# UNIVERSIDAD INTERNACIONAL DEL ECUADOR

## FACULTAD DE CIENCIAS TÉCNICAS

## ESCUELA DE INGENIERÍA MECATRÓNICA

# DISEÑO DE UN ROBOT ASISTENCIAL PARA LA COMPRA EN UN SUPERMERCADO

**PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN MECATRÓNICA**

**MATEO SEBASTIÁN VERNAZA ORTIZ**

**DIRECTOR: ING. PABLO ANÍBAL VELARDE RUEDA, Ph.D.**

**D. M. Quito**,

**2022**

# CERTIFICACIÓN

El docente de la Facultad de Ciencias Técnicas, escuela de Ingeniería Mecatrónica Pablo Aníbal Velarde Rueda encargado de la revisión del documento final,

CERTIFICA QUE:

El proyecto de investigación denominado “Robot asistencial para la compra en supermercados”, fue desarrollado por el Sr. Mateo Sebastián Vernaza Ortiz y ha sido debidamente revisado y está en condiciones de ser entregado para que siga lo dispuesto por la carrera de Ingeniería Mecatrónica, correspondiente a la sustentación y defensa del mismo.

Firmado electrónicamente por:
PABLO ANIBAL
VELARDE RUEDA

**Ing. Pablo Velarde Rueda, Ph.D.**

**DIRECTOR DE PROYECTO**

*Dedico a mi familia por su motivación y apoyo incondicional.*

# Agradecimientos

Primero agradezco a mi familia por su cariño y confianza en mis años de carrera. A mi hermana por las risas, la confidencia y el ayuda en los momentos más difíciles; a mi madre por los valores, el cariño diario y la confianza ciega en mí; a mi padre por el ánimo, la protección, y por enseñarme que con esfuerzo y sacrificio se consiguen grandes cosas; y a mi madrastra por su afecto y apoyo en este camino.

A mi tutor agradezco por su apoyo constante en el proyecto de titulación y en mi formación. Recibir su conocimiento, sus consejos y sus valores, primero como profesor y luego como tutor, han marcado mi formación profesional y mi forma de ser.

A mis amigos de carrera les agradezco por todos los momentos vividos en estos años, por todas las risas, las malas noches y todas las experiencias que pasamos.

Finalmente, a mis amigos de toda la vida les agradezco por ser mi otra familia que siempre esta pendiente de mí, listos para brindarme su cariño y apoyo cuando lo necesito.

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS



## ÍNDICE DE ANEXOS

# ROBOT ASISTENCIAL PARA LA COMPRA EN SUPERMERCADOS

## 1. Tema

Diseño de un robot asistencial para la compra en un supermercado.

## 2. Objetivos

### 2.1. General

Diseñar un robot asistencial en la compra de bienes de consumo en un supermercado.

### 2.2. Específicos

- Investigar y documentar los principios de funcionamiento de los robots autónomos.
- Seleccionar el mecanismo de traslación del robot móvil.
- Determinar el método de estimación de posición y orientación para el prototipo.
- Definir el sistema de sensores para la evasión de obstáculos.
- Seleccionar el mecanismo de recolección de productos del robot móvil.
- Construir un prototipo que demuestre el concepto del sistema.
- Diseñar los elementos mecánicos y electrónicos del sistema.
- Implementar el algoritmo para el control del robot autónomo.
- Diseñar una aplicación web para realizar el pedido de los bienes de consumo.
- Ejecutar un protocolo de pruebas de funcionamiento del sistema.

## 3. Problema

En diciembre de 2019 se identificó el primer brote de SARS-CoV-2, un virus que puso en emergencia sanitaria a la mayoría de los países en el mundo. Esta pandemia obligó a los

gobiernos a imponer medidas como confinamientos, aislamiento social, cuarentenas, entre otros, para la prevención del contagio. Estas medidas han provocado pérdidas enormes en las economías de los países más afectados debido a la baja o casi nula actividad comercial, pero a partir de la nueva realidad que se vive en el mundo, centrado en el distanciamiento social, se puede observar una evolución en la forma de hacer negocios. Las empresas, rápidamente, se adaptaron a nuevas tecnologías para mantenerse en el juego, ocasionando que el comercio electrónico crezca rápidamente y se use el Internet como medio para la adquisición de bienes y servicios [1] [2].

Específicamente, en los supermercados de Ecuador no existió un cierre completo debido a que ofertan productos de primera necesidad, pero tomaron medidas como horarios restringidos y una reducción de hasta el 50 % del aforo, lo cual no ha afectado significativamente en las ventas; pero sí ha afectado la calidad del servicio ya que el tiempo necesario para realizar las compras ha aumentado debido a largas filas para el ingreso a los establecimientos [3].

Por otro lado, se puede resaltar el crecimiento que ha tenido el comercio electrónico, antes de la pandemia el 33 % de los ciudadanos de Estados Unidos compraban en línea al menos una vez por semana y 69 % compraban en línea por lo menos una vez al mes en el 2015 [4]. Esto se podría interpretar como una necesidad del consumidor para tener diferentes opciones al momento de realizar sus compras, ya que se pierde demasiado tiempo realizando compras de forma física, no solo en el establecimiento, sino también en el tiempo invertido en el tráfico. Según [5], los colombianos pasan cuatro años de su vida haciendo mercado.

Debido a los problemas pre y post-pandemia que se mencionan, han surgido empresas con una gran visión, por ejemplo, Wanlla [6], Disgralec [7], Tipti [8] e incluso grandes cadenas de supermercados como Supermaxi [9], que ofrecen un servicio de supermercado online en Ecuador. Mediante las aplicaciones para smartphones o páginas web de estas empresas se pueden ordenar los productos típicos que se encuentran en un supermercado y entregarán el pedido a domicilio, eliminando las incomodidades de ir personalmente a comprar estos productos.

Pero esta solución aún genera riesgos de contagio para los clientes y entre el perso-

nal encargado de recolectar los productos, ya que aún existe contacto entre personas que puedan ser portadoras del virus, además que puede ser un trabajo agotador para los encargados al tener que cargar las compras y transportarlas durante todo su horario de trabajo.

## 4. Hipótesis

El sistema consistirá en una aplicación móvil, mediante la cual el usuario elegirá los productos que desea adquirir en un supermercado. Una vez realizado el pedido, un robot de manera autónoma recolectará los productos y los preparará para su empacado y entrega para su despacho.

Se construirá un prototipo a escala del robot, que será capaz de cargar un peso máximo de 3 kg y desplazarse a una velocidad máxima de 20 cm/s. Además, el robot podrá evadir obstáculos que se puedan presentar en la trayectoria definida a la hora de recolectar los productos en los pasillos del lugar.

## 5. Estado del Arte

La aplicación de robots como asistentes en tiendas o supermercados aún se encuentra en un estado inicial de pruebas. Sin embargo, hay empresas que han dado un paso al frente y están probando la capacidad de los robots para ayudar a sus clientes a tener un mejor servicio. Por ejemplo, el robot “Marty” (de la empresa Badger Technologies [10]) se desplegó en el 2019 en 172 tiendas de la cadena Giant y 325 tiendas Stop  Shop en Estados Unidos. Este robot de servicio es capaz de reconocer derrames o algún riesgo de tropiezo en el piso y alertar a los empleados para su limpieza. Además, Marty escanea la estantería para reconocer espacios vacíos en las perchas o identificar si los productos están correctamente etiquetados con sus precios correspondientes. La Figura 1 muestra el robot Marty frente a una estantería de un supermercado.


**Figura 1.** Robot asistencial "Marty" [10].

Los robots asistenciales de las tiendas *7Fresh*, abiertas a finales del 2019, que son tiendas de alta tecnología abiertas por el gigante del comercio electrónico JD en China, como se muestra en la Figura 2. En esta tienda los clientes deben descargarse una aplicación móvil y escanear un código QR que se encuentra en el robot, esto enlazará el robot con el cliente y actuará como un carrito de compras autónomo que seguirá al cliente alrededor de la tienda y además escaneará los productos introducidos en él, para que cuando el cliente termine de comprar pueda pagar con reconocimiento facial.


**Figura 2.** Robots asistenciales de las tiendas *7Fresh*.

Otra aplicación donde los robots asistenciales para compras están comenzando a probarse son los robots autónomos para el transporte y entrega de alimentos. Las empresas *Starship* [11] y *Marble* [12] están probando robots autónomos para entregar comida y otros

productos directamente a sus clientes. En Washington D.C., *Starship* ha lanzado varios de sus robots que poseen seis ruedas y son completamente eléctricos, con una altura de aproximadamente medio metro y con una capacidad de carga de hasta 9 kg. Estos robots son capaces de moverse por las aceras con cámaras y tecnología de rastreo para evitar chocarse con personas, vehículos u otros obstáculos, la Figura 3 muestra un grupo de robots de la empresa *Starship*. Otro ejemplo similar son los robots de la empresa *Marble* que en conjunto con la compañía de entrega de comida *Yelp Eat24* ofrecen un servicio puerta a puerta mediante el uso de robots autónomos, estos robots están siendo probados en San Francisco. Sin embargo, una de las limitaciones de estos robots es que deben realizar las entregas en lugares cercanos, a unos pocos kilómetros de distancia de su centro de control.

**Figura 3.** Robots repartidores de la empresa *Starship* [11].

En el área de investigación tecnológica, [13] y [14] exponen un sistema de compras de alimentos de forma remota en un supermercado, el cual será teleoperado por el cliente para que escoja el producto que necesite sin encontrarse físicamente en la tienda. En la investigación se construye un sistema que consta de un manipulador móvil, una pinza succionadora, un localizador láser como escáner para identificar la forma del producto y una interfaz de comunicación, que a su vez involucra el desarrollo de sistemas navegación autónoma, percepción del ambiente, manipulación de objetos y teleoperación, para permitir a los clientes escoger comida fresca de forma remota. En [13] se presenta el mecanismo de agarre, que se muestra en la Figura 4, el cual consiste en un sistema de ventosas que exitosamente cumplen el objetivo de manipular varios objetos con diferentes formas y texturas,

conociendo previamente la forma y curvatura de los productos.

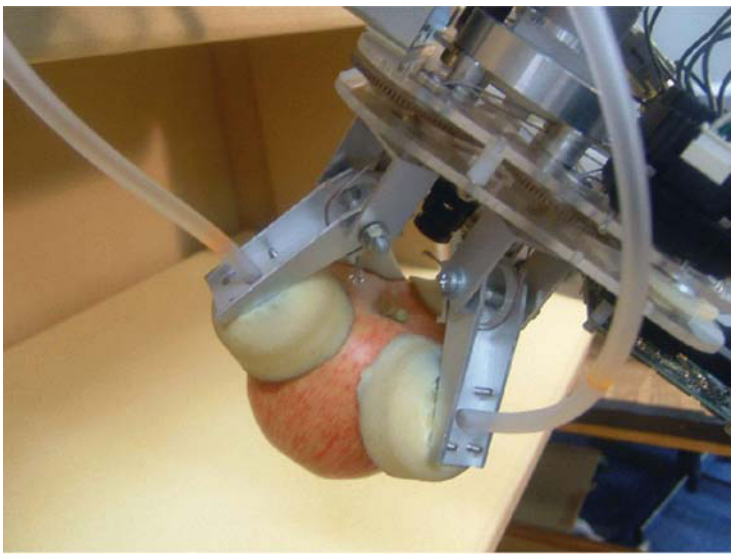


**Figura 4.** Mecanismo de sujeción [13].

Un sistema de control para carritos de compras que sigue automáticamente a los clientes [15], está diseñado para personas con edad avanzada o con discapacidad motriz que tengan problemas para cargar sus compras o manejar un carrito por su cuenta. En su investigación implementaron un grupo de sensores en su robot como encoders rotatorios, sensores de giroscopio y sensor de alcance láser para identificar al cliente y sus movimientos. Además, se presenta una posible forma de identificar obstáculos y de realizar un sistema de mapeo mediante el uso de sensores láser.

Pueden existir diferentes configuraciones para implementar robots móviles, por ejemplo, se plantea un sistema automático de transporte de equipaje mediante un robot de tipo péndulo invertido y plantean una función de navegación en un ambiente real [16]. Su sistema implementado consiste en dos subsistemas, un control balancín y de desplazamiento y un subsistema de navegación. El prototipo maneja una carga máxima de 7.5 kg; presenta una interesante forma de corregir los errores en la posición estimada debido al giroscopio y al cambio en el centro de gravedad al cargar el equipaje. Un estudio similar implementa un robot asistente con ruedas del tipo péndulo invertido con control en los movimientos para que el robot se ponga de pie, se siente y se mueva [17], como se muestra en la Figura 5.

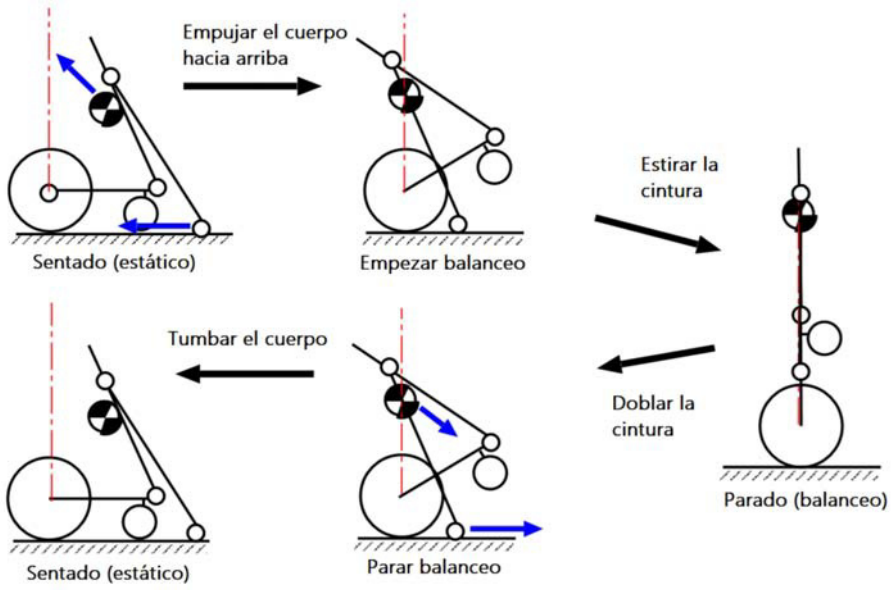



**Figura 5.** Secuencia de movimientos para un robot balancín [17].

Sistemas de navegación en ambientes de interior para robots móviles autónomos usando una red de sensores inalámbricos que cuenta con un algoritmo de navegación que no necesita un mapa del ambiente, es presentado en [18]. Se basa en una red de sensores de radiofrecuencia (RF) mediante el cual un robot autónomo podrá estimar su posición al adquirir información de emisores de radio preconfigurados. El algoritmo conceptualmente es simple y de fácil implementación, puede calcular la posición estimada del robot al medir la distancia hacia los sensores y realizar una triangulación. De forma similar, en [19] y [20] se presentan sistemas basados en identificación por radiofrecuencia (RFID) para estimar la posición del robot, por ejemplo Robocart, el cual es un prototipo de un robot asistencial de compras para personas con discapacidad visual. Este prototipo utiliza etiquetas RFID desplegadas a lo largo de una tienda para determinar su localización, con el propósito ayudar a las personas con discapacidad visual a navegar por la tienda y encontrar los productos que necesitan.

Un enfoque para permitir que un robot asistente autónomo alcance y transporte objetos mientras evita obstáculos es presentado en [21], para esto en la investigación han generalizado el enfoque dinámico de un atractor que suele ser establecido para vehículos, para la formación de trayectorias en brazos robóticos. Esto permite que el robot se desenvuelva en ambientes que varían en el tiempo, como suele ser una tienda en la que están presentes

personas. Por otro lado, en [22] se demuestra por simulación y mediante experimentación en el mundo real un programa para la planificación de movimiento en un robot articulado, similar a un brazo robótico, diseñado para recoger y colocar productos de forma automática en un almacén. El sistema propuesto es capaz de seleccionar automáticamente objetos de un estante y colocarlas en una bolsa y viceversa, según un listado de pedidos.

Para la planificación de la trayectoria se presenta en [23], un enfoque basado en depredadores y presas para una planificación adaptiva en situaciones como la introducción repentina de obstáculos estacionarios o dinámicos. El enfoque está diseñado a partir de tres funciones de recompensa, una inspirada en un modelo de depredador y presa, la segunda en una recompensa por continuar el movimiento en una dirección recta y la tercera está relacionada con la cobertura de los límites. Además, un algoritmo para un robot de tipo monociclo para que siga una trayectoria deseada, incluyendo la capacidad de evitar obstáculos en el camino [24]. El control se basa en la teoría de estabilidad de Liapunov, en técnicas de *backstepping* y aborda de forma explícita la dinámica del robot.

Además, en [25] se enfrenta el problema de la navegación de robots móviles en ambientes interiores desconocidos o parcialmente conocidos, basado en la combinación de comportamientos elementales alcanzados a partir de sistemas difusos inferenciales, proponiendo dos tipos de comportamientos para evitar obstáculos, uno para obstáculos convexos y otro para obstáculos cóncavos.

## 6. Análisis de Alternativas

Se utilizan métodos sistemáticos de análisis de calidad y alternativas para definir técnicamente cuales son las cualidades o requerimientos más importantes que debe cumplir el prototipo del robot, y también, de acuerdo con estos requerimientos, establecer las características o criterios técnicos más importantes. También, se analizan posibles soluciones para los diferentes componentes de los sistemas, y como se pueden combinar las diferentes opciones para determinar cual es la mejor solución para el problema.

## 6.1. Despliegue de la Función de la Calidad (QFD)

En la Figura 6 se muestra el despliegue de la función de calidad (QFD), donde se definen los requerimientos más importantes para el diseño del prototipo en la parte izquierda de la matriz. Además, se plantean las características técnicas que se deben considerar para cumplir con los requerimientos y se evalúa su relación con cada uno de ellos. Según, la relación establecida y la importancia del requerimiento se puede determinar el orden de importancia de cada criterio técnico, como se muestra en la parte inferior de la matriz, este análisis permite estructurar claramente cuales son las características más importantes cuando se diseñe el sistema.

| | | | | CARACTERÍSTICAS/CRITERIOS | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | IMPORTANCIA | PONDERACIÓN RELATIVA | ORDEN DE IMPORTANCIA | Conexión a Internet | Capacidad de la batería | Generador de trayectorias | Localizador | Controlador de posición y velocidad | Planeador local | Control de velocidad de las ruedas | Diseño de mecanismos | Torque/potencia de motores | Capacidad de procesamiento | Sensores | Diseño adaptable y modular | Diseño orientado al peso | Diseño orientado al tamaño | Diseño orientado a la estética | Diseño para procesos y piezas de bajo costo |
| **REQUERIMIENTOS** | | | | | | | | | | | | | | | | | | | |
| Pedidos en línea | **5** | **11%** | **1** | 9 | | | | | | | | | 9 | | | | | | |
| Buena autonomía de carga | **3** | **7%** | **10** | 3 | 9 | | | | | 3 | | 8 | 8 | 2 | | 5 | | | |
| Movimiento autónomo | **5** | **11%** | **2** | | 2 | 9 | 9 | 9 | 9 | 9 | 2 | 2 | 5 | 8 | | | | | |
| Movimiento eficiente | **3** | **7%** | **11** | | | 9 | 9 | 9 | 9 | 8 | 3 | | | 4 | | 5 | | | |
| Silencioso | **4** | **9%** | **5** | | | | | | | 9 | 9 | | 8 | 3 | | 2 | | | 7 |
| Agarre de productos | **5** | **11%** | **3** | | | 7 | 7 | 8 | | | 8 | 9 | | | | | | | |
| Bajo costo | **5** | **11%** | **4** | 2 | 8 | | | | | | 9 | 7 | 9 | 9 | 4 | 5 | 9 | | 9 |
| Evasión de obstáculos | **4** | **9%** | **6** | | | 7 | 5 | 7 | 8 | | | | 3 | 9 | | | | | |
| Modular/Adaptable | **4** | **9%** | **7** | | | | | | | | 9 | | 4 | | 9 | | | 2 | 8 |
| Peso reducido | **2** | **5%** | **12** | | 5 | | | | | | 8 | 2 | | | 4 | 9 | 6 | | 2 |
| Tamaño reducido | **4** | **9%** | **8** | | 7 | | | | | | 8 | 5 | 5 | 6 | 9 | 4 | 9 | | 2 |
| Diseño atractivo | **1** | **2%** | **13** | 2 | | | | | | | 8 | | | 2 | 4 | | 4 | 9 | |
| Facilidad de fabricación | **4** | **9%** | **9** | | 5 | | | | | | 9 | 5 | 8 | 5 | 9 | 2 | 4 | 5 | 8 |
| **PONDERACIÓN ABSOLUTA** | | | | **66** | **135** | **135** | **127** | **140** | **104** | **114** | **268** | **158** | **251** | **197** | **140** | **105** | **113** | **37** | **149** |
| **PONDERACIÓN RELATIVA** | | | | **3%** | **6%** | **6%** | **6%** | **6%** | **5%** | **5%** | **12%** | **7%** | **11%** | **9%** | **6%** | **5%** | **5%** | **2%** | **7%** |
| **ORDEN DE IMPORTANCIA** | | | | **15** | **8** | **9** | **10** | **6** | **14** | **11** | **1** | **4** | **2** | **3** | **7** | **13** | **12** | **16** | **5** |

**Figura 6.** Matriz de despliegue de la función de la calidad.

## 6.2. Matriz Morfológica

La matriz morfológica es una técnica utilizada para definir las posibles alternativas de solución a los diferentes componentes de un problema, y encontrar cuales pueden ser las diferentes combinaciones que permitan una solución factible. En la Figura 7 se muestra la matriz morfológica para los algoritmos de control del sistema, los componentes electrónicos

principales y los mecanismos del robot. En la parte de algoritmos se puede evidenciar que los diferentes componentes analizados se pueden combinar con todas las soluciones posibles excepto por el método SLAM (*Simultaneous Localization And Mapping*), ya que este método por sí solo es una solución para estos componentes. También, se debe mencionar que en el caso de los componentes electrónicos no existe la posibilidad de utilizar una placa Arduino para trabajar con una cámara de video, ya que estos elementos no tienen la capacidad de procesamiento necesaria para alojar un modelo de visión artificial adecuado.

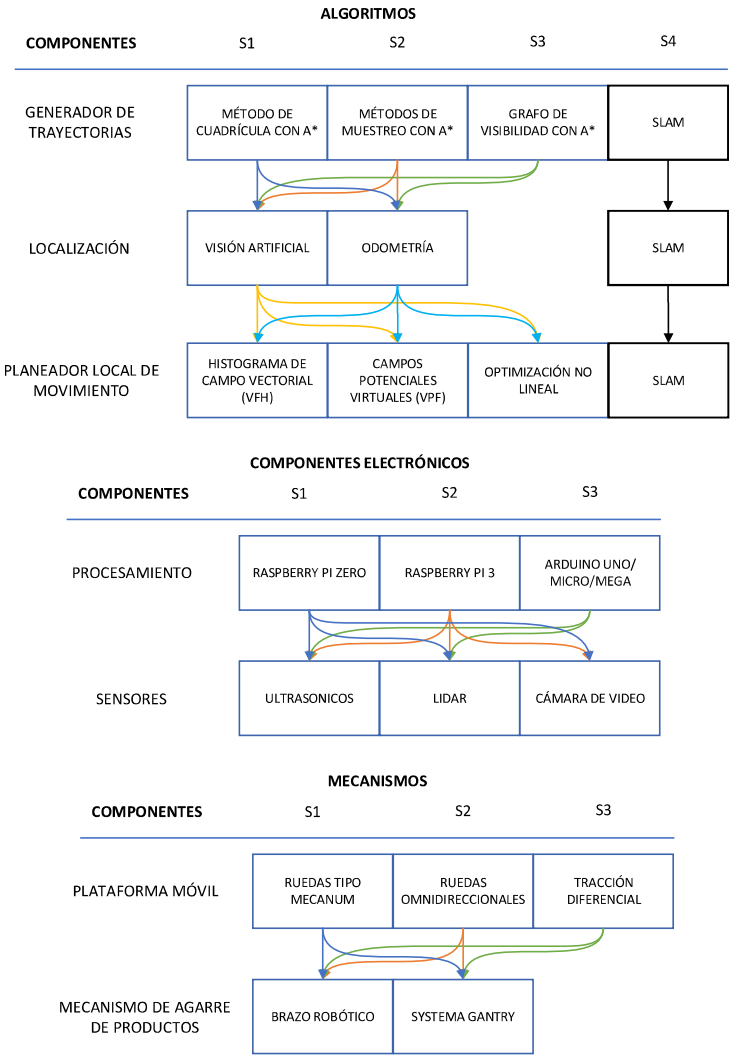



**Figura 7.** Matriz morfológica.

## 6.3. Matriz de Pugh

Partiendo del orden de importancia de los requerimientos y características técnicas obtenidas de la matriz QFD que se muestra en la Figura 8, y de las posibles soluciones mos-

trados en la matriz morfológica de la Figura , se analiza mediante la matriz de Pugh como es el rendimiento de cada solución respecto a diferentes criterios. A partir de los puntajes otorgados se selecciona la mejor solución como la solución con mayor puntaje para cada componente analizado.

| | | ALTERNATIVAS ALGORITMOS | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | GENERADOR DE TRAYECTORIAS | | | | LOCALIZACIÓN | | | | PLANEADOR LOCAL | | | |
| | | S1 | S2 | S3 | S4 | S1 | S2 | S3 | S4 | S1 | S2 | S3 | S4 |
| CRITERIOS | Costo | 1 | 1 | 2 | -2 | -2 | 2 | - | -2 | 2 | 2 | -2 | -2 |
| | Facilidad de diseñar | 2 | 2 | 2 | -1 | 0 | 1 | - | -1 | 0 | 2 | -2 | -1 |
| | Facilidad de implementar | 0 | 1 | 2 | -2 | 0 | 2 | - | -2 | 1 | 2 | -1 | -2 |
| | Procesamiento | 0 | 0 | 2 | -1 | -2 | 2 | - | -1 | 1 | 2 | 0 | -1 |
| | Funcionalidad para aplicación | -1 | -2 | 1 | 2 | 1 | -1 | - | 2 | 1 | 0 | 1 | 2 |
| | **SUMA** | **2** | **2** | **9** | **-4** | **-3** | **6** | **-** | **-4** | **5** | **8** | **-4** | **-4** |
| | **RANKING** | **2** | **3** | **1** | **4** | **2** | **1** | **-** | **3** | **2** | **1** | **3** | **4** |

| | | ALTERNATIVAS COMPONENTES ELECTRÓNICOS | | | | | |
|---|---|---|---|---|---|---|---|
| | | PROCESAMIENTO | | | SENSORES | | |
| | | S1 | S2 | S3 | S1 | S2 | S3 |
| CRITERIOS | Costo | 0 | -2 | 2 | 2 | -2 | -1 |
| | Disponibilidad | -1 | -2 | 2 | 2 | -2 | 0 |
| | Complejidad de uso | 2 | 2 | 2 | 2 | -1 | -1 |
| | Funcionalidad para aplicación | 2 | 2 | -2 | -1 | 2 | 2 |
| | Conexión a Internet | 2 | 2 | -2 | - | - | - |
| | **SUMA** | **3** | **2** | **2** | **5** | **-3** | **0** |
| | **RANKING** | **1** | **2** | **3** | **1** | **3** | **2** |

| | | ALTERNATIVAS SISTEMA MECÁNICO | | | | |
|---|---|---|---|---|---|---|
| | | PLATAFORMA MÓVIL | | | MECANISMO DE AGARRE | |
| | | S1 | S2 | S3 | S1 | S2 |
| CRITERIOS | Costo | -2 | 0 | 2 | -2 | 2 |
| | Disponibilidad de elementos | -1 | -1 | 2 | 0 | 2 |
| | Facilidad de diseñar | 2 | 2 | -2 | -2 | 1 |
| | Facilidad de implementar | 2 | 0 | 0 | -2 | 2 |
| | Funcionalidad para aplicación | 2 | 0 | -1 | 2 | -1 |
| | **SUMA** | **3** | **1** | **1** | **-4** | **6** |
| | **RANKING** | **1** | **2** | **3** | **2** | **1** |

**Figura 8.** Matriz de Pugh.

Según la matriz presentada en la Figura 8 se determina que el sistema se diseña con las siguientes soluciones:

- para la generación de trayectorias se escogen algoritmos de grafos de visibilidad junto con un algoritmo de búsqueda A*,
- para estimar la posición del robot se escoge la odometría,
- como planeador local se utiliza los métodos de campos potenciales virtuales,
- el procesamiento se diseña para un Raspberry Pi Zero,
- para la detección de obstáculos se eligen los sensores ultrasónicos,

- el sistema de la plataforma móvil es un robot holonómico con ruedas tipo mecanum,
- y para el mecanismo de agarre se utiliza un sistema tipo gantry.

### 6.4. Esquema del Robot

A partir de las alternativas seleccionadas, principalmente en los mecanismos y en los componentes electrónicos, se diseña un esquema del prototipo que se muestra en la Figura 9.

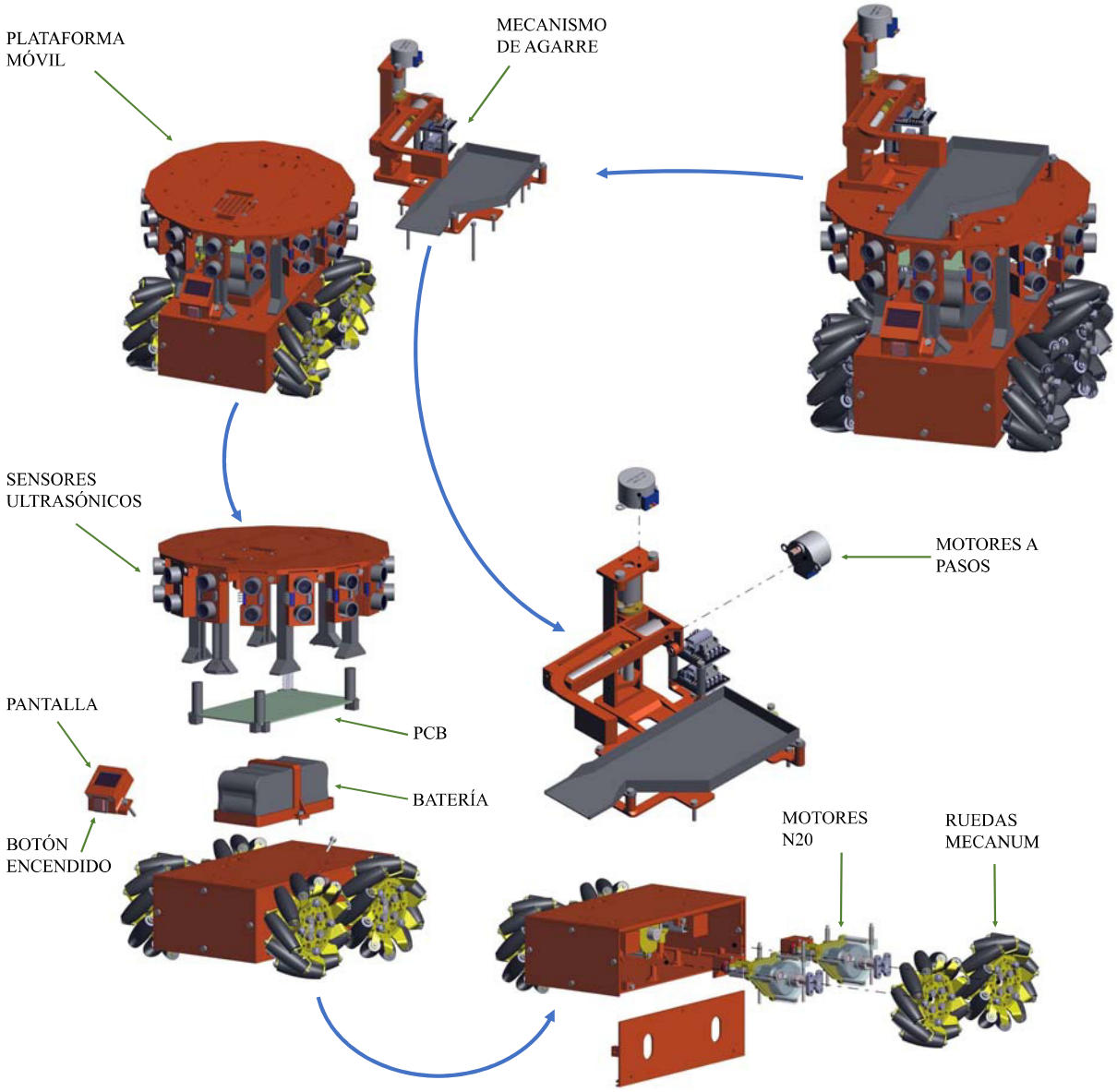



**Figura 9.** Esquema del prototipo.

Se puede evidenciar que el prototipo se divide en dos subsistemas principales; el primero

es una plataforma móvil que contiene el mecanismo de locomoción y la mayoría de los componentes electrónicos; y el segundo es el mecanismo de agarre de productos que está montado sobre la plataforma móvil, donde se encuentran los motores a pasos y los *drivers* respectivos. El mecanismo de locomoción cuenta con 4 *mecanum wheels* para que el robot sea de tipo holonómico, es decir, el robot no tiene restricciones de movimiento en el plano bidimensional.

## 7. Estudio Teórico

### 7.1. Matriz Antisimétrica

Dado un vector $x = [x_1 \ x_2 \ x_3]^T$, la representación matricial antisimétrica de $x$ permite representar el vector como una matriz antisimétrica, la cual se expresa como $[x]$ y se define mediante (1).

$$[x] = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{1}$$

Donde la característica que define esta matriz es que cumple que:

$$[x] = -[x]^T$$

### 7.2. Representación de Sistemas de Referencia

Existen varias formas de representar la orientación o rotación de un sistema de referencia en el espacio. La forma más común es mediante una matriz de rotación, pero también se tiene la representación en coordenadas exponenciales, ambas formas serán empleadas en este trabajo.

### 7.3. Matriz de Rotación $SO(3)$

Las matrices de rotación, representadas por $R$, son un grupo de matrices que pertenecen al subconjunto denominado $SO(3)$, donde las características que definen a este conjunto de matrices es que cumplen la condición de las matrices ortogonales y cuyo determinante

es uno.

$$R^T R = I$$

$$det(R) = 1$$

Las matrices de rotación se utilizan para definir orientación o rotación de un sistema de referencia en el espacio, ya que cada fila o columna representa el giro respecto a cada uno de los tres ejes principales del sistema de referencia.

### 7.4. Representación en Coordenadas Exponenciales $so(3)$

A diferencia de la representación mediante una matriz de rotación, cuando se representa la orientación o rotación de un sistema de referencia en $so(3)$ se utiliza un solo eje unitario de rotación, denominado $\hat{\omega} \in \mathbb{R}^3$, y un ángulo $\theta$, como se muestra en la Figura 10. Donde el vector $\hat{\omega}\theta \in \mathbb{R}^3$ cumple la función de representar la rotación mediante tres parámetros.

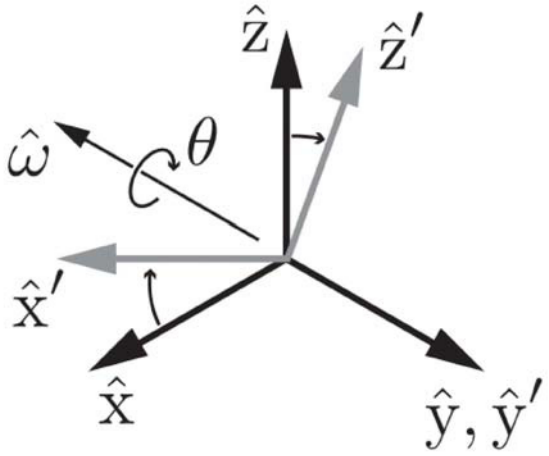



**Figura 10.** Representación mediante un eje de rotación unitario $so(3)$ [26].

Para representar el sistema de referencia en forma matricial en $so(3)$ se utiliza la representación matricial antisimétrica de $\hat{\omega}$ multiplicada por el ángulo $\theta$, representado como $[\hat{\omega}]\theta$.

La representación en coordenadas exponenciales tiene varias interpretaciones [26], las cuales son:

- si se considera a $\hat{\omega}$ como un eje y $\theta$ como un ángulo, entonces se puede pensar en un sistema de referencia inicial $\{s\}$ que es girado respecto el eje $\hat{\omega}$ por un ángulo $\theta$, su orientación final respecto a $\{s\}$ se puede expresar por una matriz de rotación $R$.

- si se considera a $\hat{\omega}\theta$ como una velocidad angular expresada en $\{s\}$, entonces se puede pensar en un sistema de referencia inicial coincidente con $\{s\}$ que se mueve a $\hat{\omega}\theta$ por una unidad de tiempo (básicamente una integración de la velocidad para conocer la posición final después de una unidad de tiempo), donde la orientación final se puede expresar por una matriz de rotación $R$.

- si se considera solo $\hat{\omega}$ como una velocidad angular expresada en $\{s\}$, entonces se puede pensar en un sistema de referencia inicial coincidente con $\{s\}$ que sigue $\hat{\omega}$ por $\theta$ unidades de tiempo (una integración de la velocidad para conocer la posición final después de $\theta$ unidades de tiempo), donde la orientación final se puede expresar por una matriz de rotación $R$.

Las dos últimas interpretaciones consideran a las coordenadas exponenciales dentro de un marco de ecuaciones diferenciales lineales.

Según [26], una ecuación diferencial lineal de la forma $\dot{x}(t) = Ax(t)$, donde $A \in \mathbb{R}^{n\times n}$ es constante y $x(t) \in \mathbb{R}^n$, y con condición inicial $x(0) = x_0$, tiene como solución (2). Además, la función $e^{At}$ tiene a (3) como su expansión en serie.

$$x(t) = e^{At}x_0 \tag{2}$$

$$e^{At} = I + tA + \frac{t^2}{2!}A^2 + \frac{t^3}{3!}A^3 + \cdots \tag{3}$$

Por lo tanto, si se considera un vector tridimensional $p(0)$ y se lo rota respecto a $\hat{\omega}$ por $t = \theta$ hasta $p(\theta)$, como se muestra en la Figura 11, y bajo la última interpretación de las coordenadas exponenciales de rotación, entonces la velocidad está dada por (4).

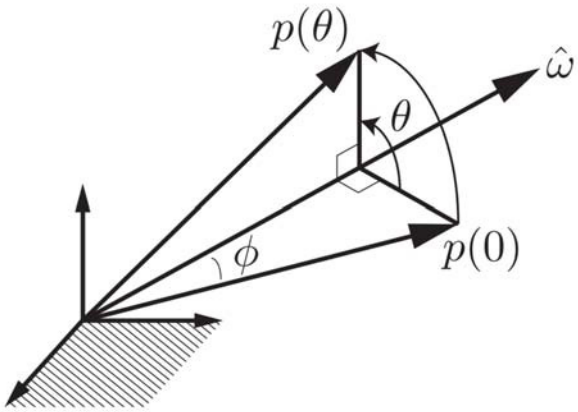



**Figura 11.** Representación del movimiento de $p(0)$ a $p(\theta)$ [26].

$$\dot{p}(t) = \hat{\omega} \times p(t) \tag{4}$$

La ecuación (4) puede ser expresada como $\dot{p} = [\hat{\omega}]p$, la cual es una ecuación diferencial lineal de la forma $\dot{x} = Ax$, con solución (2). Es decir, la solución de (4) es $p(t) = e^{[\hat{\omega}]t}p(0)$, lo que es equivalente a (5).

$$p(\theta) = e^{[\hat{\omega}]\theta}p(0) \tag{5}$$

De manera similar se puede expandir la serie de (5) mediante (3) y con algunas simplificaciones se puede deducir que para un vector $\hat{\omega}\theta \in \mathbb{R}^3$, donde $\theta$ es un escalar y $\hat{\omega} \in \mathbb{R}^3$ es un vector unitario, entonces la matriz exponencial de $[\hat{\omega}]\theta$ es (6) [26].

$$R = Rot(\hat{\omega}, \theta) = e^{[\hat{\omega}]\theta} = I + sen(\theta)[\hat{\omega}] + (1 - cos(\theta))[\hat{\omega}]^2 \in SO(3) \tag{6}$$

Donde la interpretación de (6) es que la matriz exponencial $e^{[\hat{\omega}]\theta}$ es el factor que relaciona una posición inicial $p(0)$ con la posición final $p(\theta)$. Esta ecuación (6) es conocida como la fórmula de Rodrigues para rotaciones.

### 7.5. Matriz Logarítmica de Rotación

Es el inverso de la matriz exponencial. Por lo tanto, se obtiene la rotación respecto a un eje de giro $[\hat{\omega}]\theta \in so(3)$ a partir de una matriz de rotación $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \in SO(3)$.

La matriz logarítmica de rotación se calcula mediante el Algoritmo 1 en el Anexo A, donde $tr(R)$ es la traza de la matriz $R$.

### 7.6. Representación de Orientación y Posición

Además de poder representar la orientación o rotación de un sistema de referencia, es necesario poder representar la posición o traslación del sistema. Para esto se suele utilizar la matriz de transformación $T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \in SE(3)$, donde $R$ es una matriz de rotación y $p \in \mathbb{R}^3$ es un vector que representa la posición o traslación espacial.

