, se trabaja directamente en la Raspberry Pi a través de periféricos conectados al microcontrolador, para la elaboración del programa.

Por otro lado, para poder trabajar y realizar pruebas con el robot se hace también necesario un control remoto de la Raspberry Pi. Esto se consigue a través del programa Virtual Networking Computing (VNC), el cual es un software libre basado en una estructura cliente-servidor que permite observar las acciones del microcontrolador remotamente a través de un ordenador o dispositivo móvil. De esta forma atreves de una misma conexión wifi y conociendo la IP de la Raspberry Pi, se puede establecer conexión remota (ver Figura 25) como si de conexión directa se tratara (ver Figura 25).

A lo hora de configurar la red wifi, debido a que el modelo 2 B de la Rapsberry Pi no cuenta con modulo wifi, y necesita un adaptador externo para detectar redes wifi, ha sido necesario establecer la conexión de forma manual. Después de intentar diferentes métodos, gracias a la creación y modificación de algunos documentos del sistema se ha logrado una conexión permanente a la red.

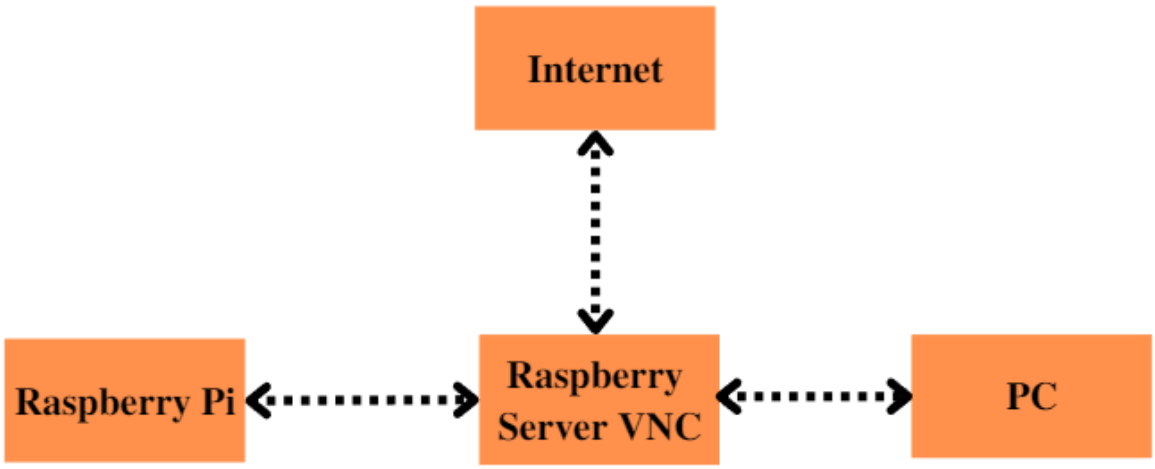


Figura 25.Esquema de conexión VNC
Fuente: Elaboración Propia

CAPITULO. 3 DISEÑO, MODELADO Y ENSAMBLADO

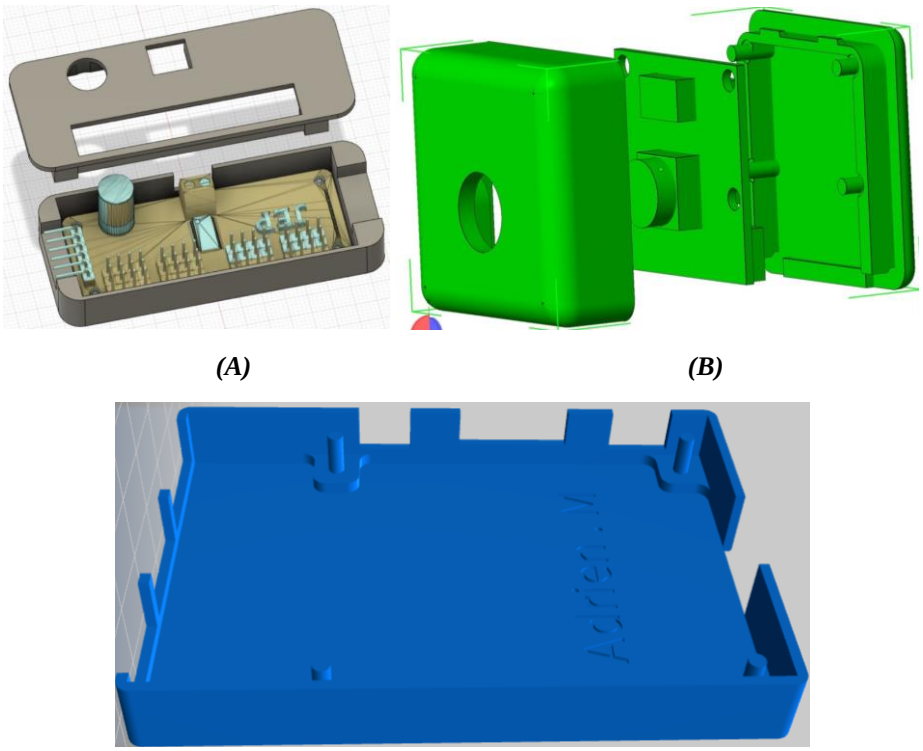
3.1. ALCANCE

A la hora de desarrollar este proyecto se ha tenido como meta un diseño funcional y escalable, capaz de ser fácilmente modificado para llevar a cabo funciones adicionales. Todo esto ciñéndose a un aspecto humanoide, sin alejarse del aspecto de un viandante. Al mismo tiempo que se ha intentado sacarle el máximo partido al kit de SunFounder del que parte todo este proyecto.

3.2. DISEÑO 3D

3.2.1. Carcasas

Para facilitar el manejo temprano y la protección de los componentes, se han utilizado diseños de carcasas para la Raspberry Pi, el módulo de cámara y el controlador de los servos (ver Figura 26). Estos diseños han sido descargados desde la página web Thingverse, destinada a compartir diseños para impresión 3D. Gracias a esto, se ha ahorrado tiempo y trabajo, además de espacio.



(A) ***(B)***

(C)

Figura 26.Carcasas de protección. (A) Carcas PCA9685. (B) Carcas del módulo de la cámara. (C) Carcasa inferior para la Raspberry Pi

Fuente: (A) ***https://www.thingiverse.com/thing:4924461*** || (B) ***https://www.thingiverse.com/thing:92208*** || ***(C) https://www.thingiverse.com/thing:559858***

3.2.2. Cabeza

La cabeza comporta la estructura que soporta la cámara del prototipo. El diseño está compuesto por una pieza que une la cabeza con el cuerpo (ver Figura 27 pieza gris), y otra donde se acopla la cámara (ver Figura 27 pieza azul), proporcionando a la estructura capacidad de rotación en loa ejes X y Z. El diseño permite a ajustar el ángulo de visión de la cámara, gracias al uso de la carcasa y un soporte a presión (ver Figura 27 pieza azul).

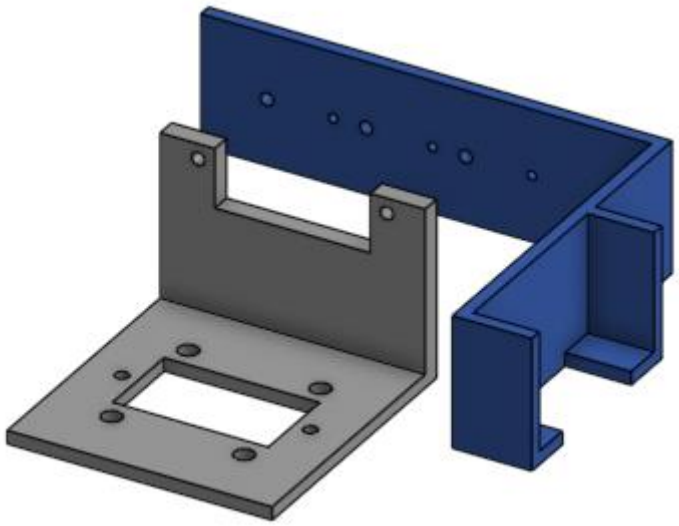


Figura 27.Conjunto de la Cabeza
Fuente: Elaboración Propia

A continuación, se presentan las especificaciones de impresión de cada pieza:

	Soporte del Cuello	Soporte de la Cámara
Filamento (m)	*1,55*	*2,53*
Filamento(g)	*4,61*	*7,54*
Coste Pieza	*0,14*	*0,23*
Tiempo de impresión	*43 min*	*43 min*

Tabla 2. Especificaciones y tiempos de impresión del conjunto de la cabeza
Fuente: Rebanador PrusaSlicer

3.2.3. Cuerpo

El cuerpo contiene en su interior todas las piezas principales del prototipo. Con el objetivo de ahorrar espacio y facilitar el montaje, se ha optado por un diseño vertical, donde los elementos internos van sujetos por guías sin necesidad de tornillos (ver Figura 29). Así, se consigue un aspecto alargado y la altura deseada. También un cierre de cuerpo con huecos que permiten la conexión de periféricos (ver Figura 28)

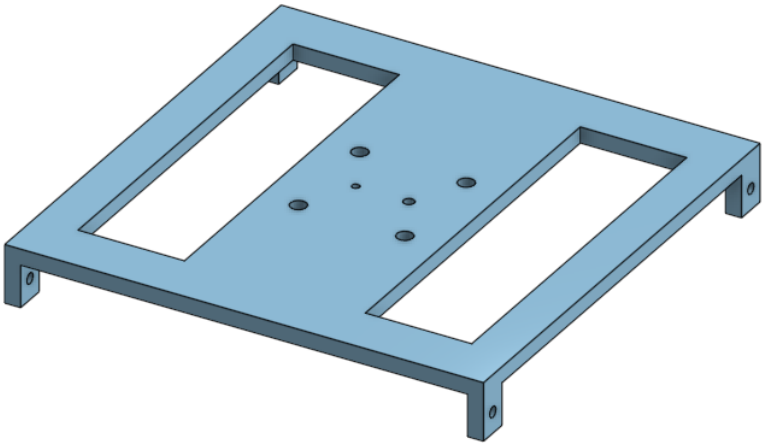


Figura 28. Cierre del cuerpo
Fuente: Elaboración Propia

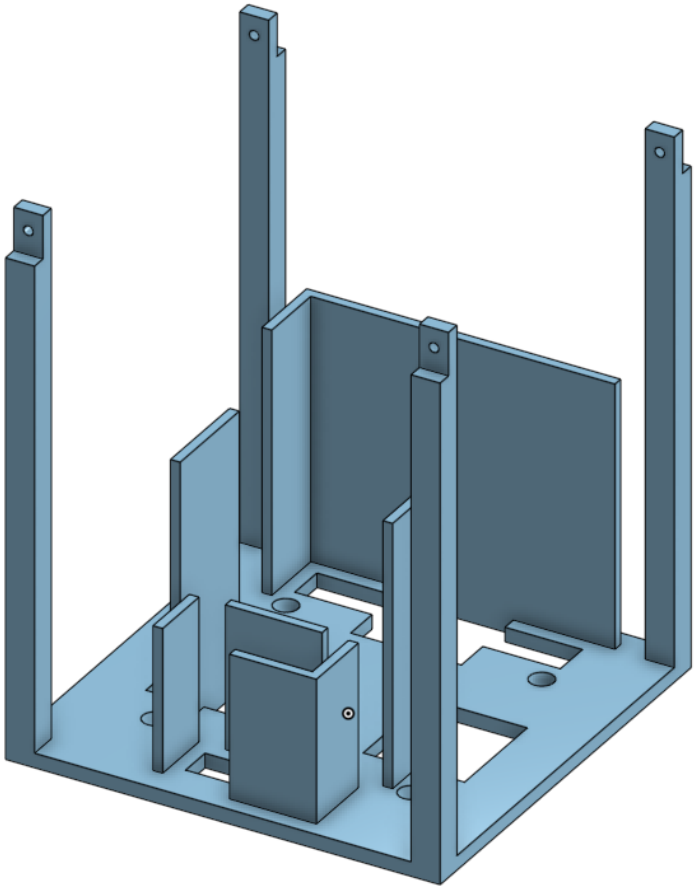


Figura 29. Cuerpo
Fuente: Elaboración Propia

La carencia de paredes laterales permite manipular el circuito eléctrico sin necesidad de desmontar el prototipo. Al mismo tiempo, permite conectarse directamente al microcontrolador o conectarlo para cargar las baterías.

A continuación, se presentan las especificaciones y tiempos de impresión de las piezas:

	Cuerpo	Cierre del Cuerpo
Filamento (m)	*16,56*	*3,38*
Filamento(g)	*49,40*	*10,09*
Coste Pieza	*1,49*	*0,31*
Tiempo de impresión	*5h 59 min*	*58 min*

Tabla 3. Especificaciones y tiempos de impresión de las piezas del cuerpo
Fuente: Rebanador PrusaSlicer

3.2.4. Piernas

Como se ha comentado anteriormente, se ha intentado sacar el máximo partido al kit original. Sin embargo, debido al peso de los componentes y la naturaleza de los servos originales de SunFounder, ha sido necesario modificar los pies.

Se han diseñado unos pies de mayor tamaño (ver Figura 30), consiguiendo una mayor área de apoyo y más equilibrio; al mismo tiempo, se ha añadido un elemento de apoyo que facilita al robot apoyarse en unos de sus pies y así poder dar pasos. Sin este elemento de apoyo, los motores de "SunFounder" no son capaces de aguantar el peso del prototipo a la hora de bascular para caminar.

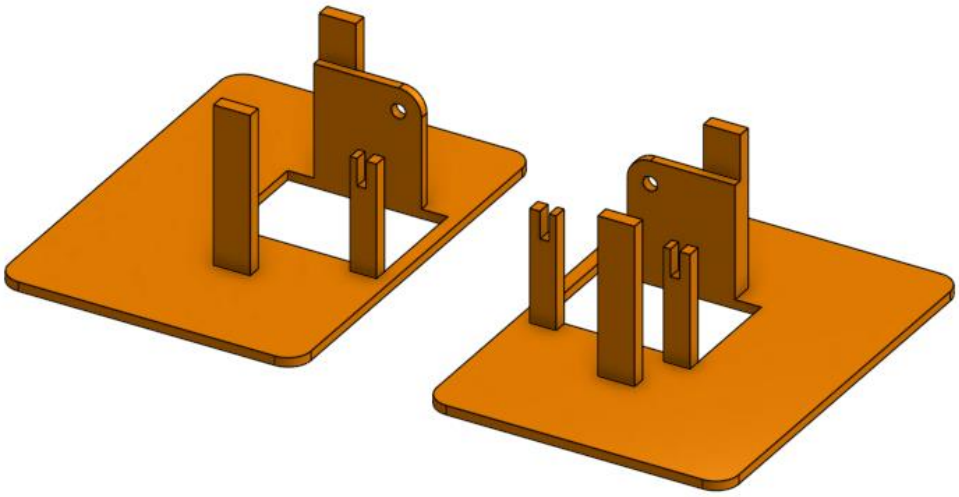


Figura 30. Pies
Fuente: Elaboración Propia

A continuación, se presentan las especificaciones de las piezas:

		Pies
Filamento (m)		7,34
Filamento(g)		21,90
Coste	Filamento	
	Pieza	0,66
Tiempo de impresión		2h 24 min

Tabla 4. Especificaciones y tiempos de impresión de ambos pies simultáneamente
Fuente: Rebanador PrusaSlicer

3.2.5. Semáforos

En el diseño de los semáforos, se ha buscado que sea compacto, y que integre todos los elementos que se necesitan para controlar los ledes. La estructura se puede dividir en tres secciones:

1. La cabeza, en la que se ubican los ledes.
2. La base, en la que se aloja la circuitería.
3. El báculo por el que circulan los cables de la base a la cabeza.

La cabeza está conformada por dos piezas que se unen a través de un sistema de deslizamiento con una pestaña final, que hace de tope del recorrido y cierre del conjunto (Figura 31).

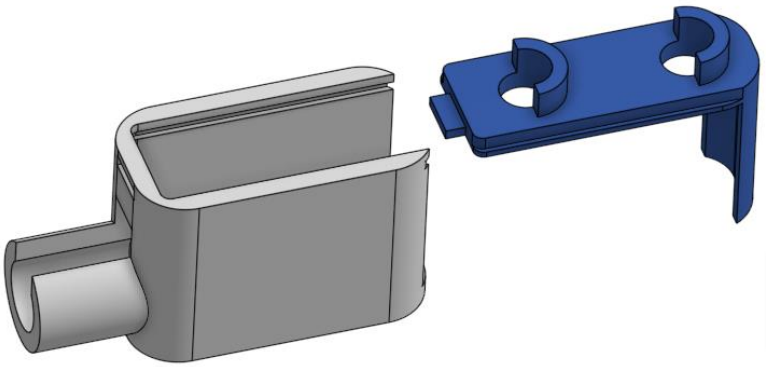


Figura 31. Cabeza del semáforo
Fuente: Elaboración Propia

Los cables se sitúan en el báculo del semáforo (ver Figura 33) que acaba en el soporte de la circuitería (ver Figura 32). El soporte de la circuitería consta de agujeros que reducen la cantidad de material y el tiempo de impresión.

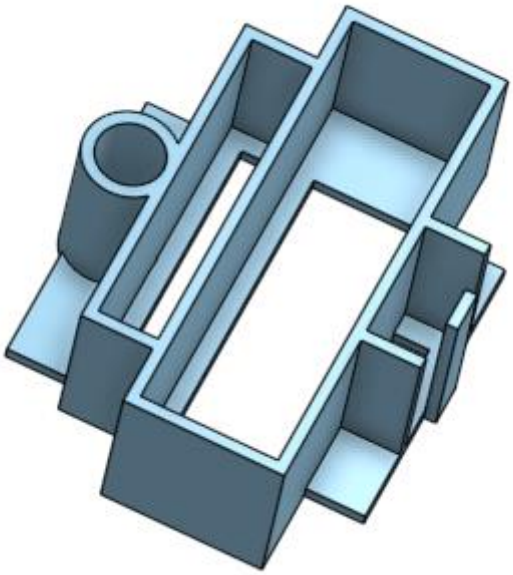


Figura 32. Soporte del semáforo
Fuente: Elaboración Propia

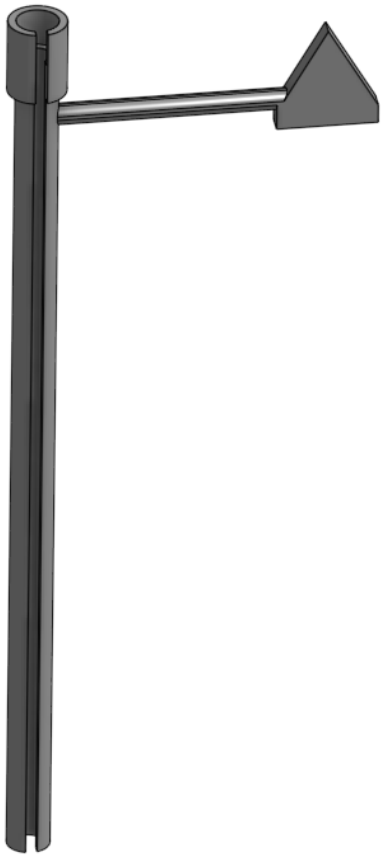


Figura 33. Báculo del semáforo
Fuente: Elaboración Propia

A continuación, se presenta los detalles de la impresión de las piezas:

	Cabeza semáforo	Soporte semáforo	Baculo semáforo
Filamento (m)	*2,04*	*8,52*	*7,91*
Filamento(g)	*6,08*	*25,41*	*23,60*
Coste (€) Pieza	*0,18*	*0,77*	*0,71*
Tiempo de impresión	*47 min*	*2h 9 min*	*2h 11 min*

Tabla 5. Especificaciones y tiempos de impresión d las partes del semáforo
Fuente: Rebanador PrusaSlicer

3.2.6. Impresión 3D

Una vez diseñadas las piezas en Onshape, se exportan como archivos STL y se llevan al rebanador PrusaSlicer para generar el fichero GCODE. El GCODE, como se comenta en la sección de impresión 3D, contiene todas las instrucciones que ha de seguir la impresora para imprimir las piezas.

Los parámetros usados en el rebanador PrusaSlicer para crear el GCODE son los siguientes:

Altura de capa		*0,2 mm*
Nº de perímetros y de capas superiores e inferiores		*2*
Patrón de relleno	*Para piezas no planas*	*Cúbico Adaptativo*
	Para piezas planas	*Rejilla*
Patrón de capas superiores e inferiores		*Rectilíneo*

Tabla 6.Parámetros de impresión
Fuente: Rebanador PrusaSlicer

La mayoría de las piezas diseñadas debido, a la existencia agujeros pasantes, voladizos o cavidades, necesitan de la utilización de material de soporte. Por ello es necesario indicar la generación automática de material de soporte a la hora de obtener el GCODE en el rebanador PrusaSlicer. Con esto se evitarán problemas de deformación comentados en la sección de Impresión 3D.

3.2.7. Esquemas eléctricos

En este apartado se muestran los esquemas de conexiones eléctricas de los subsistemas del proyecto.

En primer lugar, se muestra el del androide (ver Figura 34). Todos los servomotores se conectan a la PCA9685 y está a la Raspberry Pi la, cual tiene acoplado el SIA UPS HAT. La cámara se conecta en un puerto especifico situado entre la entra HDMI y la salida de audio. La conexión de los servos a la PCA9685 es la siguiente:

- Pierna izquierda. El servo que rota la pierna (el más cercano al cuerpo del robot) se conecta a la salida número 5, mientras que el servo que controla la rotación del pie del robot se conecta a la salida número 6.
- Pierna derecha. El servo más cercano al cuerpo se conecta a la salida 7 y el servo que controla el pie se conecta a la salida 8.
- Conjunto de la cabeza. El servo que aporta la rotación en el eje Z se conecta a la salida número 1, y el servo que rota la cámara en el eje X se conecta a la salida número 3.