### 7.7. Representación de Velocidades

En muchas ocasiones se requiere trabajar con las velocidades angulares y velocidades lineales, las cuales se pueden representar de varias formas, una de ellas mediante un $twist$. Un $twist$ es un vector representado por $\Upsilon$, igual a (8), donde los tres primeros términos corresponden a la velocidad angular y los tres últimos términos equivalen a la velocidad lineal.

$$\Upsilon = \begin{bmatrix} \omega \\ v \end{bmatrix} \in \mathbb{R}^6 \tag{7}$$

### 7.8. Interpretación de un Twist como un Movimiento de Tornillo

Un $twist$ puede ser interpretado como un “eje de tornillo” unitario representado por $S$ y una velocidad angular $\dot{\theta}$ alrededor de $S$. Un eje de tornillo es llamado así ya que representa los movimientos que ocurren al mover un tornillo, un movimiento giratorio alrededor de un eje mientras al mismo tiempo se traslada linealmente a lo largo del mismo. De esta forma se puede representar en el mismo vector rotación y traslación de una forma mas intuitiva.

Una representación de $S$ puede ser mediante los elementos $(q, \hat{s}, h)$, donde $\hat{s} \in \mathbb{R}^3$ es un vector unitario que muestra la dirección del eje de rotación, $q \in \mathbb{R}^3$ es cualquier punto que se encuentre sobre el eje y $h$ es el avance o la relación entre el desplazamiento lineal respecto al desplazamiento angular. Como se muestra en la Figura 12, se puede representar un $twist$ $\Upsilon$ mediante (8). Donde $S$ es el eje unitario de tornillo definido por (9), y se debe cumplir que $||\hat{\omega}|| = 1$ o en el caso de traslación pura, es decir, cuando $||\hat{\omega}|| = 0$, entonces se debe cumplir que $||v_u|| = 1$.

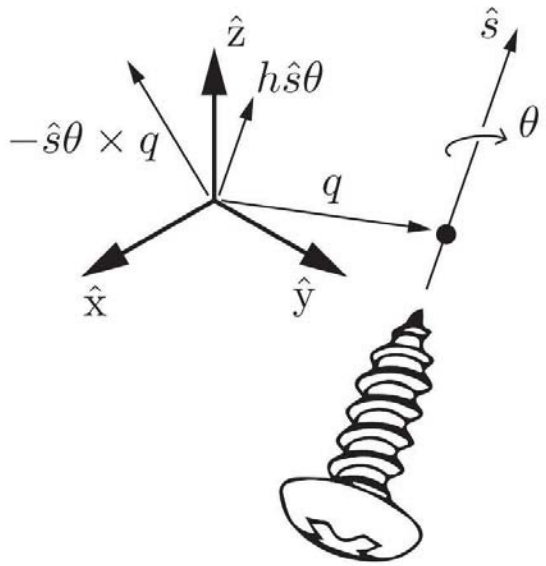



**Figura 12.** Representación de un twist como un eje de tornillo [26].

$$\Upsilon = \begin{bmatrix} \omega \\ v \end{bmatrix} = \begin{bmatrix} \hat{s}\dot{\theta} \\ -\hat{s}\dot{\theta} \times q + h\hat{s}\dot{\theta} \end{bmatrix} = S\theta \tag{8}$$

$$S = \begin{bmatrix} \hat{\omega} \\ v_u \end{bmatrix} = \Upsilon/\dot{\theta} \in R^6 \tag{9}$$

También se puede representar $S$ en forma matricial como (10).

$$[S] = \begin{bmatrix} [\hat{\omega}] & v_u \\ 0 & 0 \end{bmatrix} \in se(3) \tag{10}$$

## 7.9. Matriz Exponencial de Rotación y Traslación

Según el teorema de Charles-Mozzi [26], cualquier transformación de una pose a otra se puede expresar como un desplazamiento a lo largo de un eje de tornillo fijo $S$ en el espacio. Es decir, se pueden definir unas coordenadas exponenciales en seis dimensiones de cualquier matriz de transformación homogénea $T$ como $S\theta$; donde $S$ es el eje de tornillo y $\theta$ el giro alrededor de $S$ para llevar a un sistema de referencia desde su posición inicial a $T$.

Para aplicar el teorema de Charles-Mozzi se aplica según las dos condiciones de $S$.

- Si $||\hat{\omega}|| = 1$

$$T = e^{[S]\theta} = \begin{bmatrix} e^{[\hat{\omega}]\theta} & (I\theta + (1 - cos(\theta))[\hat{\omega}] + (\theta - sen(\theta))[\hat{\omega}]^2)v_u \\ 0 & 1 \end{bmatrix}. \tag{11}$$

  Donde $e^{[\hat{\omega}]\theta}$ es la fórmula de Rodrigues (6).

- Si $\hat{\omega} = 0$ y $||v_u|| = 1$

$$T = e^{[S]\theta} = \begin{bmatrix} I & v_u\theta \\ 0 & 1 \end{bmatrix}. \tag{12}$$

## 7.10. Matriz Logarítmica de Rotación y Traslación

Si se tiene el par $(R, p)$ expresado en una sola matriz como $T$, se puede encontrar mediante la matriz logarítmica un $\theta \in [0, \pi]$ y un eje de tornillo unitario $S = (\hat{\omega}, v_u)$, donde por lo menos uno debe ser un vector unitario y que cumpla la condición $e^{[S]\theta} = T$. El algoritmo de aplicación se muestra en el Algoritmo 2 en el Anexo A.

En resumen se puede decir que:

$$Exponencial: \ [S]\theta \in se(3) \rightarrow T \in SE(3)$$

$$Logaritmo: \ T \in SE(3) \rightarrow [S]\theta \in se(3)$$

### 7.11. Pseudoinversa de Moore-Penrose

La pseudoinversa de Moore-Penrose establece que para cualquier ecuación de la forma $Jy = z$, donde $J \in R^{m \times n}$, se tiene que:

$$y^* = J^+ z \tag{13}$$

En el cual se cumple uno de los dos casos siguientes:

- $y^*$ satisface exactamente $Jy^* = z$, y para cualquier solución se tiene que $||y^*|| \leq ||y||$. Es decir, la pseudoinversa encuentra la solución más pequeña de todas las posibles soluciones que satisfacen la ecuación (13).
- No existe una $y$ que satisfaga la ecuación, en este caso $y^*$ minimiza el error, $||Jy^* - z|| \leq ||Jy - z||$. Es decir, la pseudoinversa encuentra la solución más cercana posible aunque esta no satisfaga la ecuación (13).

La pseudoinversa se calcula mediante (14), si $m < n$, donde se cumple que $JJ^\dagger = I$. Y se calcula con (15) si $m > n$, donde se cumple que $J^\dagger J = I$.

$$J^\dagger = J^T (JJ^T)^{-1} \tag{14}$$

$$J^\dagger = (J^T J)^{-1} J^T \tag{15}$$

## 8. Selección de Motores

### 8.1. Motores a Pasos del Mecanismo de Agarre

El mecanismo de agarre de productos, que se muestra en la Figura 13, se compone de dos tornillos de potencia; uno para el movimiento en el eje $x$; y el otro para el movimiento en el eje $y$. Cada tornillo de potencia tiene un motor a pasos para controlar con precisión la distancia lineal que recorre el mecanismo. Debido a que el mecanismo de agarre que se traslada es liviano, se buscó en el mercado local el tornillo de potencia de menor diámetro para analizar el factor de seguridad dadas las condiciones, y se encontró que el tornillo de

potencia más pequeño es uno de acero inoxidable de $8\ \mathrm{mm}$ de diámetro externo, con un paso de $8\ \mathrm{mm}$, con rosca de tipo Acme y de $4$ hilos. A continuación, se calcula el torque requerido para cada motor a pasos para seleccionar los actuadores.

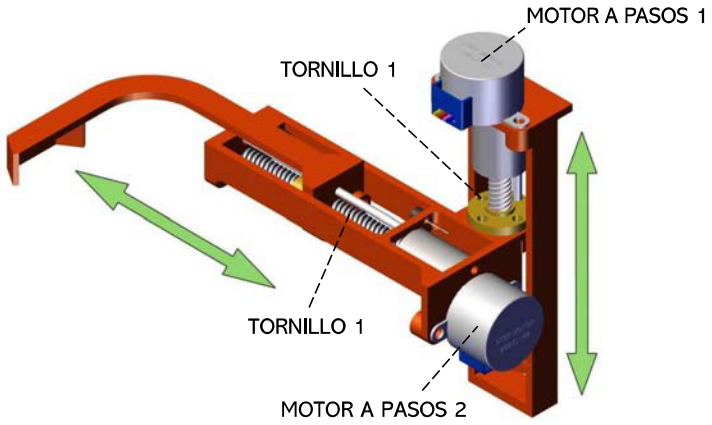



**Figura 13.** Mecanismo de agarre de productos.

### Motor a Pasos Eje *Y*

Primero, se analiza el torque requerido del motor a pasos para el eje $y$ dado el tornillo de potencia encontrado en el mercado. En la Figura 14 se muestra un diagrama, donde $T_{M1}$ es el torque del motor a pasos 1 y $P_1$ es el peso de la carga que debe levantar el mecanismo del eje $y$. Además, en la Tabla 1 se muestran los datos iniciales para dimensionar el motor.

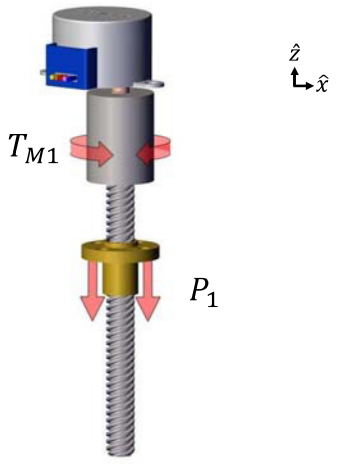



**Figura 14.** Elementos del mecanismo para el eje $y$.

**Tabla 1.** Datos iniciales para el dimensionamiento del mecanismo del eje $y$.

| Denominación | Variable | Valor |
|---|---|---|
| Diámetro exterior del tornillo | $d$ | $8\ \mathrm{mm}$ |
| Paso del tornillo | $p$ | $8\ \mathrm{mm}$ |
| Número de hilos | $n$ | $4$ |
| Masa de elevación para el motor a pasos 1 | $m_1$ | $120\ \mathrm{g}$ |
| Coeficiente de fricción para par roscado acero/bronce [27] | $f$ | $0{,}23$ |
| Factor de seguridad | $f_s$ | $1{,}5$ |
| Ángulo rosca Acme [27] | $2\alpha$ | $29°$ |
| Número de roscas en contacto | $n_t$ | $1$ |

Entonces, a partir de los datos del tornillo de potencia y de la masa de carga que debe elevar el mecanismo que se muestran en la Tabla 1, se calcula: el diámetro medio del tornillo $d_m$ mediante (16); la longitud de una vuelta de rosca $l$ con (17); y el peso de elevación $P_1$ según (18). A partir de estos valores se puede determinar el torque requerido para elevar la carga $T_{M1}$ mediante (19) [27], considerando que la tuerca utilizada es de bronce, y un factor de seguridad $f_s$ alto para mayor confianza en la selección del motor a pasos.

$$d_m = d - \frac{p}{2 \cdot n} = 7\ \mathrm{mm} \tag{16}$$

$$l = n \cdot p = 32\ \mathrm{mm} \tag{17}$$

$$P_1 = m_1 \cdot g = 1{,}175\ \mathrm{N} \tag{18}$$

$$T_{M1} = \frac{P_1 \cdot d_m}{2}\left(\frac{l + \pi \cdot f \cdot d_m \cdot sec(\alpha)}{\pi \cdot d_m - f \cdot l \cdot sec(\alpha)}\right) \cdot f_s = 0{,}016\ \mathrm{Nm} = 0{,}163\ \mathrm{kgf \cdot cm} \tag{19}$$

Con el resultado obtenido en (19) se seleccionó el motor a pasos *28byj-48*, con características de: torque de $0{,}3\ \mathrm{kgf \cdot cm}$; alimentación de $5\ \mathrm{V}$; $4$ fases; y una reducción mecánica interna de $1:64$ [28]. Se escogió este modelo debido a que es el motor que tiene el menor tamaño encontrado en el mercado local, a bajo costo y que satisface el torque requerido.

### Motor a Pasos Eje *X*

Para dimensionar el actuador del eje $x$ se debe proceder con criterios diferentes a los del eje $y$ debido a que este eje se encuentra paralelo al plano horizontal, entonces, la carga está aplicada perpendicularmente al tornillo de potencia. En la Figura 14 se muestra el diagrama del eje $x$, donde $P_2$ es el peso de los elementos sobre el tornillo de potencia en el extremo más alejado de la sujeción y $T_{M2}$ es el torque requerido del motor a pasos 2.

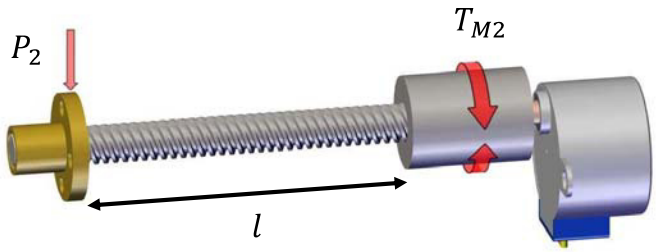



**Figura 15.** Elementos del mecanismo para el eje $x$.

Para calcular el torque requerido para desplazar los elementos del eje $x$ se considera la fuerza de fricción entre el tornillo de potencia y la tuerca. Ya que los dos elementos son iguales a los utilizados en el eje $y$, se utilizan algunos valores de la Tabla 1. Por lo tanto, $P_2$ se obtiene a partir de (20), considerando una masa $m_2 = 25$ g, y la fuerza de fricción se calcula con (21).

$$P_2 = m_2 \cdot g = 0{,}245 \text{ N} \tag{20}$$

$$F_{fr} = f \cdot m_2 \cdot g = 0{,}056 \text{ N} \tag{21}$$

Ya que la fricción se opone al movimiento, se puede calcular el torque requerido para el motor a pasos 2 mediante (22).

$$T_{M2} = F_{fr} \cdot d = 4{,}511 \times 10^{-4} \text{ Nm} = 0{,}005 \text{ kgf} \cdot \text{cm} \tag{22}$$

El torque requerido para el motor a pasos 2, $T_{M2}$, es menor que el motor a pasos 1, $T_{M1}$, y como se mencionó anteriormente, el motor *28byj-48* es el más pequeño y económico que se encontró en el mercado local, entonces, se utiliza el mismo modelo de actuador para este mecanismo.

## 8.2. Motores de la Plataforma Móvil

Para seleccionar los motores de la plataforma móvil se parte del cálculo de torque, velocidad angular y potencia necesaria en cada rueda para el movimiento del robot. En la Figura 16, se muestra un diagrama de cuerpo libre de la plataforma con las fuerzas presentes en el sistema; donde $a$ es la aceleración del robot; $N$ es la fuerza normal de reacción; $F_{fr}$ es la fuerza de fricción por rodadura entre las ruedas y la superficie; $F_P$ es la fuerza debido al peso del prototipo, que se descompone en $F_{Px}$ y $F_{Py}$; $F_M$ es la fuerza requerida para mover el robot; y $\theta$ es la inclinación máxima de la superficie. Los datos necesarios para el dimensionamiento se muestra en la Tabla 2.

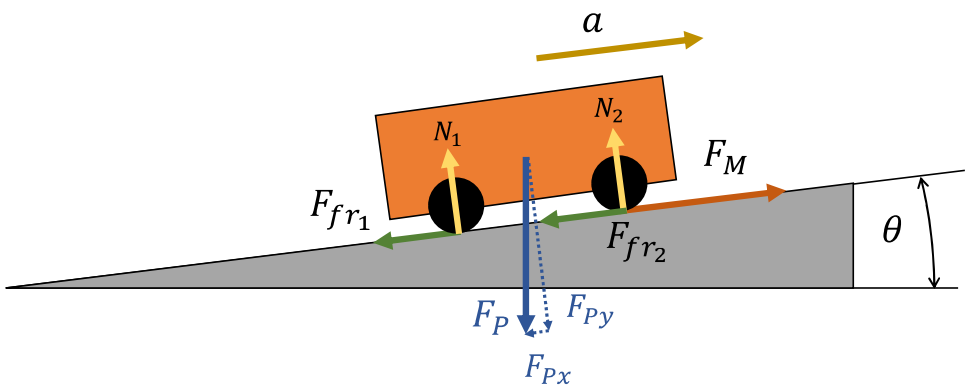



**Figura 16.** Diagrama de cuerpo libre de la plataforma móvil.

**Tabla 2.** Datos iniciales para el dimensionamiento de los motores de la plataforma móvil.

| Denominación | Variable | Valor |
|---|---|---|
| Coeficiente de rodadura [29] | $\mu_r$ | $0{,}004$ |
| Masa total | $m$ | $3{,}25\ \mathrm{kg}$ |
| Máxima inclinación de superficie [30] | $\theta$ | $8\,\% = 4{,}574°$ |
| Gravedad | $g$ | $9{,}81\ \mathrm{m/s^2}$ |
| Aceleración máxima del robot | $a_{max}$ | $0{,}5\ \mathrm{m/s^2}$ |
| Factor de seguridad | $f_s$ | $1{,}5$ |
| Radio de las ruedas [31] | $R_{rueda}$ | $48\ \mathrm{mm}$ |
| Número mínimo de motores actuando en simultáneo | $n_{motores}$ | $2$ |
| Velocidad máxima del robot | $v_{max}$ | $20\ \mathrm{cm/s}$ |

La fuerza de fricción por rodadura se obtiene a partir de (23), y la componente en $\hat{x}$ del peso del robot se obtiene mediante (24). Entonces, a partir del diagrama de cuerpo libre mostrado en la Figura 16 y por la segunda ley de Newton se calcula la fuerza que ejerce el robot para subir una pendiente máxima de $\theta = 8\,\%$, con una aceleración $a = 0{,}5\ \mathrm{m/s^2}$, mediante (25).

$$F_{fr} = \mu_r \cdot N = \mu_r \cdot m \cdot g \cdot cos(\theta) = 0{,}127\ \mathrm{N} \tag{23}$$

$$F_{Px} = m \cdot g \cdot sen(\theta) = 2{,}547\ \mathrm{N} \tag{24}$$

$$F_M = m \cdot a + F_{fr} + F_{Px} = 4{,}299\ \mathrm{N} \tag{25}$$

Entonces, se puede determinar el torque requerido para cada motor de la plataforma móvil mediante (26), y la velocidad angular máxima de cada rueda como (27). Se utiliza un factor de seguridad de $f_s = 1{,}5$ para compensar algunas variables que se desprecian, tales como el arrastre del aire, fricción entre elementos rotatorios dentro del prototipo, entre otros, y así poder elegir un modelo de motor con mayor seguridad. También, los robots con *mecanum wheels* utilizan cada rueda a diferentes velocidades y sentidos de giro dependiendo del tipo de movimiento que se quiere tener, es por esto que se establece como el mínimo de motores funcionando en simultáneo es $n_{motores} = 2$. Adicionalmente, la potencia mecánica se calcula mediante (28).

$$T = \frac{F_M \cdot R_{rueda} \cdot f_s}{n_{motores}} = 0{,}155 \text{ Nm} = 1{,}578 \text{ kgf} \cdot \text{cm} \tag{26}$$

$$\omega = \frac{v_{max}}{R_{rueda}} \cdot \frac{30}{\pi} = 4{,}167 \text{ rad/s} = 39{,}789 \text{ RPM} \tag{27}$$

$$P = T \cdot \omega = 0{,}645 \text{ W} \tag{28}$$

### Motores Seleccionados

A partir de los datos calculados anteriormente, se seleccionaron los motores N20 debido a su tamaño reducido, costo relativamente bajo, a sus características y a su sensor de efecto *Hall* integrado que se utilizan como encoder. Las características de estos motores se muestran en la Tabla 3 [32].

**Tabla 3.** Características técnicas de los motores N20.

| Característica | Valor |
|---|---|
| Voltaje nominal | 12 V |
| Torque de parada | 5,04 kgf · cm |
| Velocidad sin carga | 90 RPM |
| Corriente sin carga | 0,07 A |
| Corriente de parada | 1,6 A |

Con estos parámetros se puede obtener la característica del motor, que se muestra en la Figura 17, donde la potencia y la eficiencia son calculadas mediante (28) y (29), respectivamente.

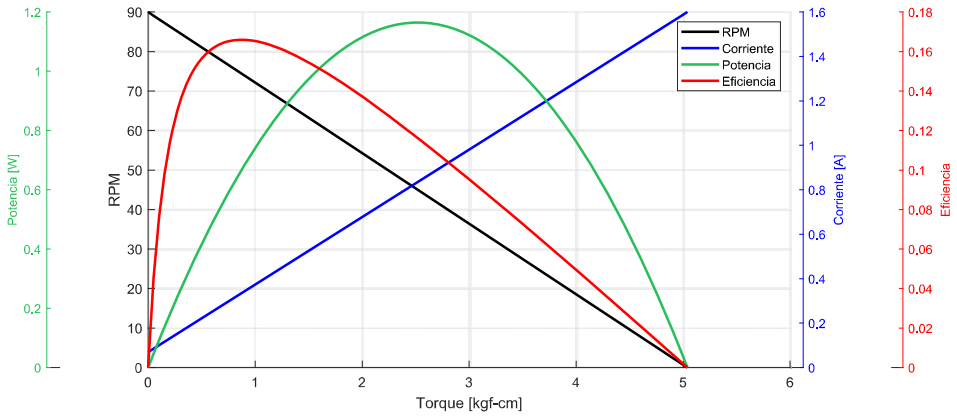



**Figura 17.** Curvas características de los motores DC N20.

$$\eta = \frac{P}{V \cdot I} \tag{29}$$

Mediante la Figura 17 se determina la eficiencia máxima de los motores N20; el torque; la velocidad angular; y la corriente a máxima eficiencia. En la Tabla 4 se resumen los resultados.

**Tabla 4.** Características de los motores DC N20 a máxima eficiencia.

| Característica | Valor | Variable |
|---|---|---|
| Eficiencia máxima | $17\,\%$ | $\eta_{max}$ |
| Torque a máxima eficiencia | $0{,}87\ \mathrm{kgf \cdot cm}$ | $T_\eta$ |
| Velocidad angular a máxima eficiencia | $74{,}55\ \mathrm{RPM}$ | $\omega_\eta$ |
| Corriente a máxima eficiencia | $0{,}33\ \mathrm{A}$ | $I_\eta$ |

Al comparar los resultados obtenidos en (26) - (27) con los de la Tabla 4, se puede identificar que con una reducción mecánica mediante engranajes con una relación de 1:2 $GR_{eng} = 2$ se pueden obtener una velocidad angular y torque aproximados a la eficiencia máxima.

Finalmente, se calcula cuánta masa extra puede cargar la plataforma móvil del robot mediante (30). Donde $T_p$ es el torque de parada y $f_p = 40\,\%$ es un factor para considerar un porcentaje del torque de parada como torque máximo, ya que no es recomendable que los motores sean sobrecargados, ya sea porque la caja reductora interna del motor puede sufrir daños, o porque la parte eléctrica del motor puede quemarse por mucha corriente (por el bobinado interno o los cables de alimentación).

$$m_{extra} = \frac{f_p \cdot GR_{eng} \cdot T_p \cdot n_{motores}}{R_{ruedas} \cdot f_s \cdot (a_{max} + \mu_r \cdot g \cdot cos(\theta) + g \cdot sen(\theta))} - m = 5\ \mathrm{kg} \tag{30}$$

## 9. Diseño Mecánico

### 9.1. Dimensionamiento de Mecanismo de Agarre

#### Dimensionamiento Eje *Y*

A partir del cálculo de $T_{M1}$ con (19), se calcula el factor de seguridad por el criterio del cortante máximo (ECM) $n_{ECM}$ y por la teoría de la energía de distorsión (ED) $n_{ED}$ [27], considerando un límite a la fluencia para el acero inoxidable de $Sy_{inox} = 241\ \mathrm{MPa}$ y un

límite a la fluencia para el bronce de $Sy_{br} = 150\ \mathrm{MPa}$ . Para esto se comienza calculando el diámetro de la raíz $d_r$ del tornillo de potencia mediante (31).

$$d_r = d - \frac{p}{n} = 6\ \mathrm{mm} \tag{31}$$

Entonces, se puede calcular el esfuerzo cortante de torsión $\tau$ en el cuerpo del tornillo de potencia (provocado por el torque ejercido por el motor a pasos $T_{M1}$) mediante (32), donde $T$ es el torque ejercido en el elemento, $r$ es el radio del tornillo y $J$ es el segundo momento polar de área de la sección transversal del tornillo [27].

$$\tau = \frac{T \cdot r}{J} = \frac{16 \cdot T_{M1}}{\pi \cdot d_r^3} = 376{,}326\ \mathrm{kPa} \tag{32}$$

También, el tornillo estará sometido a un esfuerzo de tracción $\sigma$ por el peso de la carga $P_1$, y se calcula con (33).

$$\sigma = \frac{F}{A} = \frac{4 \cdot P_1}{\pi \cdot d_r^2} = 41{,}562\ \mathrm{kPa} \tag{33}$$

Las roscas sufrirán un esfuerzo de apoyo $\sigma_B$, que se calcula mediante (34). Donde $A_B$ es el área de la cara de la rosca donde se apoya la tuerca (y por consiguiente el peso). De acuerdo con [27], se han realizado experimentos que muestran que la primera rosca en contacto soporta más carga que el resto, con un $38\,\%$ de la carga total, entonces, se dimensiona considerando que solo una rosca está en contacto $n_t$ con una carga $F_B = 0{,}38 \cdot P_1$.

$$\sigma_B = -\frac{F_B}{A_B} = -\frac{0{,}38 \cdot P_1}{\pi \cdot n_t \cdot (d^2 - d_r^2)} = -5{,}077\ \mathrm{kPa} \tag{34}$$

La fuerza de apoyo sobre las roscas también generan un esfuerzo flector $\sigma_b$ en la raíz de la rosca, que se obtiene mediante (35). Donde $M$ es el momento flector, $c$ es la distancia máxima desde el eje neutro e $I$ es el momento de inercia de la sección transversal de la rosca (se aproxima la sección transversal de la rosca Acme como una rosca rectangular).

$$\sigma_b = \frac{M \cdot c}{I} \cdot \frac{1}{n_t} = \frac{3 \cdot (0{,}38 P_1)}{\pi \cdot d_r \cdot (d - d_r)} \cdot \frac{1}{n_t} = 93{,}515\ \mathrm{kPa} \tag{35}$$

Finalmente, también debido a la fuerza de apoyo sobre la rosca, existe un esfuerzo cortante $\tau_r$ en la raíz de la rosca, que se obtiene a partir de (36), donde $V$ es el cortante y $A$ es el área en la que se aplica el cortante (en este caso también se aproxima la rosca Acme como un rosca cuadrada, para que el área sea el ancho de la rosca por la circunferencia del tornillo en su raíz).

$$\tau_r = \frac{3 \cdot V}{2 \cdot A} = \frac{3 \cdot (0{,}38 P_1)}{2 \cdot \pi \cdot d_r \cdot (d - d_r)} \cdot \frac{1}{n_t} = 46{,}758 \text{ kPa} \tag{36}$$

Entonces, se puede definir el estado tensional en la raíz de la rosca del tornillo de potencia (según el sistema de referencia de la Figura 14) como (37), y determinar los esfuerzos principales al resolver la ecuación (38) para $\sigma$ (círculo de Mohr para esfuerzos tridimensionales).

$$[T] = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} = \begin{bmatrix} \sigma_b & \tau & \tau_r \\ \tau & 0 & 0 \\ \tau_r & 0 & \sigma \end{bmatrix} \tag{37}$$

$$\begin{aligned} &\sigma^3 - (\sigma_x + \sigma_y + \sigma_z) \cdot \sigma^2 + (\sigma_x \cdot \sigma_y + \sigma_x \cdot \sigma_z + \sigma_y \cdot \sigma_z - \tau_{xy}^2 - \tau_{yz}^2 - \tau_{zx}^2) \cdot \sigma \\ &- (\sigma_x \cdot \sigma_y \cdot \sigma_z + 2 \cdot \tau_{xy} \cdot \tau_{yz} \cdot \tau_{zx} - \sigma_x \tau_{yz}^2 - \sigma_y \tau_{zx}^2 - \sigma_z \tau_{xy}^2) = 0 \end{aligned} \tag{38}$$

Los tres esfuerzos principales que se obtuvieron de (38) son: $\sigma_1 = 429{,}126$ kPa, $\sigma_2 = 40{,}945$ kPa y $\sigma_3 = -334{,}993$ kPa. Con los esfuerzos principales se puede calcular el esfuerzo de von Misses mediante (39), y el esfuerzo cortanto máximo con (40).

$$\sigma' = \sqrt{\frac{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2}{2}} = 667{,}682 \text{ kPa} \tag{39}$$

$$\tau_{max} = \frac{\sigma_1 - \sigma_3}{2} = 382{,}341 \text{ kPa} \tag{40}$$

Por lo tanto, se puede determinar el factor de seguridad por el criterio de la energía de distorsión con (41), y mediante el criterio del esfuerzo cortante máximo con (42). Se puede ver que en ambos casos se tiene un factor de seguridad muy grande, esto se debe a que

el peso de las cargas es bajo en relación con el tamaño y material del tornillo de potencia, aunque este tornillo sea el más pequeño dentro del mercado local.

$$n_{ED} = \frac{Sy_{inox}}{\sigma'} = 360{,}95 \tag{41}$$

$$n_{ECM} = \frac{Sy_{inox}}{2 \cdot \tau_{max}} = 315{,}164 \tag{42}$$

También, se puede determinar el factor de seguridad para la tuerca de bronce considerando que la mayoría de esfuerzos son iguales al del tornillo de potencia. Entonces, se define el estado tensional de la rosca como (43), considerando que las diferencias con (37) es que no existe esfuerzo cortante por el torque de motor y que la tuerca se encuentra a compresión.

$$[T] = \begin{bmatrix} \sigma_b & 0 & \tau_r \\ 0 & 0 & 0 \\ \tau_r & 0 & -\sigma \end{bmatrix} \tag{43}$$

Resolviendo (38) para (43), se obtiene que los esfuerzos principales son: $\sigma_1 = 108{,}121\ \mathrm{kPa}$, $\sigma_2 = 0\ \mathrm{kPa}$ y $\sigma_3 = -56{,}168\ \mathrm{kPa}$. Por lo que se obtiene un esfuerzo de von Misses $\sigma' = 144{,}631\ \mathrm{kPa}$ con (39), y un esfuerzo cortante máximo $\tau_{max} = 82{,}145\ \mathrm{kPa}$ mediante (40). Entonces, los factores de seguridad para la tuerca del tornillo de potencia son (44) y (45), que son valores aún mayores que los obtenidos para el tornillo de potencia.

$$n_{ED} = \frac{Sy_{br}}{\sigma'} = 1{,}037 \times 10^3 \tag{44}$$

$$n_{ECM} = \frac{Sy_{br}}{2 \cdot \tau_{max}} = 913{,}023 \tag{45}$$

**Dimensionamiento Eje *X***

A continuación, se debe determinar el factor de seguridad para el tornillo de potencia, para poder estimar si es adecuado para las condiciones del mecanismo. De acuerdo con la Figura 15 y los datos de la Tabla 5, se puede determinar que el tornillo de potencia

está sometido una condición de carga puntual en voladizo, ya que está sujeta solo por un extremo, pero se debe considerar una carga completamente reversible debido a que el tornillo de potencia gira cíclicamente (se hace una simplificación considerando el peor caso ya que si el tornillo de potencia gira, entonces la carga se trasladaría linealmente).

**Tabla 5.** Datos iniciales para el dimensionamiento del eje $x$.

| Denominación | Variable | Valor |
|---|---|---|
| Masa de elementos para motor a pasos 2 | $m_2$ | $25$ g |
| Resistencia última del acero inoxidable [27] | $S_{ut}$ | $601$ MPa |
| Factor $a$ para condición superficial [27] | $a$ | $4{,}51$ |
| Exponente $b$ para condición superficial [27] | $b$ | $-0{,}265$ |
| Factor de modificación de la carga [27] | $k_c$ | $1$ |
| Factor de modificación de la temperatura [27] | $k_d$ | $1$ |
| Factor de confiabilidad [27] | $k_e$ | $1$ |
| Sensibilidad a la muesca [27] | $q$ | $0{,}6$ |
| Factor de concentración del esfuerzo para barra redonda ranurada en flexión [27] | $K_t$ | $2{,}6$ |
| Longitud del tornillo de potencia 2 | $l$ | $80{,}5$ mm |
| Fracción de resistencia a la fatiga [27] | $f$ | $0{,}86$ |

En la Figura 18 se muestra un diagrama de cuerpo libre del tornillo de potencia. Con (46) y (47) se determina la fuerza y momento de reacción debido a la carga, y se identifica que el punto crítico del elemento es en la parte del anclaje, donde existe un cortante de $V = 0{,}25N$ y un momento flector de $M = -19{,}7Nmm$. a partir del cual se puede determinar el diagrama de cortantes y de momentos que se muestran en la parte derecha de la Figura 18.

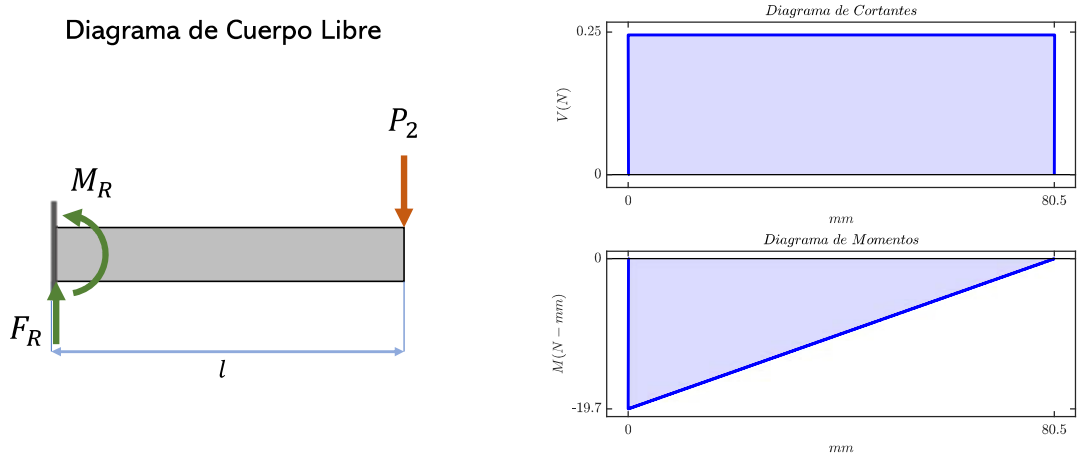



**Figura 18.** Cargas y diagramas del tornillo de potencia 2.

$$\sum F_y = 0 \tag{46}$$

$$\sum M = 0 \tag{47}$$

Las cargas en el punto crítico generan un esfuerzo cortante $\tau_c$ y un esfuerzo flector $\sigma_f$ que se calculan mediante (48) y (49) respectivamente.

$$\tau_c = \frac{3 \cdot V}{2 \cdot A} = \frac{6 \cdot P_2}{\pi \cdot d_r^2} = 13{,}006 \text{ kPa} \tag{48}$$

$$\sigma_f = \frac{M \cdot c}{I} = -\frac{32 \cdot P_2 \cdot l}{\pi \cdot d_r^3} = -930{,}685 \text{ kPa} \tag{49}$$

Bajo los esfuerzos que está sometido el tornillo se define el estado tensional en un plano bidimensional mediante (50). Por lo tanto, se puede calcular los esfuerzos principales con la solución de (51) (círculo de Mohr para esfuerzos bidimensionales).

$$[T] = \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{yx} & \sigma_y \end{bmatrix} = \begin{bmatrix} \sigma_f & \tau_c \\ \tau_c & 0 \end{bmatrix} \tag{50}$$

$$\sigma_1, \sigma_2 = \frac{\sigma_x + \sigma_y}{2} \pm \sqrt{(\frac{\sigma_x - \sigma_y}{2})^2 + \tau_{xy}^2} \tag{51}$$

Los esfuerzos principales calculados son $\sigma_1 = 181{,}732$ Pa y $\sigma_2 = -930{,}867$ kPa. Entonces, mediante (52) se calcula el esfuerzo de von Misses para un esfuerzo plano.

$$\sigma' = \sqrt{\sigma_1^2 - \sigma_1 \cdot \sigma_2 + \sigma_2^2} = 930{,}958 \text{ kPa} \tag{52}$$

Como se mencionó anteriormente, se debe considerar la carga a fatiga. Para esto, según [27] se estima el límite de resistencia mediante (53) ($S_{ut} \leq 1{,}4$ GPa).

$$Se' = 0{,}5 \cdot S_{ut} = 300{,}5 \text{ MPa} \tag{53}$$

Además, se consideran factores modificadores del límite de resistencia a fatiga. El factor de superficie $k_a$ se calcula según (54); el factor de tamaño $k_b$ con (55); el factor de carga $k_c$,

el factor de temperatura $k_d$ y el factor de confiabilidad $ke$ se muestran en la Tabla 5; y para el factor de efectos varios $k_f$ se calcula con (56) a partir de la sensibilidad a la muesca $q$ y el factor de concentración del esfuerzo $K_t$ que se muestran en la Tabla 5.

$$k_a = a \cdot (S_{ut})^b = 0{,}828 \tag{54}$$

$$k_b = 1{,}24 \cdot (d)^{-0{,}107} = 0{,}993 \tag{55}$$

$$k_f = 1 + q \cdot (K_t - 1) = 1{,}96 \tag{56}$$

Por lo tanto, mediante la ecuación de Marin [33] se determina el límite de resistencia a fatiga $Se$ según (57), donde se puede evidenciar que es mayor al esfuerzo de von Mises $\sigma'$ calculado en (52), por lo tanto, la pieza tiene vida infinita bajo las condiciones de funcionamiento.

$$Se = k_a \cdot k_b \cdot k_c \cdot k_d \cdot k_e \cdot k_f \cdot Se' = 285{,}442 \text{ MPa} \tag{57}$$

### 9.2. Dimensionamiento de Ejes de la Caja Reductora

Como se mencionó en la selección de motores, se diseña un sistema de engranajes para una reducción mecánica de los motores de 1:2, como se muestra en la Figura 19, según los datos presentados en la Tabla 6. Los engranajes se ensamblan en ejes de tipo D para poder asegurar un ajuste que no permita el deslizamiento entre los engranajes y los ejes. De manera similar a los elementos de los tornillos de potencia, los ejes se seleccionaron según la disponibilidad del mercado local, donde el material es acero inoxidable con un diámetro $d = 6 \text{ mm}$.

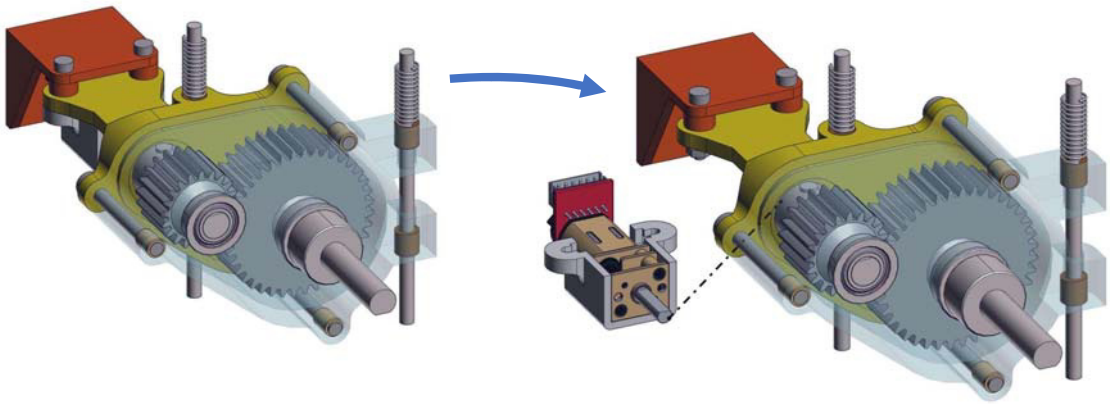


**Figura 19.** Fuerzas en los engranajes.

**Tabla 6.** Datos iniciales para el dimensionamiento de los ejes de los engranajes.

| Denominación | Variable | Valor |
|---|---|---|
| Diámetro del piñón | $d_2$ | $16$ mm |
| Diámetro del engranaje | $d_3$ | $32$ mm |
| Ángulo de presión de los engranajes | $\phi$ | $20°$ |
| Módulo de los engranajes | $m$ | $0{,}8$ |
| Longitud del eje del piñón | $l$ | $20$ mm |
| Longitud del eje del engranaje al soporte | $l_1$ | $25{,}4$ mm |
| Longitud del eje del engranaje a la carga | $l_2$ | $32{,}1$ mm |
| Diámetro de los ejes | $d$ | $6$ mm |
| Factor $a$ para condición superficial [27] | $a$ | $4{,}51$ |
| Exponente $b$ para condición superficial [27] | $b$ | $-0{,}265$ |
| Factor de modificación de la carga [27] | $k_c$ | $1$ |
| Factor de modificación de la temperatura [27] | $k_d$ | $1$ |
| Factor de confiabilidad [27] | $k_e$ | $1$ |
| Factor de efectos varios [27] | $k_f$ | $1$ |
| Fracción de resistencia a la fatiga [27] | $f$ | $0{,}86$ |
| Masa máxima de carga | $m_{max}$ | $8{,}25$ |

Para calcular los ciclos de vida de los ejes se analizan las fuerzas presentes en el sistema de engranajes como se muestran en la Figura 20, donde $F_{2a}$ es la fuerza del engrane $2$ sobre el eje $a$; $F_{3b}$ es la fuerza del engrane $3$ sobre el eje $b$; y $P$ es la fuerza sobre el eje $b$ debido al peso del prototipo.