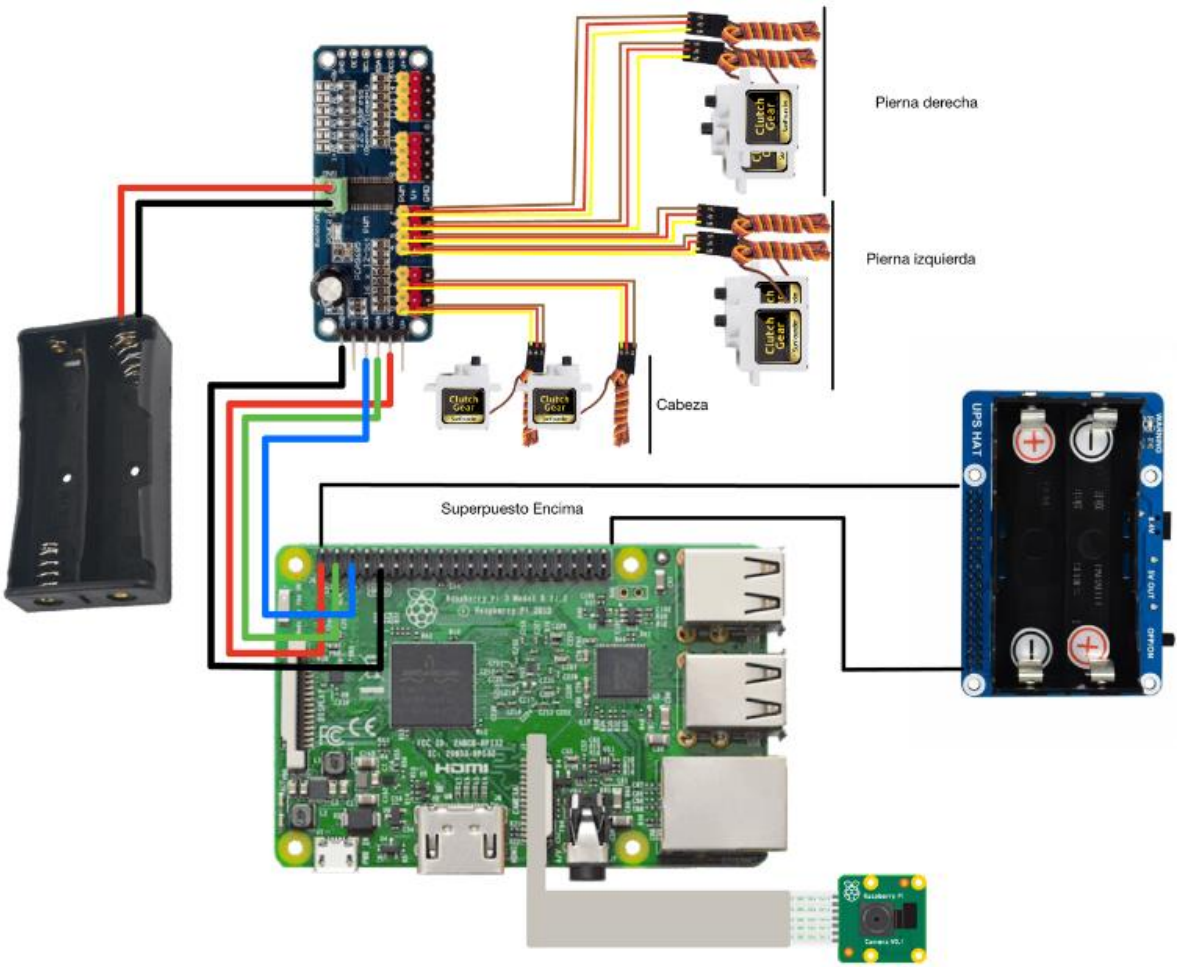


Figura 34.Esquema eléctrico del prototipo
Fuente: Elaboración Propia

Por último, la Figura 35 muestra el esquema de conexiones del conjunto de los semáforos del banco de pruebas. Observamos dos circuitos, uno para los dos ledes rojos y otro para los otros dos ledes verdes. Ambos circuitos son iguales, los ledes están conectados en paralelo, y en serie con una resistencia. A su vez se conecta un pin de salida de Arduino nano con los ledes (salida D8 para los ledes verdes y salida D9 para los rojos). Por último el adaptador de fuente alimenta al Arduino a través del pin de 5V,

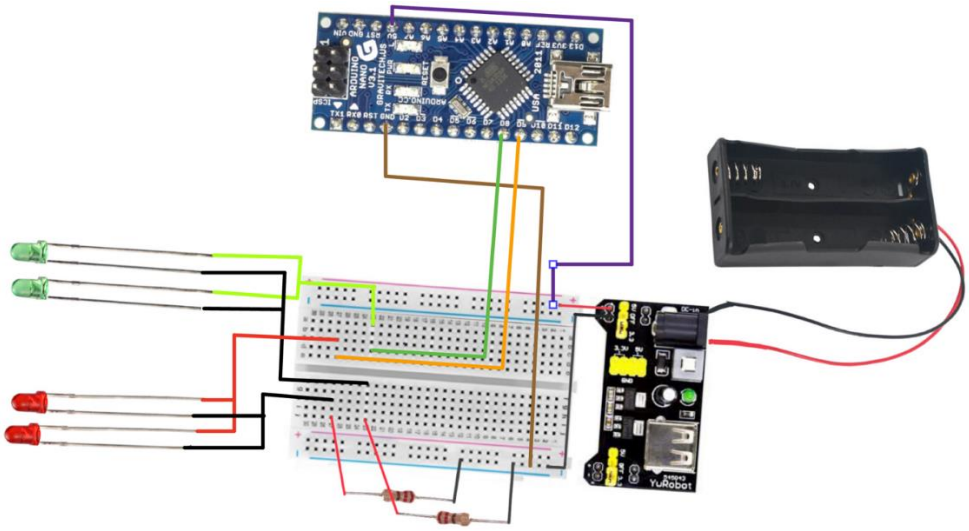


Figura 35. Esquema eléctrico del sistema de semáforos
Fuente: Elaboración Propia

3.3. MONTAJE

El montaje del prototipo empieza por las piernas. Como se han utilizado las piezas originales del kit, hay que seguir las instrucciones oficiales, sustituyendo los pies originales por los diseñados para este proyecto (ver Figura 36).

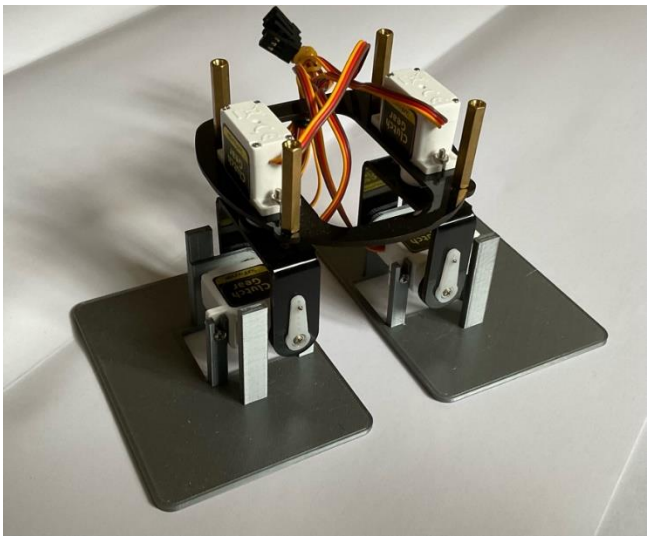


Figura 36. Montaje de las piernas
Fuente: Elaboración Propia

Posteriormente se ensambla mediante tornillos el cuerpo del prototipo encima del conjunto de las piernas (ver Figura 37), como si de la pieza original del kit se tratara.

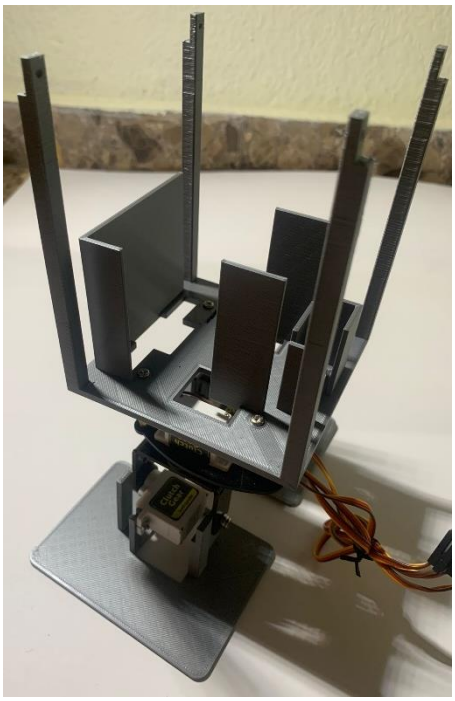


Figura 37. Acoplamiento del cuerpo con las piernas
Fuente: Elaboración Propia

Una vez con la estructura principal, se monta el esquema eléctrico como en la Figura 38 y se desliza por las guías presentes en el cuerpo (ver Figura 39).

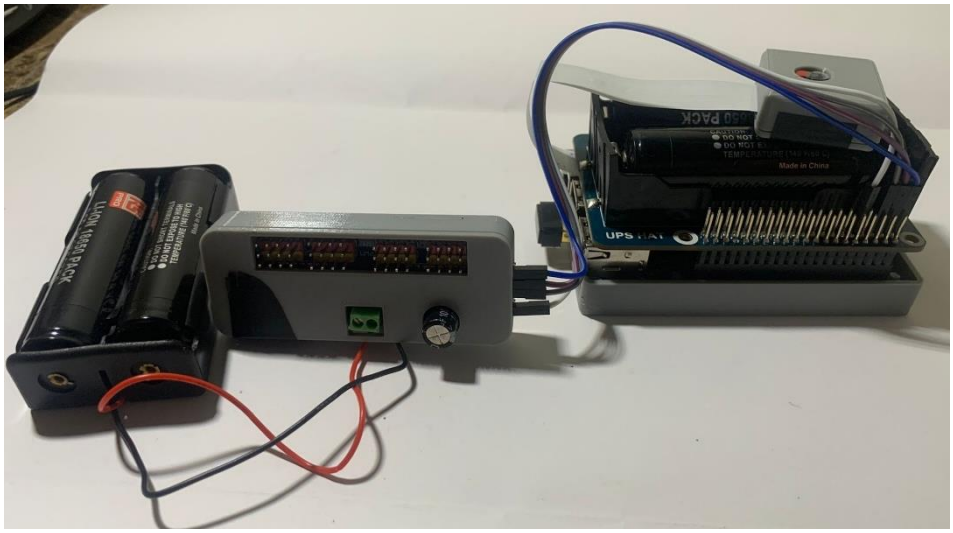


Figura 38. Montaje del esquema eléctrico del prototipo
Fuente: Elaboración Propia

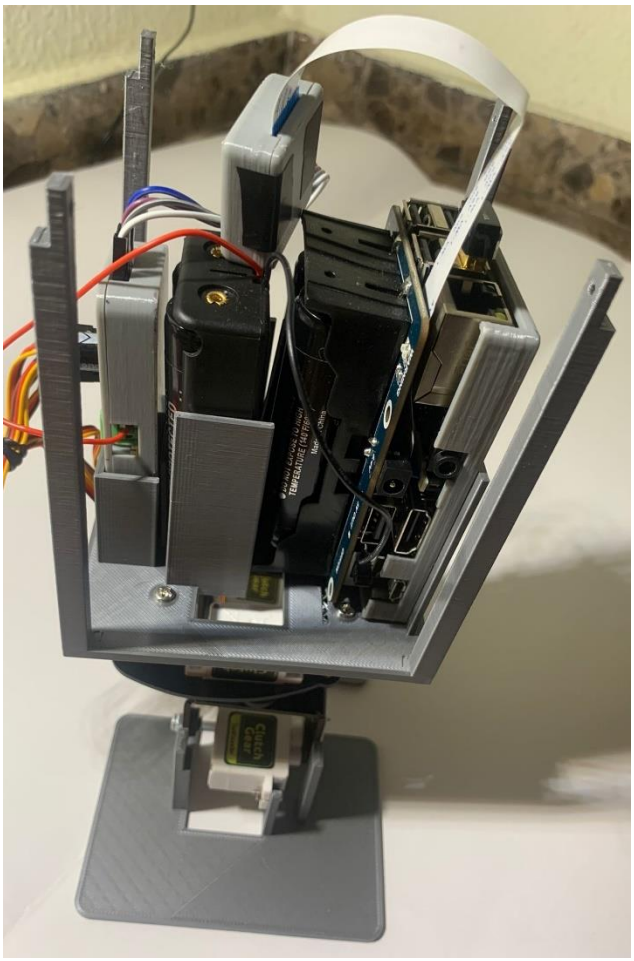


Figura 39. Montaje de los componentes electrónicos en el cuerpo
Fuente: Elaboración Propia

Por otro lado, se atornillan ambos servomotores a la pieza que hace la función de cuello, y posteriormente el soporte de la cámara al eje del servo (ver Figura 40).

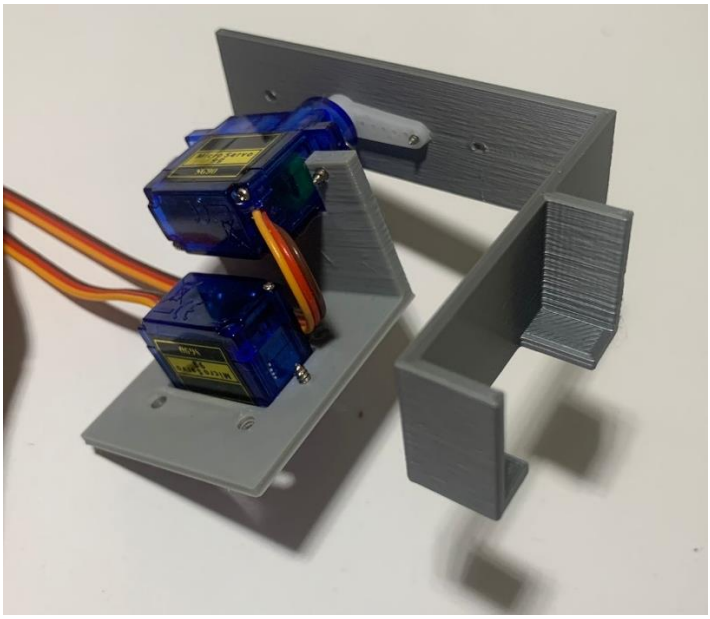


Figura 40. Montaje del conjunto de la cabeza
Fuente: Elaboración Propia

Así, una vez montado el conjunto de la cabeza, se atornilla el servomotor inferior la pieza que hace de tapa para el cuerpo (ver Figura 41).

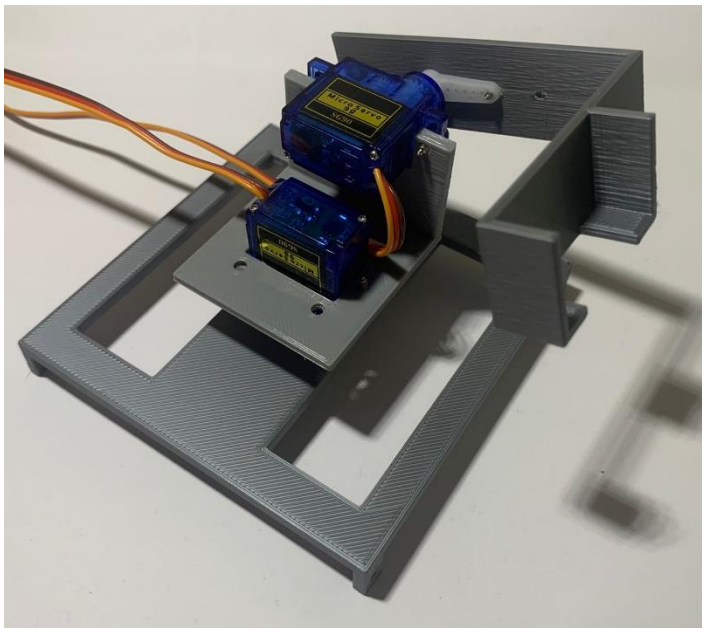
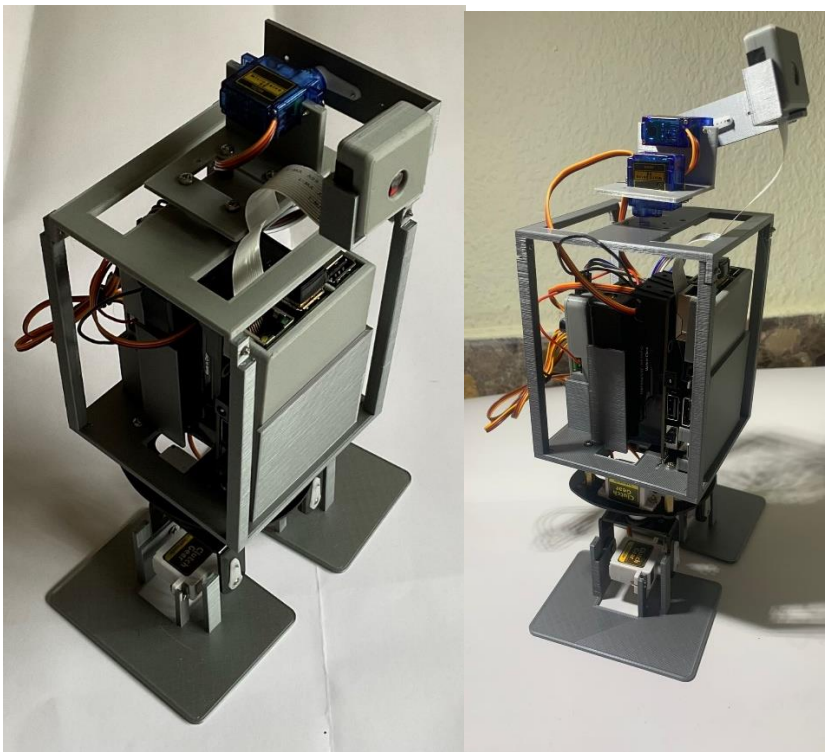
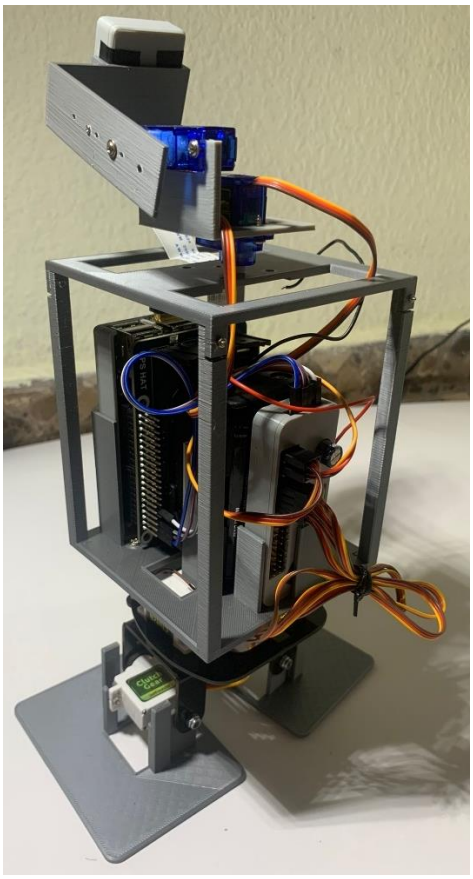


Figura 41. Unión de la cabeza con el cierre del cuerpo
Fuente: Elaboración Propia

Por último, se ensambla mediante tornillos las dos partes del prototipo (cabeza y cuerpo) completando el montaje del mismo (ver *Figura 42*).



(A) ***(B)***



(C)

Figura 42. Montaje completo del prototipo. (A) Vista frontal desde arriba. (B) Vista lateral. (C) Vista trasera
Fuente: Elaboración Propia

CAPITULO. 4 VISIÓN ARTIFICIAL

La visión artificial es una potente disciplina científica, que busca ser capaz de, a partir de imágenes del mundo real, obtener información numérica que pueda ser tratada y procesada por una computadora, de forma similar a lo que se produce en los ojos humanos. El objetivo último es conseguir estrategias automáticas para el reconocimiento de patrones complejos en imágenes en múltiples dominios.

4.1.1. OPENCV

Para la implementación de visión artificial a un nivel sencillo, se ha utilizado la biblioteca *OpenCV* de *Python*. *OpenCV* es una biblioteca *open-source* desarrollada por Intel que actualmente es la más usada, debido a la gran cantidad de áreas de aplicación que tiene. Gracias a que engloba un gran número de funciones de tratamiento de imágenes, presenta una gran versatilidad a la hora de realizar funciones como el *machine learning*, reconocimiento facial, segmentación, *tracking*, etc.

Junto con esta biblioteca, se hace necesaria, la biblioteca *Numpy*. Puesto que las imágenes se tratan como matrices y vectores multidimensionales, se necesitan operaciones capaces de tratar con matrices. Numpy aporta la capacidad de operar con matrices multidimensionales grandes, siendo en definitiva estas matrices representaciones de la imagen que captura la cámara.

4.1.2. FUNCIONES

En el presente proyecto tan solo se ha utilizado una porción de la amplia gama de funciones que ofrece la biblioteca *OpenCV*. En este capítulo se describen algunas de las funciones clave en este proyecto.