Se comienza el dimensionamiento calculando la potencia transmitida en el caso más crítico con (58), que como se analizó en la sección anterior, se considera con el peso del prototipo y la masa extra máxima (30), a la velocidad máxima del motor que se muestra en la Tabla 3.

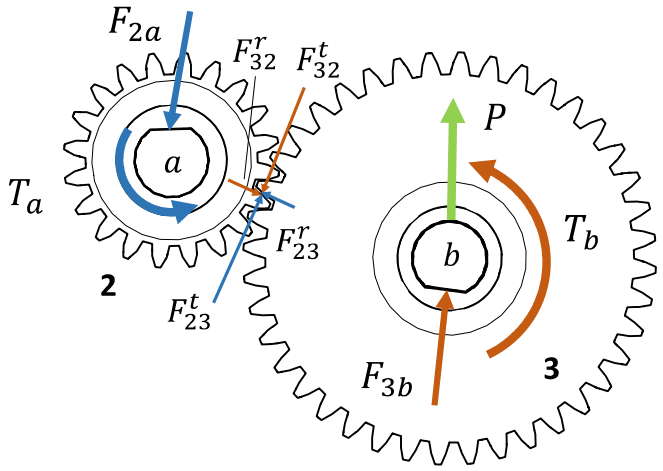



**Figura 20.** Fuerzas en los ejes de los engranajes.

$$P = T \cdot \omega = 1{,}863 \text{ W} \tag{58}$$

A continuación, se calcula la fuerza tangencial en los engranajes mediante (59), y la fuerza resultante sobre el eje del piñón con (60).

$$F_t = \frac{2 \cdot P}{d_2 \cdot \omega} = 24{,}713 \text{ N} \tag{59}$$

$$F_{2a} = \frac{F_t}{cos(\phi)} = 26{,}299 \text{ N} \tag{60}$$

Con la fuerza resultante (60) se pueden analizar las cargas sobre el eje del piñón. En la Figura (21), en la parte superior se muestra el diagrama de cuerpo libre para el elemento. Se puede evidenciar que, debido a las restricciones del eje en sus extremos, la estructura se encuentra en un estado de hiperestaticidad, por lo tanto, se utiliza el método de superposición para encontrar las reacciones del sistema. Entonces, se analiza primero únicamente las fuerzas de reacción, calculando los diagramas de cortantes y momentos; por separado, se calcula los diagramas de los momentos de reacción. Luego, se suman los diagramas para obtener los cortantes y momentos resultantes, como se muestra en la Figura (21). Analizando los diagramas se puede determinar que el punto crítico se encuentra en B, en la mitad de los ejes, donde se aplica la fuerza debido a los engranajes, con una fuerza cortante de $V = 26{,}3$ N, un momento $M = 52{,}6$ N $\cdot$ mm y un torque máximo $T = 19{,}77$ N $\cdot$ cm

.

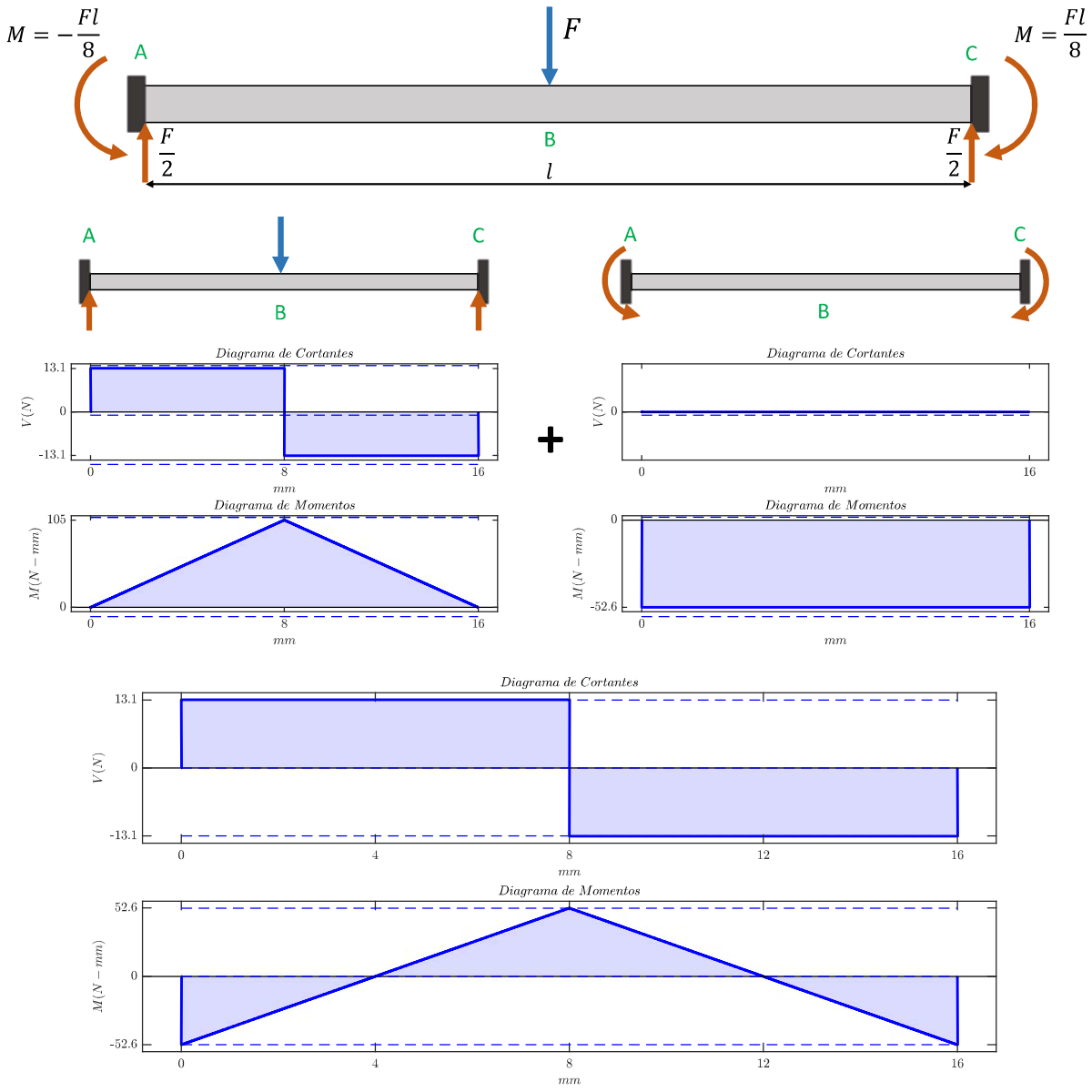



**Figura 21.** Cargas y diagramas del eje $a$.

Para poder determinar el esfuerzo flector y torsor en el punto crítico se debe determinar la inercia de la sección transversal, pero el eje tiene una sección de tipo D. Entonces, si se considera a la sección de tipo D como una figura compuesta, como se muestra en la Figura 22, se puede calcular la inercia de la sección mediante el teorema de Steiner (61) y (62). En (61) y (62), $I_0$ es la inercia de la figura $i$; $A_i$ es el área de la figura $i$; $X_{CG}$ y $Y_{CG}$ es el desplazamiento del centro de gravedad de la figura compuesta en $\hat{x}$ e $\hat{y}$, y se calcula mediante (63) y (64), respectivamente; $\bar{X}_i$ e $\bar{Y}_i$ son la posición del centro de gravedad en la

figura $i$; y en la Figura 22 $d$ es el diámetro del eje.

$$I_x = \sum_{i=1}^{n} (Ix_{0i} + A_i \cdot (X_{CG} - \bar{X}_i)^2) \tag{61}$$

$$I_y = \sum_{i=1}^{n} (Iy_{0i} + A_i \cdot (Y_{CG} - \bar{Y}_i)^2) \tag{62}$$

$$X_{CG} = \frac{\sum_{i=1}^{n} \bar{X}_i \cdot A_i}{\sum_{i=1}^{n} A_i} = \frac{2 \cdot d}{3} \cdot \frac{sin^3(\pi - \alpha)}{2(\pi - \alpha) - sin(2(\pi - \alpha))} \tag{63}$$

$$Y_{CG} = \frac{\sum_{i=1}^{n} \bar{Y}_i \cdot A_i}{\sum_{i=1}^{n} A_i} = 0 \tag{64}$$

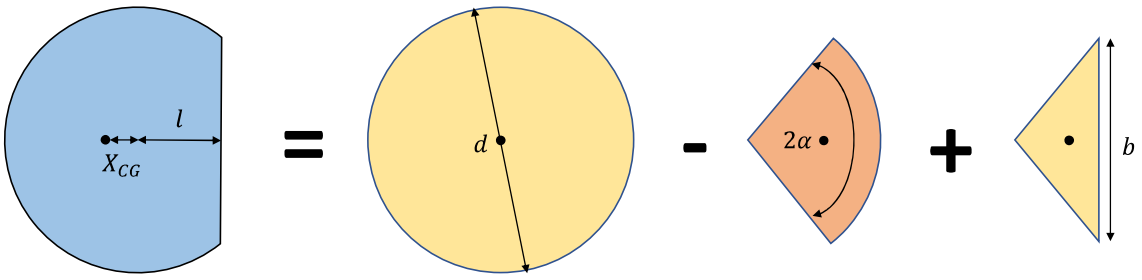



**Figura 22.** Inercia de figuras compuestas para sección transversal de los ejes.

A partir de la Figura 22, se desarrolla (61) y (62), para obtener (65) y (66), donde $\alpha$ se calcula según (67). El segundo momento polar de inercia $J$ se calcula con (68).

$$I_x = \frac{d^4}{128} \cdot (2(\pi - \alpha) - sen(2(\pi - \alpha)) + 2 \cdot sen(2(\pi - \alpha)) \cdot sen(\pi - \alpha)^2) = 55{,}385 \text{ mm}^4 \tag{65}$$

$$I_y = \frac{d^4}{384} \cdot (6(\pi - \alpha) - 3 \cdot sen(2(\pi - \alpha)) - 2 \cdot sen(2(\pi - \alpha)) \cdot sen(\pi - \alpha)^2) = 62{,}985 \text{ mm}^4 \tag{66}$$

$$\alpha = cos^{-1}(\frac{2 \cdot l}{d}) \tag{67}$$

$$J = I_x + I_y = 118{,}37 \text{ mm}^4 \tag{68}$$

El esfuerzo cortante $\tau_c = -1{,}295$ MPa, flector $\sigma_f = 2{,}849$ MPa y torsor $\tau = 5{,}011$ MPa se obtiene con (48), (49) y (32), respectivamente, donde $c = d/2 = 3$ mm. El eje se para un estado de carga completamente reversible, por lo que se utiliza el criterio de fallas a fatiga utilizado para dimensionar el tornillo de potencia del eje $x$.

Se pueden determinar los esfuerzos principales con (38) para calcular el esfuerzo de von Mises $\sigma' = 9{,}2$ MPa según (39). Ya que el material del eje es igual al material de los tornillos de potencia, el límite de resistencia de la probeta es $Se' = 300{,}5$ MPa. También, las contantes y factores modificadores de la resistencia a la fatiga se muestran en la Tabla 6, donde $k_a = 0{,}828$ y $k_b = 1{,}024$ se calculan con (54) y (55). Entonces, según (57) se obtiene el límite de resistencia a la fatiga $Se = 254{,}553$ MPa, el cual es mayor al esfuerzo de von Misses $\sigma'$, por lo que se concluye que el elemento tiene vida infinita.

En el caso del eje $b$ se tiene un estado de carga diferente al eje $a$, como se muestra en la Figura 23, ya que se tiene la carga del engranaje sobre el eje $F_{3b}$ que es igual en magnitud pero con sentido contrario a $F_{2a}$; el soporte C se encuentra una distancia $l_1$ de A, y no en el otro extremo del eje; y se tiene la carga $P = 20{,}226$ N que es el peso máximo que cada rueda soporta debido al peso total del prototipo. De manera similar al caso del eje $a$, se utiliza el método de superposición para evaluar las cargas, los cortantes y momentos de forma separada, para combinarlos en los diagramas resultantes que se muestran en la parte inferior de la Figura 23.

A partir de la Figura 23 se determina que el punto critico es el C, donde se encuentra el soporte derecho del eje. En este punto se tiene un cortante $V = -20{,}226$ N, un momento $M = 732{,}76$ N $\cdot$ mm y un torque $T = 39{,}54$ N $\cdot$ cm. El esfuerzo cortante $\tau_c = -1{,}073$ MPa, flector $\sigma_f = 39{,}691$ MPa y torsor $\tau = 10{,}021$ MPa se obtiene con (48), (49) y (32), respectivamente. Se procede a calcular el esfuerzo de von Misses $\sigma' = 43{,}36$ MPa con (39), y ya que el elemento tiene la misma geometría (excepto por el largo del eje, sin embargo, este

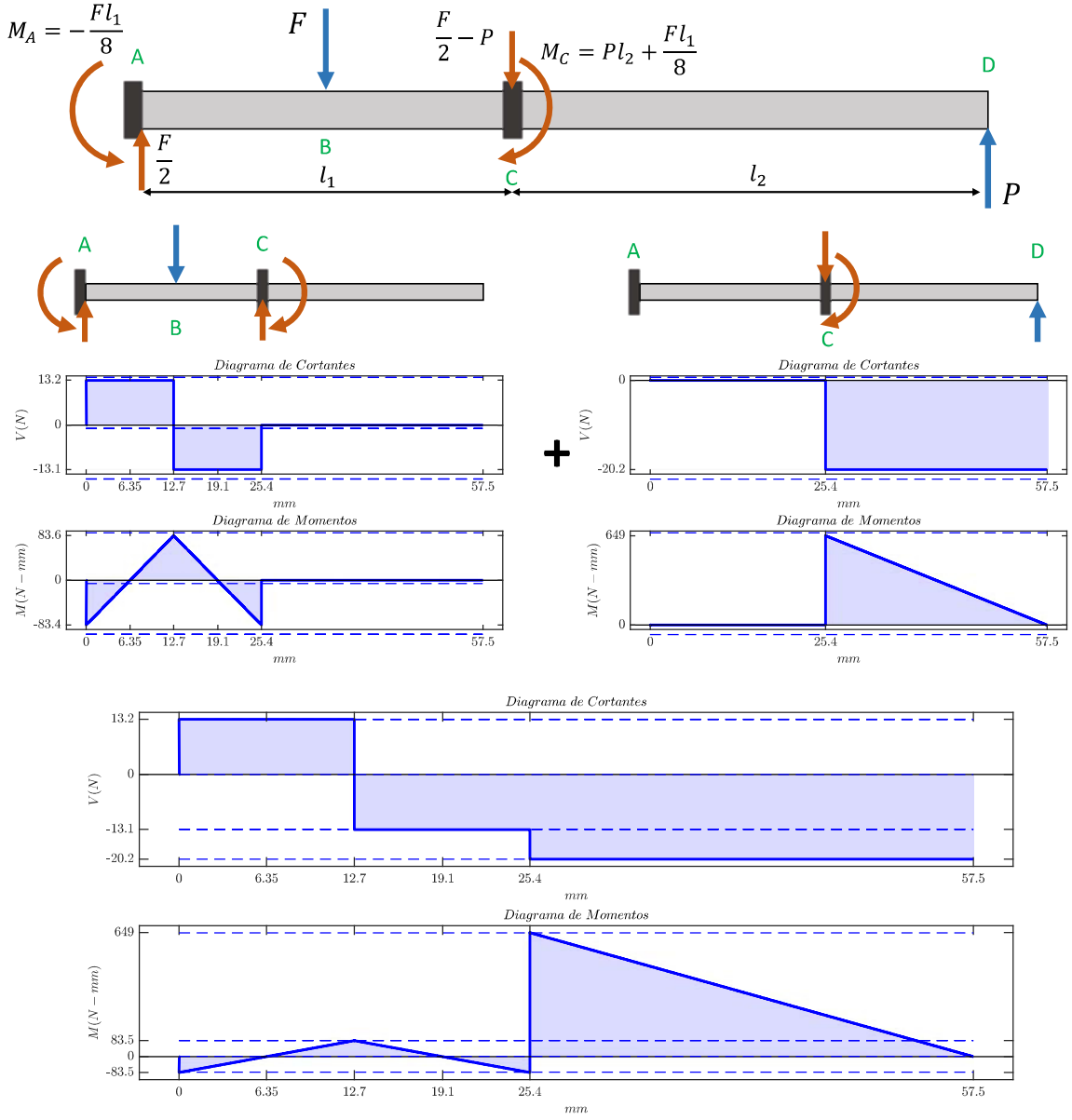



**Figura 23.** Cargas y diagramas del eje $b$.

factor no influye en la resistencia a la fatiga) y material, la resistencia a la fatiga es igual a la calculada para el eje $a$, $Se = 254{,}6$ MPa, donde $Se > \sigma'$, por lo tanto el elemento tiene vida infinita.

### 9.3. Sistema de Suspensión

El robot estima su posición mediante odometría, pero como se analizará en la Sección 10.4, este método tiene un error acumulativo, donde un factor muy importante para reducir el error es que las llantas giren siempre en contacto con el suelo para que exista tracción. Para

esto se diseña un sistema de suspensión simple como se muestra en la Figura 24, con dos ejes por cada caja reductora para guiar el movimiento del mecanismo y dos resortes para mantener una fuerza constante en las 4 ruedas. En la Tabla 7 se muestra las características de los resortes utilizados para el sistema.

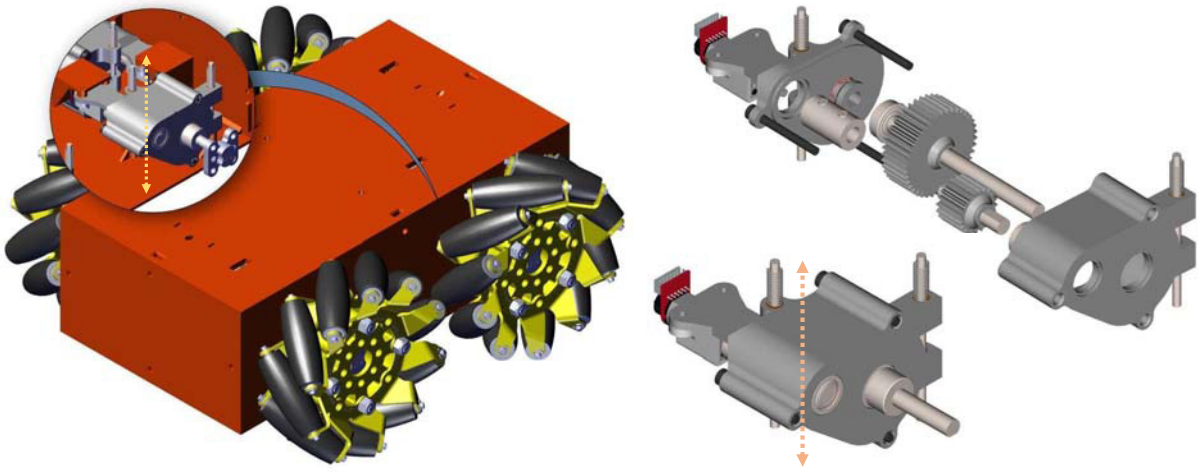


**Figura 24.** Plataforma móvil y caja reductora.

**Tabla 7.** Características técnicas de los resortes.

| Característica | Valor |
|---|---|
| Número de resortes | $n_R = 8$ |
| Longitud de resortes | $L_R = 13\ \text{mm}$ |
| Constante del resorte | $k = 1{,}524\ \text{lbf/plg}$ |
| Carga máxima por resorte | $m_{max} = 0{,}472\ \text{kg}$ |

Analizando la Tabla 7, se puede evidenciar que los resortes tienen una carga máxima, por lo que se determina la masa extra máxima considerando este factor según (69), el cual es un valor menor a la masa extra calculada anteriormente.

$$m_{extra} = m_{max} \cdot n_R - m = 0{,}526\ \text{kg} \tag{69}$$

Los ejes seleccionados para el sistema de suspensión son de acero inxidable de diámetro $d = 3\ \text{mm}$. Para dimensionar estos elementos se prefirió realizar un análisis FEA en el software *SolidWorks Simulation* debido a que el mecanismo es un sistema hiperestático complejo. Como se muestra en la Figura 25, se introdujeron las cargas correspondientes al peso del sistema (considerando la masa extra $m_{extra} = 5\ \text{kg}$) sobre el eje de la rueda, y las fuerzas de los resortes. Además, se establecieron soportes fijos en los extremos de

los ejes y se realizó un modelo simplificado de la caja reductora para poder definirla como un cuerpo rígido (como se muestra en la figura central de la Figura 25). En la Figura 25, en la parte derecha se muestran los resultados, con un esfuerzo máximo de von Mises $\sigma' = 27{,}34\ \mathrm{MPa}$. Finalmente, se calcula el factor de seguridad por el criterio de la energía de distorisón $n_{ED} = 8{,}815$ por (41), el cuál es un valor conservador, sin embargo, es el eje de menor diámetro en el mercado local, por lo que es la mejor opción.

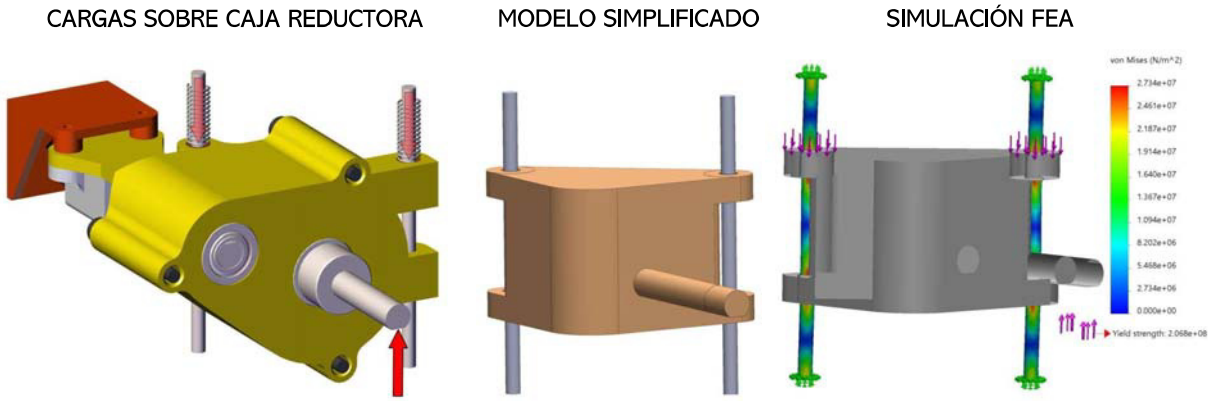



**Figura 25.** Análisis de elementos finitos de los ejes en el sistema de suspensión .

## 10. Diseño del Sistema de Control

El sistema de control se diseñó considerando que el robot debe ser capaz de desplazarse a una serie de posiciones preestablecidas donde se encuentran los diferentes productos que debe recolectar, pero dentro del entorno que el robot se desplaza existen obstáculos fijos cuya posición es conocida, como la estantería o paredes que limitan el entorno. Además, en el entorno hay obstáculos dinámicos, es decir, obstáculos que se mueven por el entorno y por lo tanto no se conoce previamente su posición, como personas, otros robots, entre otros. Por lo tanto, el sistema se diseña para calcular una trayectoria que pase por todas las posiciones de los productos (metas) evitando los obstáculos conocidos y con un componente reactivo que calcule rápidamente un movimiento para evitar la colisión del robot con un obstáculo que previamente no se conocía su posición y que puede moverse a través del entorno.

## 10.1. Generación de Trayectorias

### Grafo de Visibilidad

Para generar una trayectoria que vaya desde una coordenada inicial a una meta sin colisionar con obstáculos conocidos en un entorno se puede generar un mapa que caracterice el entorno, es decir, que determine espacios ocupados por obstáculos y espacios libres por los que el robot puede desplazarse, además este mapa debe ser computable para que un algoritmo pueda utilizar esta información. Debido a esto, una opción ampliamente utilizada son los grafos de visibilidad [34]. Los grafos son una estructura matemática que tiene dos elementos principales: nodos y aristas. Las aristas tienen la característica de unir dos nodos, y en el caso de un grafo ponderado las aristas tienen un valor relacionado a ellas, donde este valor se puede considerar como una penalización para pasar por esta arista. Estas estructuras matemáticas permiten describir muchos sistemas prácticos de una forma sencilla y computable, en otras palabras, permite aplicar algoritmos a partir de ellos. Por esta razón se utiliza un grafo para caracterizar el entorno, donde los nodos representan posiciones o coordenadas en el espacio euclidiano del entorno; y las aristas representan la conectividad entre dos nodos, es decir, si es posible ir de una coordenada a otra sin colisionar con obstáculos. En la Figura 26 se muestra una representación de cómo un grafo puede caracterizar la conectividad de un entorno con obstáculos, los círculos representan a los nodos y las flechas son las aristas del grafo.

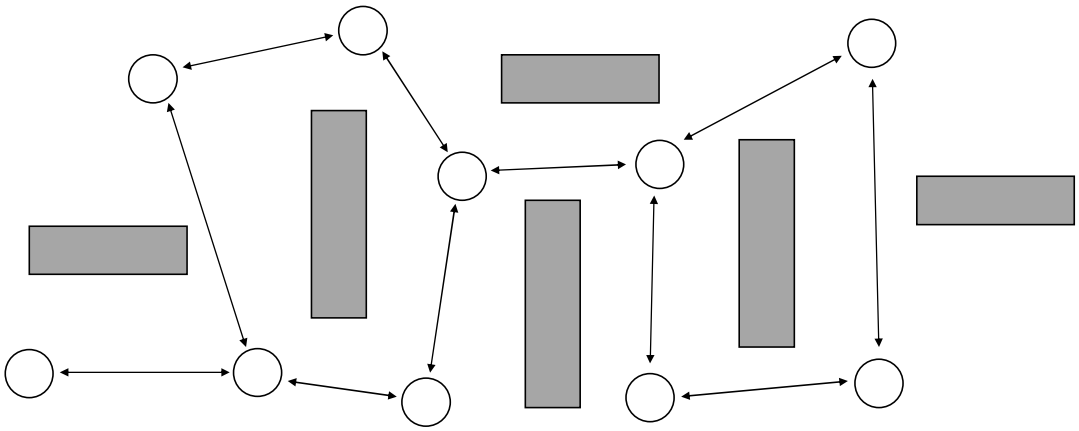


**Figura 26.** Representación de un grafo para caracterizar un entorno con obstáculos.

Entonces, se debe determinar cómo definir el grafo para representar correctamente la

configuración del entorno, en otras palabras, dónde localizar los nodos para que el grafo exprese correctamente los espacios libres del entorno. Para esto, se utilizan los grafos de visibilidad, donde los nodos se posicionan en todos los vértices de los obstáculos, analizando la conectividad de estos nodos, de esta forma los nodos se unen entre si con aristas solo si pueden “ver” al otro nodo, por esto su nombre. En la práctica, lo que se hace primero es “agrandar” los obstáculos para poder definir los nodos debido a que el robot tiene un radio (o un ancho dependiendo de la forma del robot), por lo que si no se agrandan los obstáculos esto provocaría que en las coordenadas de los nodos el robot colisione con los obstáculos [35].

A partir del grafo de visibilidad se puede agregar un nodo que represente la coordenada inicial y otro nodo para la meta, y se puede utilizar algún algoritmo de búsqueda en grafos para encontrar un camino entre estos dos nodos. La Figura 27 muestra una representación de un grafo de visibilidad donde: las líneas punteadas representan los obstáculos aumentados; los puntos negros son los nodos en cada esquina de los obstáculos; las líneas azules son las aristas del grafo de visibilidad; el punto y líneas naranjas representan el nodo inicial y sus respectivas aristas para conectar al grafo; y el punto y líneas verdes son el nodo meta y su conexiones.

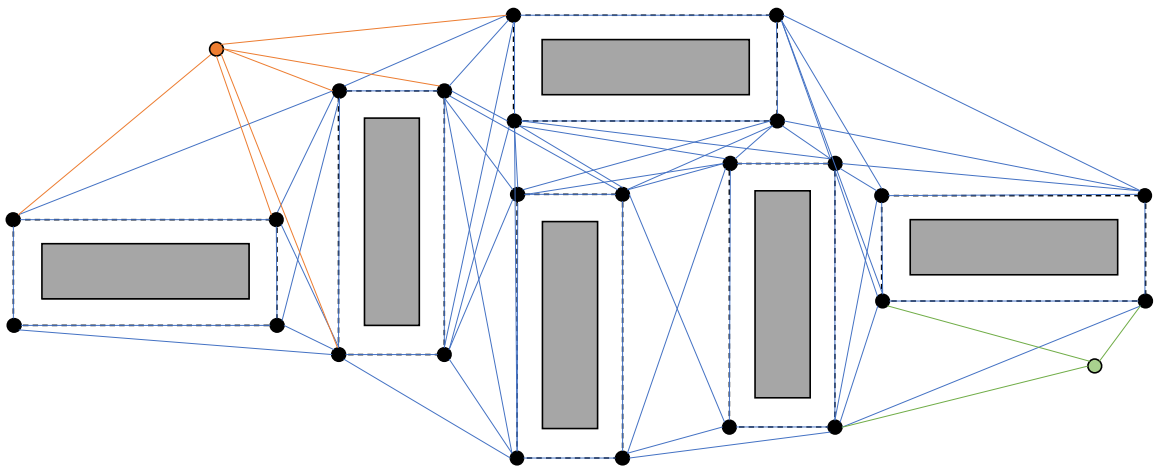


**Figura 27.** Representación de un grafo de visibilidad.

Es necesario implementar un algoritmo para calcular el grafo de visibilidad a partir de los nodos en las esquinas de los obstáculos. Este algoritmo debe ser capaz de determinar si existe conectividad entre los diferentes nodos del grafo, es decir, debe calcular si una

arista que conecte dos nodos no colisiona con un obstáculo, para todos los nodos. Este procedimiento se muestra en el Algoritmo 3, donde como entrada debe recibir la variable $nodos$ que es un vector con las coordenadas de los nodos, $vectores\_obstaculos$ que es un vector de vectores de las aristas de los obstáculos y $puntos\_obstaculos$ es un vector de los vértices de los obstáculos. El algoritmo verifica todas las combinaciones de nodos que puedan tener conectividad mediante la función $colision()$ que se muestra en el Algoritmo 4 en el Anexo A, donde si no existe colisión se agrega la distancia euclidiana entre los nodos en la matriz $mat\_pesos$ para ser utilizada posteriormente en un algoritmo de búsqueda de grafos ponderados. Finalmente, el algoritmo retorna $mat\_pesos$.

Para calcular si existe colisión se implementó el Algoritmo 4, que es una modificación de la metodología planteada en [36]. Esta metodología fue desarrollada para determinar si existe intersección entre dos segmentos de línea en un espacio tridimensional; sin embargo, se realiza modificaciones para aplicarlo en un espacio bidimensional. Se parte considerando a las aristas de los obstáculos y grafo como un segmento de línea definida por un punto inicial más un vector, como se muestra en la Figura 27. Entonces, cualquier punto que se encuentre sobre el segmento de línea puede ser definido por el punto inicial del segmento más el vector multiplicado por un parámetro escalar con un rango de $[0, 1]$. En la Figura 28, $p_1$ y $p_2$ son los puntos iniciales del segmento de línea, $p_1 + \vec{r}$ y $p_2 + \vec{s}$ son los puntos finales de los segmentos de línea y $t$, $u$ son los parámetros escalares. Si los dos segmentos de línea se intersecan, entonces por lo menos hay un punto común entre los dos segmentos, por lo tanto, se puede definir el punto común como (70).

$$p_1 + t \cdot \vec{r} = p_2 + u \cdot \vec{s} \tag{70}$$

Si a (70) se aplica el producto cruz a ambos lados por $\vec{s}$ y se utiliza la propiedad distributiva, se obtiene (71). Donde, por las propiedades del producto cruz se tiene que $\vec{s} \times \vec{s} = 0$.

$$p_1 \times \vec{s} + t \cdot \vec{r} \times \vec{s} = p_2 \times \vec{s} + u \cdot \vec{s} \times \vec{s} \tag{71}$$

Entonces, se puede despejar el factor escalar $t$ como (72). De manera similar, se puede desarrollar para el factor $u$ como (73).

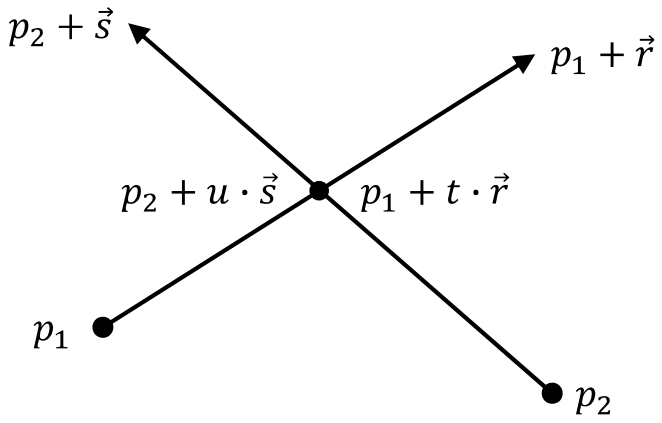



**Figura 28.** Intersección de dos segmentos de línea.

$$t = \frac{\|(p_2 - p_1) \times \vec{s}\|}{\|\vec{r} \times \vec{s}\|} \tag{72}$$

$$u = \frac{\|(p_2 - p_1) \times \vec{r}\|}{\|\vec{s} \times \vec{r}\|} \tag{73}$$

Donde se tiene que:

1. Si $\|\vec{r} \times \vec{s}\| = 0$ y $\|(p_2 - p_1) \times \vec{r}\| = 0$, entonces los segmentos de línea son colineales.
2. Si $\|\vec{r} \times \vec{s}\| = 0$ y $\|(p_2 - p_1) \times \vec{r}\| \neq 0$, entonces los segmentos de línea son paralelos pero no se intersecan.
3. Si $\|\vec{r} \times \vec{s}\| \neq 0$, $0 \leq t \leq 1$ y $0 \leq u \leq 1$, entonces los segmentos de línea se intersecan en $p_1 + t \cdot \vec{r} = p_2 + u \cdot \vec{s}$.
4. Caso contrario, los segmentos de línea no son paralelos pero no se intersecan

Para la aplicar esta metodología de detección de colisión de una arista en la construcción de un grafo de visibilidad, es útil el caso 3, pero se debe considerar para $0 < t < 1$ y $0 < u < 1$ ya que existirá uno o dos puntos comunes entre las aristas del grafo y los obstáculos, y estos son los nodos, pero esta intersección en los nodos se da solo cuando $t$ o $u$ son $0$ o $1$, es decir, cuando se intersecan solo en los extremos de las aristas. En el Algoritmo 4 en el Anexo A se muestra el pseudocódigo para detectar si dos nodos pueden conectarse con una arista sin colisionar con los obstáculos. Este algoritmo recibe como entrada $nodo1$ y $nodo2$

que son las coordenadas de los nodos que se quiere conectar; $vectores\_obstaculos$ es un vector de vectores de las aristas de los obstáculos (equivalente a $\vec{r}$ o $\vec{s}$ en el desarrollo de la metodología); y $puntos\_obstaculos$ es un vector de los vértices de los obstáculos. Entonces, el algoritmo aplica la metodología desarrollada para cada arista de los obstáculos, y si se detecta una intersección en cualquiera de estas aristas se devuelve $check$ como $verdadero$, caso contrario, no existe colisión y se devuelve $check$ como $falso$.

### Algoritmo de Búsqueda A*

El algoritmo A* es un algoritmo de búsqueda en grafos para encontrar una ruta entre un nodo inicial y un nodo final con un costo mínimo dentro de todas las distintas posibilidades de rutas. Este algoritmo es ampliamente utilizado debido a que utiliza una heurística para encontrar la solución en menos tiempo comparado con otros algoritmos similares como el algoritmo de Dijkstra. Sin embargo, esta eficiencia mejorada depende de escoger correctamente la variable que se utilizará como heurística. En el caso de generación de trayectorias en un espacio euclidiano de dos dimensiones una heurística común es la distancia euclidiana hacia el nodo final, es decir, la distancia en línea recta hacia la meta.

El algoritmo A* se basa en expandir un árbol desde el nodo inicial hacia los nodos vecinos (nodos que tengan conectividad) y analiza cuál es el nodo con menor función de costo para expandirse a través de este hasta que finalmente llega al nodo meta, y así poder definir la ruta con menor costo al analizar la jerarquía del árbol, iniciando desde la raíz (nodo inicial) hasta el nodo meta. Lo especial de este algoritmo es la definición de la función de costo, que es $costo = pesos[nodo] + heuristica[nodo]$. La componente heurística hace que los nodos más cercanos a la meta tengan una prioridad para expandir el árbol comparado con nodos que se alejan de la meta. Por esta razón, el algoritmo encuentra una solución en menor o igual tiempo a algoritmos que no utilizan una heurística [37].

El Algoritmo 5 en el Anexo A muestra el pseudocódigo para aplicar el método A*. Este algoritmo toma como entrada las variables $pesos$ que es la matriz de pesos del grafo de visibilidad (costo de cada arista del grafo); $nodos$ que es una lista de los nodos del grafo; $n\_inicial$ que es el nodo desde donde parte la búsqueda; y $n\_final$ que es el nodo meta. El resultado del algoritmo es la variable $camino$, que es una lista con el orden de los nodos

para ir del nodo inicial al nodo final; y $costo\_total$ que es el costo en ir desde el nodo inicial hasta el nodo final (esta variable se utilizará para el Algoritmo 6 del Anexo A).

**Problema del Agente Viajero**

El algoritmo A* permite encontrar una ruta desde un nodo inicial hacia un nodo meta en un grafo. Sin embargo, para las necesidades del proyecto se requiere que el algoritmo calcule una ruta que vaya desde un nodo inicial hasta una serie de nodos meta, ya que el robot debe ser capaz de recolectar varios productos en una trayectoria para ser eficiente. Además, el algoritmo debe calcular una ruta que sea lo suficientemente eficiente; para esto debe encontrar una trayectoria que minimice la distancia recorrida del robot y que pase por todas las metas. Este problema se lo conoce como el ”Problema del Agente Viajero” (*Travelling Salesman Problem* - TSP), el cuál es uno de los problemas más estudiados en el campo de la optimización combinatoria [38].

El problema del agente viajero es computacionalmente complicado, ya que el tiempo de cómputo aumenta significativamente dependiendo del número de metas que se analizan y del algoritmo que se utilice. Sin embargo, se han desarrollado muchos algoritmos que abordan este problema, unos heurísticos y otros exactos, donde los heurísticos generan soluciones en menor tiempo y con altas probabilidades de encontrar la solución óptima; mientras que los exactos suelen tener mayor tiempo de cómputo, pero siempre encuentran la solución óptima.

Debido a que el entorno en el que se diseñó el robot no tiene un número considerable de metas y nodos por analizar (en el sistema que se trabaja se tiene un número de nodos del orden de varias decenas hasta varias centenas, en condiciones del orden de las decenas de miles o millones de nodos es preferible utilizar un método heurístico) se implementó el método exacto más sencillo. El método consiste en analizar todas las permutaciones a las metas comenzando desde el nodo inicial y avanzar hacia la meta menos costosa, luego, a partir de esta meta analizar nuevamente todas las permutaciones posibles y continuar con la iteración hasta que se haya encontrado una ruta que pasa por todas las metas.

El Algoritmo 6 en el Anexo A muestra el pseudocódigo para encontrar un ruta que pase por todas las metas requeridas. El algoritmo toma como entradas: la variable $metas$, que es

una lista de los nodos meta; $pesos$ que es una matriz de pesos del grafo de visibilidad; $nodos$ que es una lista de los nodos del grafo; y $n\_inicial$ que es el nodo desde donde comienza la búsqueda. El algoritmo comienza la búsqueda desde $n\_inicial$ y mediante el Algoritmo 5 (A*) se calcula el costo de las rutas hacia todas las metas, se analiza cuál es menor costo de todas las rutas y se pasa a esta meta. Entonces, se toma la meta con menor costo como el nuevo nodo inicial y se elimina de la lista $metas\_no\_revisadas$ que almacena todas las metas que aún no han sido agregadas a la ruta. Se realiza el mismo proceso iterativamente hasta que todas las metas se agregaron a la ruta, es decir, cuando $metas\_no\_revisadas$ sea una lista vacía. Finalmente, el algoritmo retorna la variable $camino$ que almacena el orden de los nodos que se debe seguir para pasar por todas las metas.

## 10.2. Campos Potenciales Virtuales

Ahora que se tiene una ruta con una serie de coordenadas por las que el robot debe pasar, es necesario definir un planeador de trayectoria, es decir, un algoritmo que calcule cómo debe desplazarse el robot para llegar a las coordenadas establecidas. Además, este planeador debe evitar colisionar con obstáculos dinámicos que aparezcan en el entorno modificando la trayectoria. Para este problema se utilizó el concepto de ”Campos Potenciales Virtuales” (*Virtual Potential Fields)*) [39].