Para la captura del video se ha utilizado `<< cap = cv2.VideoCapture(0) >>` para establecer la conexión con el módulo de cámara, y a través de `<<ret,frame=cap.read()>>` se han capturado los datos de cada fotograma para posteriormente procesarlos. En esta última función, `ret` es una variable booleana que indica si ha recibido datos correctamente, y `frame` es el fotograma del video (ver Figura 43) representado en una matriz tridimensional (RGB). Esta matriz tiene como tamaño la resolución del video (640x480) y las tres dimensiones representan la tonalidad de cada pixel en formato RGB (*Red*,*Green*,*Blue*) entre 0 y 255.

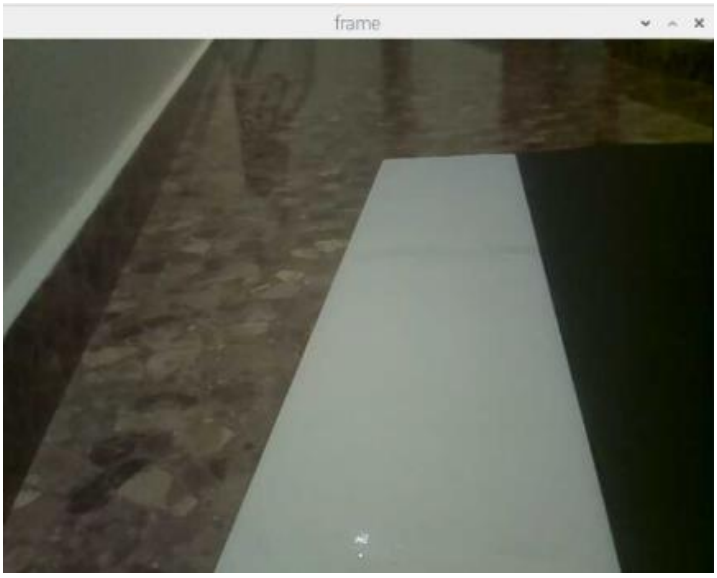


Figura 43.Imagen original capturada
Fuente: Elaboración Propia

Una vez separado el video en imágenes (fotogramas) se deben procesar estas para poder obtener información fiable de las mismas.

Para la detección de la acera se comienda aplicando un filtro gaussiano `<<gaussiano=cv2.GaussianBlur(frame,(k,k),α)>>`. Se trata de un filtro paso bajo que deja pasar los elementos más comunes, y que ocupan mayor superficie en la imagen; así se elimina ruido de la cámara que emborrona la imagen. Como se observa en la Figura 44, se obtiene una imagen un poco más borrosa, con menos detalles. Esto facilita el procesado del fotograma y el reconocimiento de objetos.

Figura 44.Filtro Gaussiano
Fuente: Elaboración Propia

A continuación, se atenúan los colores de la imagen haciendo uso de `<<gray=cv2.cvtColor (gaussiano, cv2.COLOR_BGR2GRAY) >>`. De esta forma, la imagen de color (RGB) pasa a una imagen en escala de grises (ver Figura 45), que se representa mediante una matriz unidimensional con valores de 0 a 255, siendo 0 negro y 255 blanco.

Figura 45.Imagen en escala de grises
Fuente: Elaboración Propia

Tras esto se puede detectar fácilmente bordes gracias a `<<edges=cv2.Canny(gray,threshold_one,threshold_two,aperture_size)>>`. Este filtro se encarga de detectar y acentuar los saltos de grises que se pueden presentar en la imagen (bordes). La imagen resultante (ver Figura 46) aún está en escala de grises, y por ello puede presentar bordes o píxeles en grises que no se perciban a simple vista.

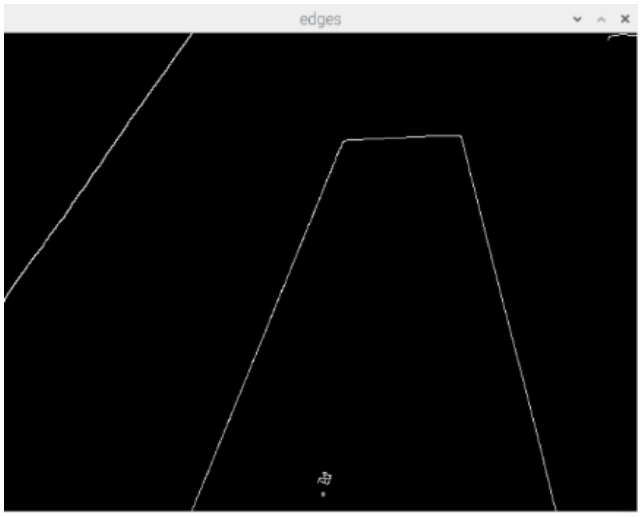


Figura 46. Imagen tras el reconocimiento de bordes
Fuente: Elaboración Propia

Por último, la imagen se pasa a blanco y negro mediante `<<_,bin=cv2.threshold (edges, threshold_one, threshold_two,cv2.THRESH_BINARY)>>,` que clasifica los valores de todos los píxeles de la imagen, y en función de si sobrepasa un valor (`threshold_one`) fija el valor de este pixel al valor que uno quiera (`threshold_two`), si no lo fija a negro (0).

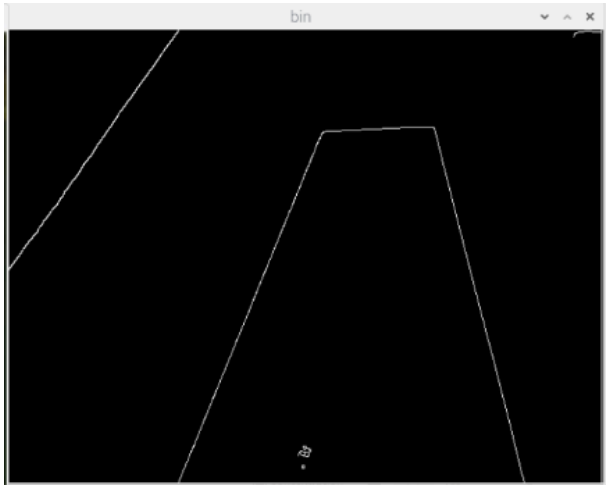
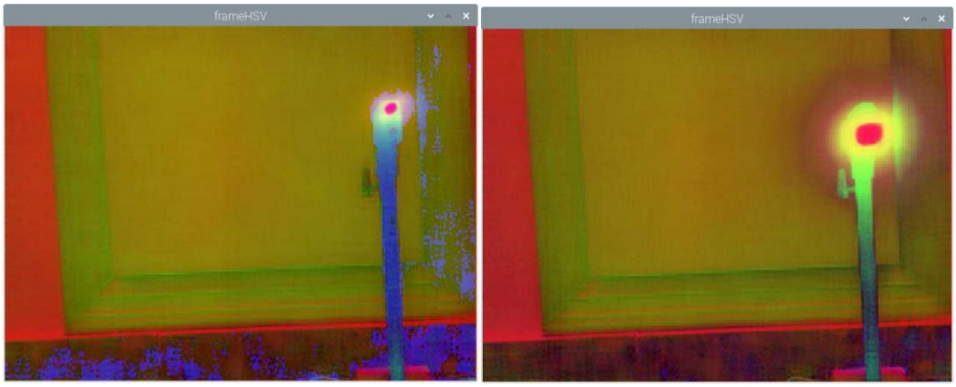


Figura 47.Imagen final en blanco y negro
Fuente: Elaboración Propia

Pasado todo este proceso, la imagen resultante es una matriz unidimensional de 480x640 con valores de 0 a 255, con tan solo dos líneas blancas que delimitan el arcén y un fondo negro (ver Figura 47).

Por otro lado, el reconocimiento de los colores del semáforo se ha llevado a cabo de una forma muy distinta. Dado que lo que se pretende detectar son fuentes de luz, lo primero que se hace es convertir la imagen del formato RGB al HSV (del inglés *Hue*, *Saturation*, *Value* – Matiz, Saturación, Valor), con la función `<<frameHSV=cv2.cvtColor (frame,cv2.COLOR_BGR2HSV)>>`. Trabajar con una imagen en formato HSV (ver Figura 48) ofrece la ventaja principal de trabajar con los valores de saturación. Como el semáforo es una fuente de luz, se distingue del resto a partir de su saturación, y posteriormente en función de su color.



(A) ***(B)***

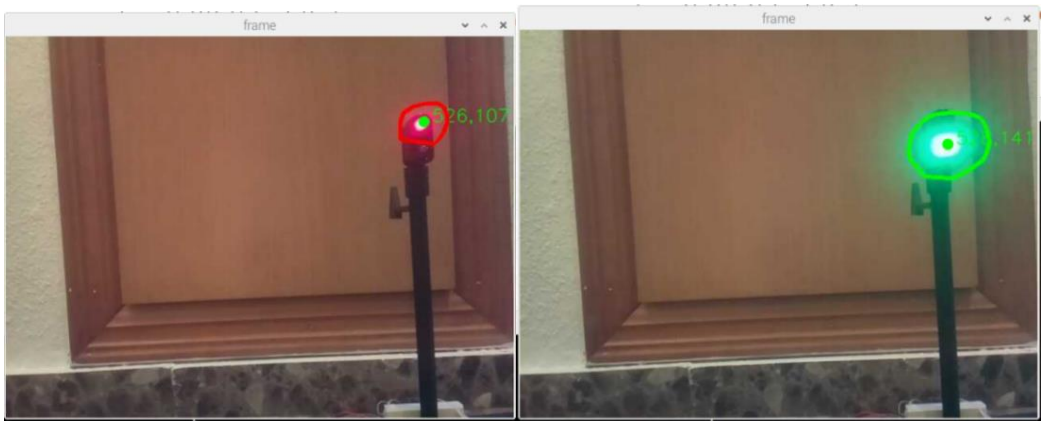
Figura 48. Imágenes del semáforo en formato HSV. (A) Semáforo en rojo. (B) Semáforo en verde
Fuente: Elaboración Propia

A continuación, se establecen los umbrales superiores e inferiores de HSV del color a detectar (ver Tabla 7) que depende de las condiciones de iluminación. y, a través de la función `<<maskRed=cv2.inRange (frameHSV, umbrale_inf, umbrale_sup)>>`, aislamos los píxeles de interés correspondientes ledes. Esta función fija a blanco aquellos píxeles que estén dentro del rango establecido por los umbrales, y deja en negro los que no.

	Matiz	Saturación	Valor
Verde	30-100	0-255	230-255
Rojo	160-179	0-255	200-255

Tabla 7. Valores de HSV habituales para la detección del estado del semáforo
Fuente: Elaboración Propia

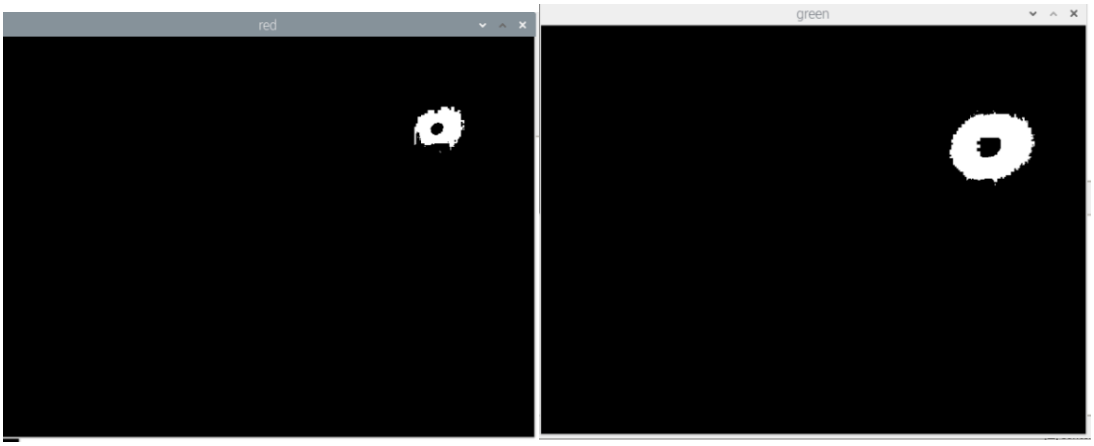
Por último, con <<contornos, _ = cv2.findContours(maskRed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)>> junta grupos de píxeles como un objeto y proporciona datos dimensionales como el área que puede ser útil para segmentar. Al conseguir una muy buena clasificación a través del HVS, tan solo obtenemos un objeto que ya es el de interés. A partir del área se puede incluso calcular la posición del objeto (ver Figura 49).



(A) **(B)**

Figura 49.Detección del estado y posición del semáforo. (A)Semáforo en rojo. (B)Semáforo en verde
Fuente: Elaboración Propia

Tras estos procesos se obtiene una imagen en negro en el que el led iluminado resalta como un grupo de píxeles blancos (ver Figura 50).



(A) **(B)**

Figura 50.Grupo de píxeles discriminados. (A)Semáforo en rojo. (B Semáforo en verde)
Fuente: Elaboración Propia

Para realizar pruebas de detección, con `<<cv2.imshow("frame", frame)>>` se muestra cualquier imagen que se esté manejando a través de una ventana. Y, una vez terminado, se desconecta la cámara con `<<cap.release>>` y se cierran las ventanas con *`<<cv2.destroyAllWindows()>>`*.

4.2. PROGRAMA DE CIRCULACIÓN

En este capítulo se va a describir el procedimiento con el que se ha elaborado el sistema de circulación autónomo, dotando al proyecto de capacidad de decisión ante ciertos estímulos.

4.2.1. APROXIMACIÓN

El problema de la circulación autónoma se ha planteado como un diagrama de control con etapas y transiciones. A través de un GRAFCET, se ha hecho un modelo gráfico del proceso que se va a automatizar (circulación y cruce de calles), contemplando entradas, acciones y procesos intermedios. GRAFCET tan solo se presenta como un lenguaje de programación grafico para autómatas. Como en este proyecto se va a trabajar con Python, se ha realizado una adaptación a este lenguaje.

Así, se ha trabajado con bloques de programación formados por bubles, los cuales realizan las mismas acciones hasta que se cumpla una cierta condición. De esta forma, cada bloque se presenta como un pequeño programa que puedo modificar mucho más libremente que si de una etapa del GRAFCET se tratara. Además, mediante clases de Python se elaboran las acciones que se han de llevar a cabo en cada etapa o, en este caso, en cada bloque.

Salvando las distancias, con esta aproximación se consigue la sencillez de los grafos a la hora de representar la automatización y diseñarla; manteniendo la profundidad de edición y modificación de un lenguaje de programación tan complejo como Python.

4.2.2. PARAMETROS DE REFERENCIA

A la hora de automatizar, es necesario establecer una serie de parámetros de referencia que nos indiquen cuándo realizar o no una acción. A continuación, se explican los parámetros elegidos como referencias.

En primer lugar, saber si el semáforo está en verde o rojo es clave para saber cuándo cruzar, sin embargo, también se ha tenido en cuenta la posición de los ledes iluminados del semáforo. Con este último parámetro se pretende controlar la a posición del semáforo en el campo de visón del robot.

En segundo lugar, hay que la posición de la acera, para poder rectificar posibles desviaciones a la hora de circular, conocer cuándo se acaba la acera y cuándo el robot ya a cruzado. Además, se establece como parámetro la distancia al cruce, para que el robot sea capaz de teneres a una distancia segura del cruce.

Al tener un control completo de la pista por donde se circula, se pueden establecer tantos parámetros de referencia como sea necesario. Aun así, en este proyecto se ha mantenido el menor número de parámetros posible para simplificar la pista de pruebas.

4.2.3. BLOQUES FUNCIONALES

Como se ha comentado antes, se ha organizado la programación en bloques en bucles, y se han creado clases en las que se definen acciones concretas. A continuación, se describen los bloques y clases que conforman el programa.

Todos los movimientos de los servos se han unificado en una clase llamada *'Move'*. En ella, tomando como referencia la biblioteca '*Robo Hat'* de SunFounder que utiliza el kit original se han creado bloques de movimientos predefinidos. Con '*Start*', todos los servos vuelven a su posición inicial. Mediante *'Walk*' se establece el número de pasos que da el robot, y con '*Turn*' el robot gira en un sentido u otro.

La visión artificial se divide en dos clases. Por un lado, *'Carril*' recoge la detección de la acera, el reconocimiento de la presencia de un cruce y el final de la acera (ver Figura 51). Por su parte, *'Detect'* engloba el proceso de detección del estado de los ledes del semáforo y su posición (ver Figura 52).

De esta forma, la clase *'Carril'* permite la circulación por la pista de pruebas. Se analiza una columna o fila de píxeles y se devuelve el primer y último píxel con valor 255. Así, partiendo de la imagen en binario, y recortando las zonas de interés, se determina el punto medio de la acera y compararla con

un valor de referencia. Este mismo proceso se utiliza para detectar la presencia del cruce de peatones y el final de la acera. Por último, la clase devuelve varias variables booleanas (True o False) que indican si es necesario corregir el rumbo, si se ha acabado la acera y si está ante en un cruce.

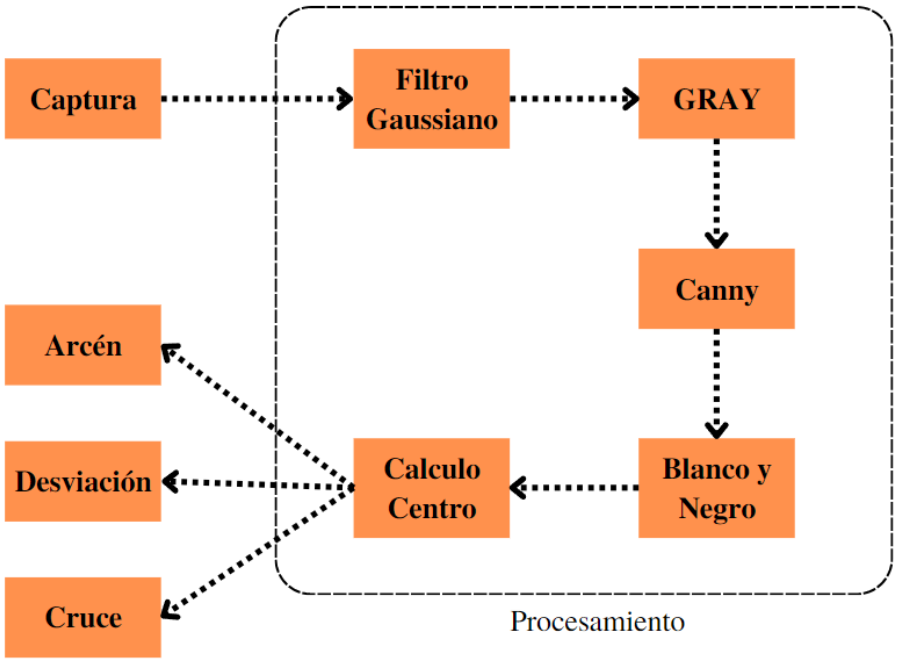


Figura 51. Diagrama de la función Carril()
Fuente: Elaboración Propia

Por otro lado, la clase *'Detect'*, tomando como parámetros de entrada los umbrales superiores e inferiores del HSV del color que se debe detectar, devuelve la posición del centro de objeto detectado y una variable booleana que indica si ha detectado o no el objeto de interés. La discretización del objeto se hace mediante los comandos antes comentados, y el cálculo de la posición del centro a través de los momentos.