Los campos potenciales virtuales se crearon inspirados en energías potenciales existentes en la naturaleza, como la energía potencial gravitacional o campos magnéticos, que provocan el movimiento de diferentes elementos al transformar la energía potencial en energía cinética. A partir de este concepto, se asignan bajos potenciales a las metas y a los obstáculos se les asignan potenciales altos, entonces, en el entorno se genera una distribución de potenciales que si se transforma en energía cinética generarán fuerzas virtuales que “atraerán” al robot hacia la meta (que arbitrariamente se lo establece como el mínimo global del campo potencial) y “alejarán” al robot de obstáculos. Aunque los campos potenciales virtuales entran en el área de los planeadores, estrictamente hablando, esta metodología no planea los movimientos, en cambio, es una metodología reactiva que responde a las diferentes fuerzas virtuales que encuentra en el entorno. Esto permite que el robot reaccione rápidamente a obstáculos (estáticos o dinámicos) que no necesariamente se conocía su

posición con anticipación, con un bajo costo computacional.

Existen varias formas de definir el campo potencial virtual para desarrollar las ecuaciones de las fuerzas virtuales, tanto para las fuerzas de atracción, como para las fuerzas de repulsión, para diferentes entornos. Una de estas formas establece las metas con un campo virtual equivalente a un campo gravitacional, donde la ecuación de la fuerza de atracción esta dada por (74). La variable $K_{atraccion}$ es una constante y $error$ es el error de posición del robot con respecto a la meta, y se calcula mediante (75). Por lo tanto, la fuerza de atracción depende linealmente del error de posición del robot.

$$\vec{F}_{atraccion} = K_{atraccion} \cdot \vec{error} \tag{74}$$

$$\vec{error} = \vec{meta} - \vec{pose} \tag{75}$$

Sin embargo, plantear la fuerza de atracción como una función lineal que dependa del error de posición tiene dos problemas principales; el primero es que cuando el robot se encuentre muy alejado de la meta el error de posición será muy grande y por lo tanto $F_{atraccion}$ será muy grande; el segundo problema es que cuando el robot empiece a acercarse a la meta, $F_{atraccion}$ irá reduciendo su valor linealmente, por lo que en los alrededores de la meta $F_{atraccion}$ será muy baja y puede que no consiga atraer al robot los suficiente como para llegar a un error de posición lo suficientemente pequeño. Debido a esto se modificó la función (74) a (76), basada en la función de un sistema de primer orden, donde la constante $C_{max}$ es la fuerza máxima de atracción (equivalente a la ganancia en un sistema de primer orden) y la constante $T$ define el error de posición al que $F_{atraccion}$ será el $63\,\%$ de $C_{max}$ (equivalente a la constante de tiempo en un sistema de primer orden). En la Figura 29 se muestra una comparación del comportamiento de los modelos, donde se puede evidenciar la ventaja de la modificación, debido a que aunque el error de posición sea grande, $F_{atraccion}$ tendrá un valor máximo igual a $C_{max}$, y mediante la constante $T$ se puede controlar el error de posición a la que $F_{atraccion}$ comienza a decrecer.

$$\vec{F}_{atraccion} = C_{max} \cdot (1 - e^{-\frac{error}{T}}) \tag{76}$$

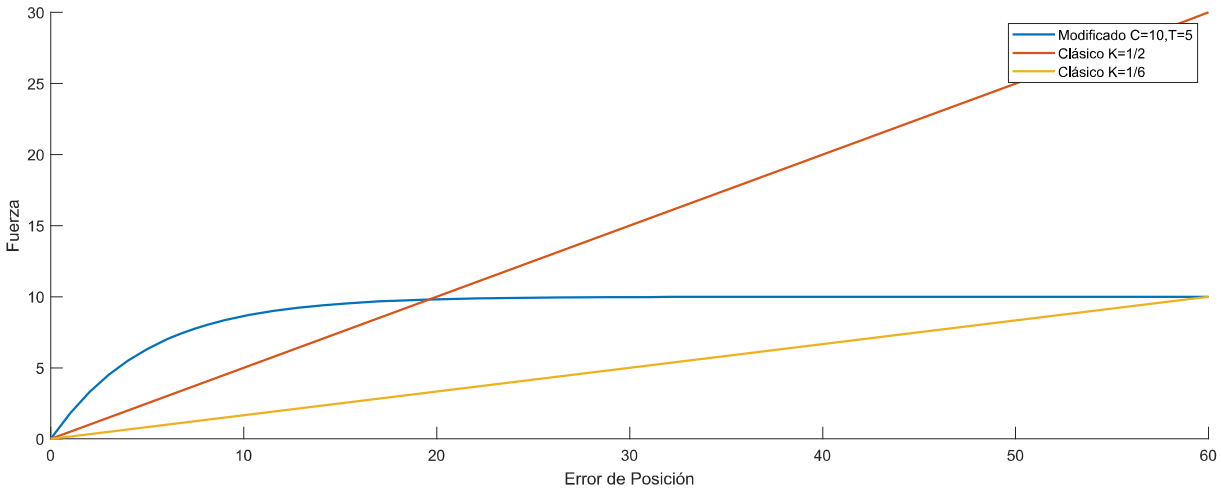



**Figura 29.** Comparación de modelos de fuerza de atracción en campos potenciales virtuales.

Por el lado de la repulsión, un modelo planteado en [26] está dado por (77), donde esta forma tiene la ventaja de que los obstáculos distantes a la posición del robot no contribuirán a la fuerza de repulsión (para distancias mayores a una variable $d_{rango}$) . En este caso también se realizó una modificación al modelo, como se muestra en (78). Sin embargo, aunque este ajuste no afecta significativamente el comportamiento de $F_{repulsion}$, sí tiene la ventaja de ser una función más sencilla y con menos operaciones. $K_{repulsion}$ es una constante utilizada para ajustar el comportamiento de $F_{repulsion}$; $d_{obstaculo}$ es la distancia del obstáculo hasta el centro del robot; y $R_{robot}$ es el radio del robot. En la Figura 30 se muestra una comparación de los dos modelos, donde se puede observar que el comportamiento de ambos es casi idéntico. También se puede analizar que la $F_{repulsion}$ es nula cuando las distancias a los obstáculos sean lejanas, pero cuando el robot se acerque a un obstáculo, $F_{repulsion}$ aumentará rápidamente, generando grandes fuerzas de repulsión si el robot se acerca demasiado.

$$\vec{F}_{repulsion} = \begin{cases} \frac{K_{repulsion}}{2} \cdot \left(\frac{d_{rango} - d_{obstaculo}}{d_{rango} \cdot d_{obstaculo}}\right)^2 & \text{si } d_{obstaculo} < d_{rango} \\ 0 & \text{caso contrario} \end{cases} \tag{77}$$

$$\vec{F}_{repulsion} = \begin{cases} \frac{K_{repulsion}}{2} \cdot \left(\frac{1}{d_{obstaculo} - R_{robot}}\right)^2 & \text{si } d_{obstaculo} < d_{rango} \\ 0 & \text{caso contrario} \end{cases} \tag{78}$$

Entonces, se puede definir la fuerza neta en cada punto del entorno mediante (79). Donde la componente de repulsión es la sumatoria de todas las fuerzas repulsivas correspondientes a cada obstáculo que se encuentra en el entorno.

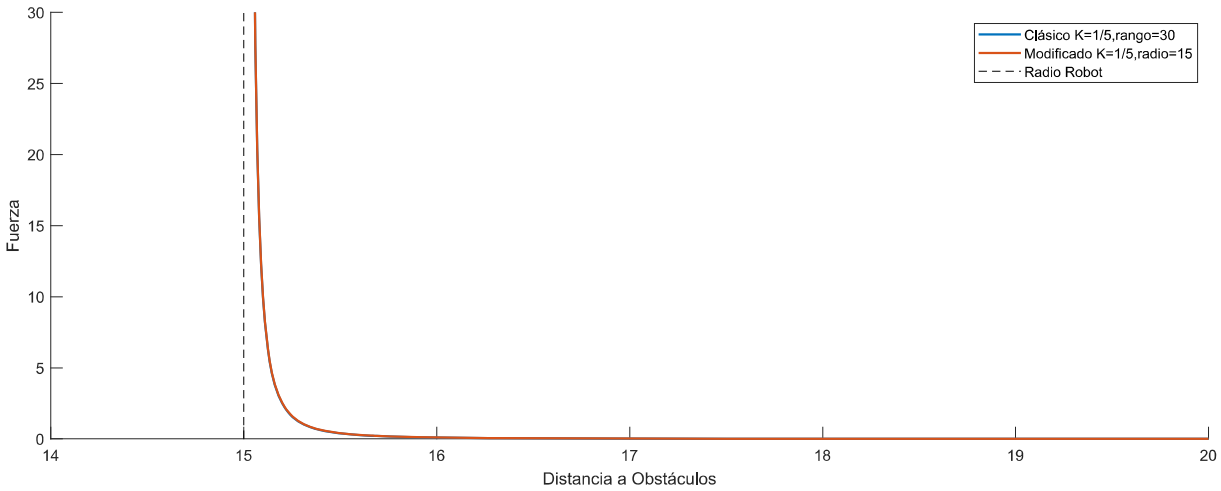



**Figura 30.** Comparación de modelos de fuerza de repulsión en campos potenciales virtuales.

$$\vec{F} = \vec{F}_{atrracion} + \sum_{i=1}^{n_{obstaculos}} \vec{F}^{i}_{repulsion} \tag{79}$$

Controlar un robot por medio de comandos de fuerza implicaría desarrollar el modelo dinámico del robot, lo cual es innecesario para la aplicación específica del proyecto, ya que es más importante controlar la posición y la velocidad del robot que la fuerza con la que se desplaza. Entonces, la opción utilizada es interpretar las fuerzas virtuales como velocidades de control, como se muestra en (80), donde $\dot{q} = (\dot{\phi}, \dot{x}, \dot{y})$.

$$\dot{q} = \vec{F} \tag{80}$$

En la Figura 31 se muestra una comparación del campo de velocidades de control para un entorno arbitrario con obstáculos y con una meta en el centro. En la izquierda se presenta el método clásico y en la derecha el método modificado. Se puede identificar como las velocidades de control alrededor de los obstáculos se desvían de estos, y no existe mayor diferencia entre los dos métodos. Sin embargo, la atracción en el método clásico disminuye significativamente en los alrededores de la meta, con velocidades de control muy pequeñas en posiciones con errores de posición considerables. Pero en el método modificado se puede evidenciar velocidades de control homogéneas (de valor $C_{max}$) por todo el entorno, y con reducciones en las velocidades de control solo en los alrededores inmediatos de la meta (controlado por el factor $T$).

El método de campos potenciales virtuales es una gran herramienta para otorgar una

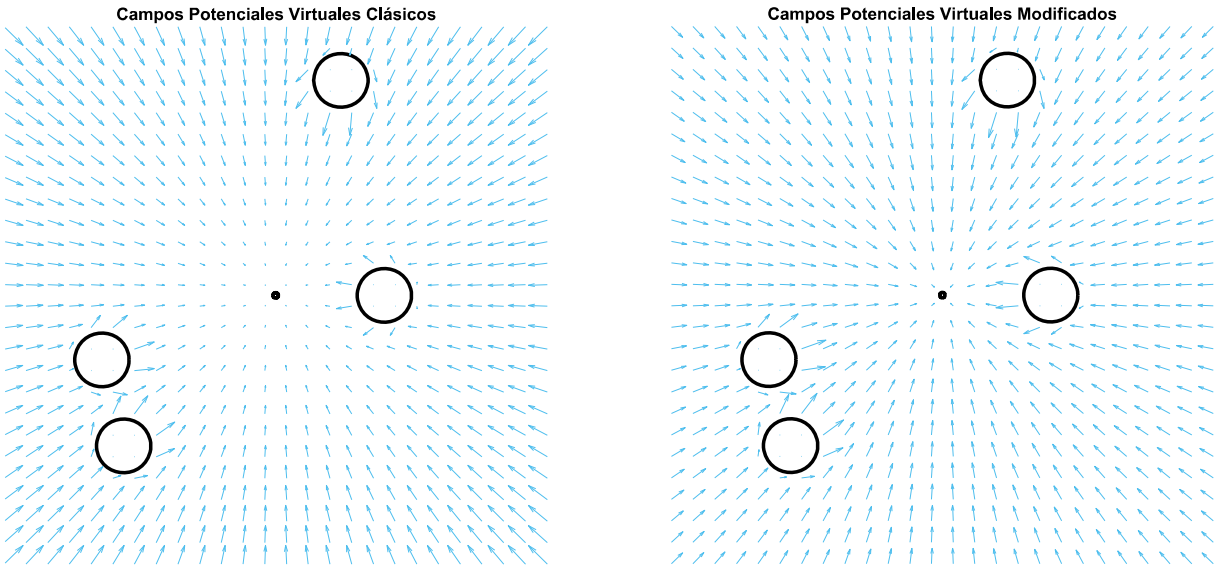



**Figura 31.** Comparación de modelos de velocidades de control con campos potenciales virtuales.

capacidad reactiva a la trayectoria del robot. Sin embargo, tiene la desventaja de que puede generar mínimos locales diferentes a la meta, los cuales pueden provocar que el robot se quede “atascado” en una posición diferente a la posición final deseada.

### 10.3. Modelo Cinemático de la Plataforma Móvil

Es necesario conocer el modelo cinemático de la plataforma móvil para poder utilizar esta información y encontrar una expresión que relacione el movimiento de las ruedas (que en este caso son del tipo *mecanum*, las cuales tienen una serie de rodillos que giran libremente unidos alrededor de la rueda en un ángulo $\alpha$) con el movimiento de la plataforma móvil. Para esto se parte del análisis del comportamiento de estas ruedas, en la Figura 32 se puede evidenciar que estas ruedas tienen un movimiento motor como cualquier rueda común en $\hat{x}$ (es decir, permiten el movimiento en una dirección perpendicular a los ejes de las ruedas), pero también, debido a los rodillos que giran libremente, se tiene un movimiento de deslizamiento en un ángulo $\alpha$ que depende del ángulo de los rodillos respecto a la rueda.

A partir del diagrama de la Figura 32, mediante una sumatoria de velocidades, se puede desarrollar la velocidad en $\hat{x}$ e $\hat{y}$ de la plataforma como (81) y (82), respectivamente.

$$v_x = v_{motor} - sen(\alpha) \cdot v_{deslizamiento} \tag{81}$$

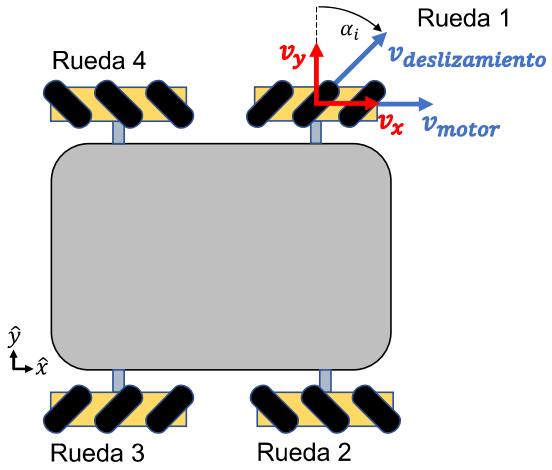



**Figura 32.** Modelo cinemático de la plataforma móvil.

$$v_y = cos(\alpha) \cdot v_{deslizamiento} \tag{82}$$

Despejando $v_{deslizamiento}$ de (82) se obtiene (83), y reemplazando en (81) se puede encontrar (84).

$$v_{deslizamiento} = \frac{v_y}{cos(\alpha)} \tag{83}$$

$$v_{motor} = v_x + tan(\alpha) \cdot v_y \tag{84}$$

Finalmente, para encontrar la velocidad a la cual debe girar la rueda para tener una determinada velocidad del chasís del robot definida por $\dot{q} = (\dot{\phi}, \dot{x_s}, \dot{y_s})$, donde $\phi$ describe la orientación del robot, y el par $(x_s, y_s)$ definen la posición del robot respecto al sistema de referencia global $\{s\}$, se aplica (85).

$$\omega_{motor} = \frac{v_{motor}}{r} = \frac{1}{r} \cdot (v_x + tan(\alpha) \cdot v_y) \tag{85}$$

## 10.4. Odometría

La odometría se utiliza para estimar la posición del robot a partir del giro de las ruedas registrado en los encoders. Es decir, se quiere estimar $q_{k+1}$ mediante la ecuación (86) donde únicamente se conoce la configuración actual del robot $q_k$ y la variación del giro en cada

rueda $\Delta\theta_i$ en en un intervalo $\Delta t$.

$$q_{k+1} = \Delta q + q_k \tag{86}$$

Si se aproxima la velocidad angular de cada rueda como constante (ya que los motores tienen un control de velocidad, como se mostrará en la Sección 10.5), se tiene que $\omega^i_{motor} = \Delta\theta_i/\Delta t$ se puede encontrar una relación entre $\omega^i_{motor}$ y $\dot{q}$ mediante (87), donde $h_i \in \mathbb{R}^3$ para cada rueda $i$.

$$\omega^i_{motor} = h_i \cdot \dot{q} \tag{87}$$

Para desarrollar $h_i$ se aplicarán una serie de transformaciones, primero se opera $\dot{q}$ que está expresado respecto a $\{s\}$ para representarlo como un $twist$ $\Upsilon$ expresado respecto al sistema de referencia de la plataforma móvil $\{b\}$, es decir, expresar $\dot{q}$ como $\Upsilon_b = (\dot{\phi}, v_{bx}, v_{by})$ mediante una matriz de rotación $R^b_s(\phi)$ que depende de la orientación del robot (88), aplicada en (89).

$$R^b_s(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sen(\phi) \\ 0 & -sen(\phi) & cos(\phi) \end{bmatrix} \tag{88}$$

$$\Upsilon_b = R^b_s(\phi) \cdot \dot{q} \tag{89}$$

Posteriormente se transforma $\Upsilon_b$ al vector $v_{ci} \in \mathbb{R}^2$ que representa la velocidad lineal en cada rueda $i$ en $\{c\}_i$, mediante (90).

$$v_{ci} = M_{ci} \cdot \Upsilon_b \tag{90}$$

La matriz de transformación $M_{ci}$ se puede desarrollar a partir de la Figura 33 donde se muestra que en $\{c\}_1$ además de las velocidad presentes en $\{b\}$ también se debe considerar las velocidades tangenciales que se generan cuando la plataforma rota a una velocidad $\dot{\phi}$ respecto a su centro geométrico. Por lo tanto, las componentes de la velocidad lineal en $\{c\}_i$ se expresan mediante (91) y (92).

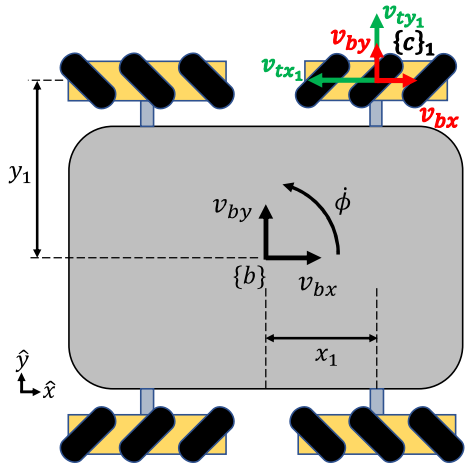



**Figura 33.** Velocidades presentes en las ruedas del robot.

$$v_{cxi} = -y_i \cdot \dot{\phi} + v_{bx} \tag{91}$$

$$v_{cyi} = x_i \cdot \dot{\phi} + v_{by} \tag{92}$$

Si (91) y (92) se representan de forma matricial (93) se puede determinar la matriz $M_{ci}$ expresada en (94) que satisface (90).

$$v_{ci} = \begin{bmatrix} v_{cxi} \\ v_{cyi} \end{bmatrix} = \begin{bmatrix} -y_i & 1 & 0 \\ x_i & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ v_{bx} \\ v_{by} \end{bmatrix} \tag{93}$$

$$M_{ci} = \begin{bmatrix} -y_i & 1 & 0 \\ x_i & 0 & 1 \end{bmatrix} \tag{94}$$

A continuación se transforma la velocidad lineal en el sistema de referencia de cada rueda $\{c\}_i$ a la velocidad angular de cada rueda $\omega^i_{motor}$ mediante el modelo cinemático expresado en (85). Representando (85) en forma matricial (95), se puede expresar $M_{\omega i}$ como (96).

$$\omega^i_{motor} = M_{\omega i} \cdot v_{ci} = \begin{bmatrix} \frac{1}{r} & \frac{tan(\alpha_i)}{r} \end{bmatrix} \cdot \begin{bmatrix} v_{cxi} \; v_{cyi} \end{bmatrix} \tag{95}$$

$$M_{\omega i} = \begin{bmatrix} \frac{1}{r} & \frac{tan(\alpha_i)}{r} \end{bmatrix} \tag{96}$$

Por último, se puede desarrollar el valor de $h_i$ en (87) mediante (89), (90) y (95), como (98).

$$\omega^i_{motor} = M_{\omega i} \cdot M_{ci} \cdot R^b_s(\phi) \cdot \dot{q} \tag{97}$$

$$h_i = M_{\omega i} \cdot M_{ci} \cdot R^b_s(\phi) = \frac{1}{r} \cdot \begin{bmatrix} l \cdot tan(\alpha_i) - w & \frac{cos(\alpha_i + \phi)}{cos(\alpha_i)} & \frac{sen(\alpha_i + \phi)}{cos(\alpha_i)} \end{bmatrix} \tag{98}$$

Generalizando (97) para las cuatro ruedas de la plataforma móvil se tiene (99), donde $\omega_{motor} = [\omega^1_{motor}\, \omega^2_{motor}\, \omega^3_{motor}\, \omega^4_{motor}]^T$ y $H = [h_1\, h_2\, h_3\, h_4]^T$.

$$\omega_{motor} = H \cdot \dot{q} \tag{99}$$

Si se asume que $\omega^i_{motor} \approx \Delta\theta_i / \Delta t$ y $\dot{q} \approx \Delta q / \Delta t$ se puede realizar una integración en $\Delta t$ para obtener una expresión que relacione $\Delta\theta$ y $\Delta q$. Para la integración se utiliza el concepto de matriz exponencial, presentada en la Sección 7.4, para obtener una matriz de transformación que represente el cambio de configuración de la plataforma, sin embargo, si se aplica la matriz exponencial a $\dot{q}$ se obtiene como resultado una matriz de transformación que representa el cambio de configuración del robot respecto a $\{s\}$ y no respecto a $q_k$. Por lo tanto, se modifica ligeramente $h_i$ como (100), donde se puede definir $H' = [h'_1 h'_2 h'_3 h'_4]^T$, para poder trabajar con la ecuación (101) que contiene el $twist\ \Upsilon_b$ que está expresado respecto al sistema de referencia de la plataforma móvil $\{b\}$, y posteriormente se transformará al sistema $\{s\}$.

$$h'_i = M_{\omega i} \cdot M_{ci} = \frac{1}{r} \cdot \begin{bmatrix} x_i \cdot tan(\alpha_i) - y_i & 1 & tan(\alpha_i) \end{bmatrix} \tag{100}$$

$$\omega_{motor} \approx \frac{\Delta\theta}{\Delta t} = H' \cdot \Upsilon_b \tag{101}$$

Para aplicar la matriz exponencial a $\Upsilon_b$ se debe despejar esta variable, pero $H' \in \mathbb{R}^{4\times 3}$, y ya que no es una matriz cuadrada no se puede calcular la inversa de la matriz. No obs-

tante, se puede calcular la pseudoinversa de Moore-Penrose $H'^{\dagger}$ mediante (15), y si se reemplazan los valores de $x_i$, $y_i$ y $\alpha_i$ en $H'$ según la Figura 34 y se calcula la pseudinversa se obtiene (102).

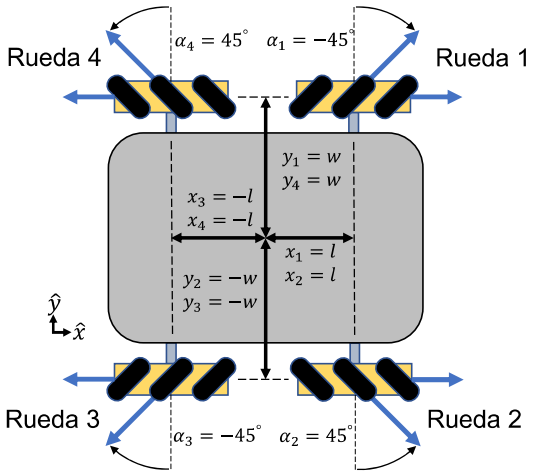



**Figura 34.** Valores del modelo de la plataforma móvil.

$$H'^{\dagger} = \frac{r}{4} \cdot \begin{bmatrix} \frac{-1}{l+w} & \frac{1}{l+w} & \frac{1}{l+w} & \frac{-1}{l+w} \\ 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \tag{102}$$

Entonces se puede calcular (103), donde $\Delta\theta = [\Delta\theta_1\ \Delta\theta_2\ \Delta\theta_3\ \Delta\theta_4]^T$.

$$\Upsilon_b = H'^{\dagger} \cdot \Delta\theta \cdot \frac{1}{\Delta t} = \frac{r}{4} \cdot \begin{bmatrix} \frac{-\Delta\theta_1+\Delta\theta_2+\Delta\theta_3-\Delta\theta_4}{l+w} \\ \Delta\theta_1 + \Delta\theta_2 + \Delta\theta_3 + \Delta\theta_4 \\ -\Delta\theta_1 + \Delta\theta_2 - \Delta\theta_3 + \Delta\theta_4 \end{bmatrix} \cdot \frac{1}{\Delta t} \tag{103}$$

A partir de (103) se puede expandir el $twist$ respecto a $\{b\}$ en las seis dimensiones espaciales como $\Upsilon_{b6} = [\omega_{bx}\ \omega_{by}\ \omega_{bz}\ v_{bx}\ v_{by}\ v_{bz}]^T = [0\ 0\ \dot{\phi}\ v_{bx}\ v_{by}\ 0]^T$, donde $\omega = [0\ 0\ \dot{\phi}]^T$ y $v = [v_{bx}\ v_{by}\ 0]^T$. Por lo tanto, a partir de (8) y (9) se define $\hat{\omega} = \omega/||\omega||$ y (104).

$$\Delta\phi = \omega_{bz} = \frac{r}{4} \cdot \frac{-\Delta\theta_1 + \Delta\theta_2 + \Delta\theta_3 - \Delta\theta_4}{l+w} \tag{104}$$

Aplicando la matriz exponencial $e^{[\Upsilon_{b6}]} = e^{[S]\Delta\phi}$ (11) a $\Upsilon_b$ se tiene (105) y (106) si $||\hat{\omega}|| = 1$, es decir, si $\Delta\phi \neq 0$.

$$R_{b'}^{b} = \begin{bmatrix} cos(\Delta\phi) & -sen(\Delta\phi) & 0 \\ sen(\Delta\phi) & cos(\Delta\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{105}$$

$$p_{b'}^{b} = \begin{bmatrix} \frac{r}{4} \cdot [sen(\Delta\phi) \cdot (\Delta\theta_1 + \Delta\theta_2 + \Delta\theta_3 + \Delta\theta_4) + (cos(\Delta\phi) - 1) \cdot (-\Delta\theta_1 + \Delta\theta_2 - \Delta\theta_3 + \Delta\theta_4)] \\ \frac{r}{4} \cdot [(1 - cos(\Delta\phi)) \cdot (\Delta\theta_1 + \Delta\theta_2 + \Delta\theta_3 + \Delta\theta_4) + sen(\Delta\phi) \cdot (-\Delta\theta_1 + \Delta\theta_2 - \Delta\theta_3 + \Delta\theta_4)] \\ 0 \end{bmatrix} \tag{106}$$

Si se tiene que $||\hat{\omega}|| = 0$ (12) , es decir, si $\Delta\phi = 0$, dando como resultado (107) y (108).

$$R_{b'}^{b} = I \tag{107}$$

$$p_{b'}^{b} = \begin{bmatrix} \frac{r}{4} \cdot (\Delta\theta_1 + \Delta\theta_2 + \Delta\theta_3 + \Delta\theta_4) \\ \frac{r}{4} \cdot (-\Delta\theta_1 + \Delta\theta_2 - \Delta\theta_3 + \Delta\theta_4) \\ 0 \end{bmatrix} \tag{108}$$

A partir de (108) se puede definir $\Delta q_b = [\Delta\phi\ \Delta x_b\ \Delta y_b]^T$ para dos casos:

- si $\Delta\phi = 0$, entonces se tiene (109).

$$\Delta q_b = \begin{bmatrix} 0 \\ \frac{r}{4} \cdot (\Delta\theta_1 + \Delta\theta_2 + \Delta\theta_3 + \Delta\theta_4) \\ \frac{r}{4} \cdot (-\Delta\theta_1 + \Delta\theta_2 - \Delta\theta_3 + \Delta\theta_4) \end{bmatrix} \tag{109}$$

- si $\Delta\phi \neq 0$, entonces se tiene (110).

$$\Delta q_b = \begin{bmatrix} \Delta\phi \\ \frac{r}{4} \cdot [sen(\Delta\phi) \cdot (\Delta\theta_1 + \Delta\theta_2 + \Delta\theta_3 + \Delta\theta_4) + (cos(\Delta\phi) - 1) \cdot (-\Delta\theta_1 + \Delta\theta_2 - \Delta\theta_3 + \Delta\theta_4)] \\ \frac{r}{4} \cdot [(1 - cos(\Delta\phi)) \cdot (\Delta\theta_1 + \Delta\theta_2 + \Delta\theta_3 + \Delta\theta_4) + sen(\Delta\phi) \cdot (-\Delta\theta_1 + \Delta\theta_2 - \Delta\theta_3 + \Delta\theta_4)] \end{bmatrix} \tag{110}$$

Pero es necesario transformar $\Delta q_b$ para que sea expresado respecto a $\{s\}$ y poder ser aplicado en (86), por lo que se utiliza una matriz de rotación (111), obteniendo como resultado (112).

$$R_{bs}(\phi_k) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi_k) & -sen(\phi_k) \\ 0 & sen(\phi_k) & cos(\phi_k) \end{bmatrix} \tag{111}$$

$$\Delta q = \begin{bmatrix} \Delta\phi \\ \Delta x_s \\ \Delta y_s \end{bmatrix} = R_{bs}(\phi_k) \cdot \Delta q_b = \begin{bmatrix} \Delta\phi \\ cos(\phi_k) \cdot \Delta x_b - sen(\phi_k) \cdot \Delta y_b \\ sen(\phi_k) \cdot \Delta x_b + cos(\phi_k) \cdot \Delta y_b \end{bmatrix} \tag{112}$$

Finalmente, con (112) se puede desarrollar (86) como (113).

$$q_{k+1} = \begin{bmatrix} \phi_k + \Delta\phi \\ x_k + cos(\phi_k) \cdot \Delta x_b - sen(\phi_k) \cdot \Delta y_b \\ y_k + sen(\phi_k) \cdot \Delta x_b + cos(\phi_k) \cdot \Delta y_b \end{bmatrix} \tag{113}$$

## 10.5. Control de Velocidad para Motores DC

### Diseño del Controlador

El sistema de retroalimentación que se implementó para el control de velocidad de los motores DC se muestra en la Figura 35. Donde el controlador cumple la ley de control (114).

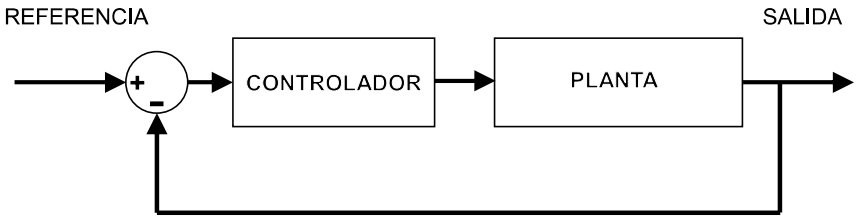



**Figura 35.** Esquema del sistema con retroalimentación y controlador.

$$u(t) = k_p \cdot e(t) + k_i \cdot (t)dt + k_d \cdot \frac{de(t)}{dt} \tag{114}$$

Para seleccionar el controlador específico se probó primero obtener el modelo de los motores DC (planta) por el método de sintonización por la ganancia crítica, pero no se obtuvieron buenos resultados ya que el periodo crítico era menor al menor periodo de muestreo posible (ya que las mediciones de velocidad mediante los encoders de efecto *Hall* son discretas), por lo que no se pudo visualizar correctamente la oscilación para obtener los valores para la sintonización. Después, se procedió a probar el método de los dos puntos para aproximar el modelo como un sistema de primer orden con retardo, pero el resultado del modelo obtenido con esta aproximación no se comportaba como el modelo real, por lo que se tuvo que descartar este método. Finalmente, se sintonizó el controlador PID mediante ajustes por prueba y error. Donde se comenzó por aumentar la constante proporcional $k_p$ hasta obtener una respuesta similar a la referencia, luego se aumentó la constante integral $k_i$ hasta que el error en estado estacionario sea cero, y finalmente se agregó la constante derivativa $k_d$ para disminuir el sobre impulso y las oscilaciones. Este proceso fue iterativo hasta encontrar una respuesta con condiciones adecuadas. Los coeficientes del controlador escogidos se muestran en la Tabla 8.

**Tabla 8.** Coeficientes para controlador PID.

| $k_p$ | $k_i$ | $k_d$ |
|---|---|---|
| 1.1 | 36 | 0.004 |

### Discretización

Para implementar el controlador PID en un microcontrolador es necesario discretizar el controlador. Existen varios métodos para discretizar un controlador PID, los más comunes son Euler I, Euler II y Tustin. Los tres métodos de discretización comparten la ley de control (115), sin embargo cambian en los coeficientes del controlador discretizado. En la Tabla 9 se muestra las ecuaciones para calcular los coeficientes del controlador discretizado mediante el método de Euler I, Euler II y Tustin, a partir de los valores mostrados en la Tabla 8. El periodo de muestreo seleccionado es de $T_m = 2\ \mathrm{ms}$.

$$u_k = u_{k-1} + q_0 \cdot e_k + q_1 \cdot e_{k-1} + q_2 \cdot e_{k-2} \tag{115}$$

**Tabla 9.** Coeficientes para controladores discretizados.

| | Euler I | Euler II | Tustin |
|---|---|---|---|
| $q_0$ | $k_p + \frac{k_d}{T_m}$ | $k_p + k_i \cdot T_m + \frac{k_d}{T_m}$ | $k_p + \frac{k_i \cdot T_m}{2} + \frac{k_d}{T_m}$ |
| $q_1$ | $-k_p + k_i \cdot T_m - 2\frac{k_d}{T_m}$ | $-k_p - 2\frac{k_d}{T_m}$ | $-k_p + \frac{k_i \cdot T_m}{2} - 2\frac{k_d}{T_m}$ |
| $q_2$ | $\frac{k_d}{T_m}$ | $\frac{k_d}{T_m}$ | $\frac{k_d}{T_m}$ |

Para seleccionar el tipo de discretización que se aplica al controlador se grafican los resultados de una prueba de seguimiento aplicada a cada uno de los controladores en un motor N20, los datos obtenidos se muestran en la Figura 36. Como se puede evidenciar, en este caso no existe mayor diferencia entre los diferentes métodos de discretización, con todos se obtienen buenos resultados, pero finalmente se opta por el método de Tustin para discretizar el controlador PID, obteniendo la ley de control dada por (116).

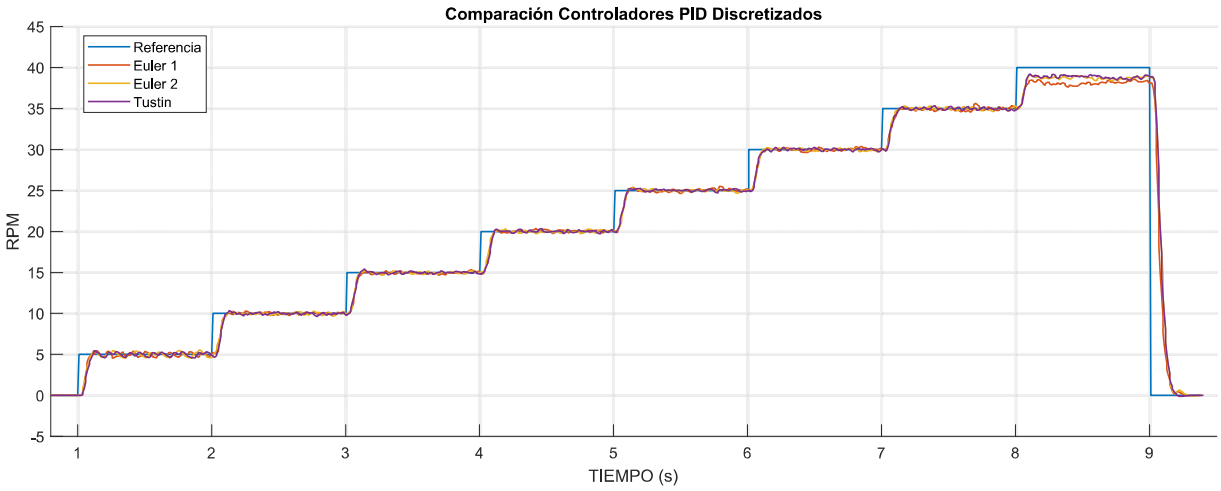



**Figura 36.** Comparación de las pruebas de seguimiento para los controladores discretizados.

$$u_k = u_{k-1} + 3{,}136 \cdot e_k - 5{,}064 \cdot e_{k-1} + 2 \cdot e_{k-2} \tag{116}$$

## 11. Diseño Electrónico

Los requerimientos para el diseño del sistema electrónico son: capacidad de procesamiento para calcular la planeación de trayectoria con evasión de obstáculos de forma local, estimación de la posición del robot por odometría, medición de distancias hacia obstáculos alrededor del robot, capacidad de controlar los actuadores y conexión Wi-Fi para recibir los pedidos que debe recolectar. Para cumplir estos requerimientos se diseñó el sistema

electrónico como se muestra en la Figura 37. Se usa un Raspberry Pi Zero como unidad de procesamiento central y un microcontrolador Atmega328P por cada motor DC de cada rueda.

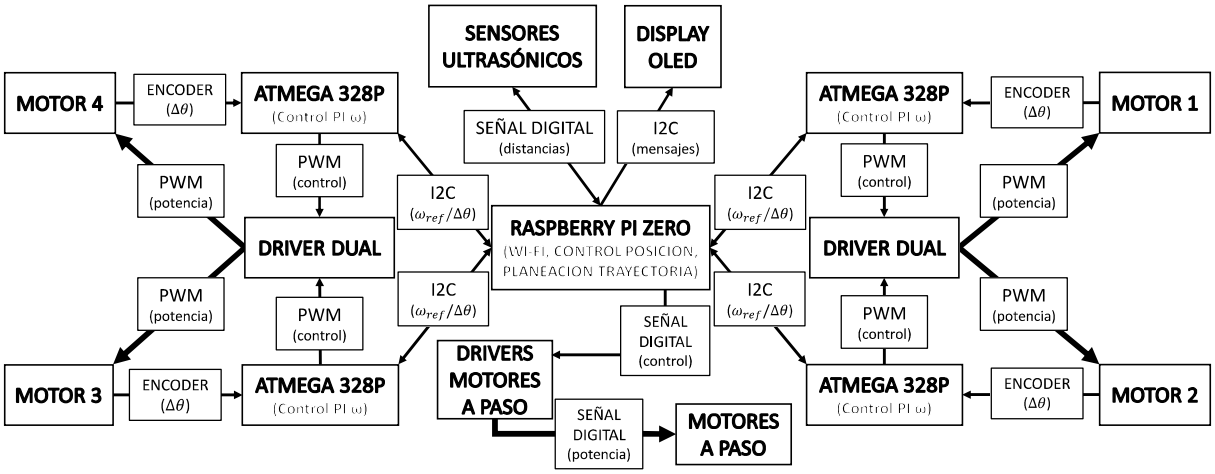



**Figura 37.** Diagrama de bloques del sistema electrónico.

El Raspberry Pi Zero se encarga principalmente de alojar la página web donde se realizan los pedidos, de estimar la posición del robot por odometría y de calcular las trayectorias del robot entre los obstáculos fijos como estantes o paredes, además se encarga de evitar colisiones del robot contra obstáculos dinámicos. Para esto se implementó un arreglo de 14 sensores ultrasónicos HC-SR04 distribuidos alrededor del robot, con los cuales el Raspberry Pi puede estimar las distancias a las cuales se encuentran los obstáculos. Por otro lado, para que el robot pueda seguir una trayectoria se necesita controlar la velocidad de giro de las ruedas, por lo que el Raspberry Pi se comunica mediante el protocolo I2C a cada uno de los microcontroladores Atmega328P, donde el microprocesador envía la velocidad de giro requerida, y los microcontroladores retornan el cambio de giro de cada llanta que se utiliza para calcular la posición del robot mediante odometría.

Cada motor N20 tiene un encoder de efecto *Hall* unido a su eje, por lo que se utilizan los microcontroladores para detectar las señales digitales de los encoders, y poder estimar el giro de cada llanta, a su vez se estima y controla la velocidad angular de cada motor mediante un controlador PID. El controlador PID programado en cada microcontrolador tiene como salida una señal PWM de control que es enviada a un driver MC33926, el cual amplifica la corriente para manejar las cargas requeridas por los motores N20, debido a que este

driver es dual, se puede utilizar un módulo por cada dos motores.

El mecanismo de recolección de productos tiene dos motores a pasos y cuatro fines de carrera para determinar los límites de movimiento del mecanismo. Por lo tanto, con el Raspberry Pi se envían las señales de control a dos drivers ULN2003 que a su vez controlan los dos motores a pasos. También se utiliza un display OLED de 0.96 pulgadas con comunicación I2C como un método de comunicación con los usuarios donde se puedan desplegar mensajes.