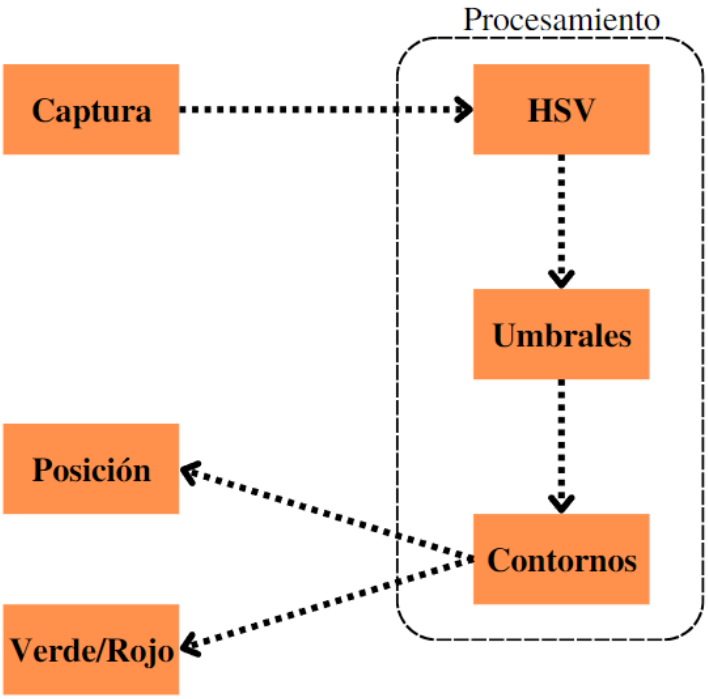
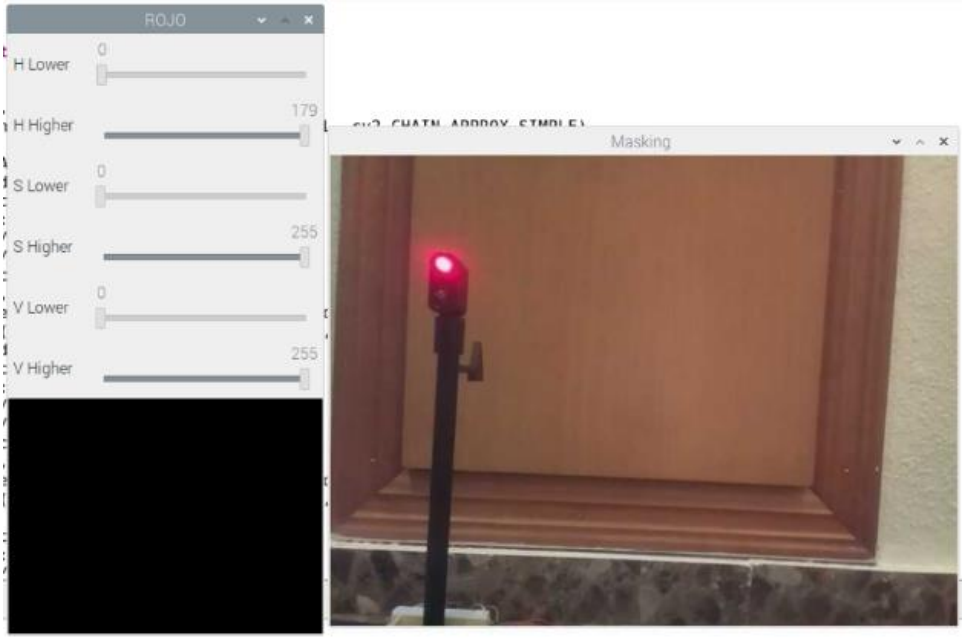
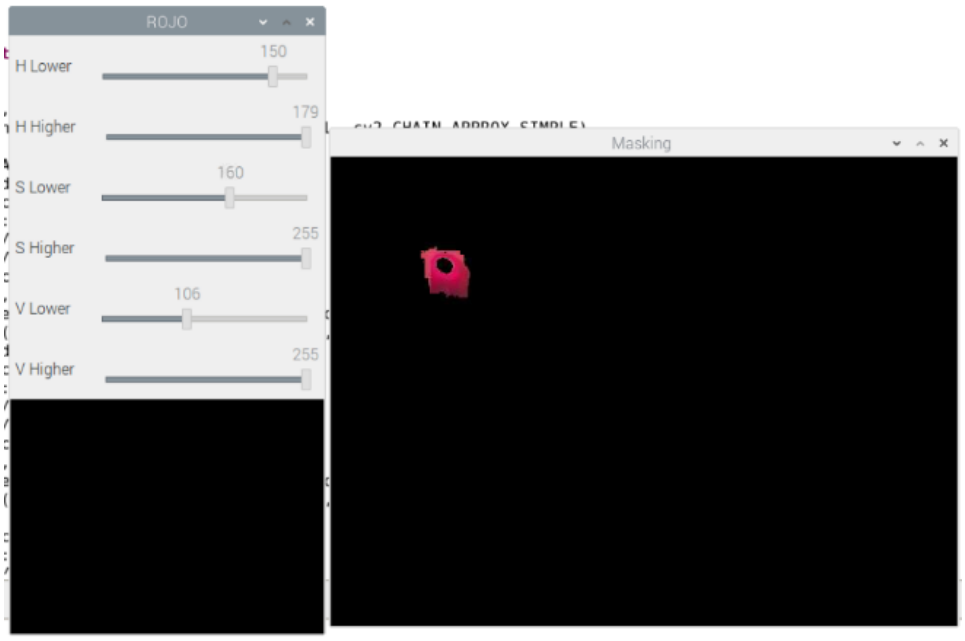


Figura 52.Diagrama de la función Detect()
Fuente: Elaboración Propia

Por último, para determinar tanto los umbrales del HSV como los valores de los parámetros necesarios para aislar los bordes de la acera (comentados en la sección FUNCIONES) se ha creado la clase *'CalibrationV2()'*. Esta clase, permite modificar los parámetros que se desean establecer y ver la imagen resultante en tiempo real a través de una ventana emergente (ver Figuras 53, 54, 55).

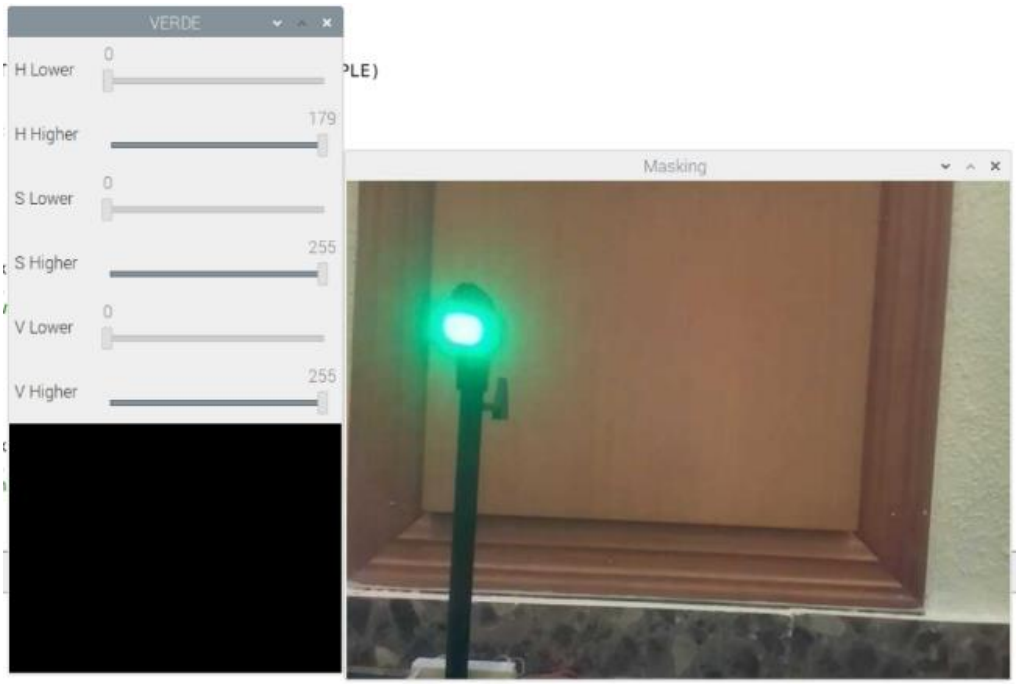


(A)

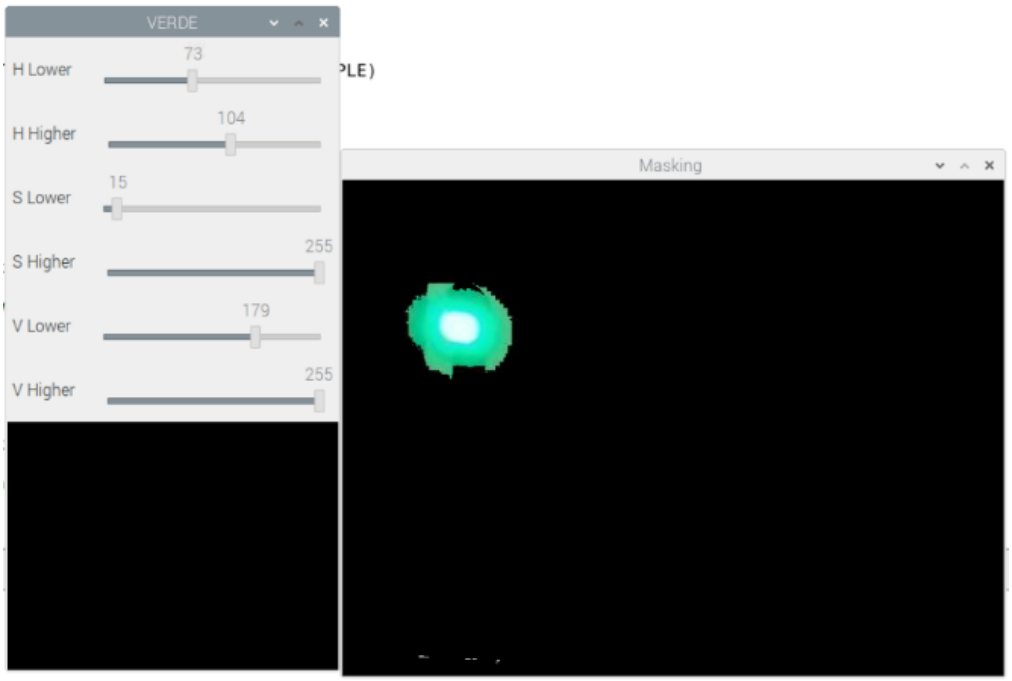


(B)

Figura 53.Funcion de CalibrationV2(). (A) Parámetros sin modificar. (B) Parámetros modificados para detectar el semáforo en rojo
Fuente: Elaboración Propia



(A)



(B)

Figura 54. Función CalibrationV2(). (A) Parámetros sin modificar. (B) Parámetros modificados para detectar el semáforo en verde.
Fuente: Elaboración Propia

4.2.4. FLUJO DE CIRCULACIÓN

En este punto se describen las distintas etapas de la circulación que se han automatizado. Como se ha comentado antes en la sección de Aproximación, se ha organizado por bloques donde se llevan a cabo distintas funciones en bucle.

El proceso comienza con una inicialización de todos los parámetros que se van a utilizar, para decidir qué acciones realizar en la etapa, o si se ha de pasar a la siguiente. Este bloque se utiliza antes y después de cada etapa, junto con el movimiento de la cabeza.

La primera etapa es la de circulación (ver Figura 55). En ella se reconoce la acera y se actúa en función de la posición del punto medio de esta. Si el robot se desvía, rectifica la dirección; y por último si detecta la presencia de un cruce, pasa a la siguiente etapa.

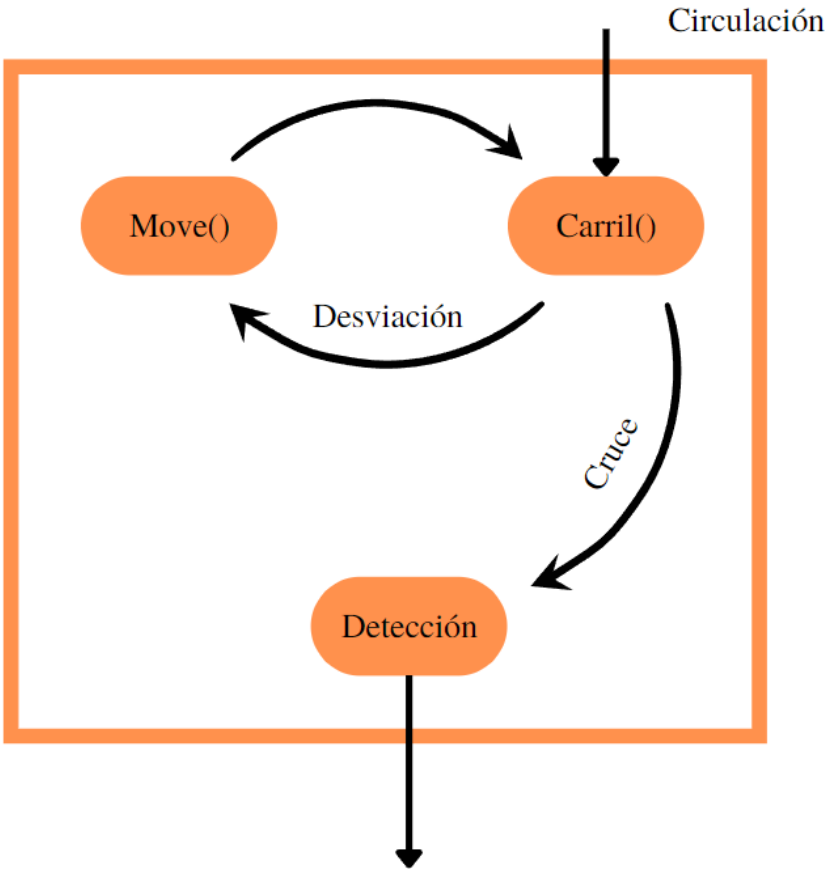


Figura 55.Diagrama de la etapa de circulación
Fuente: Elaboración Propia

La siguiente etapa es la de detección (ver Figura 56). En esta etapa; el robot gira la cámara hacia su derecha hasta que reconoce el semáforo. Posteriormente una vez el semáforo está en verde pasa a la siguiente etapa.

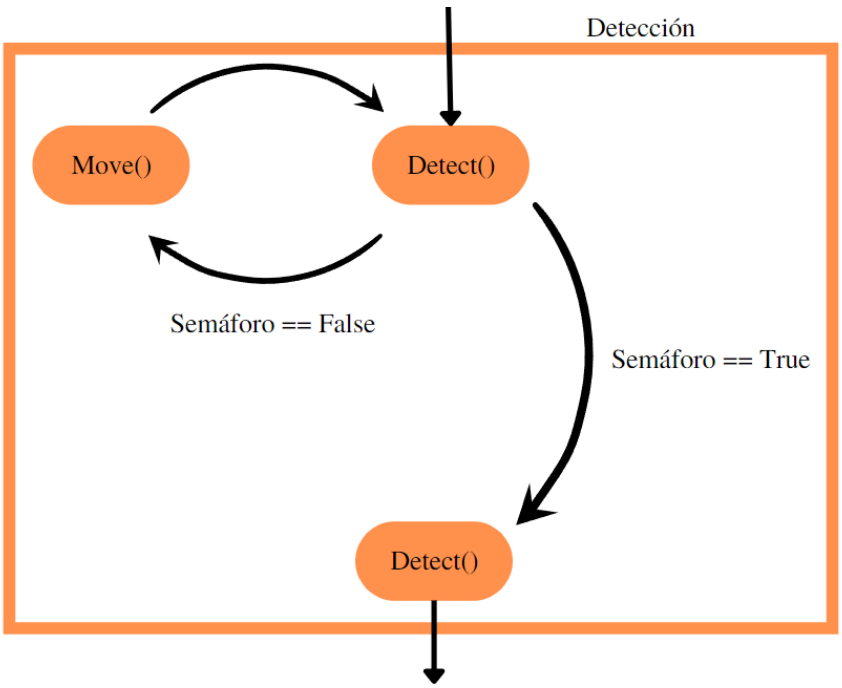


Figura 56.Diagrama de la etapa de detección

Fuente: Elaboración Propia

Una vez en verde, se pasa a la etapa de cruce (ver Figura 57). En esta etapa, al igual que la primera etapa de todo el robot circula siguiendo la deviación que tiene respecto con la línea blanca que hace de acera, tan solo que en este caso la etapa acaba cuando acaba la pista de pruebas.

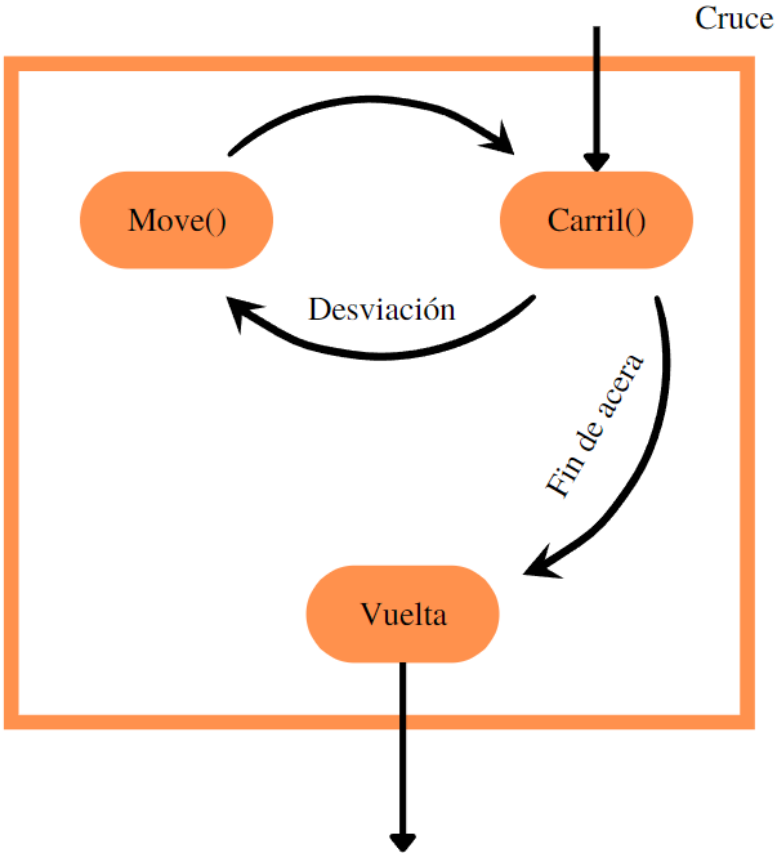


Figura 57. Diagrama de la etapa de vuelta
Fuente: Elaboración Propia

La última etapa es la de vuelta. El robot da la vuelta y espera un tiempo aleatorio hasta que decide volver a cruzar. Así, una vez pasada esta etapa, el programa comienza de nuevo, tan solo que hora el robot comienza en el otro lado del cruce.

Por último, se representan todas las etapas en conjunto, haciendo semejanza a un diagrama GRAFCET donde las acciones de las etapas son las clases anteriormente comentadas (ver ANEXO 2).

CAPITULO. 5 CONCLUSIÓN

El trabajo presentado en este proyecto permite alcanzar un resultado final de un robot autónomo programado en Python capaz de circular por un entorno urbano haciendo uso de visión artificial.

A lo largo de todo el proyecto han surgido problemas que han sido resueltos de forma concisa para poder llegar al resultado final deseado.

En lo que hace referencia al diseño del androide sido un reto la distribución de espacios y conseguir un aspecto más o menos humanoide. A través de un proceso iterativo y varias pruebas de piezas, se he logrado un equilibro entre forma y utilidad.

A lo largo de todo el proceso de programación del sistema de circulación autónomo se han subsanado una sustancial cantidad de errores y fallos propias del arte de la programación.

Finalmente, aunque han surgido problemas a lo largo del proyecto, se ha sido capaz de llegar a un resultado final satisfactorio, dejando las puertas abiertas a la mejora del código y del diseño del robot. Se ha intentado mantener una estructura del programa de circulación ordenada y sencilla, capaz de añadir fácilmente nuevas funciones y bloques, o modificar los actuales.

A pesar de todos estos problemas, he disfrutado el proyecto se ha disfrutados de principio a fin, siendo un reto continuo con el que he puesto a prueba mis habilidades y a la vez que he adquirido nuevas.

CAPITULO. 6 BIBLIOGRAFIA Y REFERENCIAS DOCUMENTALES

Bernabeu, EJ., Carbonell, PJ., Esparza, A., Picó, JA. & Ramos, C. (2018). Apuntes de Sistemas Automáticos (cód. 11407). PoliformaT: UPV

Bondía, J., Díez, JL., García, PJ., Navarro, JL. &Vallés, M. (2019). Apuntes de Tecnología Automática (cód. 11430). PoliformaT: UPV

García, M., Capuz, S., Sirvent, JI. & Cloquell, V. (2020). Apuntes de Proyectos (cód. 11419). PoliformaT: UPV

Baraza Calvo, JC. (2021). Apuntes de Impresión 3D y Fabricación (cód. 13438). PoliformaT: UPV

Hirschtick, J. & McEleney, J. (2012). Onshape. EEUU: PTC Inc. (2019). Sitio web: ww.onshape.com/en/

SunFounder (2016). SunFounder Robotic Kit for Arduino, 4-DOF Programmable DIY Sloth Robot Kit with Tutorial. Sitio web: https://www.sunfounder.com/products/arduino-sloth-robot/

SunFounder (2016). Repositorios de SunFounder en GitHub. Sitio web: github.com/sunfounder

Biblioteca open-source de visión artificial. (s.f.). Sitio web: opencv.org

Biblioteca open-source de Python para matrices y vectores. (s.f.). Sitio web: numpy.org

Crespo Cano, R. (2020). Curso de introducción a openCV y Python. Sitio web: nbviewer.jupyter.org/github/CAChemE/opencv-python/blob/master/opencv-and-python.ipynb#1.-Introducci%C3%B3n

Python Programming Language. (s.f.). Sitio web: www.geeksforgeeks.org/python-programming-language/?ref=ghm

Thingiverse.com. (s. f.). Thingiverse—Digital Designs for Physical Objects. Sitio web: https://www.thingiverse.com/

Home » omes-va.com. (s. f.). OMES. Sitio web:https://omes-va.com/

Buttu, M. (2016). El gran libro de Python. Marcombo.

Stack Overflow en español. (s. f.). Stack Overflow en español Sitio web: https://es.stackoverflow.com/

Monk, S. (2017). Ejercicios prácticos con Raspberry Pi. Marcombo.

Raspberry Pi documentation. (s. f.). Sitio web: https://www.raspberrypi.com/documentation/

II.PRESUPUESTO

CAPITULO. 1 CALCULO DEL PRESUPUESTO

A lo largo de la presente sección se va a detallar el coste de llevar a cabo un proyecto como este.

1.1. CUADRO DE MANO DE OBRA

El trabajo ha sido desarrollado por un estudiante de grado y supervisad por dos ingenieros:

Código	Ud.	Descripción	Precio (€/h)
MO001	h	Ingeniero sénior	60,00
MO002	h	Estudiante de ingeniería	30,00

Tabla 8.Mano de obra

1.2. CUADRO DE MAQUINARIA

Se ha empleado una impresora 3D MK3S de las 9 disponibles en el laboratorio para la materialización del prototipo, y herramientas tales como destornilladores. Sin embargo, pese la impresora 3D ha sido proporcionada por la universidad, la impresión de piezas ha supuesto un gasto energético. Por otro lado, mismo el kit del que parte el proyecto cuenta con un destornillador más que suficiente para el proyecto por lo que no ha sido necesarias herramientas extras. Además, las licencias necesarias para los software han sido cubierta por un convenio con universidad (Oneshape) o era de carácter gratuito (Thonny), por ello no se tendrán en cuenta como gasto.