### 11.1. Alimentación y Regulación de Voltajes

Como se muestra en la Figura 38, la fuente de alimentación para el robot es una batería de Ni-MH de $12\ \mathrm{V}$ y $3\ \mathrm{Ah}$, a la cual se le conecta un botón de encendido y un diodo 1N4007 para protección en caso de polarizar el circuito de forma inversa. Los motores N20 trabajan a un voltaje nominal de $12\ \mathrm{V}$, por lo que sin ninguna regulación de voltaje se conectan los drivers MC33926 a $+12\ \mathrm{V}$. Sin embargo, el Raspberry Pi y los Atmega328P, deben ser alimentados con $5\ \mathrm{V}$, y algunos otros elementos como el display OLED necesitan una alimentación de $3{,}3\ \mathrm{V}$. Por lo que se utilizan los integrados LM7805 y LD1086V33 para regular el voltaje a $5\ \mathrm{V}$ y $3{,}3\ \mathrm{V}$, respectivamente, utilizando capacitores tanto a la entrada como a la salida de los reguladores para mejorar la estabilidad de los voltajes. Además, se implementó un circuito separado de la regulación del resto del circuito para energizar el Raspberry Pi Zero debido a que este microprocesador es sensible a pequeñas caídas de voltaje que no permiten arrancar el sistema operativo. Como se mencionó, el Raspberry Pi Zero puede ser alimentado directamente al pin de $5\ \mathrm{V}$, pero esta placa no tiene protección en ese pin, por lo que cualquier sobre-corriente o polarización inversa dañarán la placa. Por esta razón, se conecta un fusible de $1\ \mathrm{A}$ en serie a la alimentación del microprocesador.

### 11.2. Unidad de Procesamiento Central

Aunque el Raspberry Pi Zero debe ser alimentado con $5\ \mathrm{V}$, todos los pines de comunicación de esta placa operan a $3{,}3\ \mathrm{V}$, por esta razón el display OLED puede ser conectado directamente al bus I2C del Raspberry Pi, como se muestra en la Figura 39. También los *drivers* ULN2003 para controlar los motores a paso pueden recibir señales a $3{,}3\ \mathrm{V}$, y los fi-

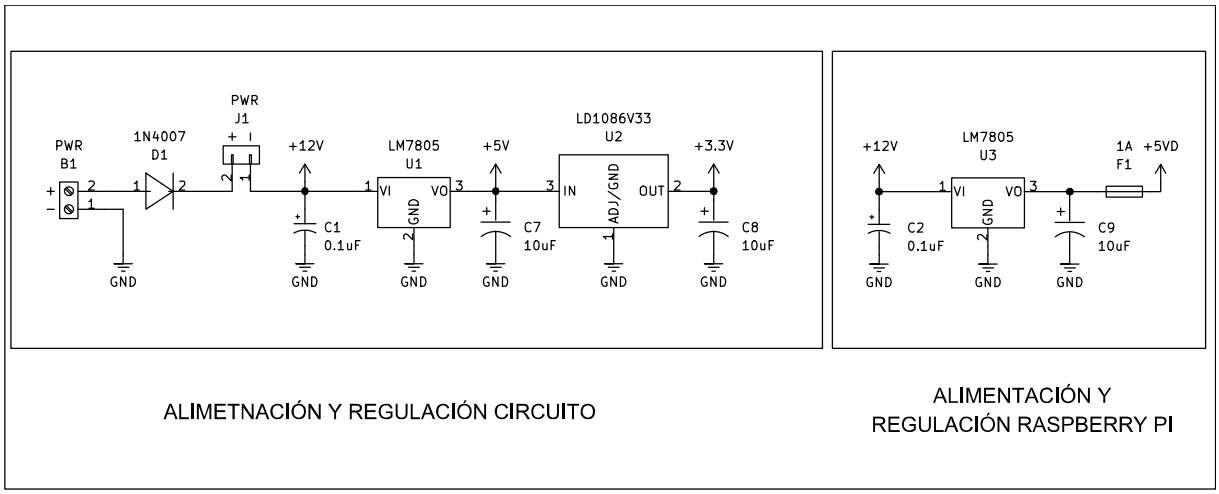



**Figura 38.** Diagrama del circuito: alimentación y regulación de voltajes.

nes de carrera del mecanismo de recolección de productos son conectados directamente al microprocesador, ya que se puede programar por software la activación de resistencias *pull-down* para asegurar el nivel lógico de estos pines. Sin embargo, algunos elementos operan a $5\ \mathrm{V}$ y no es recomendable conectarlos directamente al Raspberry Pi por seguridad de la placa. Por esta razón, se utilizan dos módulos conversores de nivel lógico bidireccional de cuatro canales, donde dos canales se usan para los sensores ultrasónicos (Trigger, Echo), cuatro canales para la selección de canal en el multiplexor el cual trabaja a $5\ \mathrm{V}$ y dos canales para la comunicación I2C (SCL, SDA), donde se agregan resistencias *pull-down* para asegurar el estado lógico del bus.

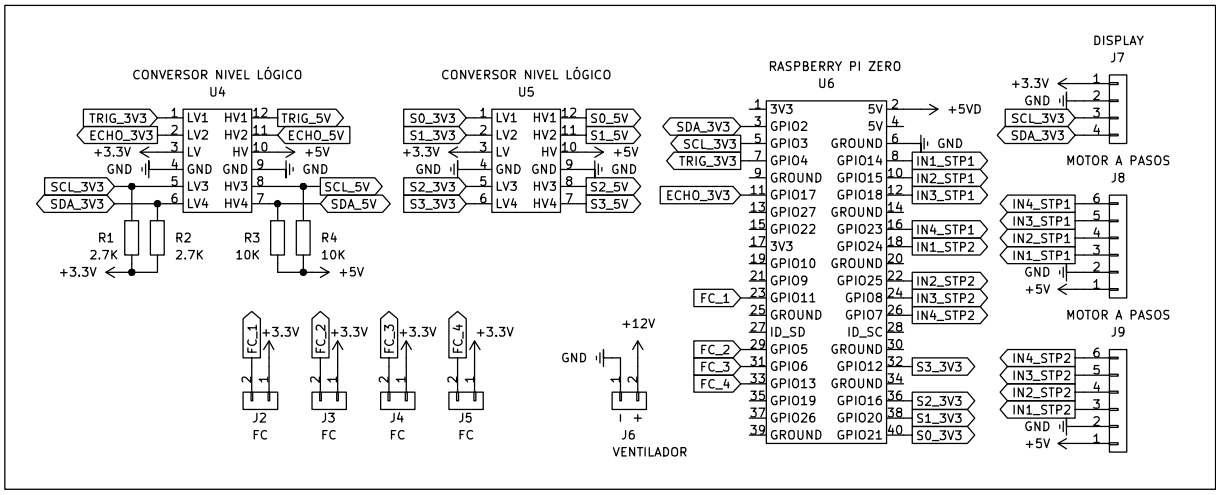



**Figura 39.** Diagrama del circuito: microprocesador, display, motores a pasos.

## 11.3. Sensores

Cada sensor ultrasónico tiene dos canales de comunicación, uno que recibe la señal para iniciar la emisión de la onda ultrasónica (Trigger) y otro para la señal recibida cuando la onda ultrasónica retorna al sensor (Echo). Si se conecta cada uno de estos canales a un pin del Raspberry Pi esto implicaría que 28 pines del microprocesador estén ocupados. Sin embargo, es posible reducir significativamente el número de pines utilizados; primero todos los canales Trigger de los sensores son conectados a la misma señal del Raspberry Pi, y segundo, se utiliza un módulo multiplexor de 16 canales basado en el integrado CD74HC4067 para acceder a un canal específico a la vez, que corresponda al sensor del cual se quiere obtener la distancia. Por lo tanto, se reduce el número de pines a 6, 2 para Trigger y Echo común para todos los sensores, y 4 para la selección del canal en el multiplexor, como se muestra en la Figura 40.

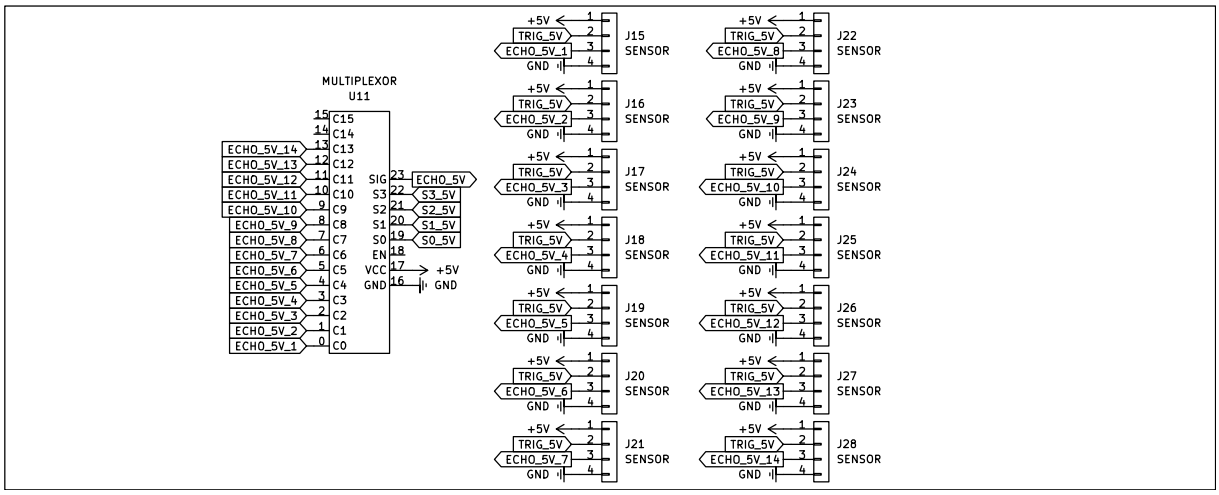



**Figura 40.** Diagrama del circuito: multiplexor y sensores ultrasónicos.

## 11.4. Microcontroladores

Como se mencionó, cada motor es controlado por un microcontrolador Atmega328P, y la razón principal por la que se ha seleccionado este elemento es por su facilidad de uso, ya que las placas Arduino Uno están basadas en estos microcontroladores, por lo que no solo se puede utilizar el Arduino IDE para programar estos elementos, sino que también se puede hacer uso de las vastas librerías disponibles para las placas Arduino. Los microcontroladores tienen un circuito básico de funcionamiento que se muestra en la Figura

41 (para más información revisar el Anexo D), con su respectivo oscilador de cristal de $16\ \mathrm{MHz}$ y sus capacitores, tanto para el oscilador como para la alimentación [40]. Además, se diseñó el circuito para que cada microcontrolador tenga conectores disponibles a los canales de comunicación serial, para que puedan ser programados fácilmente. Se incluyó un LED al pin 13 (en la numeración de correspondiente a la placa Arduino) para poder supervisar el correcto funcionamiento del microcontrolador. Todos los microcontroladores están conectados a una señal común de reseteo, que se utiliza tanto para cargar nuevos programas, como para reiniciar los microcontroladores en caso de ser necesario.

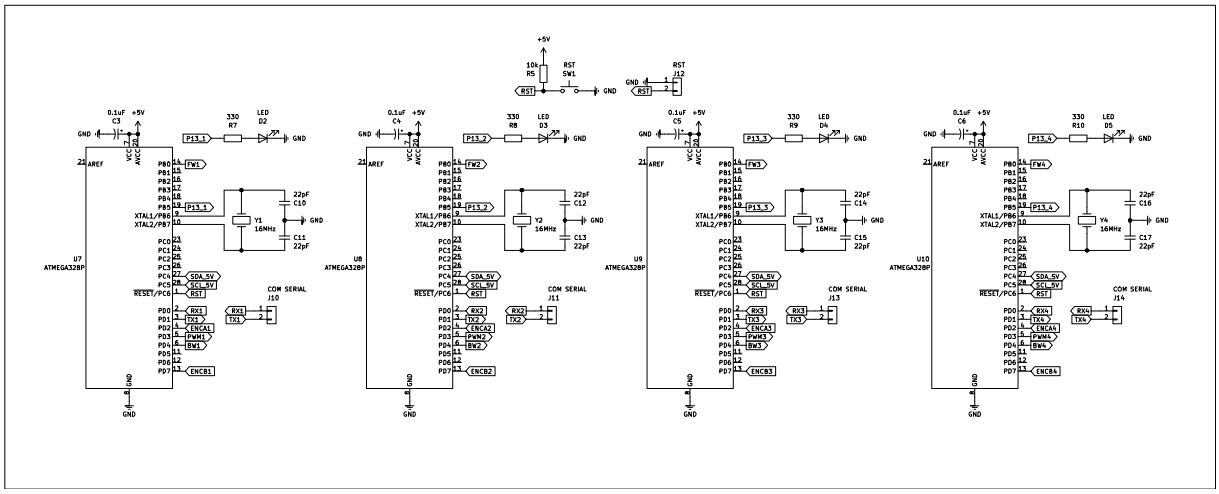



**Figura 41.** Diagrama del circuito: microcontroladores.

Los encoders emiten dos señales digitales desfasadas $90°$ que permiten, no solo medir cuánto ha girado el eje del motor, pero debido al desfase, también se puede determinar el sentido de giro del motor dependiendo de cual señal se mida primero. Para detectar las señales de forma eficiente se utilizan interrupciones por flanco ascendente en el pin ENCA y dentro de la interrupción se mide el nivel lógico de ENCB para determinar el sentido de giro del motor. También, dos pines son utilizados para controlar la polaridad, y, por tanto, el sentido de giro de los motores mediante los *drivers* MC33926, estos pines están denominados como FW y BW.

Para controlar la velocidad de los motores N20 los microcontroladores generan una señal PWM con ciclo de trabajo variable, pero sin modificaciones la frecuencia de esta señal generada por los Atmega328P es de $490\ \mathrm{Hz}$, por lo que se encuentra dentro del rango de frecuencias audibles para el ser humano, en otras palabras, al trabajar con frecuencias

PWM dentro de los $20$ Hz-$20$ KHz para controlar un motor DC existirá un sonido que es percibido por las personas. Para evitarlo se utiliza el Timer2 del microcontrolador, donde se configura el modo de corrección de fase PWM (*Phase Correct PWM Mode*), que permite establecer el registro OC2A (PB3) como tope del contador y así, junto con la definición del preescalador, definir la frecuencia de la señal PWM en el registro OC2B (PD3) según (117) [41]. Donde $f_{CLK} = 16$ MHz es la frecuencia del reloj, $N = 8$ es el preescalador y $OC2A = 51$. Se debe mencionar que los *drivers* MC33926 pueden trabajar adecuadamente a una frecuencia máxima de $20$ KHz, por lo que se configura el microcontrolador para tener una frecuencia PWM de $19{,}6$ KHz, que es la mayor frecuencia posible que se puede utilizar para un correcto funcionamiento de los drivers.

$$f_{PWM} = \frac{f_{CLK}}{2 \cdot N \cdot OC2A} = 19{,}6 \text{ KHz} \tag{117}$$

Para calcular la velocidad de giro de los motores N20 mediante detección de las señales de los encoders de efecto *Hall* se utilizan interrupciones por flanco de subida y el Timer1 de los microcontroladores. Para esto se configura las interrupciones por flanco de subida en el pin ENCA respectivo de cada microcontrolador, donde al activarse la interrupción se estimará la velocidad del motor según (118), donde $rev$ es la fracción de revolución que gira la rueda del robot por cada señal del encoder, que se obtiene mediante (119), donde $GR_{N20} = 297{,}924$ es la reducción mecánica de los motores N20, $SPR = 3$ es el número de señales por cada revolución del motor N20 previo a la reducción mecánica [32] y $GR_{eng} = 2$ es la reducción mecánica por la caja reductora. Además $t$ es el tiempo entre señales del enconder. Para obtener $t$ se utiliza el Timer1, donde se configura un preescalador $N = 1$ y se resetea el contador TCNT1 en cada interrupción, de esta forma se puede calcular $t$ mediante (120), además se configura un contador de desbordamientos del contador TCNT1 ($Of_c$), es decir, para cuando el contador sea mayor a $2^{16} = 65536$, ya que el Timer1 es de 16 bits.

$$RPM = \frac{rev}{t} \cdot 60 \tag{118}$$

$$rev = \frac{1}{GR_{N20} \cdot SPR \cdot GR_{eng}} = 559{,}427 \times 10^{-6} \tag{119}$$

$$t = \frac{f_{CLK}}{N \cdot (Of_c \cdot 2^{16} + TCNT1 + 1)} \tag{120}$$

Adicionalmente, se utiliza el Timer0 de los Atmega328P para generar interrupciones periódicas y medir las velocidades de giro de los motores (variable de control) con un periodo de muestreo $T_m = 2 \text{ ms}$, o lo que es igual a una frecuencia de muestreo de $f_{muestreo} = 500 \text{ Hz}$. Para esto se configura el Timer0 en modo CTC (*Clear Timer on Compare Match*) por medio del registro WGM01. Este modo, como su nombre lo dice, resetea el contador TCNT0 cuando el contador iguala al registro OCR0A (PD6) y se puede activar una interrupción cada que el contador es reseteado. Por lo tanto, se puede generar interrupciones periódicas a una frecuencia $f_{muestro}$ mediante (121), donde el preescalador se configura para $N = 256$ y el registro $OCR0A = 124$.

$$f_{muestreo} = \frac{f_{CLK}}{N \cdot (1 + OCR0A)} = 500 \text{ Hz} \tag{121}$$

La razón por la que se utiliza un microcontrolador por cada motor es porque para cada motor N20 se necesitan tres Timers; uno para establecer la frecuencia de la señal PWM que controla la velocidad de los motores N20; otro para determinar el tiempo que se demora en girar las ruedas entre señales de los encoders; y el último para establecer interrupciones para el muestreo a una frecuencia determinada. Pero el Atmega328P tiene tres Timers y por esta razón no se puede controlar dos o más motores con un solo microcontrolador con la funcionalidad planteada.

## 11.5. Motores y *Drivers*

En la Figura 42 se muestra la conexión de los motores N20 a los drivers duales MC33926 (para más información visualizar el Anexo C). Las señales de los encoders se diseñan con resistencias *pull-up* por recomendación de su hoja técnica [32] y en ambos drivers se hace uso de la capacidad de tener una señal (EN) para deshabilitar el funcionamiento de los cuatro motores por medio de un pulsador externo.

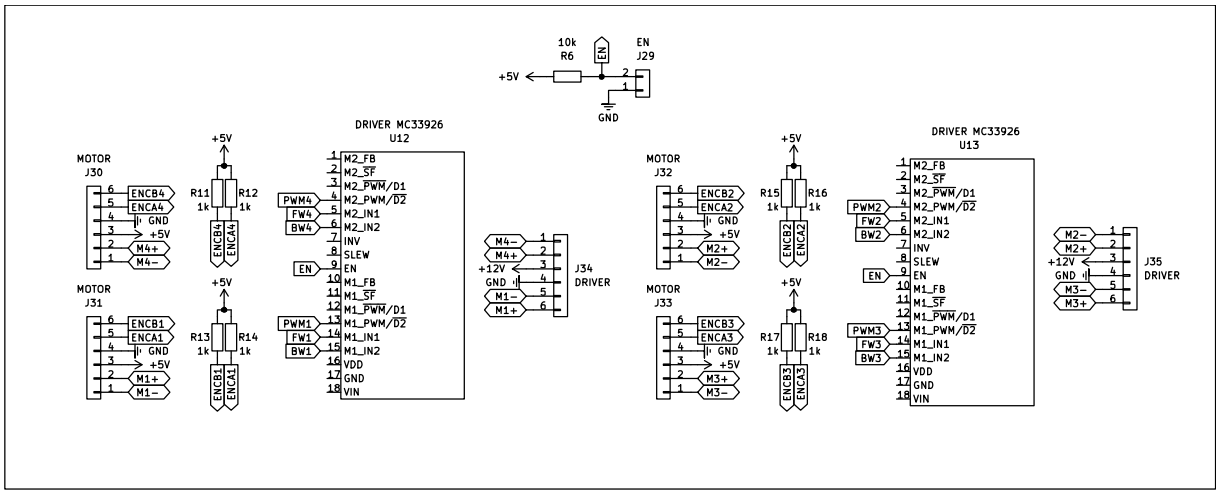



**Figura 42.** Diagrama del circuito: motores N20 y drivers MC33926.

## 12. Desarrollo de la Programación

### 12.1. Página Web y Programa Principal

En la Figura 43 se muestra un diagrama de flujo del programa principal, donde se ejecuta la página web del mercado en línea para el proyecto. El programa principal ejecuta un bucle infinito, que al recibir un pedido desde el mercado en línea, inicia el programa principal del robot para retirar todos los productos pedidos.

La página web se desarrolló en el *micro webframework* Flask [42], que está escrito en Python, para tener la facilidad de acoplar la página web con el programa del robot que se ejecuta en el Raspberry Pi. El mercado en línea tiene tres páginas principales, que se muestran en la Figura 44; la primera es la página principal del mercado en línea, donde en la parte superior tiene los accesos para el mercado y para el carrito de compras para generar el pedido; la segunda página es la del mercado, donde se muestra la información de todos los productos disponibles y los botones para agregar los productos al pedido; y la tercera página principal es el carrito de compras, donde se despliegan todos los productos agregados a la orden y el precio total del pedido, en esta página al confirmar el pedido se envía el comando para comenzar el movimiento del robot hacia todos los productos en el pedido.

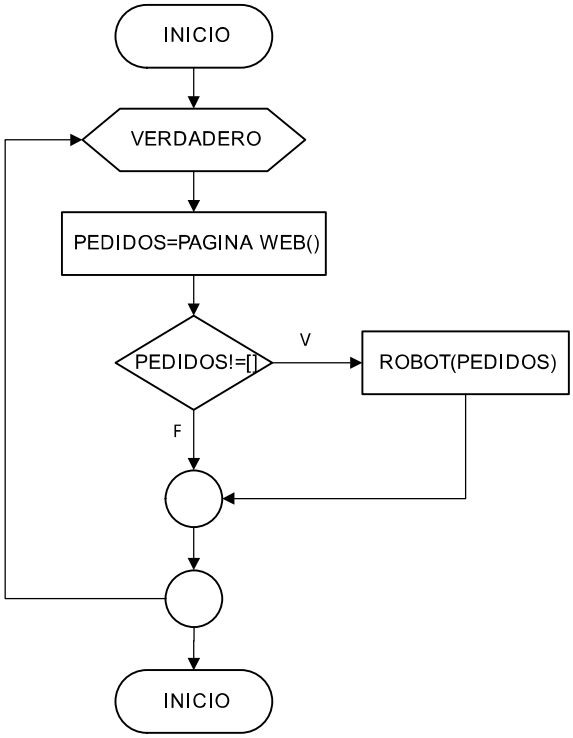
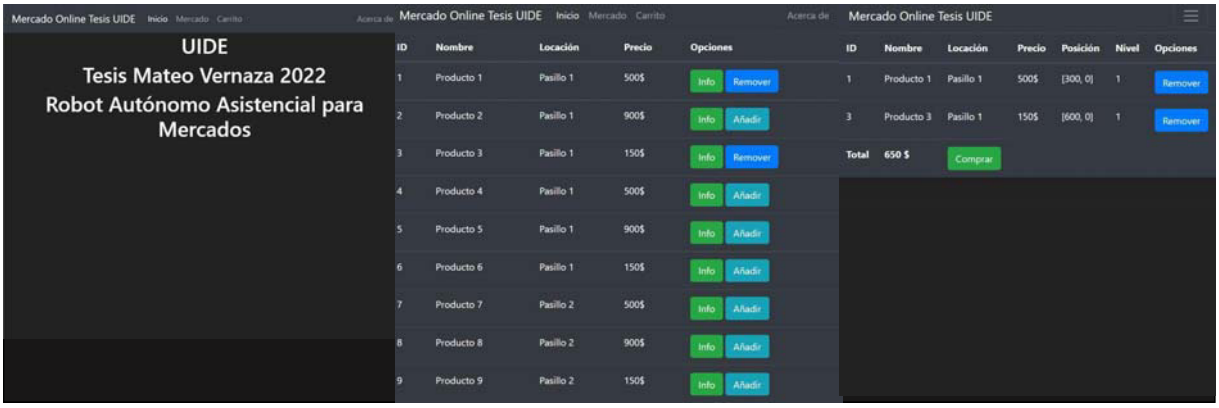



**Figura 43.** Diagrama de flujo del programa principal.

## 12.2. Programa de Control del Robot

En la Figura 45 se muestra un diagrama de flujo del programa principal de control para el robot, que recibe como entrada la orden de pedidos. El flujo de procesos comienza con el cálculo del grafo de visibilidad a partir de los obstáculos que se encuentran en el entorno. Luego, se toma los productos pedidos como las metas en el entorno y se genera un nuevo grafo agregando las metas al grafo de visibilidad.

A partir del nuevo grafo se calcula la trayectoria que el robot debe seguir para que pase por todas las metas del pedido, como se presentó en la Sección 10.1. Entonces, comienza el bucle principal, donde se estima la posición del robot por odometría según el giro de cada rueda del robot, a partir de las ecuaciones desarrolladas en la Sección 10.4, y se verifica si el robot se encuentra a una distancia (umbral) cercana a la meta, si es el caso, se activa la secuencia de los motores a pasos del mecanismo de agarre de productos; después, si la meta actual es la última dentro de la trayectoria se termina el bucle principal ya que el robot habrá recolectado todos los productos, caso contrario, se continua a la siguiente meta de la trayectoria; en el caso de que el error de posición hacia la meta no este dentro del umbral




Página principal Página del mercado Página del carrito

**Figura 44.** Página web del mercado en línea.

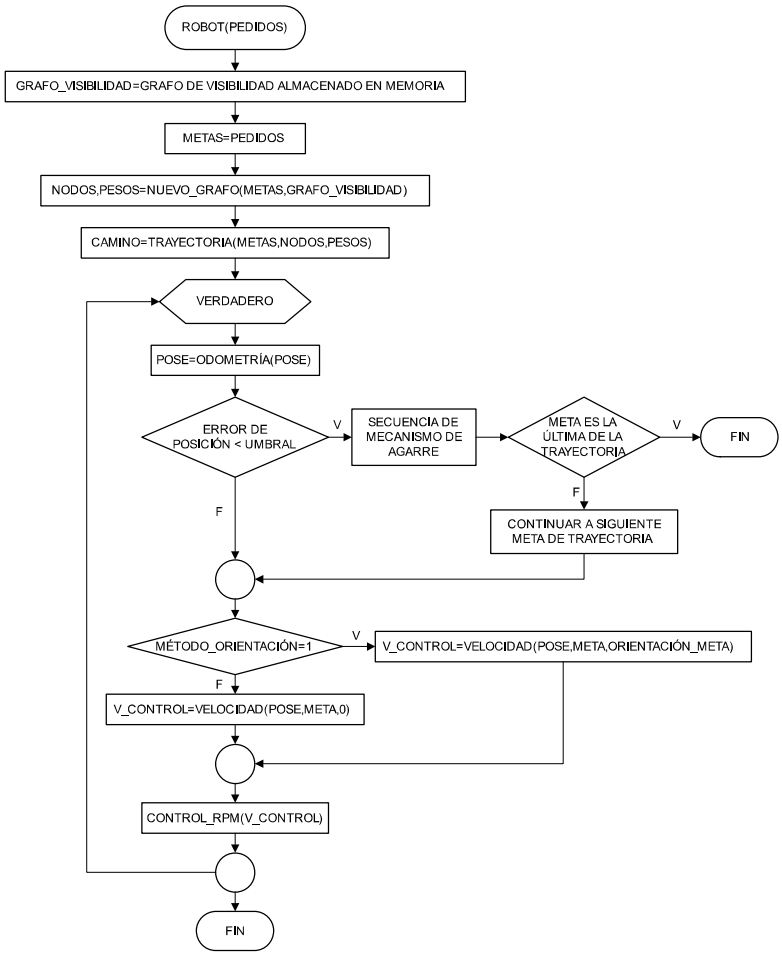



**Figura 45.** Diagrama de flujo del programa de control del robot.

se continua con el proceso.

El control del robot se diseñó con dos métodos de orientación del robot; el primer método controla el robot para ajustar la orientación del robot hacia la orientación que debe recoger

el producto mientras se desplaza hacia la meta, en otras palabras, el robot gira hacia la orientación de la meta al mismo tiempo que se desplaza linealmente; el segundo método controla el robot para que la orientación se mantenga fija mientras el robot se desplaza linealmente hacia la meta, y una vez que llegue a la posición deseada ajustará la orientación para retirar el producto. De acuerdo con el método de control de orientación seleccionado se calcula la velocidad de control del robot, según la metodología planteada en la Sección 10.2, para poder controlar la velocidad de cada motor mediante (116).

### 12.3. Programa de Generación de Grafo de Visibilidad

Como se mencionó, la metodología para generar el grafo de visibilidad se desarrolla en la Sección 10.1, y en la Figura 46 se muestra el flujo de procesos. Primero se agranda los obstáculos para considerar el radio del robot y que los nodos no estén muy cerca de los obstáculos, luego a partir de los obstáculos aumentados se generar los nodos del grafo en cada vértice de los obstáculos. Con la lista de nodos generada, se analiza la conectividad de todas las posibles permutaciones de pares de nodo mediante una función que analiza si existe colisión entre la arista del grafo que se quiere generar y los obstáculos; si existe conectividad se agrega el costo de la arista a la matriz de pesos del grafo. Finalmente, se genera el grafo de visibilidad con la lista de nodos y la matriz de pesos y se guarda en la memoria para que el programa principal del robot pueda acceder a esta información.

Para determinar si existe colisión entre una arista del grafo y los obstáculos se implementó el proceso mostrado en la Figura 47, donde esta función recibe como entrada los dos nodos que se quiere conectar mediante una arista y todos los obstáculos del entorno. Según la metodología planteada en la Sección 10.1, se expresa la arista del grafo como un vector y un punto de comienzo, luego, se analiza este vector con cada arista de cada obstáculo del entorno. En cada caso, se expresa la arista del obstáculo como un vector y un punto (como se hizo con la arista del grafo), y se aplica el criterio de Goldman [36] para determinar si existe una intersección. En el caso de existir una intersección, el algoritmo retornará un valor de $verdadero$, caso contrario, retornará $falso$.

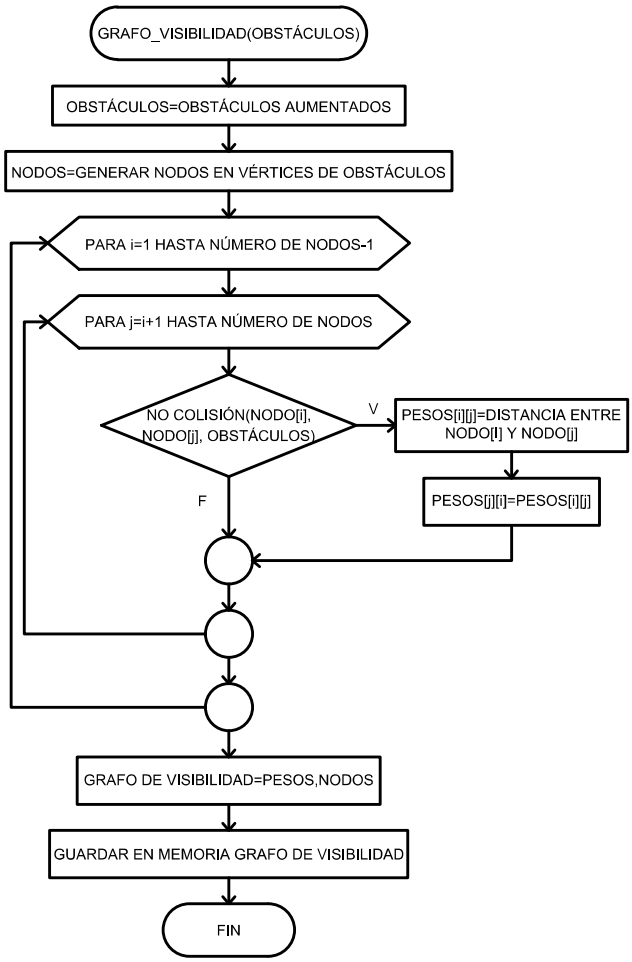



**Figura 46.** Diagrama de flujo del programa para generación del grafo de visibilidad.

## 12.4. Programa de Generación de Nuevo Grafo con Metas

El subproceso para genera un nuevo grafo, agregando las metas del pedido generado, se muestra en la Figura 48. Este subproceso es muy similar al de generación del grafo de visibilidad, con la diferencia de que el algoritmo parte del grafo de visibilidad y solo analiza los nuevos nodos agregados. Para esto, se comienza agregando las metas a la lista de nodos del grafo de visibilidad, y se analiza la conectividad de los nodos agregados.

## 12.5. Programa de Generación de la Trayectoria

En la Figura 49 se muestra el subproceso para calcular la trayectoria que pase por todas las metas. El algoritmo comienza por crear una lista de metas que no han sido agregadas

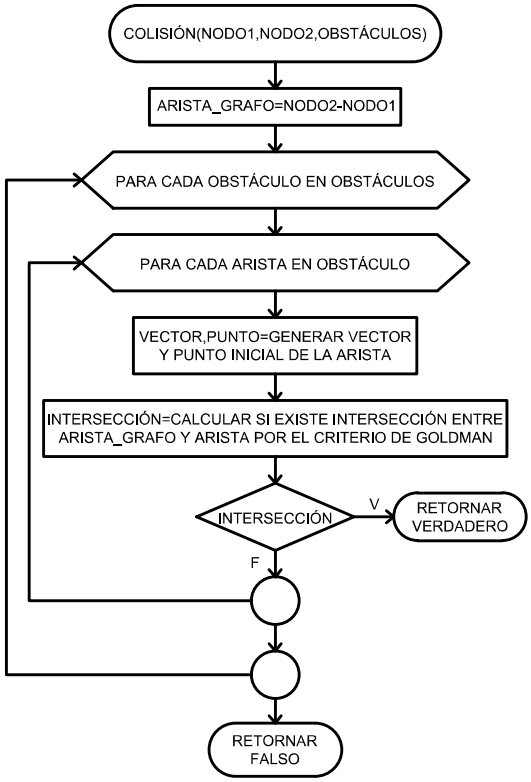



**Figura 47.** Diagrama de flujo del programa para detección de colisión.

a la trayectoria e itera el proceso hasta que esta lista este vacía. En cada iteración se inicializa una variable del costo mínimo encontrado para ir a una meta con un valor infinito. Entonces, se itera por cada meta, calculando un camino para ir desde el nodo inicial a la meta, mediante el algoritmo A* (Algoritmo 5), y almacenando el costo de este camino. Luego se analiza si el costo del camino calculado es menor al mínimo costo almacenado, en caso de que se cumpla, se reemplaza el costo, la meta y el camino como el mejor hasta el momento. Si se realiza este proceso para todas las metas se tendrá al final del bucle la meta más cercana al nodo inicial, entonces se agrega el camino de esta meta a la trayectoria; luego se remueve la meta de la lista de metas no revisadas y se cambia el nodo inicial por esta meta. De esta forma en la siguiente iteración se encontrará la siguiente meta mas cercana y así sucesivamente, hasta que la lista de metas no revisadas este vacía, para

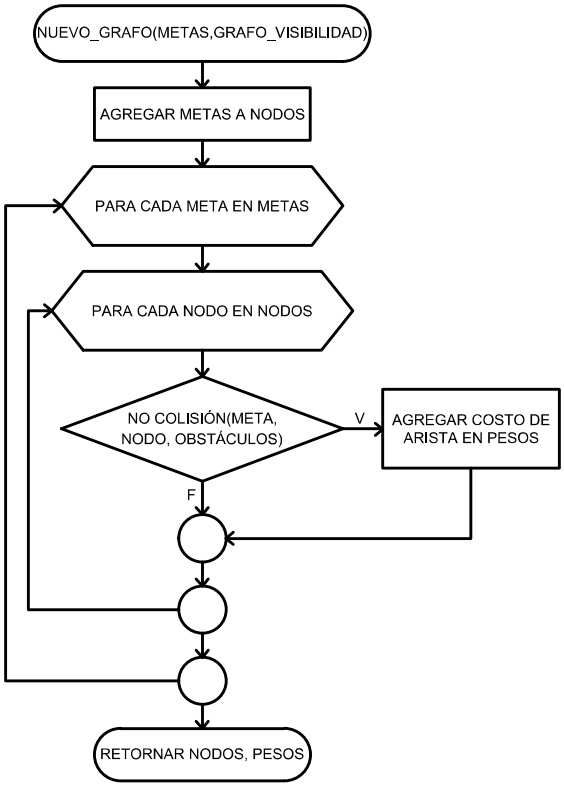



**Figura 48.** Diagrama de flujo del programa para generación del nuevo grafo con las metas.

finalmente retornar la trayectoria almacenada en la variable $camino$.

## 12.6. Programa de Actualización de Posición por Odometría

El diagrama de flujo mostrado en la Figura 50 presenta el proceso para actualizar la posición por odometría. El algoritmo primero obtiene las mediciones de cuánto ha girado cada llanta, mediante comunicación por el protocolo I2C, con cada uno de los microcontroladores de cada llanta. Con esta información se estima la posición del robot mediante (113), y se retorna la nueva posición del robot calculada.

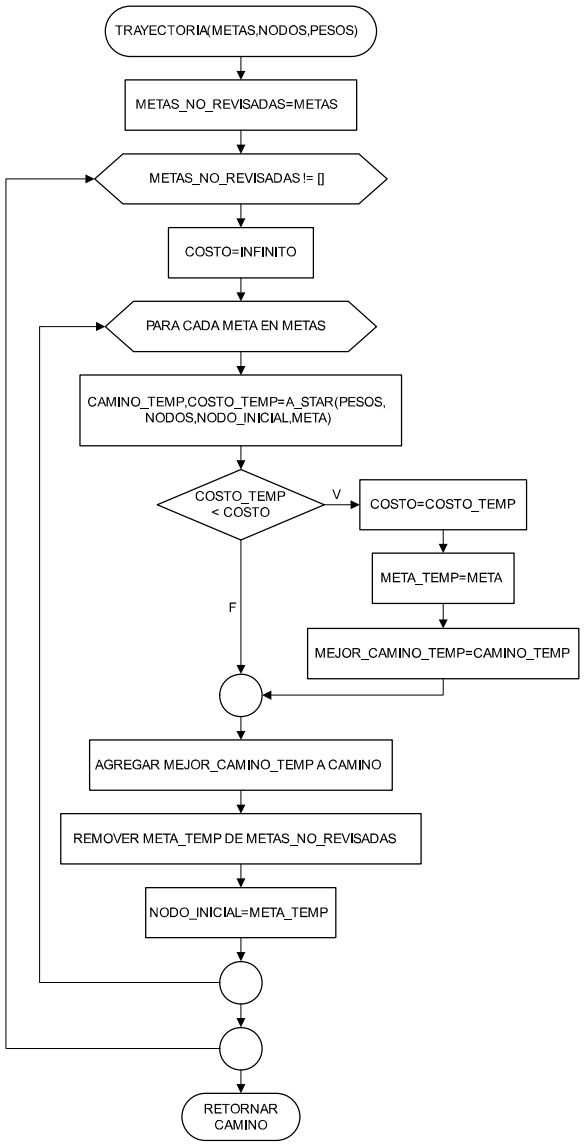



**Figura 49.** Diagrama de flujo del programa para generación de la trayectoria del robot.

## 12.7. Programa para Calcular la Velocidad de Control y Control de Velocidades de los Motores

En la Figura 51 se muestra el diagrama de flujo para el algoritmo que calcula la velocidad de control a partir del concepto de campos potenciales virtuales, desarrollado en la Sección 10.2. El proceso comienza calculando la velocidad de atracción mediante (76). Después se obtienen las mediciones de los 14 sensores ultrasónicos, accediendo individualmente a cada uno de ellos por el multiplexor como se explicó en la Sección 11.3. A los valores de distancia obtenidos se les aplica un filtro EMA (*Exponential Moving Average*) para reducir el ruido en las mediciones de los sensores, el filtro se aplica según (122) [43], donde $\alpha$

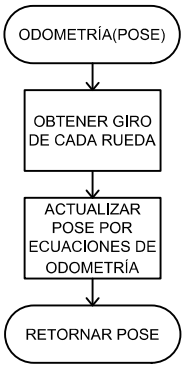



**Figura 50.** Diagrama de flujo del programa para actualizar la posición de robot por odometría.

es la constante de suavizado. Con las mediciones suavizadas se calcula la velocidad de repulsión para cada sensor mediante (78), y se obtiene la velocidad de control resultante mediante (79). Finalmente, se evalúa que la velocidad de control calculada no sea mayor a la velocidad máxima del robot para ser retornada.

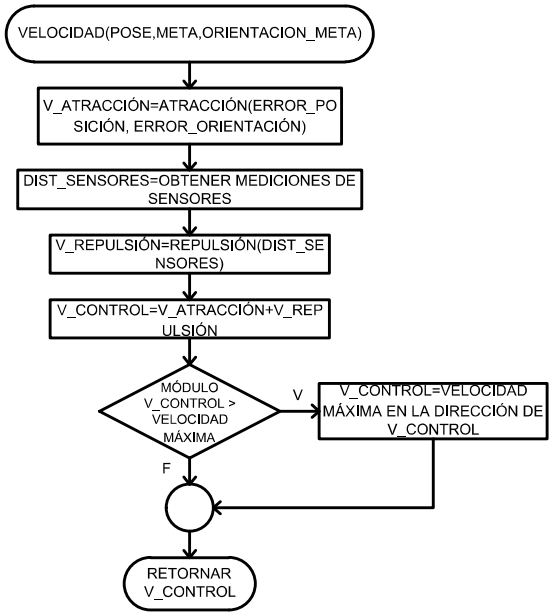



**Figura 51.** Diagrama de flujo del programa para calcular la velocidad de control.

$$D^{k}_{filtrado} = \alpha \cdot D_{sensor} + (1 - \alpha) \cdot D^{k-1}_{filtrado} \tag{122}$$

Finalmente, en la Figura 52 se muestra el subproceso para controlar las velocidades angulares de los motores para generar un desplazamiento en el robot de acuerdo con la

velocidad de control. Para esto el algoritmo recibe la velocidad de control y la transforma a velocidades angulares específicas para cada rueda según el modelo cinemático (99). Posteriormente, cada velocidad angular (en RPM) es enviada por el protocolo I2C al respectivo microcontrolador que ejecuta un controlador PID como se mencionó en la Sección 10.5.