Código	Ud.	Descripción	Precio (€/h)
MM001	Coste/h	Gasto energético de impresión	0,10

Tabla 9. Cuadro de gastos de maquinaria

1.3. CUADRO DE MATERIALES

Para materializar los prototipos se utiliza filamento PLA de color metalizado y negro. Además, se emplean elementos presentes en el kit de la marca ‘SunFounder’ y se han añadido la Raspberry Pi, la PCA9685, módulo de cámara, alimentación, ledes, resistencias, elementos de conexión, servos motores y material para manualidades.

Código	Ud.	Descripción	Precio (€/Ud.)
MT001	ud	Kit “Robotic Kit for Arduino , 4-DOF Programmable DIY Sloth Robot Kit with Tutorial”	59.99
MT002	kg	Filamento para impresora 3D FDM, PLA, 1.75mm, Negro, 1kg RS PRO	40,57
MT003	ud	Raspberry Pi 2 Model B	39,28
MT004	ud	Cartulina negra	0,50

MT005	ud	Set de elementos de conexión: 120 Piezas de Cable DuPont, 40 Pines Macho-Hembra, 40 Pines Macho-Macho, 40 Pines Hembra-Hembraç	6,99
MT006	ud	Modulo cámara Raspberry Pi	32,95
MT007	ud	Adaptador wifi TP-Link	6,99
MT008	ud	Microservo 9G S90	2,45
MT009	ud	Microservo MG90S	8,49
MT010	ud	Pilas 18650 3.7V 2600mAh	8,36
MT011	ud	Placa de prototipado	2,41
MT012	ud	Portapilas doble 18650	1,99
MT013	ud	Ledes	0,39
MT014	ud	Sistema de alimentación ininterrumpida UPS HAT	25,95
MT015	kg	Filamento para impresora 3D FDM, PLA, 1.75mm, Metalico, 1kg RS PRO	40,57
MT016	ud	MB102 Placa de prototipado adaptador de fuente	5,49

Tabla 10.Cuadro de materiales

1.4. AMORTIZACION DE EQUIPOS

Se ha tenido encuentra una amortización lineal donde el objeto se deprecia constantemente con el transcurso de los años. Antes, es necesario conocer los costes de cada los equipos utilizados.

Nombre	Descripción	Precio (€/ud.)
Ordenador	Modelo estándar	800
Impresora 3D	Modelo Original Prusa i3 MK3	1 159
Kit	Modelo “Robotic Kit for Arduino , 4-DOF Programmable DIY Sloth Robot Kit with Tutorial” que incluye las herramientas necesarias para el montaje de las placas.	59,99
Soldador	Modelo estandar	42,30
	TOTAL:	**1737.83**

Tabla 11. Amortización de equipos

En el caso de que se amortizaran los equipos en 4 años (el mínimo), entonces:

$$Coste = \frac{1737.83}{(4 \cdot 12)} = 36.20 \frac{1}{mes}$$

Como el tiempo de uso es de 6 meses, el coste de amortización es:

$$Coste = 36.20 \frac{1}{mes} \cdot 6\, meseses = 217.228$$

1.5. PRESUPUESTOS

Se elaboran los distintos capítulos que conforman el presupuesto.

Capitulo Nº1 MODELADO DEL PROTOTIPO					
Código	Ud.	Descripción	Medición	Precio (€/Ud.)	Importe (€)
1.1.-Diseño e impresión de prototipos					
MM001	Coste/h	Gasto energético de impresión	16	0,10	1,6
MT15	g	Filamento para impresora 3D FDM, PLA, 1.75mm, Metálico, 1kg RS PRO	0,09354	40,57	3,79
MT02	g	Filamento para impresora 3D FDM, PLA, 1.75mm, Negro, 1kg RS PRO	0,05509	40,57	2,24
MO01	h	Ingeniero sénior	15	60,00	900,00
MO02	h	Estudiante de ingeniería	20	30,00	600,00
		3% Costes indirectos		1507,53	45,23
Total subcapítulo 1.1.- Impresión del cuerpo del prototipo:					1552,86
1.2.-Montaje del prototipo					
MT01	ud	Kit “Robotic Kit for Arduino , 4-DOF Programmable DIY Sloth Robot Kit with Tutorial”	1	59.99	59.99
MT3	ud	Raspberry Pi 2 Model B	1	39,28	39,28
MT05	ud	Set de elementos de conexión: 120 Piezas de Cable DuPont, 40 Pines Macho-Hembra, 40 Pines Macho-Macho, 40 Pines Hembra-Hembra	1	6,99	6,99
MT06	ud	Modulo cámara Raspberry Pi	1	32,95	32,95
MT07	ud	Adaptador wifi TP-Link	1	6,99	6,99
MT08	ud	Microservo 9G S90	1	2,45	2,45
MT09	ud	Microservo MG90S	1	8,49	8,49
MT10	ud	Pilas 18650 3.7V 2600mAh	6	8,36	50,16
MT11	ud	Placa de prototipado	1	2,41	2,41
MT12	ud	Portapilas doble 18650	2	1,99	3,98
MT13	ud	Ledes	2	0,39	0,78
MT14	ud	Sistema de alimentación ininterrumpida UPS HAT	1	25,95	25,95
MT16	ud	MB102 Placa de prototipado adaptador de fuente	1	5,49	5,49
MO002	h	Estudiante de ingeniería	1	30,00	30,00
		3% Costes indirectos		271,93	8,16
Total subcapítulo 1.2.- Montaje del prototipo:					**280,9**
Total Capítulo Nº 1 MODELADO DEL PROTOTIPO:					**1833,76**

Tabla 12. Modelado del prototipo

Capítulo Nº 2 AUTONOMÍA DEL PROTOTIPO					
Código	Ud.	Descripción	Medición	Precio (€/Ud.)	Importe (€)
2.1.- Programación					
MO001	h	Ingeniero sénior	10	60,00	600,00
MO002	h	Estudiante de ingeniería	50	30,00	1500,00
		3% Costes indirectos		2100,00	63,00
		Total subcapítulo 2.1.- Programación:			**2163,00**
2.2.- Ensayos del prototipo					
MT004	ud	Cartulina negra	2	0,50	1,00
MO002	h	Estudiante de ingeniería	25	30,00	750,00
		3% Costes indirectos		751,00	22,53
		Total subcapítulo 2.2.- Ensayos del prototipo:			**773,53**
		Total Capítulo Nº 2 AUTONOMÍA DEL PROTOTIPO:			**2442,13**

Tabla 13. Autonomía del prototipo

Capítulo Nº 3 AMORTIZACIÓN DE LOS EQUIPOS					
Código	Ud.	Descripción	Medición	Precio (€/Ud.)	Importe (€)
3.1.- Amortización de los equipos					
		Total Capítulo Nº 3 AMORTIZACIÓN DE LOS EQUIPOS:			**217,228**

Tabla 14. Presupuesto de amortización de equipos

1.6. RESUMEN GENERAL

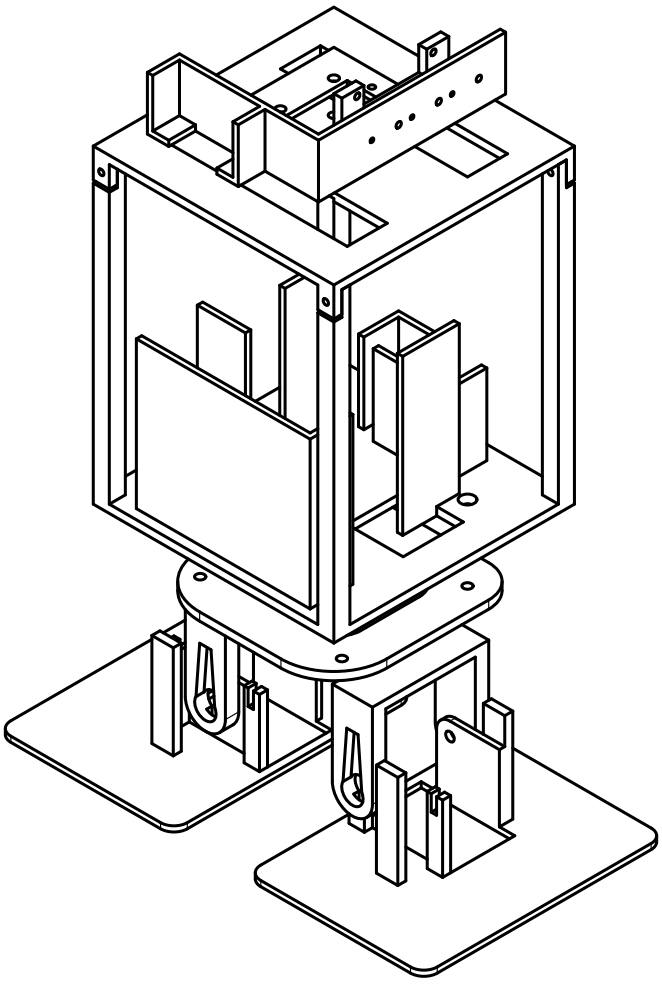
Finalmente, se expone el resumen general del presupuesto del presente proyecto.

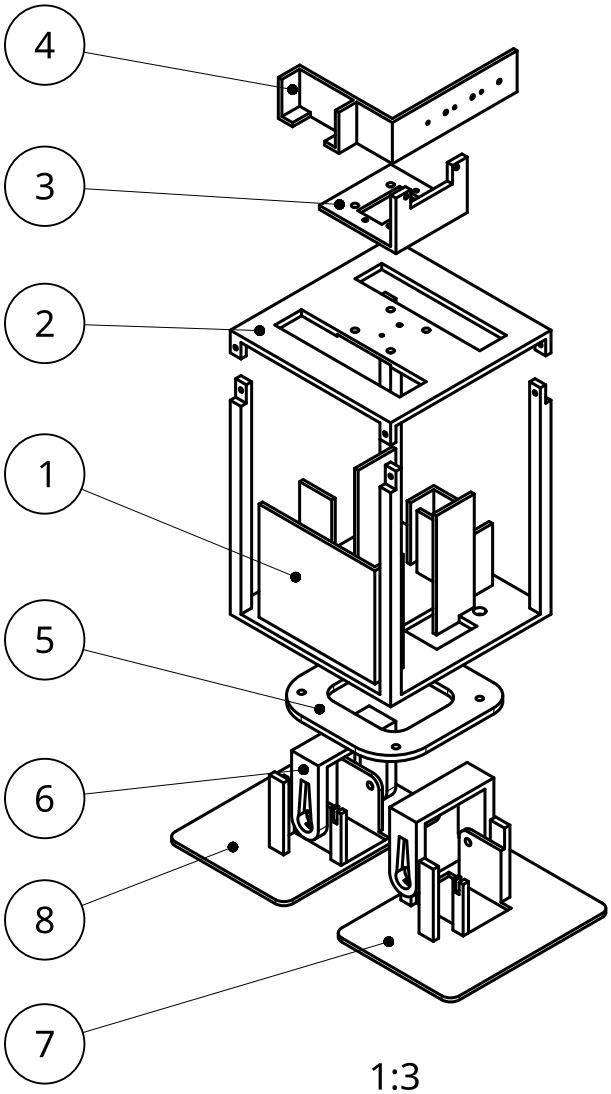
1º Modelado del prototipo	1833,76€
2º Autonomía del prototipo	2163,00€
3º Amortización de los equipos	217,228€
Presupuesto de ejecución material (PEM)	**4213,988€**
13% de gastos generales	547,818€
6% de beneficio industrial	252,839€
Presupuesto de ejecución por contrata (PEC = PEM + GG + BI)	**5014,645€**
21% IVA	1053,075 €
PRESUPUESTO BASE DE LICITACIÓN (PBL = PEC + IVA)	6067,72€

Tabla 15. Presupuesto general

III.PLANOS

Elemento n.º	Cantidad	Número de pieza	Descripción
1	1	Cuerpo	
2	1	Cierre del cuerpo	
3	1	Soporte del cuello	
4	1	Soporte de la camara	
5	1	Pieza intermedia	Pieza del kit original
6	2	Piernas	Pieza del kit original
7	1	Pie izquierdo modificado	
8	1	Pie derecho modifcado	





1:3

	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

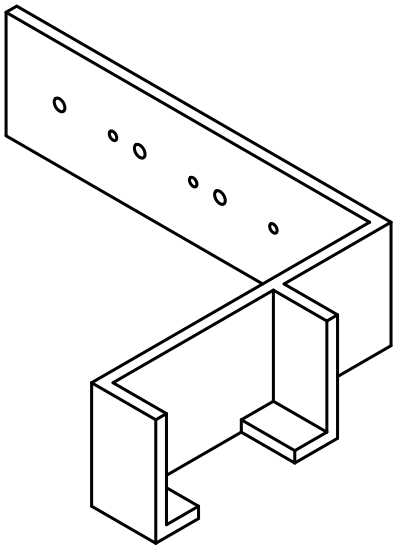
TÍTULO

Conjunto en explosión

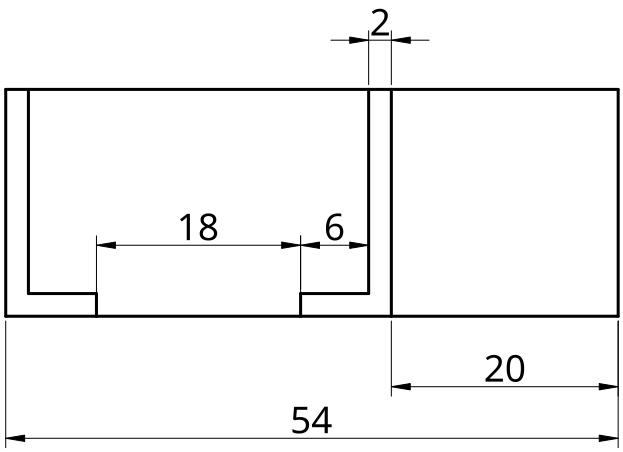
NÚMERO DE DIBUJO 1

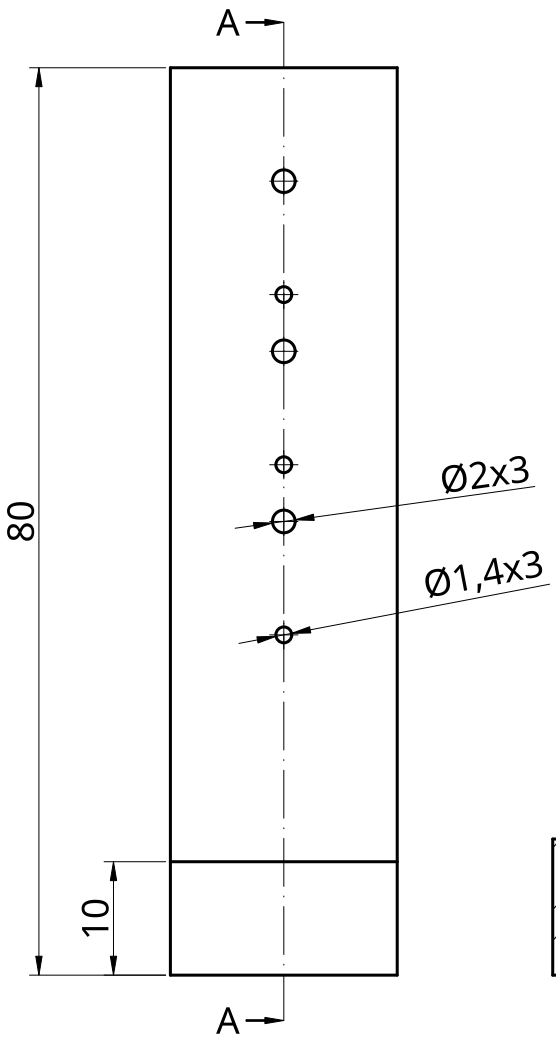
ESCALA 1:2

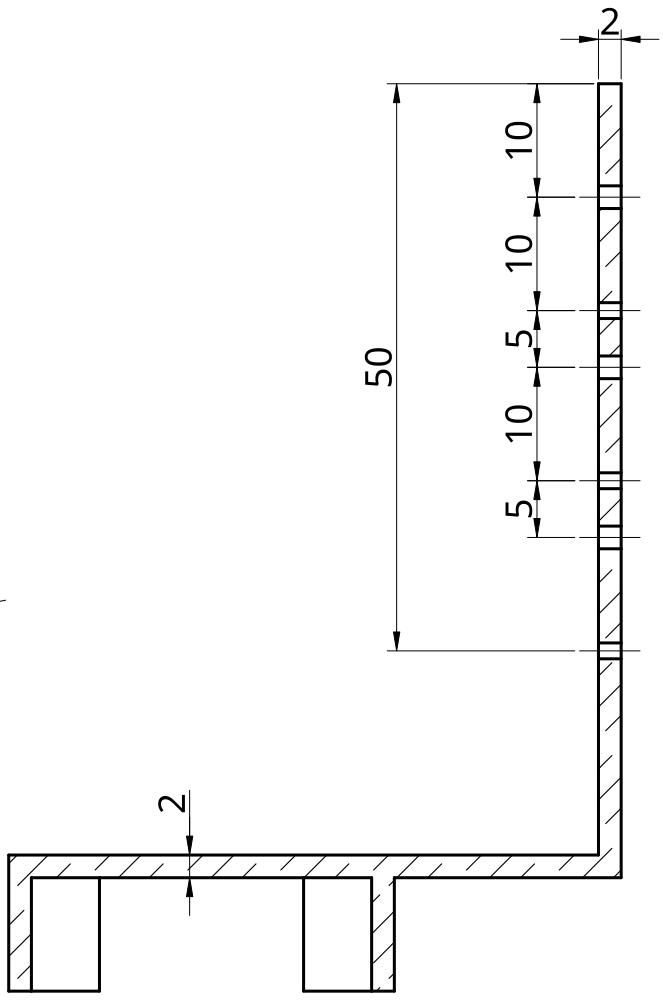
HOJA 1 de 12



1:1







	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

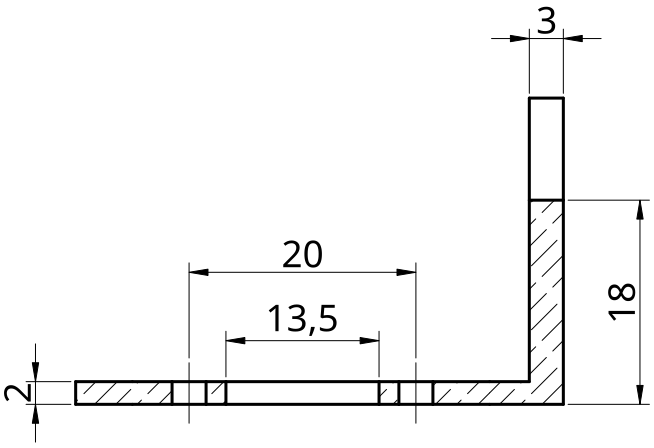
TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