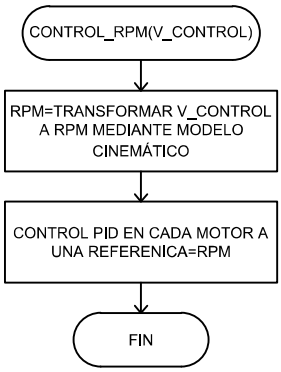



**Figura 52.** Diagrama de flujo del programa para controlar la velocidad de los motores.

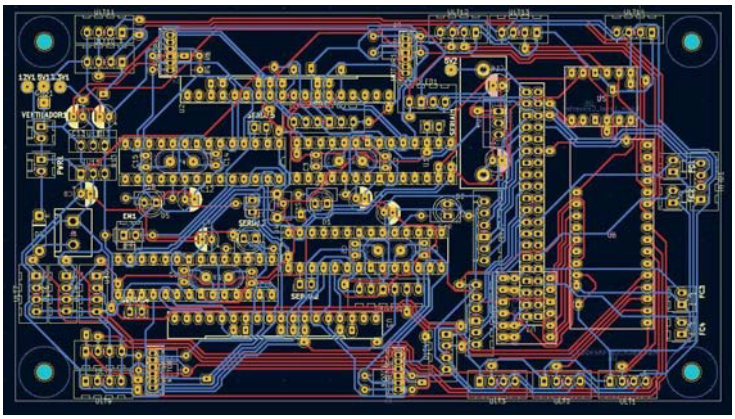
## 13. Construcción y Desarrollo del Prototipo

El prototipo del robot se desarrolló con un enfoque en el prototipado rápido para facilitar la iteración del proceso de diseñar, probar y mejorar los diferentes aspectos del sistema. Debido a esto, en el área electrónica, se diseñó el PCB en el software KiCad y se manufacturó la placa en una CNC de escritorio. El software FlatCAM se utilizó para crear el código G para el corte y la CNC se operó mediante el programa Candle. En la Figura 53, se muestra en la parte superior el diseño del PCB en KiCad, donde se observa que es una placa a doble lado, y en la parte inferior se muestra el circuito terminado. El método de fabricación de PCB mediante mecanizado es útil para prototipado rápido ya que permite la creación de placas de uno o doble lado con mayor precisión y con menor tiempo de supervisión que otros métodos utilizados para prototipado.

Para las piezas mecánicas se optó en su mayoría por elementos disponibles en el mercado local y por manufactura aditiva para la fabricación de piezas. Por lo tanto, el diseño de las piezas se realizó tomando en cuenta las mejores posiciones de impresión y evitando la utilización de soportes para tener una superficie con menor rugosidad. En la Figura 54, se muestra el prototipo construido.

Como se mencionó, debido al enfoque de prototipado rápido en todos los aspectos del

a) Diseño de PCB en KiCad



b) PCB terminado

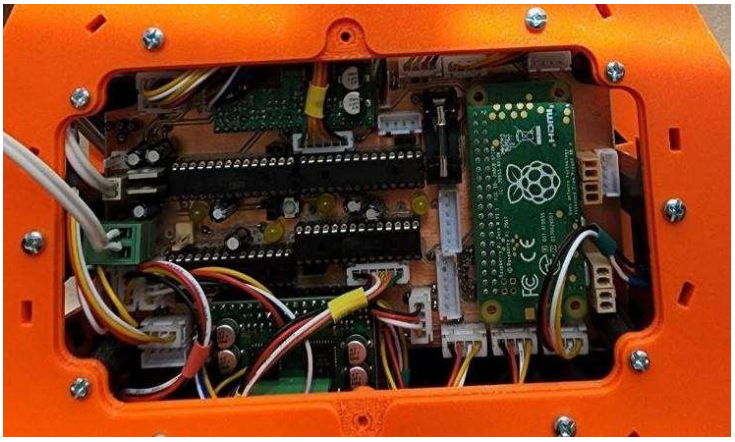


**Figura 53.** Diseño y construcción del PCB.

sistema, se programó un sistema de simulación gemelo al robot. El algoritmo se desarrolló en MATLAB y se utilizó para probar todos los elementos de control del algoritmo del prototipo antes de construirlo. Esta metodología es de gran ayuda, ya que se pudieron probar muchos aspectos como el número de sensores ultrasónicos y su posicionamiento; el algoritmo de generación de trayectorias; el algoritmo de campos potenciales virtuales; y también permite probar el funcionamiento del algoritmo del prototipo en escenarios mucho más grandes de los disponibles físicamente (por las restricciones de espacio). En la Figura 55 se muestra una simulación del prototipo, donde las figuras grises representan obstáculos; los puntos rojos son los nodos del grafo; las líneas azules representan las aristas del grafo; y el círculo negro con líneas rojas es el robot con los sensores ultrasónicos.

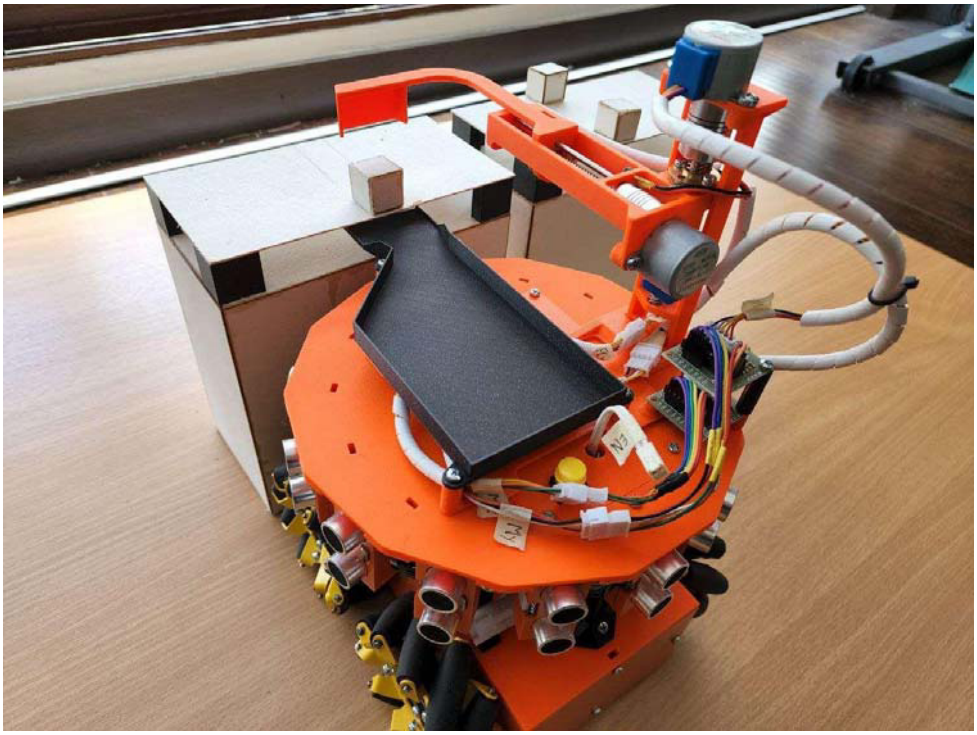


**Figura 54.** Prototipo construido del robot.

## 14. Costos

En la Tabla 10 se muestran los costos de los sistemas del prototipo, donde en la parte mecánica se incluyen los costos de motores, ruedas, ejes, pernos, entre otros; en los elementos electrónicos se incluyen el microcontrolador, microprocesadores, drivers, sensores, entre otros; y finalmente se incluye el costo estimado de impresión 3D para las piezas considerando un costo de $8.5/hora, sin embargo, este costo se puede reducir considerablemente si se tiene acceso a una impresora 3D, donde solo se consideraría el costo del material en aproximadamente $30.

**Tabla 10.** Costos de construcción del prototipo.

| **Categoría** | **Costo** |
|---|---|
| Elementos Mecánicos | $ 304 |
| Elementos Electrónicos | $ 194 |
| *Impresión 3D (30 hrs a $ 8.5/hr)* | $ 255 |
| **Total** | $ 754 |

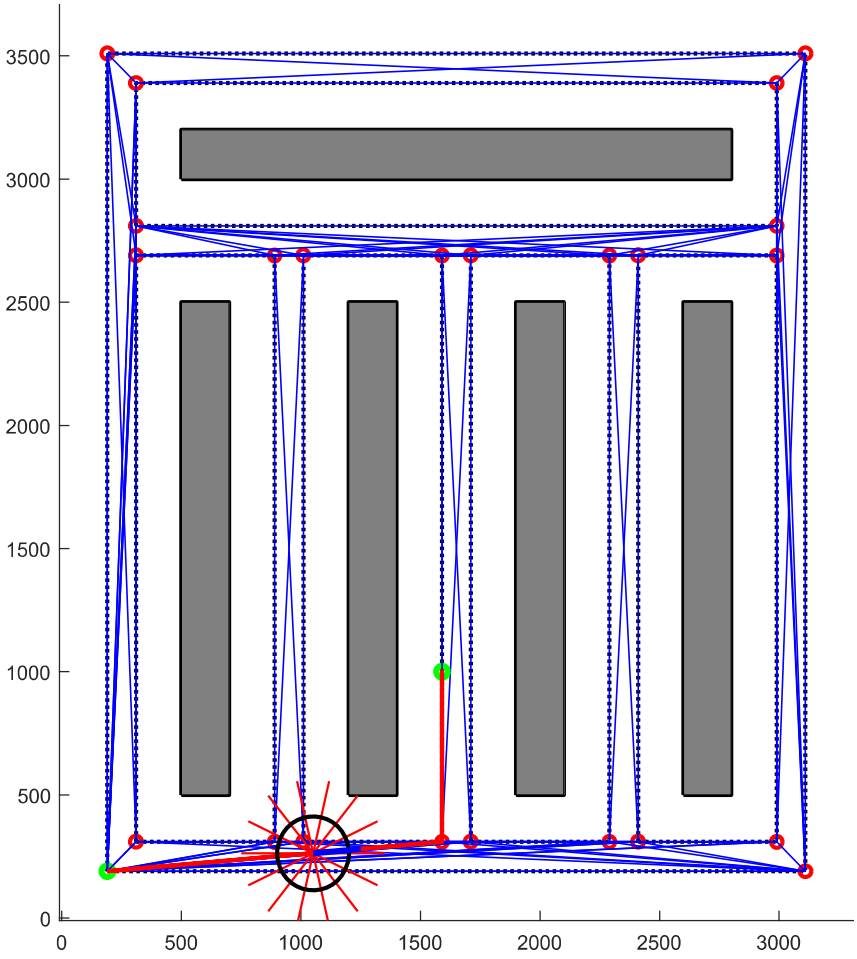



**Figura 55.** Simulación del sistema en MATLAB.

## 15. Pruebas y Resultados

Se realizaron tres pruebas para medir el error del robot en su trayectoria, en todas las pruebas se midió el error en la estimación de la pose del robot mediante odometría y el error real de la pose después de cada movimiento; en la primera prueba se comandó al robot girar sobre su propio eje en intervalos de $90°$ para determinar el error en la orientación; en la segunda prueba se comandó al robot un movimiento frontal en intervalos de $20\ \mathrm{mm}$; y en la tercera prueba se comandó al robot un movimiento lateral en intervalos de $20\ \mathrm{mm}$. Las dos últimas pruebas se realizaron para cuantificar el error en la posición del robot con diferentes tipos de movimiento.

En la Figura 56 se muestran los resultados de las pruebas mencionadas, donde se puede observar que, aunque la estimación por odometría del robot se encuentra dentro del rango de valores aceptados ($0{,}5°$ para orientación y $1\ \mathrm{mm}$ para posición), el error en la pose

real es mayor en la mayoría de las mediciones. Esto se debe a que la odometría se basa únicamente en la medición de giro de las ruedas, y cualquier factor como: deslizamiento de las ruedas, error en comunicación entre microcontrolador y microprocesadores, variaciones del modelo construido respecto al modelo ideal (modelo cinemático desarrollado), entre otros aspectos, no serán consideradas para la estimación de la pose. Por esta razón, la odometría tiene un error acumulativo, el cual provocará que el robot aumente la diferencia entre el error real y el error estimado.

Mediante los gráficos de la Figura 56 puede resultar difícil identificar cual será el error en un movimiento del robot, por lo que en la Tabla 11 se presenta una cuantificación de los movimientos probados. El error porcentual se calcula como el promedio del porcentaje del error real medido respecto al giro o desplazamiento del robot; y la relación de error se establece como el promedio del error en grados por cada $360°$ que el robot gire, o como el error de posición en $\mathrm{mm}$ por cada metro que el robot se desplace.

**Tabla 11.** Cuantificación de errores.

| Prueba | Error Porcentual | Relación Error |
|---|---|---|
| Giro | $0{,}719\,\%$ | $2{,}590\ °/360°$ |
| Frontal | $1{,}670\,\%$ | $16{,}700\ \mathrm{mm/m}$ |
| Lateral | $1{,}744\,\%$ | $17{,}441\ \mathrm{mm/m}$ |

Además, se realizaron pruebas de seguimiento de trayectoria en el robot por varios métodos de desplazamiento. Las pruebas consistieron en que el robot siga una trayectoria, dada por cuatro puntos de referencia, dos veces, para poder obtener datos de las posiciones estimadas por el robot y compararlas con las posiciones reales. En la primera prueba el prototipo se movió por la trayectoria con una orientación fija; en la segunda prueba el robot se desplazó con una orientación fija, pero en cada punto de referencia tenía una orientación deseada a la cual debía girar en su propio eje, para después regresar a la orientación original y continuar con la trayectoria; y por último, en la tercera prueba el robot giraba hacia las orientaciones deseadas de las referencias mientras se trasladaba. En la Figura 57 se muestran los resultados de las pruebas mencionadas, donde se puede observar que en todos los casos existe una desviación, debido a los errores intrínsecos de la odometría.

También se realizaron pruebas de detección y evasión de obstáculos no conocidos para comprobar la característica reactiva que otorgan los campos potenciales virtuales. Para

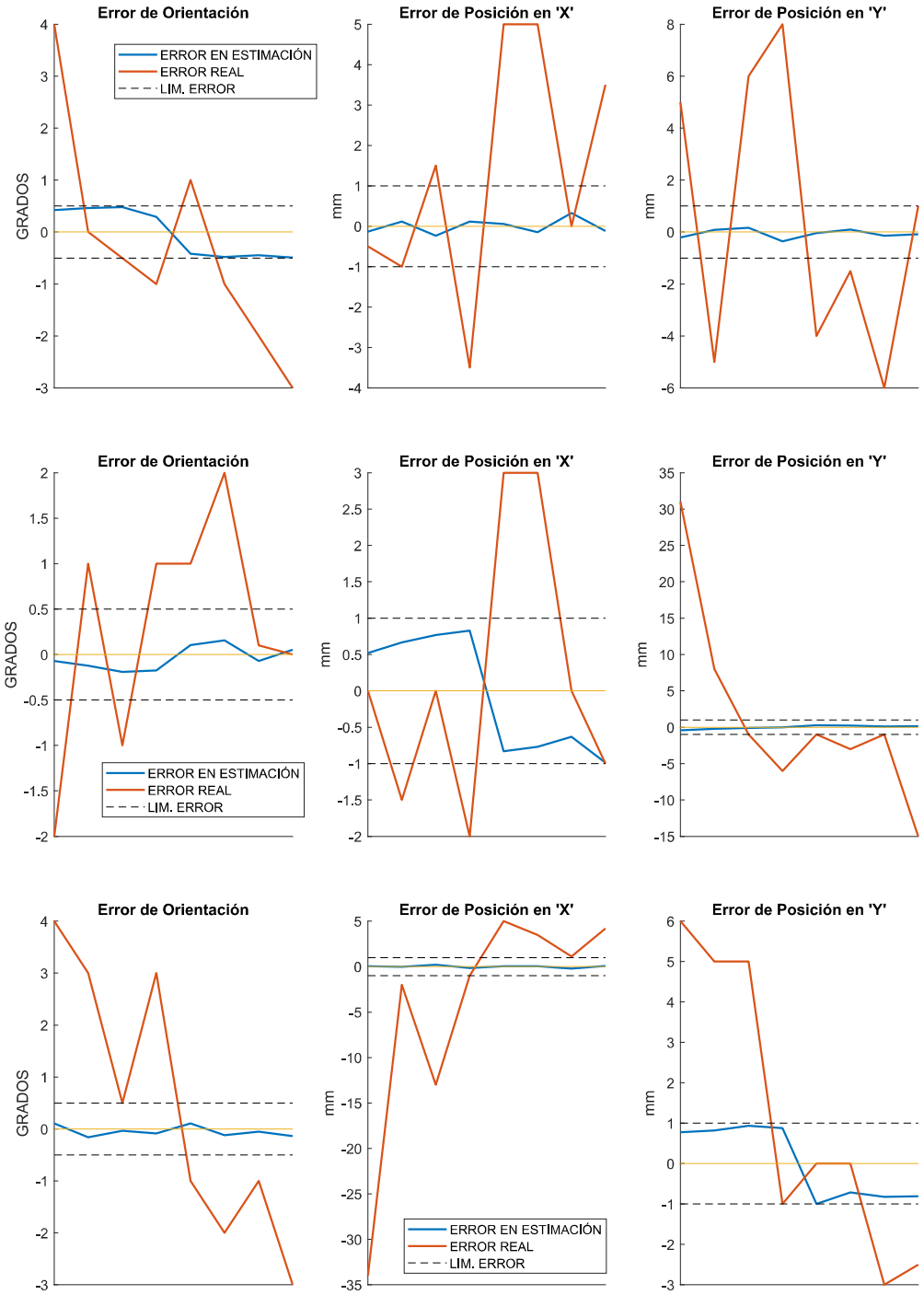



**Figura 56.** Resultados de pruebas de error en movimiento del prototipo.

esto se comandaron metas a las que el robot debía llegar en un entorno sin obstáculos conocidos, pero se introdujo un obstáculo de $20\ \mathrm{mm}$ de diámetro en el centro del entorno y se escogió arbitrariamente las metas para que la trayectoria calculada pase cerca del obstáculo, e incluso la última meta se la definió en el centro del obstáculo.

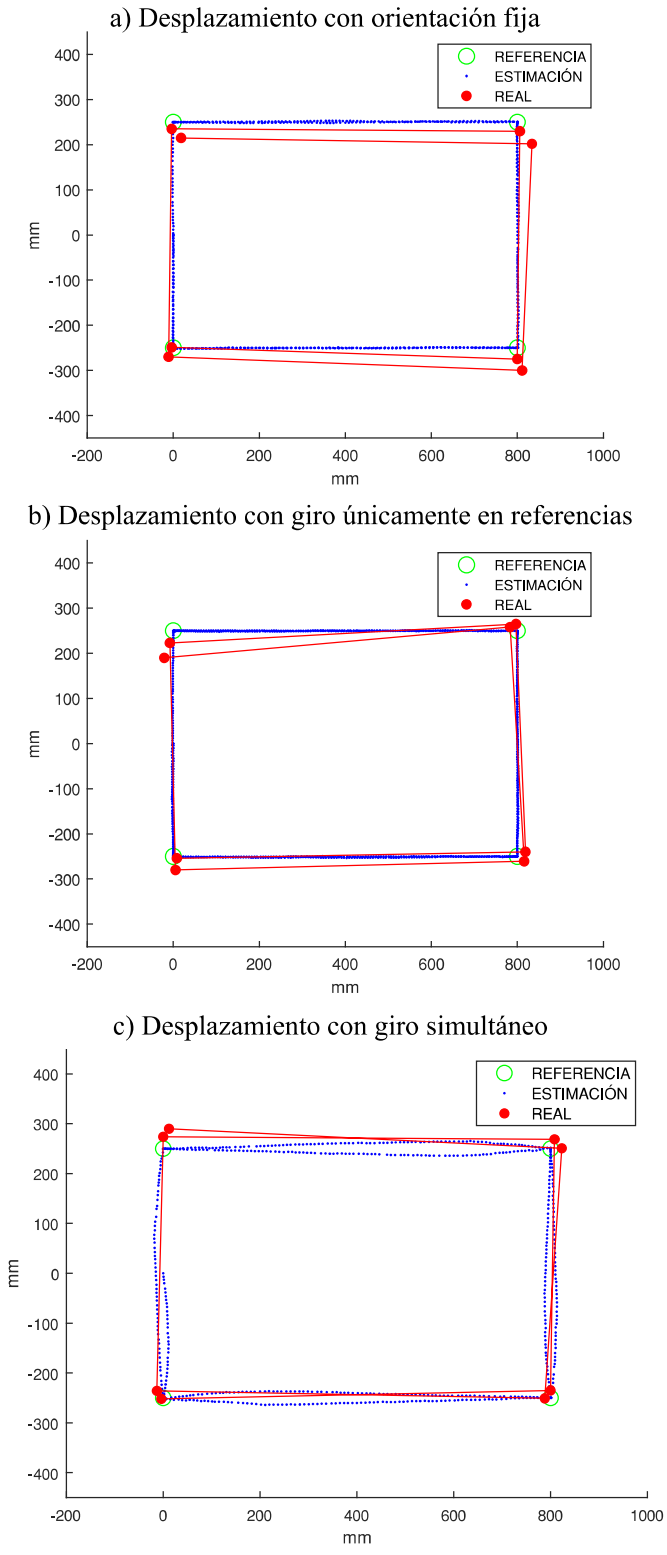



**Figura 57.** Pruebas de seguimiento de trayectoria.

En la Figura 58 se muestran los resultados, donde el obstáculo esta representado como un círculo negro; el obstáculo aumentado considerando el radio del robot se representa por el círculo con línea punteada; de verde se muestran las metas y las trayectorias calculadas;

y en azul los datos de la estimación de posición obtenidas del robot. Se puede evidenciar como la primera y segunda trayectoria pasan por dentro del obstáculo aumentado, por lo que el robot, de forma autónoma se ve repelido por el objeto y se mueve por el contorno al ser atraído hacia la meta. En la última parte de la trayectoria la meta está dentro del obstáculo, por lo que el robot no puede llegar a la meta, pero aún así no colisiona con el obstáculo. Una situación similar sucedería si una trayectoria atravesara por el medio del obstáculo, el robot se quedaría atascado en un mínimo local.

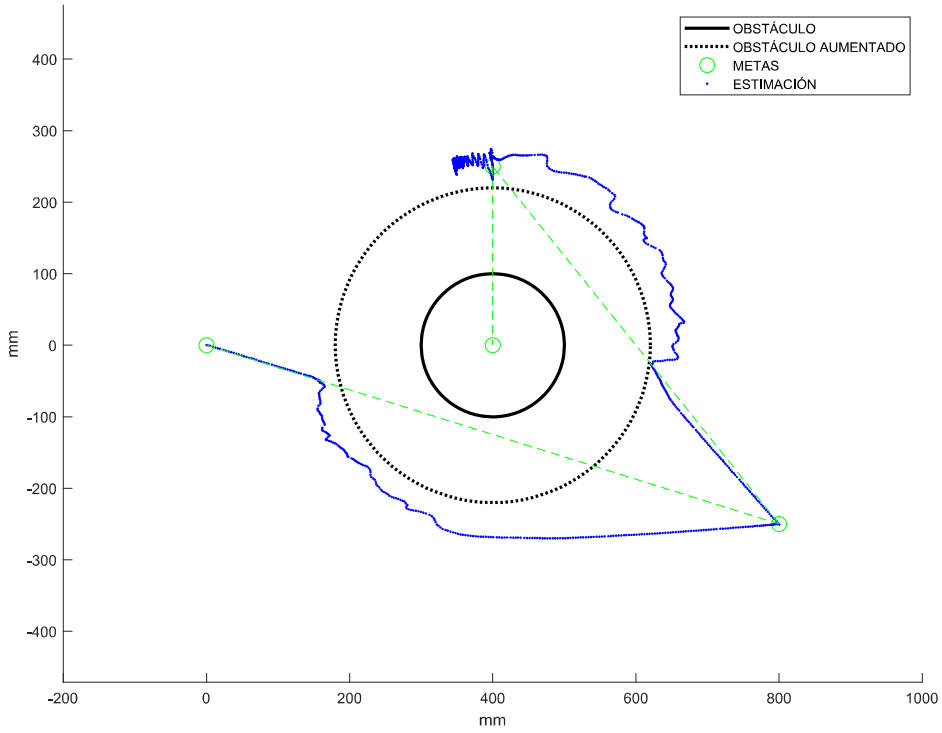



**Figura 58.** Prueba de detección y evasión de obstáculo.

El tipo de sensores utilizados contribuye en gran medida al comportamiento del algoritmo debido a que los sensores ultrasónicos no tienen un alto grado de definición para detectar obstáculos, es decir, no se puede detectar con gran precisión la geometría del obstáculo, ni tampoco obstáculos muy pequeños. Además, para que los sensores ultrasónicos midan correctamente la distancia hacia un objeto, deben estar en una posición aproximadamente paralela a la cara de este (por cómo se reflejan las ondas). Por esta razón no se escogieron obstáculos con ángulos rectos, ya que el robot era capaz de detectarlos cuando se acercaba a una cara de estos, pero cuando se aproximaba hacia una esquina o de forma diagonal a los objetos el robot chocaba con los mismos. Por otro lado, un obstáculo de

tipo cilindro siempre tendrá una aproximación de una cara perpendicular al robot. También se debe mencionar que cada ciclo de actualización del programa (odometría, sensado de obstáculos, calculo de velocidades, entre otros) tiene un tiempo aproximado de 0.1 s, por lo que se ha probado que para velocidades de desplazamiento mayores a 3 cm/s el robot no es capaz de actualizar a tiempo las mediciones de obstáculos y probablemente colisione.

Además, en la Sección 10.5 en la Figura 36 se muestran las pruebas de seguimiento del controlador PID de velocidad implementado en los microcontroladores para cada motor DC. En esta prueba se puede comprobar que se logra un correcto control de la velocidad; con oscilaciones reducidas y un sobreimpulso pequeño. Además, se puede observar como la velocidad de las ruedas tienen una velocidad máxima de $39\ \mathrm{RPM}$ hasta donde se realiza un correcto control de velocidad, por lo tanto, el robot puede desplazarse a una velocidad máxima de $19{,}6\ \mathrm{cm/s}$.

También se midió la corriente consumida por el prototipo en funcionamiento normal como $I = 0{,}96\ \mathrm{A}$. Entonces, se puede determinar la autonomía del robot en 3 horas y 7 minutos dado que tiene una batería de $3\ \mathrm{Ah}$. Sin embargo, para evitar un deterioro acelerado de la batería no es recomendable consumir completamente la energía almacenada. En las pruebas se ha utilizado el prototipo con un máximo de 2 horas en condiciones no continuas, es decir, sin que el robot se esté moviendo continuamente.

Finalmente, se muestra en la Figura 59 una secuencia de imágenes de la prueba del sistema completo.

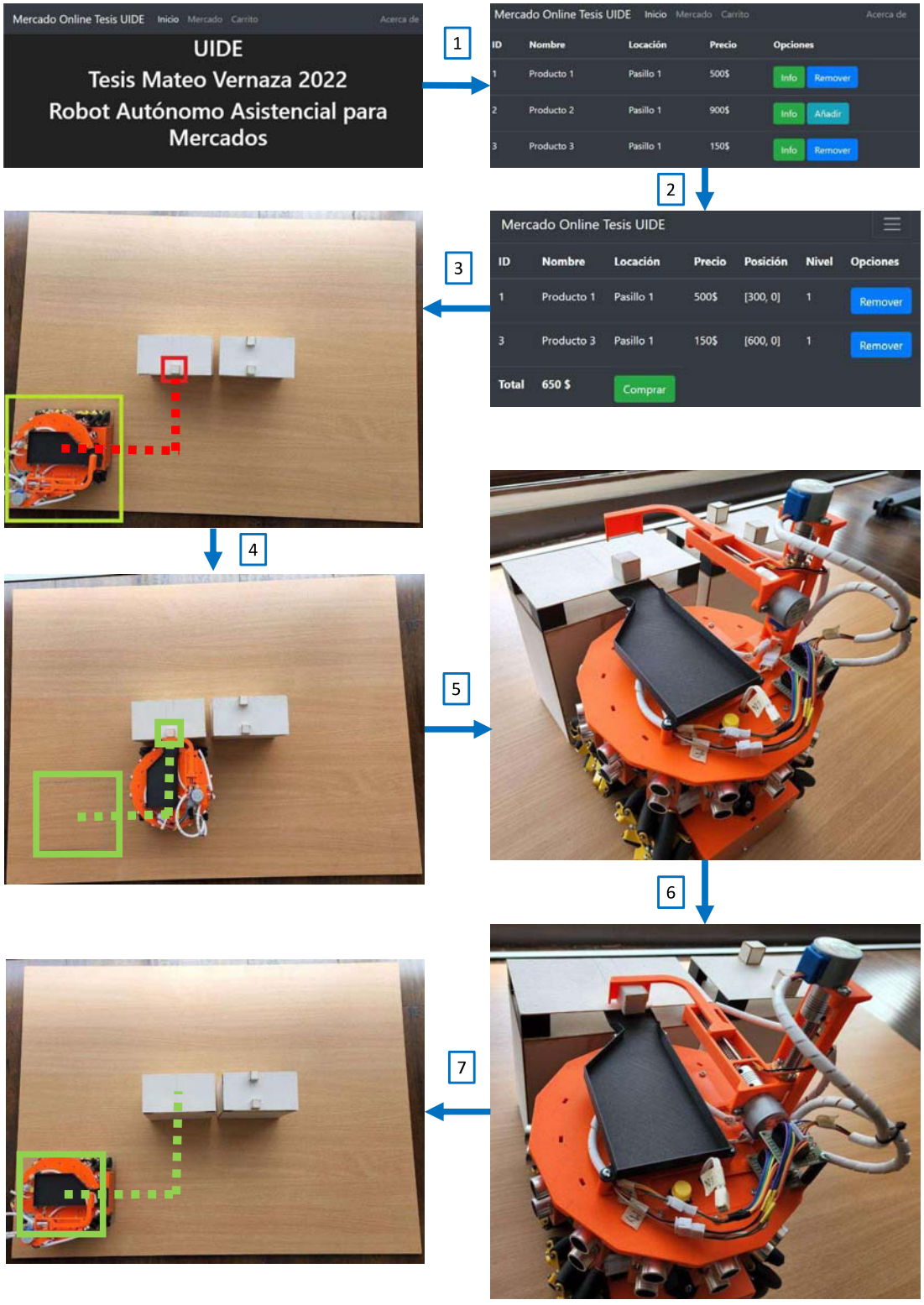



**Figura 59.** Secuencia de prueba del sistema completo.

# 16. Conclusiones

- Se diseñó y construyó un prototipo de un robot autónomo asistencial para compra en supermercados, con un peso de $3{,}25\ \mathrm{kg}$, una altura de $34\ \mathrm{cm}$ y un diámetro de $30\ \mathrm{cm}$; donde se comprobó el funcionamiento de las diferentes partes del sistema para realizar pedidos mediante una página web y que el robot recolecte los productos de forma autónoma en un entorno con obstáculos conocidos y desconocidos.

- El prototipo es capaz de: soportar un peso extra de $500\ \mathrm{g}$; desplazarse a una velocidad máxima de $19{,}6\ \mathrm{cm/s}$ sin detección de obstáculos, y a $3\ \mathrm{cm/s}$ con detección de obstáculos; y tiene una autonomía de 2 horas con funcionamiento no continuo.

- El control del robot se realiza con dos componentes principales: el primero es la generación de trayectorias en el entorno mediante el algoritmo A* aplicado a un grafo de visibilidad que caracteriza los obstáculos conocidos del entorno; y segundo el concepto de campos potenciales virtuales que otorgan una capacidad reactiva al robot para evitar colisionar con obstáculos no conocidos y que pueden ser dinámicos.

- La estimación de la pose del robot se realiza mediante odometría, donde a pesar de ser una metodología sencilla de implementar, genera un error acumulativo debido a factores como: deslizamiento de las ruedas, errores de comunicación entre los componentes electrónicos, errores en la medición de los encoders, discrepancias entre el modelo mecánico real del robot y el modelo ideal desarrollado (debido a tolerancias en construcción y ensamble), entre otros.

- El error del robot se ha cuantificado con desviaciones de entre $16 - 18\ \mathrm{mm}$ por cada metro de desplazamiento lineal, y un error de orientación de aproximadamente $2{,}6°$ por cada $360°$ girados. Se debe considerar que el error de orientación aumentará el error de desplazamiento ya que el robot se desplaza en una dirección incorrecta.

- Se diseñó y construyó el robot con un concepto de prototipado rápido para todos sus sistemas para acelerar el proceso de prueba y mejoramiento. Para esto se implementó un programa de simulación del robot en MATLAB, donde se probó el algoritmo de control completo. También, el PCB se fabricó por mecanizado para tener mayor precisión

ya que el circuito es de un tamaño considerable ($15 \times 10$ cm/s) y de doble lado. Por último, en la parte mecánica se optó por fabricar las piezas por manufactura aditiva para reducir costos y tiempo.

- Se realizaron pruebas de seguimiento de trayectoria para varios tipos de desplazamiento donde existe un error acumulativo debido a que solo se implementó la estimación de posición por odometría. Además, se realizaron pruebas de detección y evasión de obstáculos de forma exitosa para elementos con formas cilíndricas de un radio mayor a $20$ cm.

- Se programó una página web mediante el *micro web framework* Flask que se ejecuta en el microprocesador, donde se tiene la tienda (dentro de la red local) para realizar los pedidos de los productos.

## 17. Recomendaciones

- Se recomienda rediseñar el PCB con elementos SMD para reducir el tamaño del circuito, y manufacturar la placa con un método industrial para evitar errores en la medición de los encoders y errores de comunicación entre el microprocesador y los microcontroladores.

- Se recomienda no consumir la energía completa de la batería para alargar su vida útil, se ha probado hasta un uso de 2 horas de forma no continua.

- Es recomendable tener el ventilador conectado y en la posición correcta para mantener controlada la temperatura del Raspberry Pi Zero.

- Se recomienda reiniciar los microcontroladores cada que se manipule el robot o cada que exista un error de comunicación en el bus I2C.

- Cuando se coloque el robot en su posición inicial se recomienda hacerlo con la mayor precisión posible, ya que cualquier error sumará al error acumulativo en la estimación de la posición.

- Se recomienda implementar algoritmos de localización adicionales a la odometría para corregir el error acumulativo. Puede ser algoritmos de fusión de sensores o agregar otros sensores como cámaras o lidars para mejorar la estimación de la pose del robot.

# BIBLIOGRAFÍA


[1] P. Jílková y P. Králová, "Digital consumer behaviour and ecommerce trends during the covid-19 crisis," *International Advances in Economic Research*, vol. 27, no. 1, pp. 83–85, 2021.

[2] R. Y. Kim, "The impact of covid-19 on consumers: Preparing for digital sales," *IEEE Engineering Management Review*, vol. 48, no. 3, pp. 212–218, 2020.

[3] Servicio Nacional de Gestion de Riesgos y Emergencias Ecuatoriano, "Resoluciones COE Nacional 17 de marzo 2020 – Servicio Nacional de Gestión de Riesgos y Emergencias." [En línea]. Disponible: https://www.gestionderiesgos.gob.ec/resoluciones-coe-nacional-17-de-marzo-2020/

[4] Mintel, "Nearly 70 % of americans shop online regularly with close to 50 % taking advantage of free shipping," 2015. [En línea]. Disponible: https://www.mintel.com/press-centre/technology-press-centre/nearly-70-of-americans-shop-online-regularly-with-close-to-50-taking-advantage-of-free-shipping

[5] Portafolio, "Los colombianos pasan cuatro años de su vida haciendo mercado," 2018. [En línea]. Disponible: https://www.portafolio.co/innovacion/los-colombianos-pasan-cuatro-anos-de-su-vida-haciendo-mercado-518573

[6] Wanlla, "Wanlla - Supermercado Online," 2020. [En línea]. Disponible: https://www.wanlla.com/page/service

[7] Disgralec, "Disgralec - Supermercado Online," 2020. [En línea]. Disponible: http://www.disgralec.com/

[8] Tipti, "Tipti - Supermercado a domicilio," 2020. [En línea]. Disponible: https://tipti.market/formas-pago

[9] Supermaxi, "Compra Inteligente - Supermaxi," 2020. [En línea]. Disponible: https://www.supermaxi.com/compra-inteligente/

[10] Badger Technologies, “Retail Robotics,” 2020. [En línea]. Disponible: https://www.badger-technologies.com/overview.html

[11] Starship, “Starship,” 2020. [En línea]. Disponible: https://www.starship.xyz/

[12] Marble, “Marble,” 2020. [En línea]. Disponible: https://www.marble.io/

[13] T. Tomizawa, A. Ohya, y S. Yuta, “Remote shopping robot system,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 4953–4958, 2006.

[14] T. Tomizawa, K. Ohba, A. Ohya, y S. Yuta, “Remote food shopping robot system in a supermarket,” *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation, ICMA 2007*, pp. 1771–1776, 2007.

[15] T. Onozato, H. Tamura, Y. Kambayashi, y S. Katayama, “A Control System for the Robot Shopping Cart,” *Proceedings of the International Congress on Computer Applications and Computational Science (CACS 2010)*, pp. 907–910, 2010.

[16] T. Takei, R. Imamura, y S. Yuta, “Baggage transportation and navigation by a wheeled inverted pendulum mobile robot,” *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 3985–3994, 2009.

[17] S. Jeong y T. Takahashi, “Wheeled inverted pendulum type assistant robot: inverted mobile, standing, and sitting motions,” *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pp. 1932–1937, 2007.

[18] S. Fu, Z. G. Hou, y G. Yang, “An indoor navigation system for autonomous mobile robot using wireless sensor network,” *Proceedings of the 2009 IEEE International Conference on Networking, Sensing and Control, ICNSC 2009*, pp. 227–232, 2009.

[19] V. Kulyukin, C. Gharpure, y J. Nicholson, “RoboCart: Toward robot-assisted navigation of grocery stores by the visually impaired,” *Proceeding of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 979–984, 2005.

[20] S. Park y S. Hashimoto, “Autonomous mobile robot navigation using passive RFID in indoor environment,” *IEEE Transactions on Industrial Electronics*, vol. 56, no. 7, pp. 2366–2373, 2009.

[21] I. Iossifidis y G. Schöner, “Autonomous reaching and obstacle avoidance with the anthropomorphic arm of a robotic assistant using the attractor dynamics approach,” *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 4295–4300, 2004.

[22] S. Jotawar, M. Soni, y S. Kumar, “Motion planning for an automated pick and place robot in a retail warehouse,” *Proceeding of the ACM International Conference*, 2017.

[23] M. Hassan y D. Liu, “PPCPP: A Predator-Prey-Based approach to adaptive coverage path planning,” *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 284–301, 2020.

[24] L. Lapierre, R. Zapata, y P. Lepinay, “Combined path-following and obstacle avoidance control of a wheeled robot,” *International Journal of Robotics Research*, vol. 26, no. 4, pp. 361–375, 2007.

[25] H. Maaref y C. Barret, “Sensor-based fuzzy navigation of an autonomous mobile robot in an indoor environment,” *Robotics and Autonomous Systems*, vol. 38, no. 1, pp. 1–18, 2002.

[26] K. M. Lynch y F. C. Park, *Modern Robotics Mechanics, Planning, and Control*, first ed ed. Cambridge University Press, 2019.

[27] K. Budynas, R.and Nisbett, *Diseño en ingeniería mecánica de Shigley*, octava ed. Mc Graw Hill - México, 2008.

[28] Mouser Electronics, “28BYJ-48 – 5V Stepper Motor.”

[29] E. ToolBox, “Rolling Resistance,” 2008. [En línea]. Disponible: https://www.engineeringtoolbox.com/rolling-friction-resistance-d_1303.html

[30] Instituto Ecuatoriano de Normalización, “Accesibilidad De Las Personas Al Medio Físico. Rampas,” 2016. [En línea]. Disponible: https://www.habitatyvivienda.gob.ec/wp-content/uploads/downloads/2018/06/NTE-INEN-2245-RAMPAS.pdf

[31] GoBilda, “96mm Mecanum Wheel Set,” 2022. [En línea]. Disponible: https://www.gobilda.com/96mm-mecanum-wheel-set-70a-durometer-bearing-supported-rollers/

[32] Servocity, “Micro metal N20 gearmotor 298:1 Ratio, 90 RPM, with Encoder.”

[33] J. Marin, *Mechanical Behavior of Engineering Materials*. Prentice-Hall, 1962.

[34] S. H. Howie Choset, Kevin Lynch, *Principles of Robot Motion*, 2005, vol. 53.

[35] S. M. LaValle, *Planning algorithms*, 2006, vol. 9780521862.

[36] R. Goldman, “Intersection of three planes,” in *Graphics Gems*, A. S. Glassner, Ed. Academic Press, 1990, pp. 305–305.

[37] B. Siciliano y O. Khatib, *Handbook of Robotics*, 2018, vol. 109, no. 7.

[38] *The Traveling Salesman Problem*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 527–562. [En línea]. Disponible: https://doi.org/10.1007/978-3-540-71844-4_21

[39] J.-C. Latombe, *Robot Motion Planning*, 2011.

[40] Arduino, “Schematic Uno R3,” p. 1, 2019. [En línea]. Disponible: https://store.arduino.cc/products/arduino-uno-rev3

[41] Atmel, “ATmega328P,” pp. 1–294, 2015. [En línea]. Disponible: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[42] Flask, “Flask web framework.” [En línea]. Disponible: https://flask.palletsprojects.com/en/2.1.x/

[43] R. Nau, “Lecture notes on exponential smoothing.” [En línea]. Disponible: https://people.duke.edu/∼rnau/411avg.htm

# Anexo A: Algoritmos

## 1. MATRIZ LOGARÍTMICA DE ROTACIÓN

**Algoritmo 1** Matriz logarítmica de rotación

**Entrada:** $R$
**Salida:** $\theta$, $\hat{\omega}$

**si** $R = I$ **entonces**
  $\theta \leftarrow 0$;
  $\hat{\omega}\ no\ definido$;
**si no**, **si** $tr(R) = -1$ **entonces**
  $\theta \leftarrow \pi$;
  **si** $1/\sqrt{2(1+r_{33})} \in \mathbb{R}$ **entonces**
    $\hat{\omega} \leftarrow \frac{1}{\sqrt{2(1+r_{33})}} \begin{bmatrix} r_{13} \\ r_{23} \\ 1+r_{33} \end{bmatrix}$
  **si no**, **si** $1/\sqrt{2(1+r_{22})} \in \mathbb{R}$ **entonces**
    $\hat{\omega} \leftarrow \frac{1}{\sqrt{2(1+r_{22})}} \begin{bmatrix} r_{12} \\ 1+r_{22} \\ r_{32} \end{bmatrix}$
  **si no**, **si** $1/\sqrt{2(1+r_{11})} \in \mathbb{R}$ **entonces**
    $\hat{\omega} \leftarrow \frac{1}{\sqrt{2(1+r_{11})}} \begin{bmatrix} 1+r_{11} \\ r_{21} \\ r_{31} \end{bmatrix}$
  **fin si**
**si no**
  $\theta \leftarrow cos^{-1}(\frac{tr(R)-1}{2}) \in [0, \pi)$
  $[\hat{\omega}] \leftarrow \frac{1}{2sen(\theta)}(R - R^T)$
**fin si**
**devolver** $\theta$, $\hat{\omega}$

## 2. MATRIZ LOGARÍTMICA DE ROTACIÓN Y TRASLACIÓN

**Algoritmo 2** Matriz logarítmica de rotación y traslación

**Entrada:** $R$, $p$
**Salida:** $\theta$, $\hat{\omega}$, $v_u$

**si** $R = I$ **entonces**
  $\hat{\omega} \leftarrow 0$;
  $v_u \leftarrow p/||p||$;
  $\theta \leftarrow ||p||$;
**si no**
  $\theta, \hat{\omega} \leftarrow$ Matriz Logarítmica de Rotación($R$)
  $v_u \leftarrow (\frac{I}{\theta} - \frac{[\hat{\omega}]}{2} + (\frac{1}{\theta} - \frac{1}{2} \cdot cot(\frac{\theta}{2})) \cdot [\hat{\omega}]^2) \cdot p$
**fin si**
**devolver** $\theta$, $\hat{\omega}$, $v_u$

## 3. CONECTIVIDAD

**Algoritmo 3** Conectividad

**Entrada:** $nodos$, $vectores\_obstaculos$, $puntos\_obstaculos$
**Salida:** $mat\_pesos$

$n\_nodos \leftarrow dim(nodos)$;
$mat\_pesos \leftarrow [[0] * n\_nodos] * n\_nodos$;
**para** $i = 1$ hasta $n\_nodos - 1$ **hacer**
  **para** $j = i + 1$ hasta $n\_nodos$ **hacer**
    **si** no $colision(nodos[i], nodos[j], vectores\_obstaculos, puntos\_obstaculos)$ **entonces**
      $mat\_pesos[i][j] \leftarrow norm(nodos[i] - nodos[j])$;
      $mat\_pesos[j]i] \leftarrow mat\_pesos[i][j]$;
    **fin si**
  **fin para**
**fin para**
**devolver** $mat\_pesos$

## 4. COLISIÓN

**Algoritmo 4** Colisión

**Entrada:** $nodo1$, $nodo2$, $vectores\_obstaculos$, $puntos\_obstaculos$
**Salida:** $check$

1: $vector\_arista \leftarrow [nodo2 - nodo1]$;
2: **para** $i = 1$ hasta $dim(vectores\_obstaculos)$ **hacer**
3:  **si** $vector\_arista$ atraviesa $obstaculo[i]$ **entonces**
4:  **devolver** $check \leftarrow verdadero$
5:  **fin si**
6:  **para** $j = 1$ hasta $dim(vectores\_obstaculos[i])$ **hacer**
7:  $a \leftarrow puntos\_obstaculos[i][j] - nodo1$;
8:  $b \leftarrow mod(vectores\_obstaculos[i][j] \times vector\_arista)$;
9:  $t \leftarrow mod(a \times vector\_arista)/b$;
10:  $u \leftarrow mod(a \times vectores\_obstaculos[i][j])/b$;
11:  **si** $b! = 0$ y $t > 0$ y $t < 1$ y $u > 0$ y $u < 1$ **entonces**
12:  **devolver** $check \leftarrow verdadero$
13:  **fin si**
14:  **fin para**
15: **fin para**
16: **devolver** $check \leftarrow falso$

## 5. BÚSQUEDA A*

**Algoritmo 5** Algoritmo de búsqueda A*

**Entrada:** $pesos$, $nodos$, $n\_inicial$, $n\_final$
**Salida:** $camino$, $costo\_total$

**para** $i = 1$ hasta $dim(nodos)$ **hacer**
  $heuristica[i] \leftarrow norm(nodos[n\_final] - nodos[i])$
**fin para**
$nodos\_revisados \leftarrow []$;
$nodos\_no\_revisados \leftarrow [n\_inicial]$;
$min\_costos \leftarrow [infinito] * dim(nodos)$;
$min\_costos[n\_inicial] \leftarrow 0$;
$arbol \leftarrow [0] * dim(nodos)$;
$costo\_estimado \leftarrow [infinito] * dim(nodos)$;
**mientras** $nodos\_no\_revisados$ no este vacío **hacer**
  $actual \leftarrow nodos\_no\_revisados[1]$;
  agregar $actual$ a $nodos\_revisados$;
  remover $actual$ de $nodos\_no\_revisados$;
  **si** $actual = n\_final$ **entonces**
    **fin mientras**
  **fin si**
  **para** cada $vecino = 1$ en $nodos$ **hacer**
    **si** $pesos[actual][vecino]! = 0$ o $vecino$ no en $nodos\_revisados$ **entonces**
      $costo\_temporal \leftarrow min\_costos[actual] + pesos[actual][vecino]$;
      **si** $costo\_temporal < min\_costos[vecino]$ **entonces**
        $min\_costos[vecino] \leftarrow costo\_temporal$;
        $costo\_estimado[vecino] \leftarrow min\_costos[vecino] + heuristica[vecino]$;
        $arbol[vecino] \leftarrow actual$;
        insertar $vecino$ en lista ordenada $nodos\_no\_revisados$;
      **fin si**
    **fin si**
  **fin para**
**fin mientras**
$camino \leftarrow n\_final$;
**mientras** $camino[1]! = n\_inicial$ **hacer**
  isertar $arbol[camino[1]]$ en $camino$;
**fin mientras**
**devolver** $camino$, $costo\_total$

## 6. CÁLCULO DE RUTA

**Algoritmo 6** Cálculo de ruta

**Entrada:** $metas$, $pesos$, $nodos$, $n\_inicial$
**Salida:** $camino$

```
1: metas_no_revisadas ← metas;
2: mientras metas_no_revisadas no este vacío hacer
3:    menor_costo ← infinito;
4:    para meta en metas_no_revisadas hacer
5:       camino_temp, costo_temp ← a_star(pesos, nodos, n_inicial, meta);
6:       si costo_temp < menor_costo entonces
7:          menor_costo ← costo_temp;
8:          meta_temp ← meta;
9:          mejor_camino_temp ← camino_temp;
10:      fin si
11:   fin para
12:   agregar mejor_camino_temp a camino
13:   remover meta_temp de metas_no_revisadas;
14:   n_inicial ← meta_temp;
15: fin mientras
16: devolver camino
```

# Anexo B: Código de Programación

## 1. CÓDIGOS DE SIMULACIÓN EN MATLAB

### 1.1. Función Principal de la Simulación *Principal.m*

```
clc
clf
format long
%% Parametros
docEntorno = 'EntornoV2.0';
% Cargar parametros del Entorno
entorno.limites = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range','A2:A5');
rangoFin = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range','K2:K2');
if rangoFin(1) > 0
    rangoStr = strcat('L2:O',num2str(1+rangoFin(1)));
    entorno.obstaculosCon = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range',rangoStr);
else
    entorno.obstaculosCon = [];
end
rangoFin = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range','P2:P2');
if rangoFin(1) > 0
    rangoStr = strcat('Q2:T',num2str(1+rangoFin(1)));
    entorno.obstaculosDesc = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range',rangoStr);
else
    entorno.obstaculosDesc = [];
end
% Cargar parametros de los productos
rangoFin = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range','B2:B2');
if rangoFin(1) > 0
    rangoStr = strcat('C2:E',num2str(1+rangoFin(1)));
    productos.pose = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range',
```

```
    rangoStr);
27      rangoStr = strcat('F2:F',num2str(1+rangoFin(1)));
28      productos.id = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range',
    rangoStr);
29      rangoStr = strcat('G2:G',num2str(1+rangoFin(1)));
30      productos.nombre = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range'
    ,rangoStr,'OutputType','string');
31      rangoStr = strcat('H2:H',num2str(1+rangoFin(1)));
32      productos.precio = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range'
    ,rangoStr);
33      rangoStr = strcat('I2:I',num2str(1+rangoFin(1)));
34      productos.nivel = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range',
    rangoStr);
35      rangoStr = strcat('J2:J',num2str(1+rangoFin(1)));
36      productos.pasillo = readmatrix('Entornos.xlsx','Sheet',docEntorno,'Range
    ',rangoStr);
37  end
38  % Escribir a archivo
39  rangoStr = strcat('C1:J',num2str(1+rangoFin(1)));
40  T = readtable('Entornos.xlsx','Sheet',docEntorno,'Range',rangoStr);
41  writetable(T,'docs/productos.csv')
42  % Cargar parametros del robot
43  load('RobotV2.0.mat')
44  % radio_robot = en mm
45  % n_sensores = numero de sensores ultrasonicos
46  % rango_sensor = rango de deteccion de obstaculosCon para repulsion en mm
47  % max_velocidad = velocidad lineal maxima en mm/s
48  % c_atraccion = constante de desaceleracion en mm
49  % max_velocidad_angular = velocidad angular maxima en rad/s
50  % c_atraccion_angular = constante de desaceleracion en rad
51  % c_repulsion = constante de repulsion
52  % c_filtro_ema = constante para filtro EMA para mediciones de sensores
53  % dist_final = error maximo de posicion en mm
54  % ang_final = error de orientacion maximo en rad
55  % dist_estante = distancia a posicion de recoleccion de producto en mm
56  % radio_rueda = radio de las ruedas en mm
```

```
% l = largo del robot desde punto de contacto de la rueda hasta centro geometrico en mm
% w = ancho del robot desde punto de contacto de la rueda hasta centro geometrico en mm
% coord_ini = [x y] coordenadas iniciales en mm
% t_actualizacion = 0.1
%% Funcion para leer y calcular variables iniciales
[entorno,grownObs,grownLimites,vecsObs,pointsObs,nodos,matAdy,matPesos,posPrev,posProd,productos] = leerDocs(robot,entorno,productos);
%% Figura
load GraficoV1.0.mat grafico
figure(1)
maxTamano = max(entorno.limites(2),entorno.limites(4));
set(gcf,'position',[600 10 1000*entorno.limites(2)/maxTamano 1000*entorno.limites(2)/maxTamano])
axis([entorno.limites(1)-maxTamano/1e3 entorno.limites(2)+maxTamano/1e3 entorno.limites(3)-maxTamano/1e3 entorno.limites(4)+maxTamano/1e3]);
hold on
% plot obstaculosCon y limites
plotObs(entorno.obstaculosCon,grownObs,grownLimites,grafico);
% plot nodos
plotNodos(nodos,grafico);
% plot grafo de visibilidad
plotGrafo(matAdy,nodos,grafico)
% plot punto inicial del robot
plot(robot.coord_ini(1),robot.coord_ini(2),'go','LineWidth', 3)
%% Programa Principal
% Ingreso de pedidos
disp('PRODUCTOS DISPONIBLES:')
for i = 1:size(productos.id,1)
    disp(strcat("ID: ",num2str(productos.id(i))," Producto: ",productos.nombre(i)," Posicion: [",num2str(productos.pose(i,1))," ",num2str(productos.pose(i,2)),"]"));
end
% productos pedidos de pagina web
prodPedidos = input("Ingrese vector de IDs de productos pedidos: ");
```

```
86 while true
87     % obtener posiciones previas a agarrar los productos pedidos
88     goals = zeros(size(prodPedidos,2),2);
89     for i=1:size(goals,1)
90         goals(i,:) = [posPrev(prodPedidos(i),1),posPrev(prodPedidos(i),2)];
91         %   plot metas de productos pedidos
92         plot(goals(i,1),goals(i,2),'go','LineWidth', 3)
93         pause(grafico.tGoals)
94     end
95     % generar nuevo grafo de visibilidad agregando posicion inicial y metas
    de los productos
96     [nodosNew,matAdyNew,matPesosNew] = nuevoGrafo(goals,nodos,matAdy,
    matPesos,vecsObs,pointsObs,robot.coord_ini,grafico);
97     % generar mejorar trayectoria que pase por todas las metas
98     [camino,goalsOrder]=trayectoria(goals,matPesosNew,nodosNew,grafico);
99
100    % Para simulacion se genera vectores de obstaculosCon conocidos y
    desconocidos
101    % para calcular distancia con los sensores simulados
102    [knownVObs,unknownVObs]=obstDesconocidos(entorno.obstaculosCon,entorno.
    obstaculosDesc);
103    % generar vector de sensores
104    vecSensores = genVecSensores(robot);
105    % variables para simulacion
106    infr=[];
107    h=[];
108    dir=[];
109
110    % posicion inicial
111    pos = nodosNew(camino(1),:);
112    % Nodo siguiente en la ruta
113    contCamino = 2;
114    % contador de metas
115    contGoals = 1;
116    fin = true;
117    while fin
```

```
118     %     calculo de velocidad
119         [V_control,distSensores]=calcVelocidad(camino,nodosNew,contCamino,
    pos,vecSensores,knownVObs,unknownVObs,robot);
120     %     Actualizar posicion segun la velocidad -> equivalente a odometria
121         pos = pos + V_control*robot.t_actualizacion;
122 
123     %     plot robot
124         [infr,h,dir] = graficoSimulacion(pos,V_control,distSensores,robot,
    grafico,infr,h,dir);
125 
126     %     condicion de terminacion
127         if(contCamino == size(camino,1) && norm(nodosNew(camino(contCamino)
    ,:)-pos)<robot.dist_final)
128             return
129         end
130     %     chequeo para avanzar a siguiente meta
131         [contCamino,contGoals,pos,infr,h,dir]=siguienteGoal(nodosNew,
    contCamino,camino,pos,contGoals,goalsOrder,posProd,prodPedidos,robot,
    vecSensores,knownVObs,unknownVObs,infr,h,dir,grafico);
132     end
133 end
```

## 1.2. Formato de Tabla del Entorno *Entornos.xlsx*

| LIMITES | N_PRODUCTOS | POSE_X | POSE_Y | ORIENTACION | ID | NOMBRE | PRECIO | NIVEL | PASILLO | N_OBSTACULOS_C | OBSTACULOS_C_X_INI | OBSTACULOS_C_X_FIN | OBSTACULOS_C_Y_INI | OBSTACULOS_C_Y_FIN | N_OBSTACULOS_D | OBSTACULOS_D_X_INI | OBSTACULOS_D_X_FIN | OBSTACULOS_D_Y_INI | OBSTACULOS_D_Y_FIN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 500 | 400 | 1.5707963268 | 1 | Producto 1 | 3.00 | 1 | 1 | 1 | 400 | 800 | 400 | 500 | 0 | | | | |
| 1200 | | 700 | 400 | 1.5707963268 | 2 | Producto 2 | 48.00 | 1 | 1 | | | | | | | | | | |
| 0 | | 500 | 500 | -1.5707963268 | 3 | Producto 3 | 36.00 | 1 | 2 | | | | | | | | | | |
| 900 | | 700 | 500 | -1.5707963268 | 4 | Producto 4 | 21.00 | 1 | 2 | | | | | | | | | | |

**Figura 1.** Formato de tabla para definición del entorno.

## 1.3. Función de Algoritmo A* *Astar.m*

```
% Algoritmo A* para busqueda en grafos
function [camino,costoTotal] = AStar(pesosMat, nodos, nIni, nFin)
% Se calcula el costo heuristico, en este caso es la distancia euclidiana
% hacia la meta o nodo final
heuristica = zeros(size(pesosMat,1),1);
for i=1:size(pesosMat,1)
    heuristica(i) = sqrt((nodos(i,1)-nodos(nFin,1))^2+(nodos(i,2)-nodos(nFin
    ,2))^2);
end
% Se crea lista de nodos revisados y la lista de nodos no revisados
% inicializado con el nodo inicial
nodosRevisados = [];
nodosNoRevisados = [nIni];
```

```
% Se inicializa la lista de costos minimos encontrados hasta el momento
% para cada nodo, al comienzo todos menos el nodo inicial tienen un costo
% infinito
minCost(1:size(nodos,1)) = inf;
minCost(nIni) = 0;
% Se crea la lista que almacena cual es el padre de cada nodo en el arbol
arbol = zeros(size(nodos,1),1);
% Se crea una lista de costos estimados que inicialmente es infinita
costoEstimado(1:size(nodos,1)) = inf;
% Se ejecuta el programa hasta que la lista de revisados este vacia
while ~isempty(nodosNoRevisados)
%     Se elige el nodo en la primera posicion de los nodos no revisados
%     para analizarlo
    current = nodosNoRevisados(1);
%     Se agrega el nodo a la lista de revisados
    nodosRevisados = [nodosRevisados, current];
%     Se borra el nodo de la lista de no revisados
    nodosNoRevisados = nodosNoRevisados(nodosNoRevisados~=current);
%     Si el nodo que se esta analizando es igual al nodo final se termina
%     el algoritmo
    if(current == nFin)
        break
    end
%     Se analiza todo vecino del nodo que se esta evaluando
    for vecino=1:size(pesosMat,1)
%         Se chequea cual nodo es vecino y si no esta en los nodos
%         revisados
        if((pesosMat(current,vecino)~=0||(nodos(current,1)==nodos(vecino,1)
   &&nodos(current,2)==nodos(vecino,2)))&& ~ismember(vecino,nodosRevisados))
%             Se almacena un costo temporal que es el costo en ir al nodo
%             vecino desde el camino mas corto hasta el nodo actual
            costoTemporal = minCost(current)+pesosMat(current,vecino);
%             Si este camino es mas barato que el camino anteriormente
%             almacenado se reemplaza como el mejor camino hasta el nodo
%             vecino
            if(costoTemporal < minCost(vecino))
```

```
48 %                  Se actualiza el nuevo mejor costo para llegar al nodo
49 %                  vecino
50                 minCost(vecino) = costoTemporal;
51 %                  Se actualiza el arbol donde se especifica que el padre
52 %                  del nodo vecino es el nodo actualmente analizado
53                 arbol(vecino) = current;
54 %                  Se agrega el nodo vecino a nodos no revisados para ser
55 %                  eventualmente revisado, pero se lo agrega de acuerdo al
56 %                  costo estimado que tienen para llegar al nodo final
57 %                  mediante el costo heuristico. Se ordena de menor a mayor,
58                 costoEstimado(vecino) = minCost(vecino)+heuristica(vecino);
59                 if(isempty(nodosNoRevisados))
60                     nodosNoRevisados = vecino;
61                 else
62                     pos = 1;
63 %                    Se busca la posicion en la que el nodo vecino se
64 %                    inserta en la lista de no revisados
65                     while pos <= size(nodosNoRevisados,2) && costoEstimado(
   vecino) > costoEstimado(nodosNoRevisados(pos))
66                         pos = pos + 1;
67                     end
68                     nodosNoRevisados = [nodosNoRevisados(1:pos-1), vecino,
   nodosNoRevisados(pos:end)];
69                 end
70             end
71         end
72     end
73 end
74 % Recursividad para hallar el camino a partir del arbol
75 camino = nFin;
76 while(camino(1) ~= nIni)
77     if arbol(camino(1)) == 0
78         error('No se encontraron caminos')
79     else
80         camino = [arbol(camino(1)); camino];
81     end
```

```
end
% se retorna el camino de nodos para llegar a la meta y el costo total
camino(camino==nIni) = [];
costoTotal = minCost(nFin);
end
```

### 1.4. Función de Atracción *atraccion.m*

```
function [Fatr] = atraccion(goal,pos,robot)
% calculo de fuerza de atraccion
F = robot.max_velocidad*(1-exp(-(norm(goal-pos)/robot.c_atraccion)));
if(norm(goal-pos)~=0)
    Fatr = F*((goal-pos)/norm(goal-pos));
end
end
```

### 1.5. Función Cálculo de Velocidad *calcVelocidad.m*

```
function [V_control,distSensores]=calcVelocidad(camino,nodosNew,contCamino,
    pos,vecSensores,knownVObs,unknownVObs,robot)
% Fuerza de atraccion
F_atr = atraccion(nodosNew(camino(contCamino),:),pos,robot);
% Se calcula distancias obtenidas de sensores -> equivalente a medicion de
% sensoresr
distSensores = sensor(knownVObs,unknownVObs,pos,robot,vecSensores);
% Fuerza de repulsion
F_rep = repulsion(distSensores,-vecSensores,robot);
% Velocidad de Control = Fuerza Resultante = Fuerza Atraccion + Fuerza
V_control = F_atr + F_rep;
% Condicion de que la Velocidad de Control no supere la velocidad maxima
% del robot
if(norm(V_control)>robot.max_velocidad)
    V_control = V_control/norm(V_control)*robot.max_velocidad;
end
end
```

### 1.6. Función Cálculo de Velocidad sin Repulsión *calcVelocidadNoRep.m*

```
function [V_control,distSensores]=calcVelocidadNoRep(meta,pos,vecSensores,
    knownVObs,unknownVObs,robot)
% Fuerza de atraccion
F_atr = atraccion(meta,pos,robot);
% Distancias obtenidas por los sensores
distSensores = sensor(knownVObs,unknownVObs,pos,robot,vecSensores);
% Velocidad de Control = Fuerza Resultanto = Fuerza Atraccion
V_control = F_atr;
% Condicion de que la Velocidad de Control no supere la velocidad maxima
% del robot
if(norm(V_control)>robot.max_velocidad)
    V_control = V_control/norm(V_control)*robot.max_velocidad;
end
end
```

### 1.7. Función Graficación de Círculo *circle.m*

```
function h = circle(x,y,r)
th = 0:pi/50:2*pi;
xunit = r * cos(th) + x;
yunit = r * sin(th) + y;
h = plot(xunit, yunit, 'Color', 'black','LineWidth',2);
end
```

### 1.8. Función Conectividad *conectividad.m*

```
function [matAdy,matPesos]=conectividad(nodos,vecsObs,pointsObs)
nNodos = size(nodos,1);
% generar matriz de adyacencia y matriz de pesos
matAdy = zeros(nNodos);
matPesos = zeros(nNodos);
for i = 1:nNodos-1
    for j = i+1:nNodos
%         revisar si existe colision
        if(~vertColision2(nodos(i,:),nodos(j,:),vecsObs,pointsObs))
%             agregar nodos a matriz de adyacencia y de pesos
```

```
            matAdy(i,j) = j;
            matAdy(j,i) = i;
            matPesos(i,j) = norm(nodos(i,:)-nodos(j,:));
            matPesos(j,i) = matPesos(i,j);
        end
    end
end
end
```

### 1.9. Función para Borrar Gráficos *deleteGrafico.m*

```
function [] = deleteGrafico(infr,h,dir)
delete(infr)
delete(h)
delete(dir)
end
```

### 1.10. Función de Generación de Aristas *genAristasObs.m*

```
function [vecsObs,pointsObs]=genAristasObs(grownObs,grownLimites)
% Generacion vectores y puntos de obstaculos
nGrownObs = size(grownObs,1);
for i=1:nGrownObs
    vecsObs(i*6-5:i*6,:) = [0,grownObs(i,4)-grownObs(i,3)
                            grownObs(i,2)-grownObs(i,1),0
                            0,grownObs(i,3)-grownObs(i,4)
                            grownObs(i,1)-grownObs(i,2),0
                            grownObs(i,2)-grownObs(i,1),grownObs(i,4)-grownObs(
    i,3)
                            grownObs(i,1)-grownObs(i,2),grownObs(i,4)-grownObs(
    i,3)];
    pointsObs(i*6-5:i*6,:) = [grownObs(i,1),grownObs(i,3)
                      grownObs(i,1),grownObs(i,4)
                      grownObs(i,2),grownObs(i,4)
                      grownObs(i,2),grownObs(i,3)
                      grownObs(i,1),grownObs(i,3)
                      grownObs(i,2),grownObs(i,3)];
```

```
end
vecsObs((i+1)*6-5:(i+1)*6,:) = [0,grownLimites(4)-grownLimites(3)
                                grownLimites(2)-grownLimites(1),0
                                0,grownLimites(3)-grownLimites(4)
                                grownLimites(1)-grownLimites(2),0
                                0,0
                                0,0];
pointsObs((i+1)*6-5:(i+1)*6,:) = [grownLimites(1),grownLimites(3)
                  grownLimites(1),grownLimites(4)
                  grownLimites(2),grownLimites(4)
                  grownLimites(2),grownLimites(3)
                  grownLimites(1),grownLimites(3)
                  grownLimites(2),grownLimites(3)];
end
```

## 1.11. Función de Generación de Nodos *genNodos.m*

```
% Generacion de nodos en esquinas de obstaculos y limites
function [nodos]=genNodos(pointsObs,limites)
nObs = size(pointsObs,1)/6;
nodos = zeros(nObs*4,2);
% generacion de nodos
for i = 1:nObs
    for j = 1:4
        nodos(j+(i-1)*4,:) = [pointsObs(j+(i-1)*6,1),pointsObs(j+(i-1)*6,2)
    ];
    end
end
j = 1;
% eliminar nodos que se encuentran afuera de los limites del entorno
while j<= size(nodos,1)
    if(nodos(j,1)<limites(1)||nodos(j,1)>limites(2)||nodos(j,2)<limites(3)||
    nodos(j,2)>limites(4))
        nodos(j,:) = [];
    else
        j = j + 1;
    end
```

```
end
end
```

## 1.12. Función de Generación de Posición de Productos *genPosProductos.m*

```
function [posPrev,posProd,productos] = genPosProductos(robot,productos)
% generar posicion previa para agarrar producto [posPrev]=[x y direccion]
% y posicion para agarrar producto considerando radio de robot [posProd]=[x
    y direccion nivel]
nProductos = size(productos.id,1);
posPrev = zeros(nProductos,3);
posProd = zeros(nProductos,4);
for i=1:nProductos
    posPrev(i,:) = [productos.pose(i,1:2)+[-cos(productos.pose(i,3)) -sin(
    productos.pose(i,3))]*robot.offset,productos.pose(i,3)];
    posProd(i,:) = [productos.pose(i,1:2)+[-cos(productos.pose(i,3)) -sin(
    productos.pose(i,3))]*(robot.dist_estante),productos.pose(i,3),productos.
    nivel(i)];
end
% guardar variables en documentos
fileID = fopen('docs/posPrev.txt','w');
fprintf(fileID,'%f %f %f\n',posPrev');
fclose(fileID);
fileID = fopen('docs/posProd.txt','w');
fprintf(fileID,'%f %f %f %d\n',posProd');
fclose(fileID);
end
```

## 1.13. Función de Generación de Vectores de los Sensores *genVecSensores.m*

```
function [vecSensores]=genVecSensores(robot)
% Vectores de las medidas tomadas por el sensores IR
% vecLidar = [cos(2pi/cantMedidas) sen(2pi/cantMedidas)] * Distancia Maxima
vecSensores = zeros(size(robot.n_sensores,1),2);
for i = 0:robot.n_sensores-1
    vecSensores(i+1,:) = [cos(2*pi/robot.n_sensores*i)*robot.rango_sensor
    sin(2*pi/robot.n_sensores*i)*robot.rango_sensor];
```

```
end
end
```

### 1.14. Función de Graficación *graficoSimulacion.m*

```
function [infr,h,dir]=graficoSimulacion(pos,V_control,sensores,robot,grafico
    ,infr,h,dir)
% se borra grafico anterior
deleteGrafico(infr,h,dir)
% Se grafica cada sensor
for k=0:size(sensores,1)-1
    infr(k+1) = plot([pos(1) pos(1)+cos(2*pi/robot.n_sensores*k)*sensores(k
    +1)],[pos(2) pos(2)+sin(2*pi/robot.n_sensores*k)*sensores(k+1)],'r','
    LineWidth',1);
end
% plot robot
h = circle(pos(1),pos(2),robot.radio_robot);
% Graficar la direccion y magnitud de velocidad
dir = plot([pos(1) pos(1)+V_control(1)*10],[pos(2) pos(2)+V_control(2)*10],'
    blue','LineWidth',3);
pause(grafico.tSimulacion)
end
```

### 1.15. Función de Lectura de Documento *leerDocs.m*

```
function [entorno,grownObs,grownLimites,vecsObs,pointsObs,nodos,matAdy,
    matPesos,posPrev,posProd,productos]=leerDocs(robot,entorno,productos)
% guardar variables en documentos
fileID = fopen('docs/limites.txt','w');
fprintf(fileID,'%f %f %f %f\n',entorno.limites');
fclose(fileID);
% generacion de posiciones previas a agarrar productos [posPrev]=[x y
    direccion] y
% posiciones para agarrar los productos [posProd]=[x y direccion nivel]
[posPrev,posProd,productos] = genPosProductos(robot,productos);
% agrandar obstaculosCon y limites del entorno
grownObs = [entorno.obstaculosCon(:,1)-robot.offset entorno.obstaculosCon
```

```
    (:,2)+robot.offset entorno.obstaculosCon(:,3)-robot.offset entorno.
    obstaculosCon(:,4)+robot.offset];
grownLimites = [entorno.limites(1)+robot.offset entorno.limites(2)-robot.
    offset entorno.limites(3)+robot.offset entorno.limites(4)-robot.offset];
% Generacion de aristas de obstaculosCon
[vecsObs,pointsObs] = genAristasObs(grownObs,grownLimites);
% guardar variables en documentos
fileID = fopen('docs/vecsObstaculos.txt','w');
fprintf(fileID,'%f %f\n',vecsObs');
fclose(fileID);
fileID = fopen('docs/pointsObstaculos.txt','w');
fprintf(fileID,'%f %f\n',pointsObs');
fclose(fileID);
% generar nodos de grafo de visibilidad
[nodos]=genNodos(pointsObs,entorno.limites);
% guardar variable en documento
fileID = fopen('docs/nodos.txt','w');
fprintf(fileID,'%f %f\n',nodos');
fclose(fileID);
% generar grafo de visibilidad analizando conectividad entre nodos
[matAdy,matPesos] = conectividad(nodos,vecsObs,pointsObs);
% guardar variables en documentos
str = '';
for i=1:size(matAdy,1)
    if(i ~= size(matAdy))
        str = strcat(str,'%f'," ");
    else
        str = strcat(str,'%f');
    end
end
str = strcat(str,'\n');
fileID = fopen('docs/matPesos.txt','w');
fprintf(fileID,str,matPesos');
fclose(fileID);
end
```

### 1.16. Función de Movimiento del Robot *movObtenerProd.m*

```
function[pos,infr,h,dir]=movObtenerProd(meta,pos,vecSensores,knownVObs,
    unknownVObs,robot,infr,h,dir,grafico)
fin = true;
while fin
%     calculo de velocidad sin considerar repulsion
     [V_control,distSensores]=calcVelocidadNoRep(meta,pos,vecSensores,
    knownVObs,unknownVObs,robot);
     %     Actualizar posicion segun la velocidad -> equivalente a odometria
     pos = pos + V_control*robot.t_actualizacion;

%     plot robot
     [infr,h,dir] = graficoSimulacion(pos,V_control,distSensores,robot,
    grafico,infr,h,dir);

%     condicion de terminacion
     if(norm(meta-pos)<robot.dist_final)
         break
     end
end
% equivalente a movimiento de agarre de producto
fprintf('Tomando Producto\n')
pause(2)
end
```

## 1.17. Función de Generación de Nuevo Grafo *nuevoGrafo.m*

```
function[nodosNew,matAdyNew,matPesosNew] = nuevoGrafo(goals,nodos,matAdy,
    matPesos,vecObs,vertObs,coordIni,grafico)
% Nueva lista de nodos
nodosNew = [coordIni; goals; nodos];
nNodosNew = size(nodosNew,1);
nNodosAgregados = size(goals,1)+1;
% Nueva matriz de adyacencia
matAdyNew = zeros(nNodosNew);
matAdyNew(nNodosAgregados+1:end,nNodosAgregados+1:end) = matAdy;
```

```
% Nueva matriz de pesos
matPesosNew = zeros(nNodosNew);
matPesosNew(nNodosAgregados+1:end,nNodosAgregados+1:end) = matPesos;
% Agregar nuevos nodos de metas a la matriz de adyacencia y a la matriz de
% pesos
for i=1:nNodosAgregados
    for j=i+1:nNodosNew
%         revisar si arista a agregar no tiene colision
        if(~vertColision2(nodosNew(i,:),nodosNew(j,:),vecObs,vertObs))
%             agregar a matriz de adyacencia y matriz de pesos
            matAdyNew(i,j) = j;
            matAdyNew(j,i) = i;
            matPesosNew(i,j) = norm(nodosNew(i,:)-nodosNew(j,:));
            matPesosNew(j,i) = matPesosNew(i,j);
%             plot aristas nuevas
            plot([nodosNew(i,1) nodosNew(j,1)],[nodosNew(i,2) nodosNew(j,2)
    ],'b','Linewidth',0.7)
            pause(grafico.tConectividad)
        end
    end
end
end
```

## 1.18. Función de Generación de Obstáculos Desconocidos *obstDesconocidos.m*

```
function [knownVObs,unknownVObs]=obstDesconocidos(pointsObs,unknownObs)
% Generacion de vectores de los obstaculos conocidos
% knownVObs = [vecX1 vecY1 pXini1 pYini1
%              vecX2 vecY2 pXini2 pYini2
%              vecX3 vecY3 pXini3 pYini3
%              vecX4 vecY4 pXini4 pYini4];
knownVObs = zeros(size(pointsObs,1)*4,4);
for i=1:size(pointsObs,1)
    knownVObs(i*4-3:i*4,:) = [0 pointsObs(i,4)-pointsObs(i,3) pointsObs(i,1)
    pointsObs(i,3)
                 pointsObs(i,2)-pointsObs(i,1) 0 pointsObs(i,1) pointsObs(i
    ,4)
```

```
                0 pointsObs(i,3)-pointsObs(i,4) pointsObs(i,2) pointsObs(i
    ,4)
                pointsObs(i,1)-pointsObs(i,2) 0 pointsObs(i,2) pointsObs(i
    ,3)];
end
% Generacion de vectores de los obstaculos no conocidos
% Plot obstaculos
if(~isempty(unknownObs))
    Plot = [unknownObs(:,1) unknownObs(:,1) unknownObs(:,2) unknownObs(:,2)
    unknownObs(:,1) unknownObs(:,3) unknownObs(:,4) unknownObs(:,4)
    unknownObs(:,3) unknownObs(:,3)];
    for i = 1:size(Plot,1)
        plot(Plot(i,1:5), Plot(i,6:10), 'Color',[0.5 0.5 0], 'LineWidth', 2)
        fill([pointsObs(i,1) pointsObs(i,1:2) pointsObs(i,2)],[pointsObs(i
    ,3:4) pointsObs(i,4) pointsObs(i,3)],[0.5 0.5 0.5])
    end
    % unknownVObs = [vecX1 vecY1 pXini1 pYini1
    %                vecX2 vecY2 pXini2 pYini2
    %                vecX3 vecY3 pXini3 pYini3
    %                vecX4 vecY4 pXini4 pYini4];
end
unknownVObs = zeros(size(unknownObs,1)*4,4);
for i=1:size(unknownObs,1)
    unknownVObs(i*4-3:i*4,:) = [0 unknownObs(i,4)-unknownObs(i,3) unknownObs
    (i,1) unknownObs(i,3)
                                unknownObs(i,2)-unknownObs(i,1) 0 unknownObs
    (i,1) unknownObs(i,4)
                                0 unknownObs(i,3)-unknownObs(i,4) unknownObs
    (i,2) unknownObs(i,4)
                                unknownObs(i,1)-unknownObs(i,2) 0 unknownObs
    (i,2) unknownObs(i,3)];
end
end
```

## 1.19. Función de Graficación de Grafo *plotGrafo.m*

```
function[]=plotGrafo(matAdy,nodos,grafico)
```

```
% plot aristas de grafo de visibilidad
for i=1:size(matAdy,1)-1
    for j=i+1:size(matAdy,1)
        if(matAdy(i,j)~=0)
            plot([nodos(i,1) nodos(j,1)],[nodos(i,2) nodos(j,2)],'b','
    Linewidth',0.7)
        end
    end
end
pause(grafico.tConectividad)
end
```

### 1.20. Función de Graficación de Nodos *plotNodos.m*

```
function[]=plotNodos(nodos,grafico)
% Graficar nodos
for i = 1:size(nodos,1)
    plot(nodos(i,1),nodos(i,2),'ro','LineWidth', 2)
end
pause(grafico.tNodos)
end
```

### 1.21. Función de Graficación de Obstáculos *plotObs.m*

```
function []=plotObs(pointsObs,grownObs,grownLimites,grafico)
pause(grafico.tIni)
% plot obstaculos
Plot = [pointsObs(:,1) pointsObs(:,1) pointsObs(:,2) pointsObs(:,2)
    pointsObs(:,1) pointsObs(:,3) pointsObs(:,4) pointsObs(:,4) pointsObs
    (:,3) pointsObs(:,3)];
for i = 1:size(Plot,1)
    plot(Plot(i,1:5), Plot(i,6:10), 'black', 'LineWidth', 2)
    fill([pointsObs(i,1) pointsObs(i,1:2) pointsObs(i,2)],[pointsObs(i,3:4)
    pointsObs(i,4) pointsObs(i,3)],[0.5 0.5 0.5])
end
pause(grafico.tObs)
% Plot obstaculos con offset
```

```
Plot = [grownObs(:,1) grownObs(:,1) grownObs(:,2) grownObs(:,2) grownObs
    (:,1) grownObs(:,3) grownObs(:,4) grownObs(:,4) grownObs(:,3) grownObs
    (:,3)];
for i = 1:size(Plot,1)
    plot(Plot(i,1:5), Plot(i,6:10), 'black', 'LineWidth', 2, 'LineStyle',':'
    )
end
% Plot limites con offset
Plot = [grownLimites(1,1) grownLimites(1,1) grownLimites(1,2) grownLimites
    (1,2) grownLimites(1,1) grownLimites(1,3) grownLimites(1,4) grownLimites
    (1,4) grownLimites(1,3) grownLimites(1,3)];
for i = 1:size(Plot,1)
    plot(Plot(i,1:5), Plot(i,6:10), 'black', 'LineWidth', 2, 'LineStyle',':'
    )
end
pause(grafico.tObs)
end
```

### 1.22. Función de Repulsión *repulsion.m*

```
function [Frep] = repulsion(sensor,vecSensor,robot)
% calculo de fuerza de repulsion
Frep = [0 0];
for i=1:size(sensor,1)
    if(sensor(i)<robot.rango_sensor)
        F = robot.c_repulsion/2*(1/(sensor(i)-robot.radio_robot))^2;
        Frep = Frep + vecSensor(i,:)/norm(vecSensor(i,:))*F;
    end
end
end
```