TÍTULO

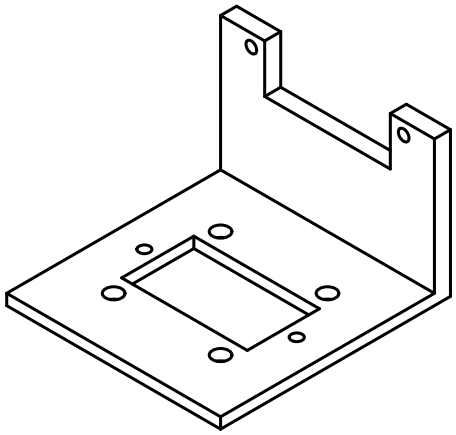
SOPORTE DE LA CAMARA

NÚMERO DE DIBUJO 2

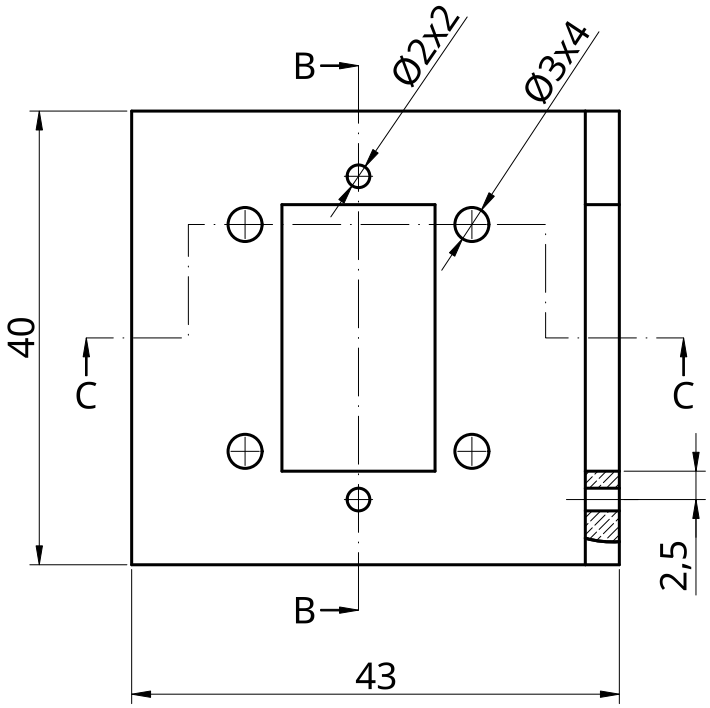
ESCALA 1.5:1

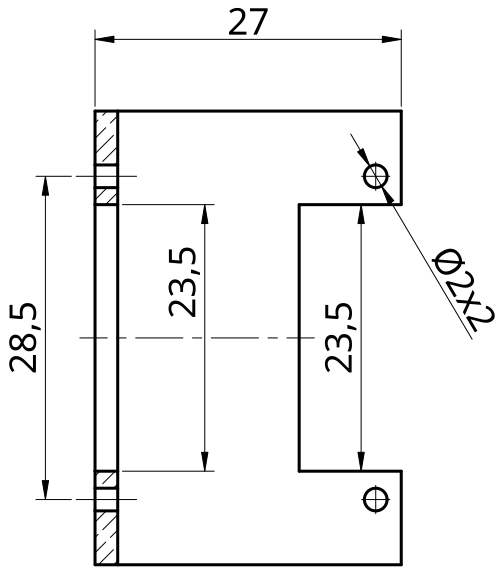


C - C



1:1





B - B

	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

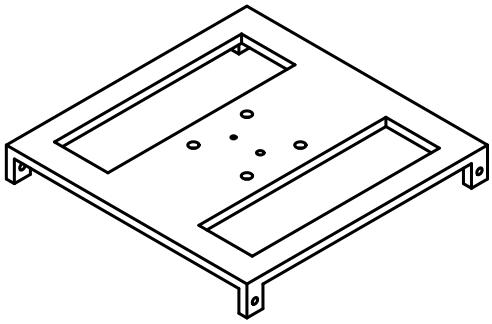
TÍTULO

SOPORTE DEL CUELLO

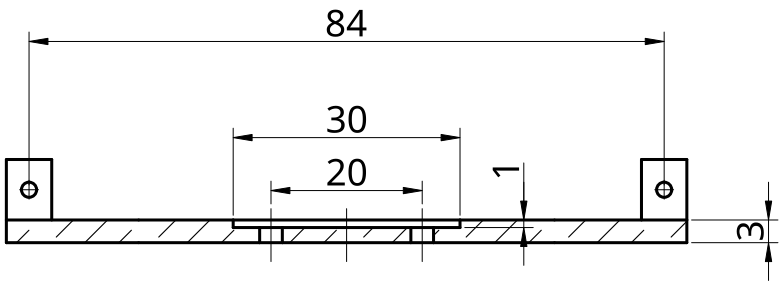
NÚMERO DE DIBUJO

3

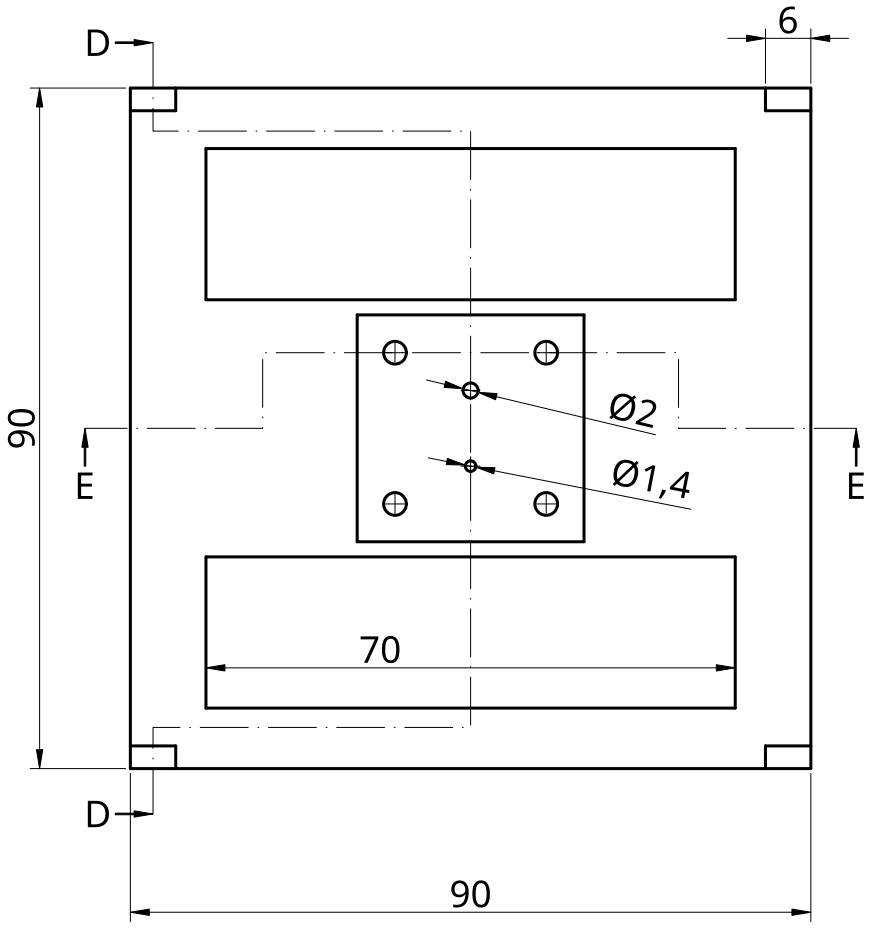
ESCALA 1.5:1

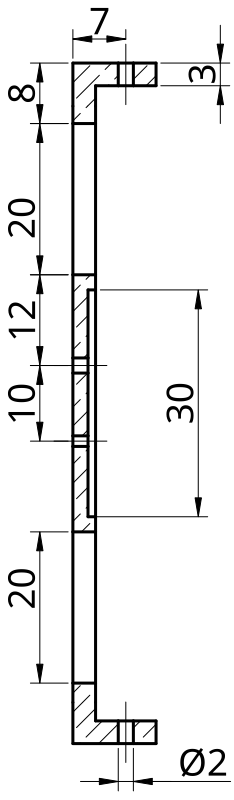


1:2



E - E





D - D

	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

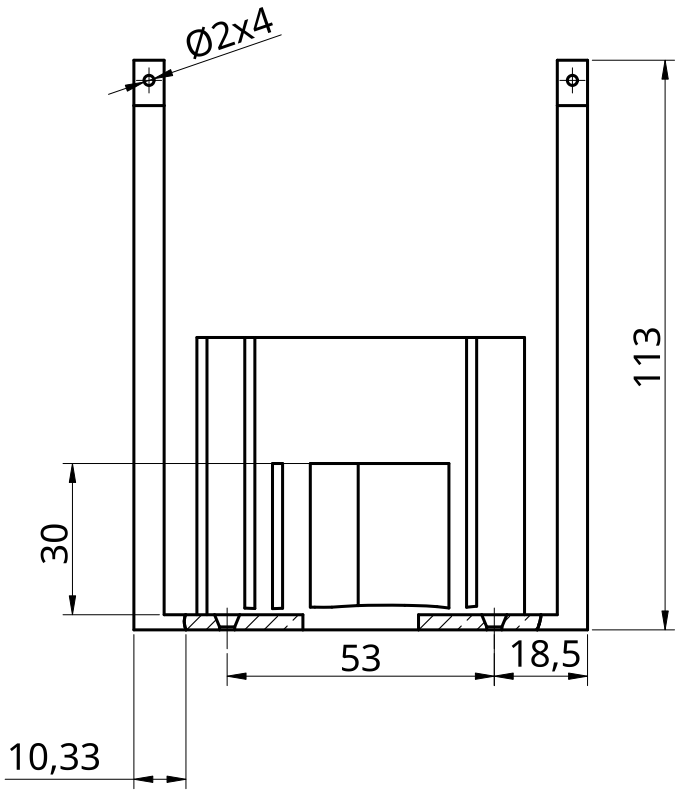
TÍTULO

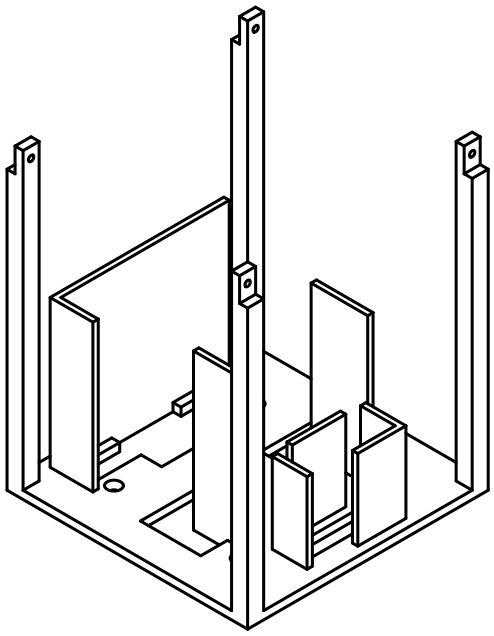
CIERRE DEL CUERPO

NÚMERO DE DIBUJO 4

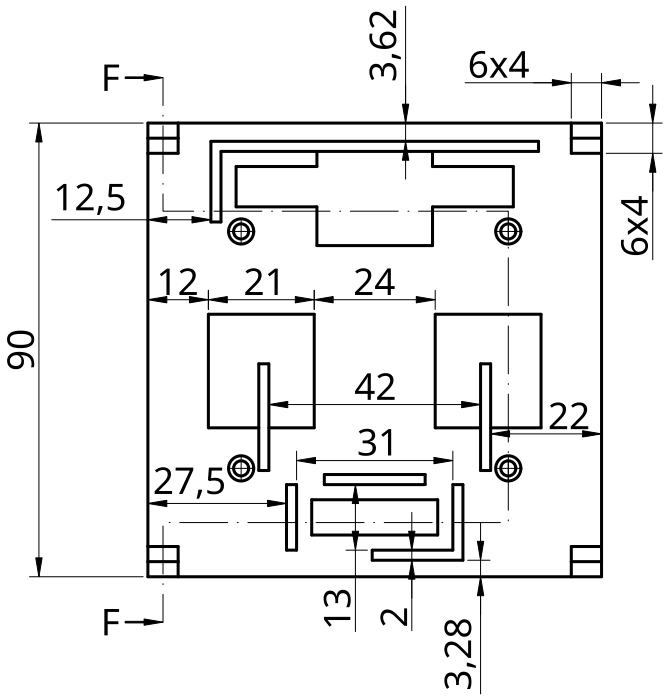
ESCALA 1:1

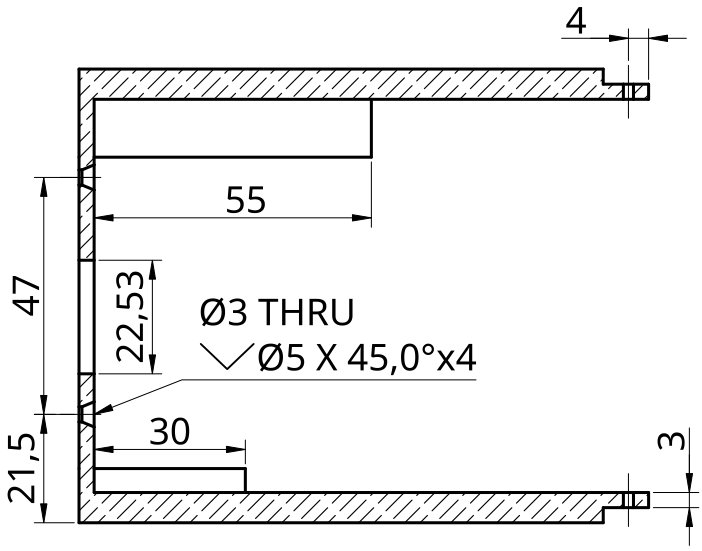
HOJA 4 de 12





1:2





F - F

*Todos los grosores de guias se entienden de 2mm a no ser que se indique lo contrario

	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

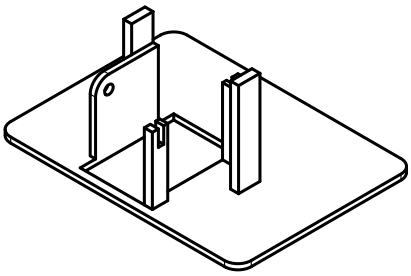
TÍTULO

CUERPO

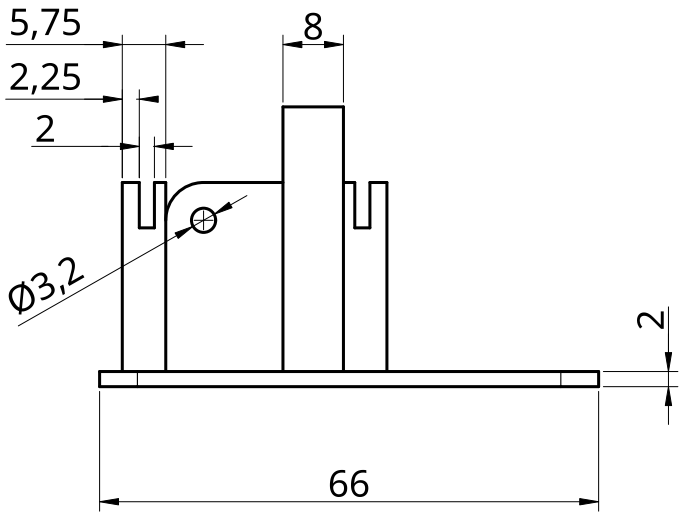
NÚMERO DE DIBUJO

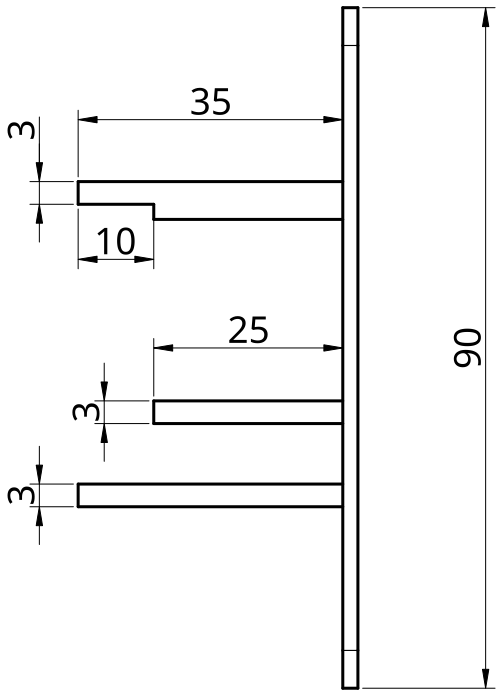
5

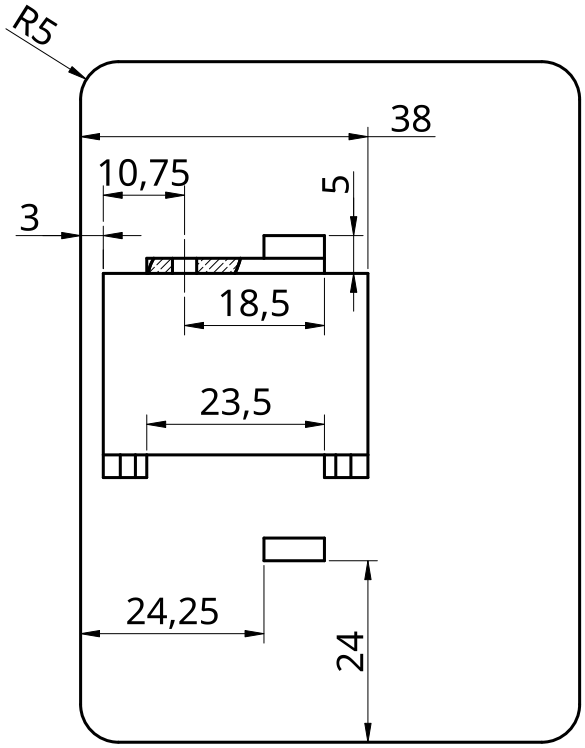
ESCALA	PESO	HOJA
1:1.5		5 de 12



1:2







	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

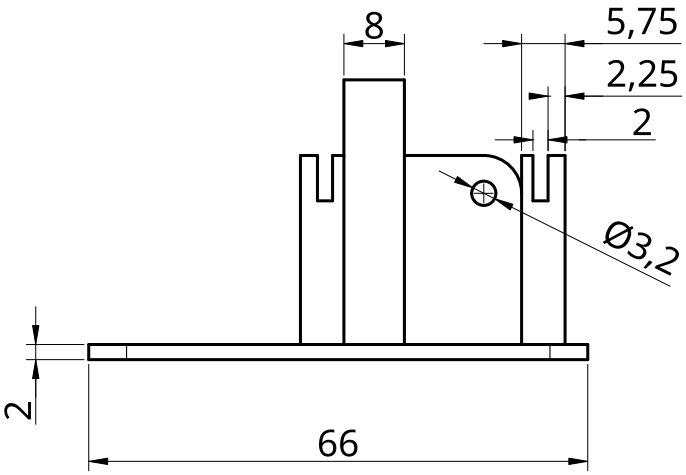
TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

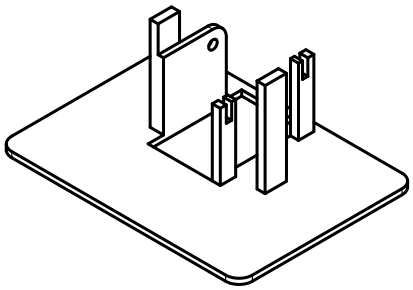
TÍTULO

PIE IZQUIERDO

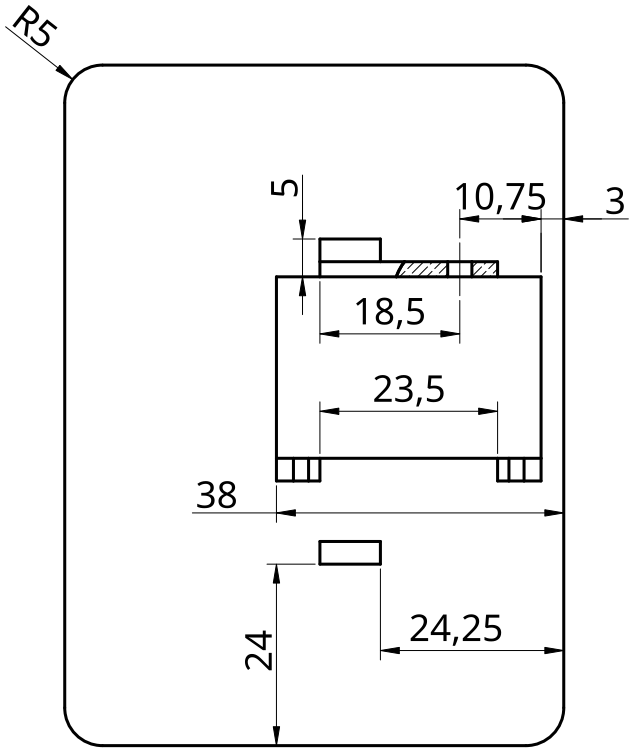
NÚMERO DE DIBUJO 6

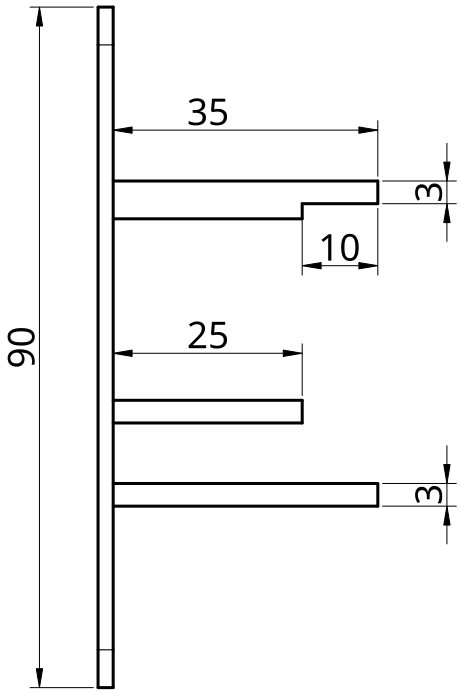
ESCALA 1:1 | PESO | HOJA 6 de 12





1:2





	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

TÍTULO

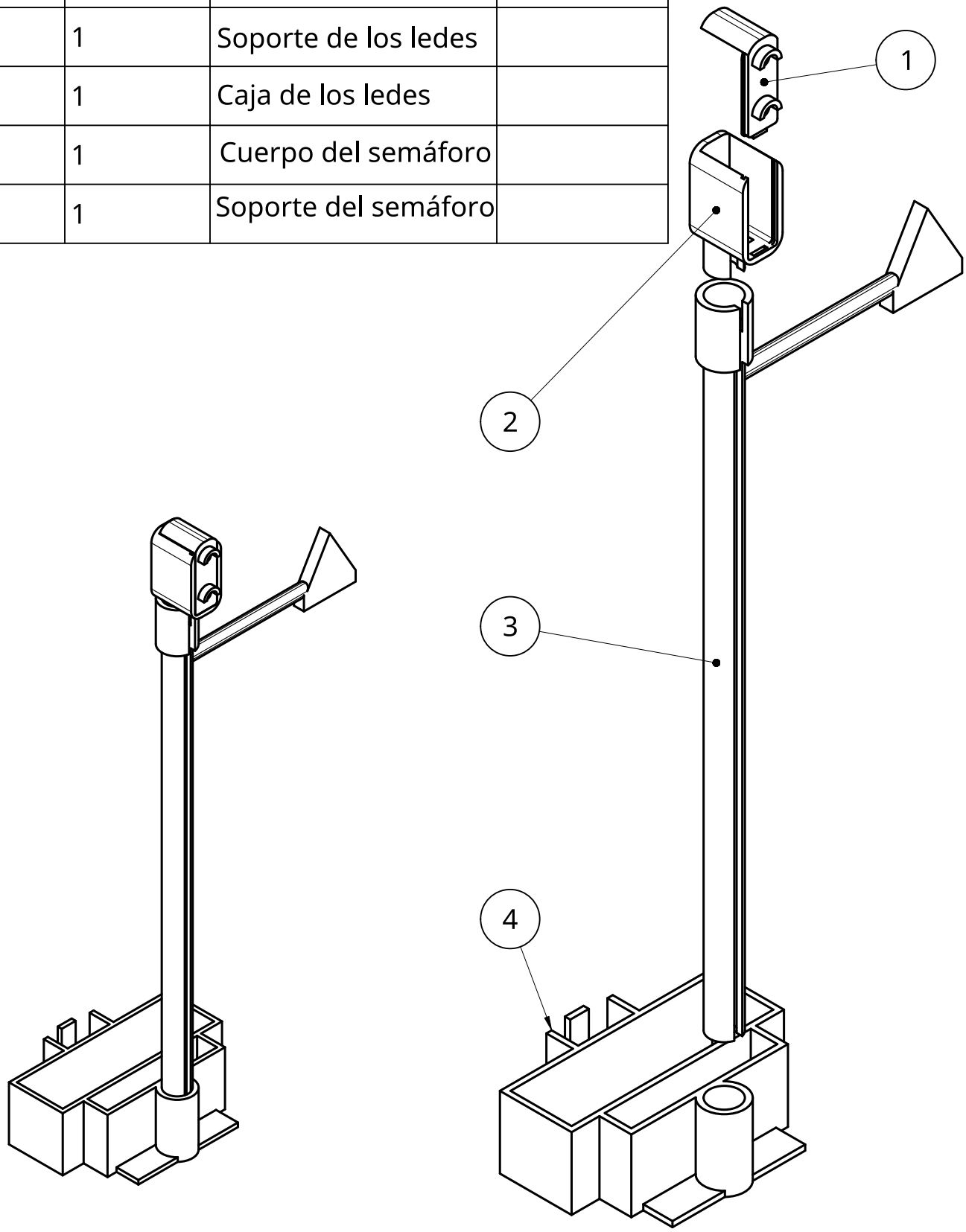
PIE DERECHO

NÚMERO DE DIBUJO 7

ESCALA 1:1

HOJA 7 de 12

Elemento n.º	Cantidad	Número de pieza	Descripción
1	1	Soporte de los ledes	
2	1	Caja de los ledes	
3	1	Cuerpo del semáforo	
4	1	Soporte del semáforo	



	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

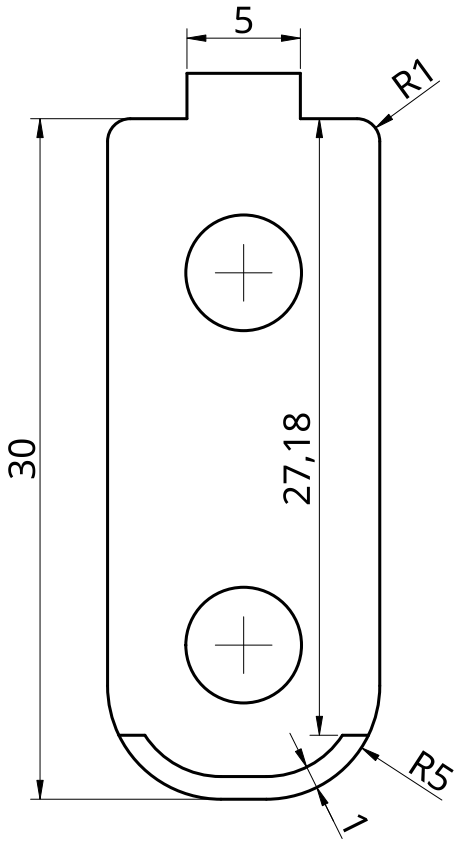
TÍTULO

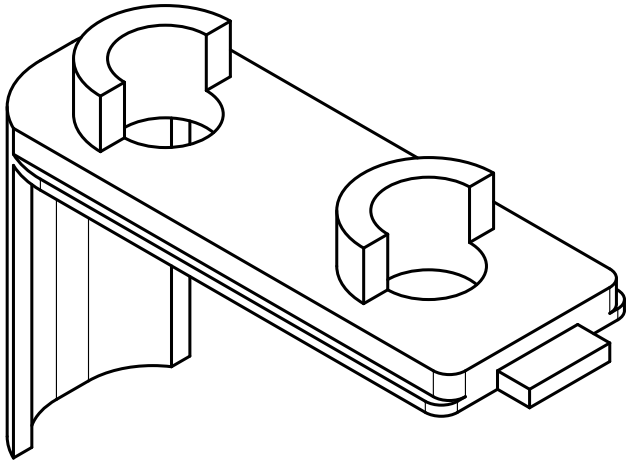
CONJUNTO DEL SEMÁFORO

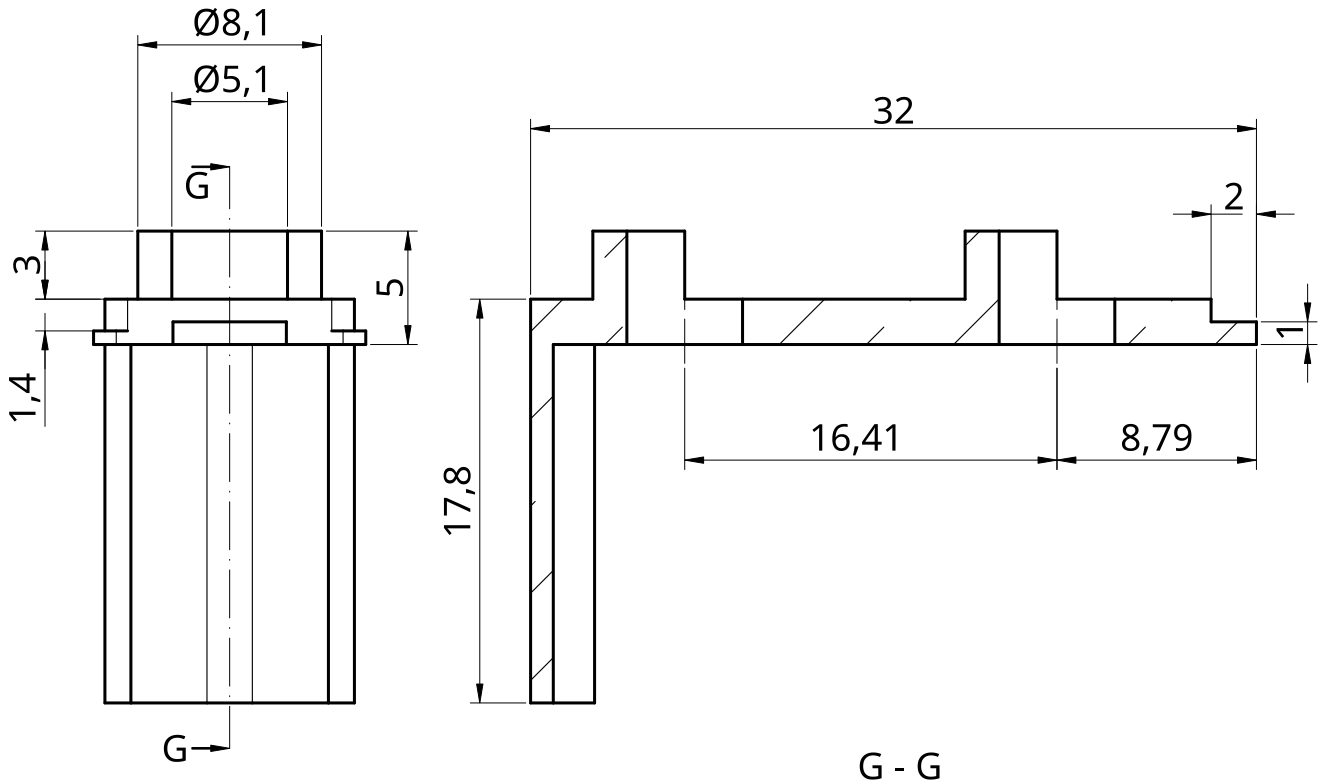
NÚMERO DE DIBUJO 8

ESCALA 1:2

HOJA 8 de 12







G - G

	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

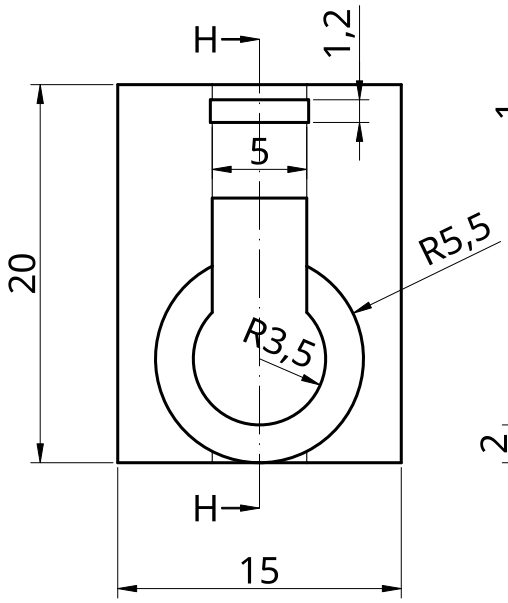
TÍTULO

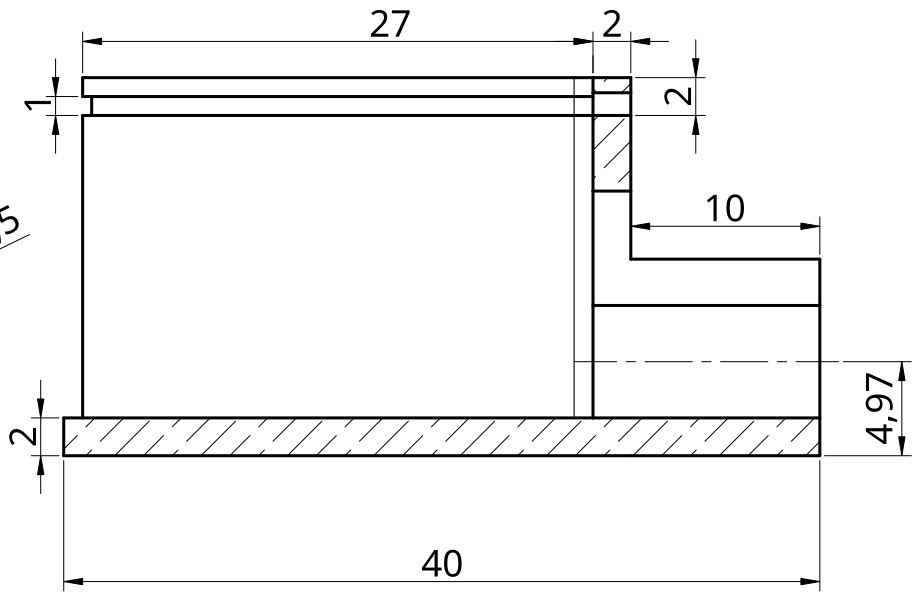
SOPORTES DE LOS LEDES

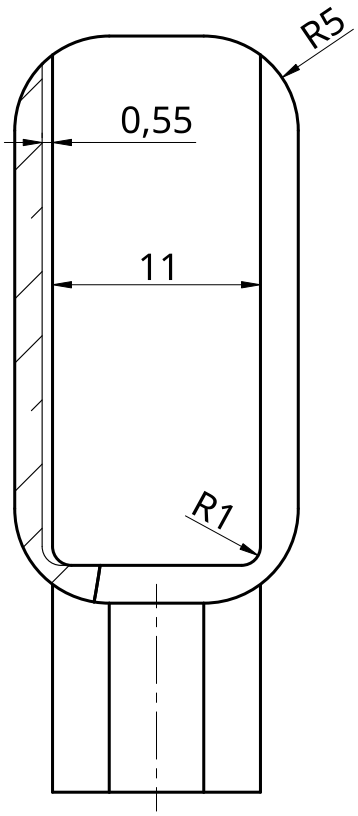
NÚMERO DE DIBUJO 9

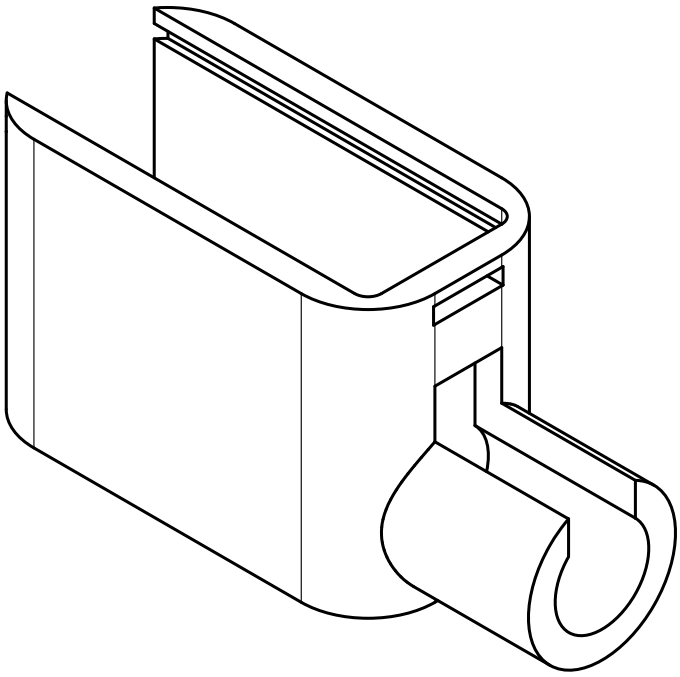
ESCALA 3:1

HOJA 9 de 12









	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

TÍTULO

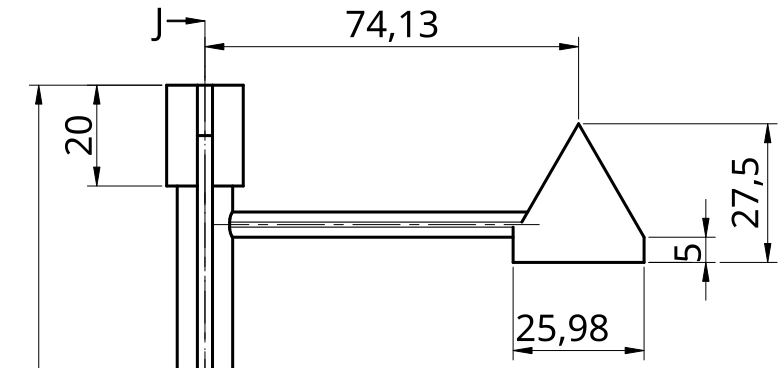
CAJA DE LOS LEDES

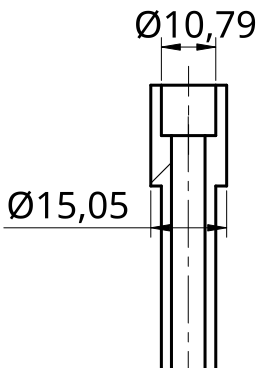
NÚMERO DE DIBUJO

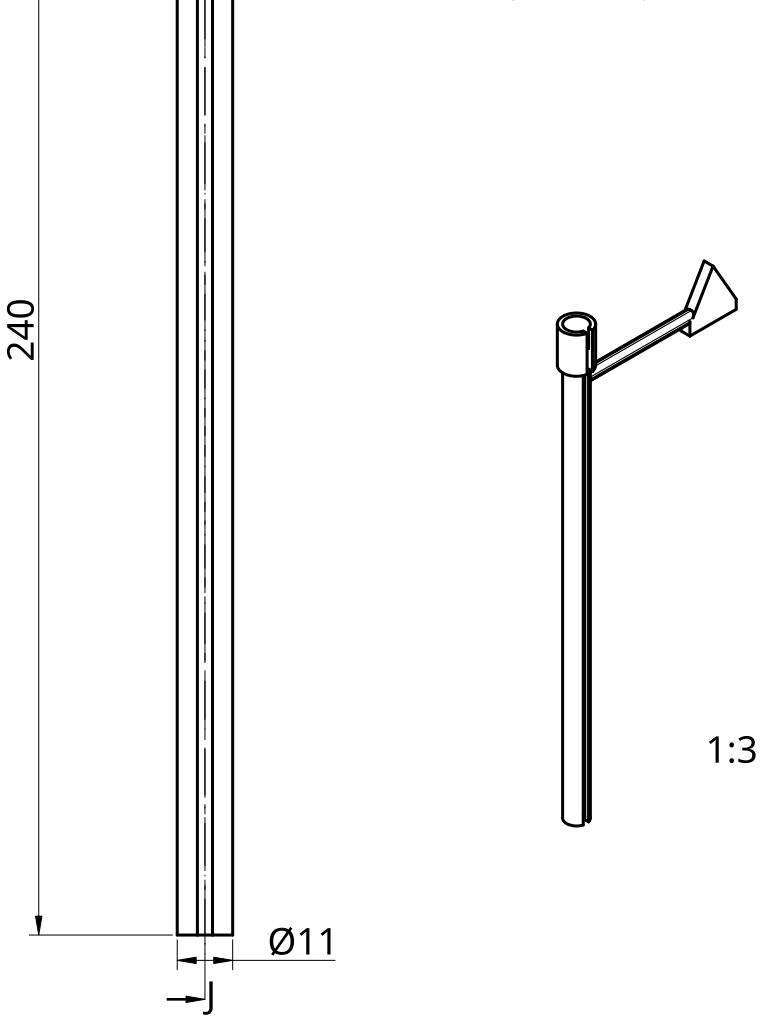
10

ESCALA 2.5:1

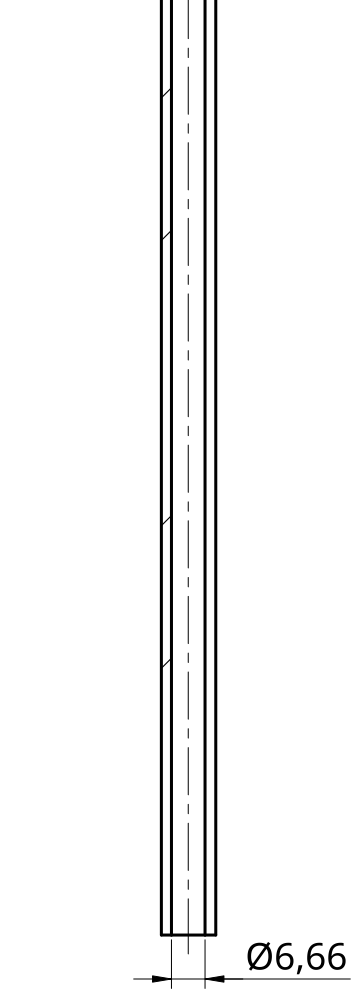
HOJA 10 de 12



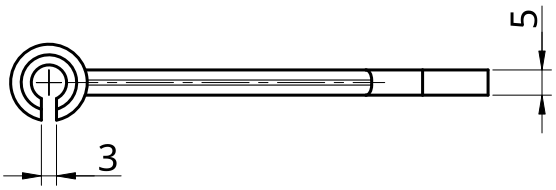




1:3



J - J



	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	
APROBADO POR	JUAN CARLOS BARAZA CALVO	

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

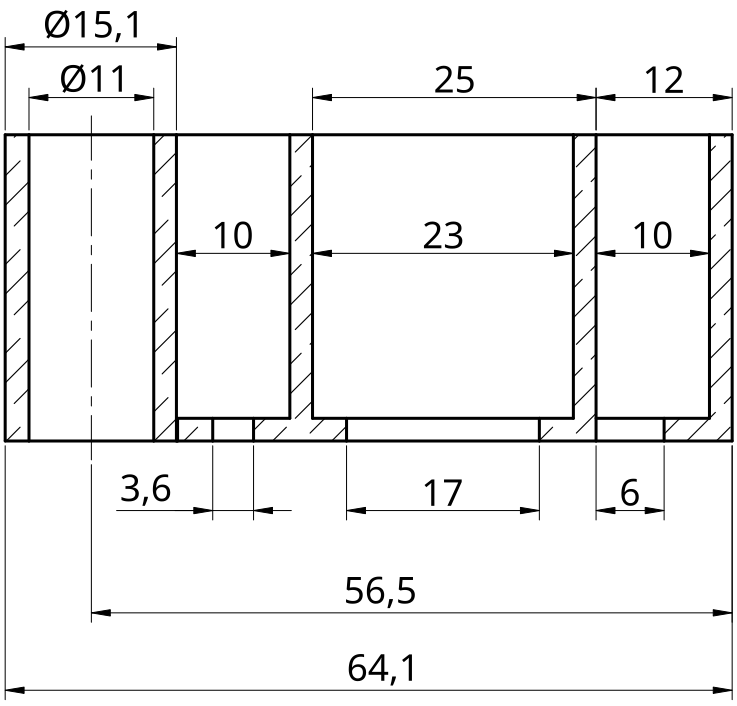
TÍTULO

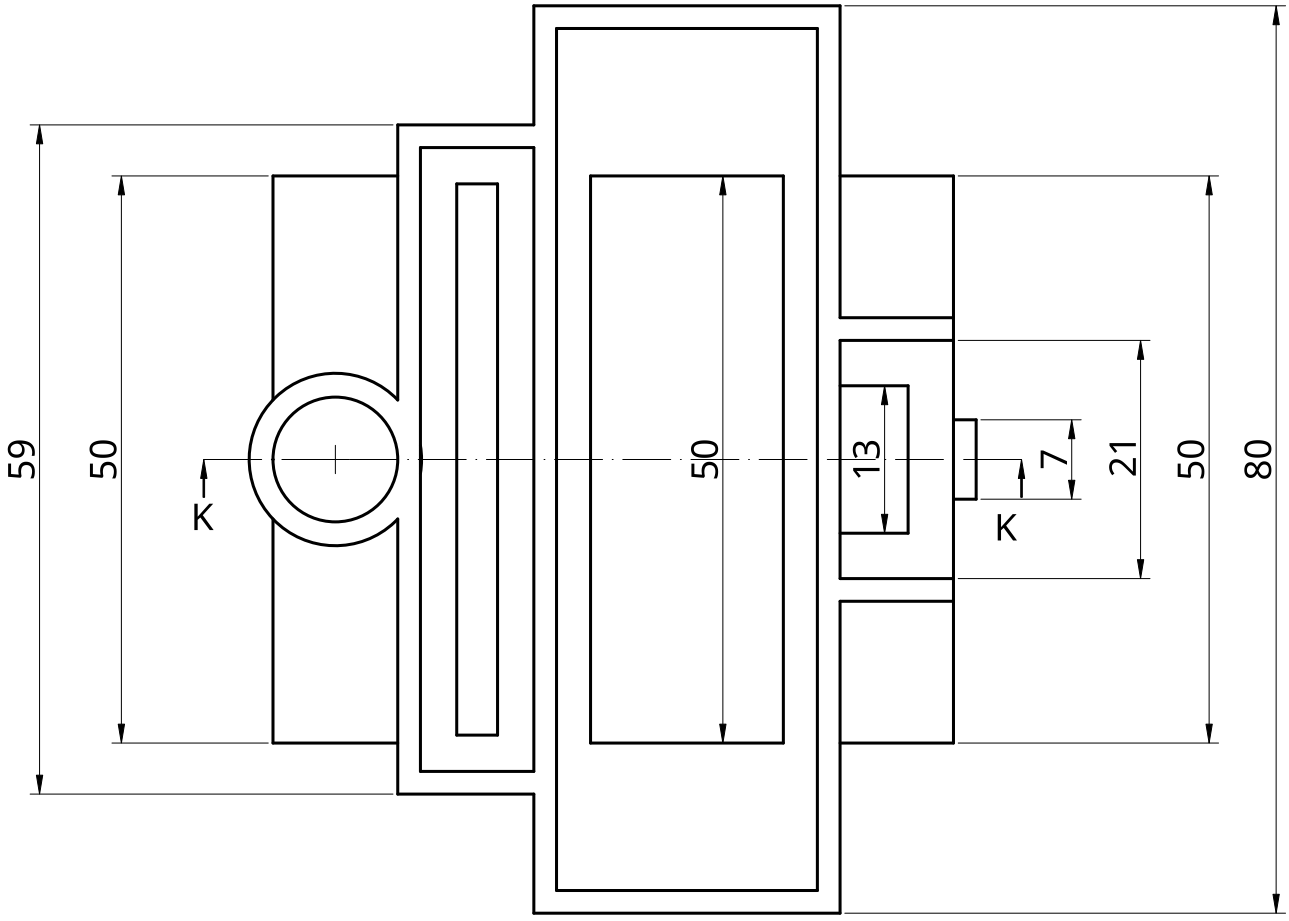
CUERPO DEL SEMÁFORO

NÚMERO DE DIBUJO 11

ESCALA 1:1.5

HOJA 11 de 12





	NOMBRE	FECHA
DIBUJADO POR	BENSON ALEJANDRO KOTEI ZURIAGA	2022-06-30
COMPROBADO POR	JUAN CARLOS BARAZA CALVO	2022-06-30
APROBADO POR	JUAN CARLOS BARAZA CALVO	2022-06-30

Desarrollo de un prototipo a escala de un robot humanoide para pruebas de laboratorio mediante impresión 3D

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIALES

SOPORTE DEL SEMÁFORO

NÚMERO DE DIBUJO: 12

ESCALA: 1.5:1

HOJA: 12 de 12

IV.ANEXOS

CAPITULO. 1 61PROGRAMACIÓN

1.1.1. Función *Detect()*

```
class Detect :
    def red(self, hL, hH, sL, sH, vL, vH) :
        import cv2
        import numpy as np
        cap = cv2.VideoCapture(0)

        x = 0
        y = 0
        red = False

        redBajo1 = np.array([hL, sL, vL], np.uint8)
        redAlto1 = np.array([hH, sH, vH], np.uint8)

        ret,frame = cap.read()

        if ret == True :
            frameHSV = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
            maskRed = cv2.inRange(frameHSV,redBajo1,redAlto1)

            contornos,_ = cv2.findContours(maskRed, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
            for c in contornos :
                area = cv2.contourArea(c)
                if area > 100:
                    red = True
                    M = cv2.moments(c)
                    if (M["m00"]==0): M["m00"]=1
```

```
                                    x = int(M["m10"]/M["m00"])
                                    y = int(M['m01']/M['m00'])
                                    cap.release()
                                    return(x,y,red)
                        return(x,y,red)

        def green(self, hL, hH, sL, sH, vL, vH) :

                import cv2
                import numpy as np
                cap = cv2.VideoCapture(0)

                x = 0
                y = 0
                green = False

                greenBajo1 = np.array([hL, sL, vL], np.uint8)
                greenAlto2 = np.array([hH, sH, vH], np.uint8)

                ret,frame = cap.read()

                if ret == True :
                        frameHSV = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
                        maskgreen = cv2.inRange(frameHSV, greenBajo1, greenAlto2)

                        contornos,_ = cv2.findContours(maskgreen, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
                        for c in contornos :
                                area = cv2.contourArea(c)
                                if area > 100:
                                        green = True
                                        M = cv2.moments(c)
                                        if (M["m00"]==0): M["m00"]=1
```

```
                    x = int(M["m10"]/M["m00"])
                    y = int(M['m01']/M['m00'])
                    cap.release()
                    return(x,y,green)
return(x,y,green)
```

1.1.2. Función *Carril()*

```
def Carril(ref_mid,tol,gauss_k,alpha,canny_down,canny_up,sigma,thres_one):

    import cv2
    import numpy as np
    from Calibration import Calibration as calibrar

    def poscarril (v1,long):
        b=[]
        for i in range(long):
            if v1[i]==255 and i != 638:
                b.append(i)
        if len(b) >= 1:
            return(b[0],b[len(b)-1],len(b))
        else:
            return(0,0,0)

    cap = cv2.VideoCapture(0)
    ret,frame = cap.read()

    if ret == True:
        gaussiano = cv2.GaussianBlur (frame,(gauss_k,gauss_k),alpha)
        gray = cv2.cvtColor (gaussiano, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny (gray, canny_down,canny_up,sigma)
        _,bin=cv2.threshold (edges,thres_one,255,cv2.THRESH_BINARY)

        recorte = bin.copy()
        recorte = recorte[280:480,:]
        x_r,y_r = recorte.shape
        carril = recorte[75,:]
        fin = recorte[:,25]
        x1,x2,c1, = poscarril(carril,y_r)
        y1,y2,c3 = poscarril(fin,x_r)

        centro_carril = int(((x2-x1)/2)+x1)

        carril = True
        recto = False
        izquierda = False
        derecha = False

        if (x1 != 0 and x2 != 0):
          if centro_carril in range (ref_mid-tol,ref_mid+tol) :

            recto = True
          elif centro_carril < ref_mid-tol :

            derecha = True
          elif centro_carril > ref_mid+tol :

            izquierda = True
          else:
            pass



        if y2 > 47 :

            carril = False
```

```
return(carril,recto,izquierda,derecha)
```

1.1.3. Función *Calibration()*

```
class Calibration() :


    def TrackBar(self,color):
        def nothing(x):
            pass


        import cv2
        import numpy as np

        camera = cv2.VideoCapture(0)

        cv2.namedWindow(color)

        cv2.createTrackbar('H Lower',color,0,179,nothing)
        cv2.createTrackbar('H Higher',color,179,179,nothing)
        cv2.createTrackbar('S Lower',color,0,255,nothing)
        cv2.createTrackbar('S Higher',color,255,255,nothing)
        cv2.createTrackbar('V Lower',color,0,255,nothing)
        cv2.createTrackbar('V Higher',color,255,255,nothing)



        while(1):
            _,img = camera.read()

            hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

            hL = cv2.getTrackbarPos('H Lower',color)
            hH = cv2.getTrackbarPos('H Higher',color)
            sL = cv2.getTrackbarPos('S Lower',color)
            sH = cv2.getTrackbarPos('S Higher',color)
            vL = cv2.getTrackbarPos('V Lower',color)
```

```
                vH = cv2.getTrackbarPos('V Higher',color)

                LowerRegion = np.array([hL,sL,vL],np.uint8)
                upperRegion = np.array([hH,sH,vH],np.uint8)

                objec = cv2.inRange(hsv,LowerRegion,upperRegion)

                kernal = np.ones((1,1),"uint8")

                obj = cv2.morphologyEx(objec,cv2.MORPH_OPEN,kernal)
                obj = cv2.dilate(obj,kernal,iterations=1)

                res1=cv2.bitwise_and(img, img, mask = obj)



                cv2.imshow("Masking ",res1)

                if cv2.waitKey(10) & 0xFF == ord('q'):
                        camera.release()
                        cv2.destroyAllWindows()
                        return(hL, hH, sL, sH, vL, vH,)


def TrackCanny(self):
        def nothing(x):
                pass

        import cv2
        import numpy as np

        camera = cv2.VideoCapture(0)
```

```
cv2.namedWindow('Carril')

cv2.createTrackbar('Gauss_k','Carril',1,100,nothing)
cv2.createTrackbar('Gauss_alpha','Carril',1,100,nothing)
cv2.createTrackbar('canny_down','Carril',0,255,nothing)
cv2.createTrackbar('canny_up','Carril',255,255,nothing)
cv2.createTrackbar('sigma','Carril',1,100,nothing)
cv2.createTrackbar('thres_one','Carril',0,255,nothing)

while(1):
        _,img = camera.read()

        gauss_k = cv2.getTrackbarPos('Gauss_k','Carril')
        alpha = cv2.getTrackbarPos('Gauss_alpha','Carril')
        canny_down = cv2.getTrackbarPos('canny_down','Carril')
        canny_up = cv2.getTrackbarPos('canny_up','Carril')
        sigma = cv2.getTrackbarPos('sigma','Carril"Carril')
        thres_one = cv2.getTrackbarPos('thres_one','Carril')

        gaussiano = cv2.GaussianBlur (img,(gauss_k,gauss_k),alpha)
        gray = cv2.cvtColor (gaussiano, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny (gray, canny_down,canny_up,sigma)
        _,bin=cv2.threshold (edges,thres_one,255,cv2.THRESH_BINARY)

        kernal = np.ones((1,1),"uint8")

        binary = cv2.morphologyEx(bin,cv2.MORPH_OPEN,kernal)
        binary = cv2.dilate(binary,kernal,iterations=1)

        res1=cv2.bitwise_and(img, img, mask = binary)
```

```
cv2.imshow("Masking ",bin)


if cv2.waitKey(10) & 0xFF == ord('q'):
        camera.release()
        cv2.destroyAllWindows()
        return(gauss_k,alpha,canny_down,canny_up,sigma,thres_one)
```

1.1.4. Función *Move()*

```
class Move :

    def start(self) :
        from adafruit_servokit import ServoKit
        import time
        kit = ServoKit(channels=16)

        kit.servo[4].angle = 90
        kit.servo[5].angle = 90
        kit.servo[6].angle = 90
        kit.servo[7].angle = 100
        kit.servo[0].angle = 50
        kit.servo[2].angle = 50

    def abajo(self) :
        from adafruit_servokit import ServoKit
        import time
        kit = ServoKit(channels=16)
        kit.servo[2].angle = 135

    def cuello(self, angulo) :
        from adafruit_servokit import ServoKit
        import time
        kit = ServoKit(channels=16)
        kit.servo[0].angle = angulo


    def walk(self,steps,direction) :

        from adafruit_servokit import ServoKit
        import time
```

```
        kit = ServoKit(channels=16)

        def poggers(servo,origen,suma,signo):
                for i in range (suma):
                        kit.servo[servo].angle = origen + (signo*i)
                        time.sleep(0.005)

        for i in range(steps) :
#Walk 1 step
                kit.servo[7].angle = 100-17
                time.sleep(0.2)
                kit.servo[5].angle = 90-32
                time.sleep(0.2)

                poggers(6,90,60,1)
                poggers(4,90,60,1)

                kit.servo[5].angle = 90
                time.sleep(0.2)
                kit.servo[7].angle = 100
                time.sleep(0.2)
                kit.servo[5].angle = 90+17
                time.sleep(0.2)
                kit.servo[7].angle = 100+32
                time.sleep(0.2)

                poggers(6,150,60,-1)
                poggers(4,150,60,-1)
                poggers(4,90,60,-1)
                poggers(6,90,60,-1)
                time.sleep(0.2)
```

```
                kit.servo[5].angle = 90
                time.sleep(0.2)
                kit.servo[7].angle = 100
                time.sleep(0.2)
                kit.servo[5].angle = 90-32
                time.sleep(0.2)
                kit.servo[7].angle = 100-17
                time.sleep(0.2)

                poggers(4,30,60,1)
                poggers(6,30,60,1)

                kit.servo[5].angle = 90
                time.sleep(0.2)
                kit.servo[7].angle = 100

def turn(self,steps,direction) :

        from adafruit_servokit import ServoKit
        import time
        kit = ServoKit(channels=16)

        def poggers(servo,origen,suma,signo):
                for i in range (suma):
                        kit.servo[servo].angle = origen + (signo*i)
                        time.sleep(0.005)


        if direction == 'right' :
                for t in range(steps):
                        kit.servo[7].angle = 100+40
                        time.sleep(0.2)
```

```
                kit.servo[5].angle = 90+20
                time.sleep(0.2)

                poggers(4,90,30,-1)

                kit.servo[7].angle = 100
                time.sleep(0.2)
                kit.servo[5].angle = 90
                time.sleep(0.2)

                poggers(4,60,30,1)

if direction == 'left' :
        for t in range(steps):
                kit.servo[5].angle = 90-30
                time.sleep(0.2)
                kit.servo[7].angle = 100-15
                time.sleep(0.2)

                poggers(6,90,30,1)

                kit.servo[7].angle = 100
                time.sleep(0.2)
                kit.servo[5].angle = 90
                time.sleep(0.2)
                kit.servo[5].angle = 90+30
                time.sleep(0.2)
                kit.servo[7].angle = 100+15
                time.sleep(0.2)


                poggers(6,120,30,-1)
```

```
                kit.servo[7].angle = 100
                time.sleep(0.2)
                kit.servo[5].angle = 90
                time.sleep(0.2)

if direction == 'vuelta' :
        for t in range(steps):
                kit.servo[5].angle = 90-30
                time.sleep(0.2)
                kit.servo[7].angle = 100-15
                time.sleep(0.2)

                poggers(6,90,50,1)

                kit.servo[7].angle = 100
                time.sleep(0.2)
                kit.servo[5].angle = 90
                time.sleep(0.2)
                kit.servo[5].angle = 90+30
                time.sleep(0.2)
                kit.servo[7].angle = 100+15
                time.sleep(0.2)

                poggers(6,140,50,-1)

                kit.servo[7].angle = 100
                time.sleep(0.2)
                kit.servo[5].angle = 90
                time.sleep(0.2)

                kit.servo[7].angle = 100+40
```

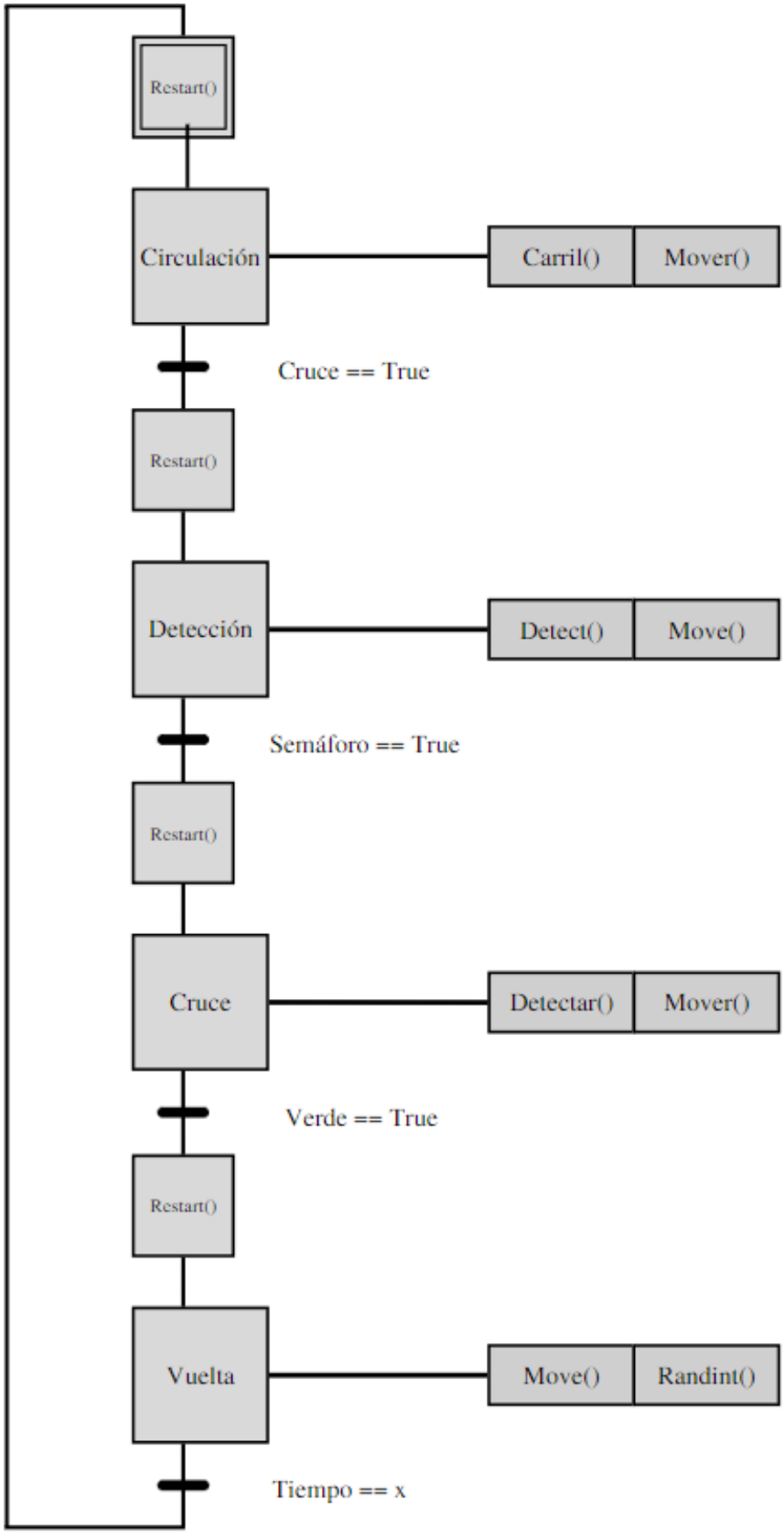
```
time.sleep(0.2)
kit.servo[5].angle = 90+25
time.sleep(0.2)

poggers(4,90,50,1)

kit.servo[7].angle = 100
time.sleep(0.2)
kit.servo[5].angle = 90
time.sleep(0.2)

poggers(4,140,50,-1)
```

1.1.5. Programa completo de circulación

```
from Detect import Detect as detectar
from Move import Move as mover
from CalibrationV2 import Calibration as calibrar
from Carril import Carril as carril
import cv2
import numpy as np
import random
import time

def Restart() :

	mover().start()
	global f
	f = 0

red = False
green = False
f = 0

Restart()

hLr, hHr, sLr, sHr, vLr, vHr = calibrar().TrackBar('ROJO') # Calibracion HSV semaforo en ROJO
hLv, hHv, sLv, sHv, vLv, vHv = calibrar().TrackBar('VERDE') # Calibracion HSV semaforo en VERDE

mover().abajo()
gauss_k,alpha,canny_down,canny_up,sigma,thres_one = calibrar().TrackCanny()

Restart()
mover().abajo()

while True:
	while True :
		acera,recto,izquierda,derecha,cruce,nothing =
carril(320,45,gauss_k,alpha,canny_down,canny_up,sigma,thres_one)
		print (acera,recto,izquierda,derecha,cruce,nothing)
		if acera == False:
			break

		if izquierda == True:
			mover().turn(1,'right')
		if derecha == True:
			mover().turn(1,'left')

		elif recto == True :
			mover().walk(1,'foward')

	Restart()

	while True :
		_,_,red = detectar().red(hLr, hHr, sLr, sHr, vLr, vHr)
		_,_,green = detectar().green(hLv, hHv, sLv, sHv, vLv, vHv)
		print (red,green)

		if red == False and green == False :
			mover().turn(1,'right')
			f=f+1
			pass

		elif red == False and green == True :
```

```
                        mover().turn(f-1,'left')
                        break

                elif red == True and green == False :
                        pass

                else:
                        pass

        Restart()
        mover().abajo()

        while True :
                acera,recto,izquierda,derecha,cruce,nothing =
carril(320,45,gauss_k,alpha,canny_down,canny_up,sigma,thres_one)
                print (acera,recto,izquierda,derecha,cruce,nothing)
                if (recto== False and izquierda == False and derecha == False and acera == True):
                        mover().walk(1,'foward')
                        break

                if izquierda == True:
                        mover().turn(1,'right')
                elif derecha == True:
                        mover().turn(1,'left')

                elif recto == True :
                        mover().walk(1,'foward')

        mover().turn(2,'vuelta')
        tiempo = random.randint(2,9)
        Restart()
        time.sleep(tiempo/2)
        mover().cuello(30)
        time.sleep(1)
        mover().cuello(80)
        time.sleep(1)
        mover().cuello(50)
        time.sleep(tiempo/2)
        mover().turn(2,'vuelta')
        mover().abajo()

        while True :
                acera,recto,izquierda,derecha,cruce,nothing =
carril(320,45,gauss_k,alpha,canny_down,canny_up,sigma,thres_one)
                print (acera,recto,izquierda,derecha,cruce,nothing)
                if (recto== True or izquierda == True or derecha == True):
                        break
                mover().turn(1,'vuelta')
```

CAPITULO. 2 DIAGRAMA GRAFCET COMPLETO



CAPITULO. 3 VIDEO DE PRUEBA

En el siguiente enlace se presenta un video demostrativo del funcionamiento del prototipo en el banco de pruebas:

https://media.upv.es/#/portal/video/6eb97080-fc51-11ec-9dc4-c734c023fbbf