### 1.23. Función de Simulación de Sensores *sensor.m*

```
function [distancias]=sensor(knownObs,unknownObs,pos,robot,vecLidar)
distancias = zeros(size(vecLidar,1),1) + robot.rango_sensor;
for i=1:size(distancias,1)
    for j=1:size(knownObs,1)
```

```
5       %     Producto cruz vecLidar X vecAristaObstaculo
6       rXs = vecLidar(i,1)*knownObs(j,2) - vecLidar(i,2)*knownObs(j,1);
7       %     Vector desde punto inicial vecAristaObstaculo (q) hacia punto
8       %     incial vecLidar (p->pos)
9       qp = knownObs(j,3:4) - pos;
10      %     Proporcion en la que los vectores se cruzan de acuerdo a
11      %     vecLidar
12      t = (qp(1)*knownObs(j,2)-qp(2)*knownObs(j,1))/rXs;
13      %     Proporcion en la que los vectores se cruzan de acuerdo a
14      %     vecAristaObstaculo
15      u = (qp(1)*vecLidar(i,2)-qp(2)*vecLidar(i,1))/rXs;
16      %     Chequeo de interseccion de vector de lidar con algun vertice
17      %     de algun obstaculo
18      if(rXs~=0 && t>=0 && t<=1 && u>=0 && u<=1 && distancias(i)>robot.
    rango_sensor*t)
19          distancias(i) = robot.rango_sensor*t;
20      end
21   end
22   for j=1:size(unknownObs,1)
23       %     Producto cruz vecLidar X vecAristaObstaculo no conocido
24      rXs = vecLidar(i,1)*unknownObs(j,2) - vecLidar(i,2)*unknownObs(j,1);
25      %     Vector desde punto inicial vecAristaObstaculo no conocido (q)
    hacia punto
26      %     incial vecLidar (p->pos)
27      qp = unknownObs(j,3:4) - pos;
28      %     Proporcion en la que los vectores se cruzan de acuerdo a
29      %     vecLidar
30      t = (qp(1)*unknownObs(j,2)-qp(2)*unknownObs(j,1))/rXs;
31      %     Proporcion en la que los vectores se cruzan de acuerdo a
32      %     vecAristaObstaculo
33      u = (qp(1)*vecLidar(i,2)-qp(2)*vecLidar(i,1))/rXs;
34      %     Chequeo de interseccion de vector de lidar con algun vertice
35      %     de algun obstaculo
36      if(rXs~=0 && t>=0 && t<=1 && u>=0 && u<=1 && distancias(i)>robot.
    rango_sensor*t)
37          distancias(i) = robot.rango_sensor*t;
```

```
38        end
39     end
40 end
41 end
```

## 1.24. Función de Condición a Siguiente Meta *siguienteGoal.m*

```
1  function [contCamino,contGoals,pos,infr,h,dir]=siguienteGoal(nodosNew,
       contCamino,camino,pos,contGoals,goalsOrder,posProd,prodPedidos,robot,
       vecSensores,knownVObs,unknownVObs,infr,h,dir,grafico)
2  % se calcula la distancia hacia la meta actual
3  dGoalActual = norm(nodosNew(camino(contCamino),:)-pos);
4  % si la distancia es menor a condicion establecida
5  if(dGoalActual<robot.dist_final)
6  %     si el nodo actual es un nodo de la meta del producto pedido
7      if(camino(contCamino) == goalsOrder(contGoals))
8  %         equivalente de movimiento de mecanismo de agarre al nivel del
       producto
9          str = strcat('Mecanismo Nivel'," ",num2str(posProd(goalsOrder(
       contGoals)-1,4)),'\n');
10         fprintf(str)
11         pause(4)
12 %         movimiento del robot hacia posicion para agarrar al producto, no
13 %         se considera fuerza de repulsion ya que el robot se acerca mucho
14 %         a estantes
15         [pos,infr,h,dir] = movObtenerProd(posProd(prodPedidos(goalsOrder(
       contGoals)-1),1:2),pos,vecSensores,knownVObs,unknownVObs,robot,infr,h,dir
       ,grafico);
16 %         se aumenta contador de metas de productos pedidos
17         contGoals = contGoals + 1;
18     end
19 %     se aumenta contador de camino de trayectoria general
20     contCamino = contCamino + 1;
21 end
22 end
```

## 1.25. Función de Generación de Trayectoria *trayectoria.m*

```
function [camino,goalsOrder]=trayectoria(goals,matPesosNew,nodosNew,grafico)
nGoals = size(goals,1);
% nodo inicial de trayectoria
Ini = 1;
% camino comienza en nodo 1
camino = [1];
goalsNoRevisados = 2:nGoals+1;
goalsOrder = [];
% bucle analiza todas las metas
while ~isempty(goalsNoRevisados)
%     comienza con un costo infinito a la meta mas cercana
    costo = inf;
%     se analiza todas las metas para obtener el costo mas bajo
    for i=1:nGoals
        % Algoritmo de Busqueda A*
        [caminoTemp,costoTemp] = AStar(matPesosNew, nodosNew, Ini,
   goalsNoRevisados(i));
%         si el costo calculado para la meta actual es mejor que el costo
   anterior
%         se reemplaza como la mejor meta para ir
        if(costoTemp < costo)
            costo = costoTemp;
            goalTemp = goalsNoRevisados(i);
            caminoTempM = caminoTemp;
        end
    end
%     se agrega el mejor camino encontrado
    camino = [camino; caminoTempM];
%     se elimina la mejor meta de la lista de no revisados
    goalsNoRevisados(goalsNoRevisados==goalTemp)=[];
%     se agrega la meta en la lista que indica el orden de las metas a
   visitar
    goalsOrder = [goalsOrder, goalTemp];
%     nodo inicial se convierte en la ultima meta
    Ini = goalTemp;
```

```
    nGoals = nGoals - 1;
end
% calculo de camino para regresar a las coordenadas iniciales del robot
[caminoTemp,costoTemp] = AStar(matPesosNew, nodosNew, Ini, 1);
camino = [camino; caminoTemp];
goalsOrder = [goalsOrder, 1];
% plot trayectoria para visitar todas las metas a los productos pedidos
for i=1:size(camino,1)-1
    plot([nodosNew(camino(i),1) nodosNew(camino(i+1),1)],[nodosNew(camino(i)
    ,2) nodosNew(camino(i+1),2)],'red','Linewidth',2)
    pause(grafico.tPath)
end
end
```

### 1.26. Función de Detección de Colisión *vertColision2.m*

```
% Adaptacion de algoritmo de Ronald Goldman para deteccion de interseccion
   de dos segmentos de linea
function [check]=vertColision2(nodo1,nodo2,vecObs,vertObs)
% generacion de vector de arista a analizar
vecVertice = [nodo2(1)-nodo1(1),nodo2(2)-nodo1(2)];
nObs = size(vecObs,1)/6;
for i=1:nObs
%     generacion de vectores y puntos de obstaculo, 1 vector por cada arista
    del obstaculo
%     y dos vectores diagonales que atraviesan el obstaculo para analizar si
    arista
%     de grafo no se cruza dentro del obstaculo
    vecs = vecObs(i*6-5:i*6,:);
    verts = vertObs(i*6-5:i*6,:);
%     deteccion de colision
    countCheck = 0;
    for j = 1:size(vecs,1)
%         verificacion de arista de grafo no cruza por dentro de obstaculos
        if(((vecs(j,1)==nodo1(1)&&vecs(j,2)==nodo1(2))||(vecs(j,1)==nodo2(1)
   &&vecs(j,2)==nodo2(2))) && (nodo1(1)~=nodo2(1)&&nodo1(2)~=nodo2(2)))
            countCheck = countCheck + 1;
```

```
        end
%         algoritmo de Ronald Goldman para deteccion de interseccion
        a = verts(j,:)-nodo1;
        b = modCross(vecVertice,vecs(j,:));
        t = modCross(a,vecs(j,:))/b;
        u = modCross(a,vecVertice)/b;
        if(b~=0 && t>0 && t<1 && u>0 && u<1)
            check = true;
            return
        end
    end
    if(countCheck>=2)
        check = true;
        return
    end
end
check = false;
end
```

## 2. CÓDIGOS DE PROGRAMACIÓN EN RASPBERRY

### 2.1. Programa Principal *app.py*

```
import smbus2 as smbus
import RPi.GPIO as GPIO
from flask import Flask, render_template
from threading import Thread
import time
import math
import csv
import Adafruit_SSD1306
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont


```

```
14 # caracterisiticas del robot
15 class Robot:
16     metodo_orientacion = 1
17     coord_ini = (190, 190, 0)  # mm
18     radio_robot = 150  # mm
19     n_sensores = 14
20     rango_sensor = 160  # mm
21     radio_sensor = 110.2  # mm
22     max_velocidad = 30  # mm/s
23     c_atraccion = radio_robot/12
24     max_velocidad_angular = math.pi/20  # rad/s
25     c_atraccion_angular = 5*math.pi/180  # rad
26     c_repulsion = 4000
27     c_filtro_ema = 0.4
28     dist_final = 2  # mm
29     ang_final = 0.5*math.pi/180  # rad
30     dist_estante = 1  # mm
31     radio_rueda = 48  # mm
32     l = 57  # mm
33     w = 79.6  # mm
34
35
36 # Definir pines fin de carrera
37 GPIO_FCX = [5, 11]
38 GPIO_FCY = [13, 6]
39 # Definir tiempo de espera entre fases
40 t_espera = 0.003
41 # Definir pines de motores a pasos
42 GPIO_StepX = [14, 15, 18, 23]
43 GPIO_StepY = [24, 25, 8, 1]
44
45 # set GPIO Pins
46 GPIO_TRIGGER = 4
47 GPIO_ECHO = 17
48 GPIO_SELECT = [21, 20, 16, 12]
49
```

```
50 # Definir numeracion de pines
51 GPIO.setmode(GPIO.BCM)
52 # Deshabilitar avisos de GPIO
53 GPIO.setwarnings(False)
54
55 # set GPIO direction (IN / OUT)
56 GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
57 GPIO.setup(GPIO_ECHO, GPIO.IN)
58 for i in GPIO_SELECT:
59     GPIO.setup(i, GPIO.OUT)
60 dist_medidas = [0] * Robot.n_sensores
61
62 # Definir pines de fin de carrera como entrada
63 GPIO.setup(7, GPIO.IN)
64 GPIO.setup(GPIO_FCX[0], GPIO.IN, GPIO.PUD_DOWN)
65 GPIO.setup(GPIO_FCX[1], GPIO.IN, GPIO.PUD_DOWN)
66 GPIO.setup(GPIO_FCY[0], GPIO.IN, GPIO.PUD_DOWN)
67 GPIO.setup(GPIO_FCY[1], GPIO.IN, GPIO.PUD_DOWN)
68 # Definir pines de motores a pasos como salidas
69 for pin in GPIO_StepX:
70     GPIO.setup(pin, GPIO.OUT)
71     GPIO.output(pin, False)
72 for pin in GPIO_StepY:
73     GPIO.setup(pin, GPIO.OUT)
74     GPIO.output(pin, False)
75 # Definir secuencia de fases motores
76 Seq = [[0]*4]*4
77 Seq[0] = [0, 0, 0, 1]
78 Seq[1] = [0, 0, 1, 0]
79 Seq[2] = [0, 1, 0, 0]
80 Seq[3] = [1, 0, 0, 0]
81
82 # Configurar display
83 # RST no se usa
84 RST = None
85 # 128x32 display con I2C
```

```
86 disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST)
87 # Inicializacion libreria
88 disp.begin()
89 # Clear display.
90 disp.clear()
91 disp.display()
92 width = disp.width
93 height = disp.height
94 image = Image.new('1', (width, height))
95 draw = ImageDraw.Draw(image)
96 # Constantes
97 padding = -1
98 top = padding
99 bottom = height-padding
100 x = 0
101 # Font
102 font = ImageFont.truetype('/home/pi/Documents/Python/Arimo-Bold.ttf', 11)
103 # Borrar imagen
104 draw.rectangle((0, 0, width, height), outline=0, fill=0)
105 # Escirbir Texto
106 draw.text((x+10, top+10), "ESPERANDO PEDIDO...",  font=font, fill=255)
107 # Display
108 disp.image(image)
109 disp.display()
110
111 # I2C addresses para los Atmega de cada motor
112 addrMotor = [0x8, 0x9, 0x10, 0x11]
113 # Indica cual I2C bus del RPI /dev/ic2-1
114 bus = smbus.SMBus(1)
115
116 app = Flask(__name__)
117
118
119 # funcion para extraer informacion de documentos
120 def leer_docs():
121     # l mites del entrno
```

```
122     with open('docs/limites.txt', 'r') as f:
123         limites = [[float(num) for num in line.split(' ')] for line in f]
124         limites = limites[0]
125     # vectores y puntos de obstaculos conocidos
126     with open('docs/vecsObstaculos.txt', 'r') as f:
127         vecs_obs = [[float(num) for num in line.split(' ')] for line in f]
128     with open('docs/pointsObstaculos.txt', 'r') as f:
129         points_obs = [[float(num) for num in line.split(' ')] for line in f]
130     # nodos del grafo de visibilidad
131     with open('docs/nodos.txt', 'r') as f:
132         nodos = [[float(num) for num in line.split(' ')] for line in f]
133     # matriz de pesos del grafo de visibilidad
134     with open('docs/matPesos.txt', 'r') as f:
135         mat_pesos = [[float(num) for num in line.split(' ')] for line in f]
136     # posiciones previas a agarrar el producto
137     with open('docs/posPrev.txt', 'r') as f:
138         pos_prev = [[float(num) for num in line.split(' ')] for line in f]
139     # posicion para agarrar el producto
140     with open('docs/posProd.txt', 'r') as f:
141         pos_prod = [[float(num) for num in line.split(' ')] for line in f]
142     items = []
143     with open('docs/productos.csv', mode='r') as file:
144         reader = csv.DictReader(file)
145         for l in reader:
146             items.append({})
147             pose = []
148             for key, value in l.items():
149                 if key == 'POSE_X':
150                     pose.append(float(value))
151                 elif key == 'POSE_Y':
152                     pose.append(float(value))
153                 elif key == 'ORIENTACION':
154                     pose.append(float(value))
155                     items[-1]['posicion'] = pose
156                 elif key == 'ID':
157                     items[-1]['id'] = int(value)
```

```
158                 elif key == 'NOMBRE':
159                     items[-1]['name'] = value
160                 elif key == 'PRECIO':
161                     items[-1]['price'] = float(value)
162                 elif key == 'NIVEL':
163                     items[-1]['nivel'] = int(value)
164                 elif key == 'PASILLO':
165                     items[-1]['pasillo'] = int(value)
166                     items[-1]['state'] = False
167     return limites, vecs_obs, points_obs, nodos, mat_pesos, pos_prev,
    pos_prod, items
168
169
170 # Funcion para apagar todas las bobinas del motor
171 def apagar_motores():
172     for pin in GPIO_StepX:
173         GPIO.output(pin, False)
174     for pin in GPIO_StepY:
175         GPIO.output(pin, False)
176
177
178 # Funcion movimiento a posicion inical en X
179 def pos_ini_x():
180     n = 0
181     while not GPIO.input(GPIO_FCX[0]):
182         for pin in range(0, 4):
183             GPIO.output(GPIO_StepX[pin], Seq[n % 4][pin])
184         time.sleep(t_espera)
185         n = n - 1
186     apagar_motores()
187
188
189 # Funcion movimiento a posicion final en X
190 def pos_fin_x():
191     n = 0
192     while not GPIO.input(GPIO_FCX[1]):
```

```
193         for pin in range(0, 4):
194             GPIO.output(GPIO_StepX[pin], Seq[n % 4][pin])
195         time.sleep(t_espera)
196         n = n + 1
197     apagar_motores()
198 
199 
200 # Funcion movimiento a posicion inicial en Y
201 def pos_ini_y():
202     n = 0
203     while not GPIO.input(GPIO_FCY[0]):
204         for pin in range(0, 4):
205             GPIO.output(GPIO_StepY[pin], Seq[n % 4][pin])
206         time.sleep(t_espera)
207         n = n - 1
208     apagar_motores()
209 
210 
211 # Funcion movimiento a posicion final en Y
212 def pos_fin_y():
213     n = 0
214     while not GPIO.input(GPIO_FCY[1]):
215         for pin in range(0, 4):
216             GPIO.output(GPIO_StepY[pin], Seq[n % 4][pin])
217         time.sleep(t_espera)
218         n = n + 1
219     apagar_motores()
220 
221 
222 # Funcion secuencia mecanismo de agarre
223 def secuencia():
224     print('-> Mecanismo en movimiento!')
225     # Borrar imagen
226     draw.rectangle((0, 0, width, height), outline=0, fill=0)
227     # Escirbir Texto
228     draw.text((x + 10, top + 5), "MECANISMO EN", font=font, fill=255)
```

```
229     draw.text((x + 10, top + 15), "MOVIMIENTO", font=font, fill=255)
230     # Display
231     disp.image(image)
232     disp.display()
233     # Posici n inicial mecanismo de agarre
234     pos_fin_x()
235     pos_fin_y()
236     pos_ini_x()
237     pos_ini_y()
238     apagar_motores()
239
240 # costo de pedido para app web
241 costo_total = 0
242 # condicion de programa del robot corriendo
243 running = 0
244 # calculo vectores de direccion de sensores
245 vecs_sensores = [[-math.cos(2 * math.pi / Robot.n_sensores * -i), -math.sin
        (2 * math.pi / Robot.n_sensores * -i)] for i in range(0, Robot.n_sensores
        )]
246 giro_ruedas = [0] * 4
247 # obtener variables precalculadas de documentos
248 limites, vecs_obs, points_obs, nodos, mat_pesos, pos_prev, pos_prod, items =
        leer_docs()
249 # Secuencia a posicion inicial mecanismo
250 pos_ini_y()
251 pos_ini_x()
252 apagar_motores()
253
254 # pagina principal
255 @app.route('/')
256 @app.route('/home')
257 def home_page():
258     return render_template('home.html')
259
260
261 # pagina de mercado
```

```
262 @app.route('/market')
263 def market_page():
264     return render_template('market.html',items=items)
265 
266 
267 # pagina de redireccion para a adir/remover un producto del mercado
268 @app.route("/mercado/<item>/<action>")
269 def action(item, action):
270     global items, costo_total
271     item = int(item)-1
272     # se cambia estado de compra del producto segun la accion a adir/
    remover y se actualiza el costo total
273     if action == 'add':
274         items[item]['state'] = True
275         costo_total += items[item]['price']
276     else:
277         items[item]['state'] = False
278         costo_total -= items[item]['price']
279     return render_template('market.html', items=items)
280 
281 
282 # pagina para remover producto de carrito de compras
283 @app.route("/carrito/<item>/<action>")
284 def remover(item, action):
285     global items, costo_total
286     item = int(item)-1
287     # se actualiza estado de costo de producto y precio de pedido
288     if action == 'remove':
289         items[item]['state'] = False
290         costo_total -= items[item]['price']
291     return render_template('compras.html', items=items, costo_total=
    costo_total)
292 
293 
294 # pagina para boton de acerca de
295 @app.route('/acerca')
```

```
296 def acerca_page():
297     return render_template('acerca_de.html')
298 
299 
300 # pagina de carrito o compras hechas
301 @app.route('/compras')
302 def compras_page():
303     global items, costo_total
304     return render_template('compras.html', items=items, costo_total=
        costo_total)
305 
306 
307 # pagina de checkout, espera mientras robot se mueve
308 @app.route('/checkout')
309 def checkout_page():
310     global running
311     # inicializacion de thread para programa del robot
312     thread = Thread(target=programa)
313     # running=0 robot no moviendos, running=1 robot moviendose, running=2
        robot termino secuencia
314     if running == 0:
315         # programa principal del robot
316         thread.start()
317         # redireccion a pagina de checkout
318         return render_template('running.html')
319     elif running == 1:
320         # redireccion a pagina de checkout
321         return render_template('running.html')
322     elif running == 2:
323         running = 0
324         # redireccion a pagina de inicio
325         return render_template('home.html')
326 
327 
328 # generacion de metas segun productos pedidos
329 def gen_goals(prod_pedidos, pos_prev):
```

```
330     goals = []
331     for pedidos in prod_pedidos:
332         goals.append(pos_prev[pedidos][0:2])
333     return goals
334 
335 
336 # m dulo del producto cruz 2D
337 def mod_prod_cruz(vec1, vec2):
338     mod = vec1[0]*vec2[1]-vec1[1]*vec2[0]
339     return mod
340 
341 
342 # deteccion de colision
343 def vert_colision(nodo1, nodo2, vecs_obs, points_obs):
344     # adaptacion de algoritmo de Ronald Goldman para deteccion de
    interseccion de dos segmentos de linea
345     # vector de arista de grafo a analizar
346     vec_vertice = [nodo2[0]-nodo1[0], nodo2[1]-nodo1[1]]
347     # se analiza colisiones en todos los obstaculos
348     for i in range(0, int(len(vecs_obs)/6)):
349         # generacion de vectores de aristas de cada obstaculos
350         vecs = vecs_obs[i*6:i*6+6]
351         # generacion de coordenadas iniciales de vector de cada obstaculo
352         verts = points_obs[i*6:i*6+6]
353         # verificacion de arista de grafo no cruza por dentro de obstaculos
354         count_check = 0
355         for j in range(0, len(vecs)):
356             if ((vecs[j][0] == nodo1[0] and vecs[j][1] == nodo1[1]) or (vecs
    [j][0] == nodo2[0] and vecs[j][1] == nodo2[1])) and (nodo1[0] != nodo2[0]
     and nodo1[1] != nodo2[1]):
357                 count_check += 1
358             # algoritmo de Ronald Goldman para deteccion de interseccion
    entre segmentos de linea
359             a = [verts[j][0]-nodo1[0], verts[j][1]-nodo1[1]]
360             b = mod_prod_cruz(vec_vertice, vecs[j])
361             if b != 0:
```

```
362                 t = mod_prod_cruz(a, vecs[j])/b
363                 u = mod_prod_cruz(a, vec_vertice)/b
364                 if 0 < t < 1 and 0 < u < 1:
365                     check = True
366                     return check
367         if count_check >= 2:
368             check = True
369             return check
370     check = False
371     return check
372 
373 
374 # generacion de nuevo grafo de visibilidad
375 def nuevo_grafo(goals, nodos, mat_pesos, vecs_obs, points_obs):
376     # crear nueva lista de nodos
377     nodos_new = [[Robot.coord_ini[0], Robot.coord_ini[1]]]
378     for i in goals:
379         nodos_new.append(i)
380     for i in nodos:
381         nodos_new.append(i)
382     n_nodos_new = len(nodos_new)
383     n_nodos_agregados = len(goals)+1
384     # crear nueva matriz de pesos de grafo
385     mat_pesos_new = [[0.0 for j in range(0, n_nodos_new)] for i in range(0,
    n_nodos_new)]
386     # agregar matriz de pesos base a la nueva matriz
387     for i in range(n_nodos_agregados, n_nodos_new):
388         for j in range(n_nodos_agregados, n_nodos_new):
389             mat_pesos_new[i][j] = mat_pesos[i-n_nodos_agregados][j-
    n_nodos_agregados]
390     # analizar conectividad de nuevos nodos y agregar a matriz de pesos
391     for i in range(0, n_nodos_agregados):
392         for j in range(i+1, n_nodos_new):
393             if not vert_colision(nodos_new[i], nodos_new[j], vecs_obs,
    points_obs):
394                 mat_pesos_new[i][j] = math.hypot(nodos_new[i][0]-nodos_new[j
```

```
    ][0], nodos_new[i][1]-nodos_new[j][1])
395                 mat_pesos_new[j][i] = mat_pesos_new[i][j]
396     return nodos_new, mat_pesos_new
397
398
399 # algoritmo de busqueda en grafos A*
400 def a_star(mat_pesos, nodos, ini, fin):
401     n_nodos = len(nodos)
402     # calculo de heuristica como la distancia euclidiana hasta la meta
403     heuristica = [math.hypot(nodos[i][0]-nodos[fin][0], nodos[i][1]-nodos[
    fin][1]) for i in range(0, n_nodos)]
404     # inicializacion de nodos revisados y no revisados
405     nodos_revisados = []
406     nodos_no_revisados = [ini]
407     # inicializacion de lista de costos minimos para cada nodo
408     min_costos = [math.inf]*n_nodos
409     min_costos[ini] = 0.0
410     # inicializacion de arbol del grafo
411     arbol = [None]*n_nodos
412     # inicializacion de costos estimados para cada nodo
413     costo_estimado = [math.inf]*n_nodos
414     # bucle se ejecuta hasta que no haya nodos vecinos a arbol para analizar
    o hasta que se llego a la meta
415     while len(nodos_no_revisados) > 0:
416         # se elige nodo en la lista nodos no revisados
417         current = nodos_no_revisados[0]
418         # se agrega nodo a lista de revisados
419         nodos_revisados.append(current)
420         # se elimina nodo de lista de no revisados
421         nodos_no_revisados.remove(current)
422         # si el nodo es igual a la meta se termina bucle
423         if current == fin:
424             break
425         # se analiza todos los nodos
426         for vecino in range(0, n_nodos):
427             # se chequea cual nodo es vecino y si no se encuentra en lista
```

```
    de revisados
428             if (mat_pesos[current][vecino] != 0 or (nodos[current][0] ==
    nodos[vecino][0] and nodos[current][1] == nodos[vecino][1])) and vecino
    not in nodos_revisados:
429                 # se almacena el costo temporal para ir a nodo vecino
430                 costo_temporal = min_costos[current]+mat_pesos[current][
    vecino]
431                 # si este costo es menor al costo almacenado previamente, se
     reemplaza valores
432                 if costo_temporal < min_costos[vecino]:
433                     min_costos[vecino] = costo_temporal
434                     arbol[vecino] = current
435                     # se agrega nodo vecino a la lista de no revisados en
    orden segun el costo estimado
436                     costo_estimado[vecino] = min_costos[vecino]+heuristica[
    vecino]
437                     n_nodos_no_revisados = len(nodos_no_revisados)
438                     pos = 0
439                     while pos < n_nodos_no_revisados and costo_estimado[
    vecino] > costo_estimado[nodos_no_revisados[pos]]:
440                         pos += 1
441                     nodos_no_revisados.insert(pos, vecino)
442     # recursividad para hallar el camino de nodo inicial a final segun el
    arbol
443     camino = [fin]
444     while camino[0] != ini:
445         camino.insert(0, arbol[camino[0]])
446     camino.remove(ini)
447     costo_total = min_costos[fin]
448     return camino, costo_total
449
450
451 # generacion de trayectoria
452 def trayectoria(goals, mat_pesos_new, nodos_new):
453     n_goals = len(goals)
454     # nodo inicial de trayectoria
```

```
455     ini = 0
456     # camino comienza en nodo 1
457     camino = [0]
458     # bucle analiza todas las metas para generar la trayectoria
459     goals_orden = []
460     goals_no_revisados = [i for i in range(1, n_goals+1)]
461     while len(goals_no_revisados) > 0:
462         # meta mas cercana comienza con costo infinito
463         costo = math.inf
464         # se analiza todas las metas para obtener el costo m s bajo
465         for i in range(0, n_goals):
466             # algoritmo de busqueda en grafos A*
467             camino_temp, costo_temp = a_star(mat_pesos_new, nodos_new, ini,
    goals_no_revisados[i])
468             # si el costo calculado es menor al costo previo almacenado se
    reemplaza como nueva mejor meta
469             if costo_temp < costo:
470                 costo = costo_temp
471                 goal_temp = goals_no_revisados[i]
472                 camino_temp_aux = camino_temp
473         # se agrega el mejor camino encontrado al camino general
474         for i in camino_temp_aux:
475             camino.append(i)
476         # se elimina la meta de las metas no revisadas
477         goals_no_revisados.remove(goal_temp)
478         # se agrega la meta en la lista del orden de las metas
479         goals_orden.append(goal_temp)
480         # nodo incial se convierte en la meta seleccionada
481         ini = goal_temp
482         n_goals -= 1
483     # c lculo del camino para retornar a las coordenadas de la posicion
    inicial
484     camino_temp, costo_temp = a_star(mat_pesos_new, nodos_new, ini, 0)
485     for i in camino_temp:
486         camino.append(i)
487     goals_orden.append(0)
```

```
488     return camino, goals_orden
489
490
491 # actualizacion de posicion por odometria
492 def odometria(pose, v_control):
493     giro_ruedas[0] = read_i2c(addrMotor[0])
494     giro_ruedas[1] = read_i2c(addrMotor[1])
495     giro_ruedas[2] = read_i2c(addrMotor[2])
496     giro_ruedas[3] = read_i2c(addrMotor[3])
497     giro_robot = Robot.radio_rueda/4*(-giro_ruedas[0]+giro_ruedas[1]+
    giro_ruedas[2]-giro_ruedas[3])/(Robot.l+Robot.w+1.738702903)
498     a = giro_ruedas[0]+giro_ruedas[1]+giro_ruedas[2]+giro_ruedas[3]
499     b = -giro_ruedas[0]+giro_ruedas[1]-giro_ruedas[2]+giro_ruedas[3]
500     if giro_robot == 0:
501         d_x = Robot.radio_rueda/4*a*1
502         d_y = Robot.radio_rueda/4*b*0.95
503     else:
504         d_x = Robot.radio_rueda/4*(math.sin(giro_robot)*a+(math.cos(
    giro_robot)-1)*b)/giro_robot*1
505         d_y = Robot.radio_rueda/4*((1-math.cos(giro_robot))*a+math.sin(
    giro_robot)*b)/giro_robot*0.95
506     pose[0] = pose[0]+math.cos(pose[2])*d_x-math.sin(pose[2])*d_y
507     pose[1] = pose[1]+math.sin(pose[2])*d_x+math.cos(pose[2])*d_y
508     pose[2] = pose[2]+giro_robot
509     if pose[2] > math.pi:
510         pose[2] = pose[2] - 2*math.pi
511     pose_print = [pose[0], pose[1], pose[2]*180/math.pi]
512     print("pose " + str([round(i, 2) for i in pose_print]))
513     return pose
514
515
516 # velocidad de giro
517 def atraccion_giro(orientacion_deseada, orientacion_actual):
518     # calcular error en orientacion
519     error_orientacion = orientacion_deseada-orientacion_actual
520     if abs(error_orientacion) > math.pi:
```

```
521         s = error_orientacion/abs(error_orientacion)
522         giro = -error_orientacion+s*math.pi
523     else:
524         giro = error_orientacion
525     # calculo modulo velocidad de giro
526     vel_angular = Robot.max_velocidad_angular * (1 - math.exp(-abs(giro) /
    Robot.c_atraccion_angular))
527     # determinacion de sentido de giro
528     if giro >= 0:
529         return vel_angular
530     else:
531         return -vel_angular
532
533
534 # velocidad lineal
535 def atraccion(goal, pose, orientacion_deseada):
536     # calculo velocidad lineal de atraccion
537     v_atraccion = Robot.max_velocidad * (1 - math.exp(-math.hypot(goal[1] -
    pose[1], goal[0] - pose[0]) / Robot.c_atraccion))
538     # angulo hacia meta
539     theta = math.atan2(goal[1]-pose[1], goal[0]-pose[0])
540     # calculo velocidad de giro para orientacion deseada
541     vel_giro = atraccion_giro(orientacion_deseada, pose[2])
542     return [math.cos(theta)*v_atraccion, math.sin(theta)*v_atraccion,
    vel_giro]
543
544
545 # velocidad sin considerar repulsion
546 def calc_velocidad_no_rep(meta, pose):
547     # calculo velocidad de control = atraccion
548     v_control = atraccion(meta[0:2], pose, meta[2])
549     # verificar que velocidad de control no supere velocidad maxima del
    robot
550     mod_v_control = math.hypot(v_control[0], v_control[1])
551     if mod_v_control > Robot.max_velocidad:
552         v_control = [Robot.max_velocidad * v_control[0] / mod_v_control,
```

```
553                     Robot.max_velocidad * v_control[1] / mod_v_control,
    v_control[2]]
554     return v_control
555
556
557 # filtro EMA
558 def ema_filtro(valores_filtrados, valores):
559     # filtro EMA - Exponential Moving Average
560     for i in range(0, Robot.n_sensores):
561         valores_filtrados[i] = Robot.c_filtro_ema * valores[i] + (1 - Robot.
    c_filtro_ema) * valores_filtrados[i]
562     return valores_filtrados
563
564
565 def sensor_select(sensor):
566     # Selecci n de sensores
567     if sensor == 1:
568         GPIO.output(GPIO_SELECT[0], False)
569         GPIO.output(GPIO_SELECT[1], False)
570         GPIO.output(GPIO_SELECT[2], False)
571         GPIO.output(GPIO_SELECT[3], False)
572     elif sensor == 2:
573         GPIO.output(GPIO_SELECT[0], True)
574         GPIO.output(GPIO_SELECT[1], False)
575         GPIO.output(GPIO_SELECT[2], False)
576         GPIO.output(GPIO_SELECT[3], False)
577     elif sensor == 3:
578         GPIO.output(GPIO_SELECT[0], False)
579         GPIO.output(GPIO_SELECT[1], True)
580         GPIO.output(GPIO_SELECT[2], False)
581         GPIO.output(GPIO_SELECT[3], False)
582     elif sensor == 4:
583         GPIO.output(GPIO_SELECT[0], True)
584         GPIO.output(GPIO_SELECT[1], True)
585         GPIO.output(GPIO_SELECT[2], False)
586         GPIO.output(GPIO_SELECT[3], False)
```

```
587     elif sensor == 5:
588         GPIO.output(GPIO_SELECT[0], False)
589         GPIO.output(GPIO_SELECT[1], False)
590         GPIO.output(GPIO_SELECT[2], True)
591         GPIO.output(GPIO_SELECT[3], False)
592     elif sensor == 6:
593         GPIO.output(GPIO_SELECT[0], True)
594         GPIO.output(GPIO_SELECT[1], False)
595         GPIO.output(GPIO_SELECT[2], True)
596         GPIO.output(GPIO_SELECT[3], False)
597     elif sensor == 7:
598         GPIO.output(GPIO_SELECT[0], False)
599         GPIO.output(GPIO_SELECT[1], True)
600         GPIO.output(GPIO_SELECT[2], True)
601         GPIO.output(GPIO_SELECT[3], False)
602     elif sensor == 8:
603         GPIO.output(GPIO_SELECT[0], True)
604         GPIO.output(GPIO_SELECT[1], True)
605         GPIO.output(GPIO_SELECT[2], True)
606         GPIO.output(GPIO_SELECT[3], False)
607     elif sensor == 9:
608         GPIO.output(GPIO_SELECT[0], False)
609         GPIO.output(GPIO_SELECT[1], False)
610         GPIO.output(GPIO_SELECT[2], False)
611         GPIO.output(GPIO_SELECT[3], True)
612     elif sensor == 10:
613         GPIO.output(GPIO_SELECT[0], True)
614         GPIO.output(GPIO_SELECT[1], False)
615         GPIO.output(GPIO_SELECT[2], False)
616         GPIO.output(GPIO_SELECT[3], True)
617     elif sensor == 11:
618         GPIO.output(GPIO_SELECT[0], False)
619         GPIO.output(GPIO_SELECT[1], True)
620         GPIO.output(GPIO_SELECT[2], False)
621         GPIO.output(GPIO_SELECT[3], True)
622     elif sensor == 12:
```

```
623         GPIO.output(GPIO_SELECT[0], True)
624         GPIO.output(GPIO_SELECT[1], True)
625         GPIO.output(GPIO_SELECT[2], False)
626         GPIO.output(GPIO_SELECT[3], True)
627     elif sensor == 13:
628         GPIO.output(GPIO_SELECT[0], False)
629         GPIO.output(GPIO_SELECT[1], False)
630         GPIO.output(GPIO_SELECT[2], True)
631         GPIO.output(GPIO_SELECT[3], True)
632     elif sensor == 14:
633         GPIO.output(GPIO_SELECT[0], True)
634         GPIO.output(GPIO_SELECT[1], False)
635         GPIO.output(GPIO_SELECT[2], True)
636         GPIO.output(GPIO_SELECT[3], True)
637 
638 
639 # obtencion distancia sensores
640 def ultrasonicos(dist_filtradas):
641     # lectura de sensores
642     for i in range(1, Robot.n_sensores + 1):
643         check = False
644         while not check:
645             distancia = 0
646             check = True
647             counter = 0
648             # set MUX selection
649             sensor_select(i)
650             # set Trigger to HIGH
651             GPIO.output(GPIO_TRIGGER, True)
652             # set Trigger after 0.01ms to LOW
653             time.sleep(0.00001)
654             GPIO.output(GPIO_TRIGGER, False)
655             time.sleep(0.000002)  # this time delay to avoid interference
    between trigger and echo
656 
657             StopTime = time.time()
```

```
658
659             while GPIO.input(GPIO_ECHO) == 0:
660                 # pass
661                 counter += 1
662                 if counter == 5000:
663                     check = False
664                     break
665             StartTime = time.time()
666
667             # save time of arrival
668             while GPIO.input(GPIO_ECHO) == 1:
669                 StopTime = time.time()
670
671             # time difference between start and arrival
672             TimeElapsed = StopTime - StartTime
673             if (TimeElapsed < 0):
674                 check = False
675             distancia = (TimeElapsed * 343000) / 2  # mm
676             time.sleep(0.002)
677         distancia = distancia + Robot.radio_sensor
678         if distancia <= Robot.radio_robot:
679             distancia = Robot.radio_robot+1
680         dist_medidas[i - 1] = distancia
681     # retornar datos filtrados
682     return ema_filtro(dist_filtradas, dist_medidas)
683
684
685 # calculo velocidad desplazamiento
686 def calc_velocidad(meta, pose, orientacion_deseada, dist_filtradas):
687     # calcular velocidad de atraccion
688     v_atraccion = atraccion(meta, pose, orientacion_deseada)
689     # obtener distancias a obstaculos
690     dist_filtradas = ultrasonicos(dist_filtradas)
691     # calcular velocidad de repulsion
692     v_repulsion = repulsion(dist_filtradas, pose[2])
693     # calcular velocidad de control = atraccion + repulsion
```

```
694     v_control = [v_atraccion[0]+v_repulsion[0], v_atraccion[1]+v_repulsion
    [1], v_atraccion[2]]
695     # verificar que velocidad de control no supere velocidad maxima del
    robot
696     mod_v_control = math.hypot(v_control[0], v_control[1])
697     if mod_v_control > Robot.max_velocidad:
698         v_control = [Robot.max_velocidad * v_control[0] / mod_v_control,
    Robot.max_velocidad * v_control[1] / mod_v_control, v_control[2]]
699     return v_control, dist_filtradas
700
701
702 # conversion velocidad robot a velocidad ruedas
703 def velocidad_2_rpm(pose, v_control):
704     a = Robot.l + Robot.w
705     b = math.sin(pose[2])+math.cos(pose[2])
706     c = math.sin(pose[2])-math.cos(pose[2])
707     d = 60/(2*math.pi*Robot.radio_rueda)
708     # conversion_2_rpm = 60/(2*math.pi)
709     return [d*(v_control[2]*(-a)+v_control[0]*b+v_control[1]*c),
710             d*(v_control[2]*a+v_control[0]*(-c)+v_control[1]*b),
711             d*(v_control[2]*a+v_control[0]*b+v_control[1]*c),
712             d*(v_control[2]*(-a)+v_control[0]*(-c)+v_control[1]*b)]
713
714
715 def velocidad(pose, v_control):
716     rpm = velocidad_2_rpm(pose, v_control)
717     write_i2c(addrMotor[0], rpm[0])
718     write_i2c(addrMotor[1], rpm[1])
719     write_i2c(addrMotor[2], rpm[2])
720     write_i2c(addrMotor[3], rpm[3])
721
722
723 # calculo movimiento con repulsion
724 def movimiento_con_repulsion(pose, meta_prev, orientacion, dist_filtradas):
725     fin = True
726     # se inicializa velocidad de control
```

```
727     v_control = [0, 0, 0]
728     velocidad(pose, v_control)
729     while fin:
730         # se actualiza pose por odometria
731         pose = odometria(pose, v_control)
732         # condicion de terminacion
733         if math.hypot(meta_prev[0] - pose[0], meta_prev[1] - pose[1]) <
    Robot.dist_estante and abs(orientacion - pose[2]) < Robot.ang_final:
734             v_control = [0, 0, 0]
735             velocidad(pose, v_control)
736             break
737         # calculo de velocidad de control
738         v_control, dist_filtradas = calc_velocidad(meta_prev[0:2], pose,
    orientacion, dist_filtradas)
739         velocidad(pose, v_control)
740     return pose, dist_filtradas
741
742
743 # calculo movimiento sin repulsion
744 def movimiento_no_repulsion(pose, meta):
745     fin = True
746     # se inicializa velocidad de control
747     v_control = [0, 0, 0]
748     velocidad(pose, v_control)
749     while fin:
750         # se actualiza pose por odometria
751         pose = odometria(pose, v_control)
752         # condicion de terminacion
753         if math.hypot(meta[0] - pose[0], meta[1] - pose[1]) < Robot.
    dist_estante and abs(meta[2] - pose[2]) < Robot.ang_final:
754             v_control = [0, 0, 0]
755             velocidad(pose, v_control)
756             break
757         # calculo de velocidad de control sin considerar repulsion de
    sensores
758         v_control = calc_velocidad_no_rep(meta, pose)
```

```
759         velocidad(pose, v_control)
760     return pose
761 
762 
763 # movimiento cuando robot debe moverse a estante
764 def movimiento_estante(meta, meta_prev, pose, dist_filtradas):
765     # giro del robot hacia orientacion deseada segun metodo de orientacion
766     if Robot.metodo_orientacion != 1:
767         print('Orientando')
768         pose, dist_filtradas = movimiento_con_repulsion(pose, meta_prev,
    meta[2], dist_filtradas)
769     pose = movimiento_no_repulsion(pose, meta)
770     print('--> Tomando Producto <--')
771     secuencia()
772     if Robot.metodo_orientacion != 1:
773         pose, dist_filtradas = movimiento_con_repulsion(pose, meta_prev,
    pose[2], dist_filtradas)
774         pose, dist_filtradas = movimiento_con_repulsion(pose, meta_prev,
    Robot.coord_ini[2], dist_filtradas)
775     return pose
776 
777 
778 # determinacion si cambiar a siguiente meta
779 def siguiente_meta(nodos_new, cont_camino, camino, pose, cont_goals,
    goals_orden, pos_prod, prod_pedidos, dist_filtradas):
780     # distancia a meta actual
781     dist_goal_actual = math.hypot(nodos_new[camino[cont_camino]][0]-pose[0],
     nodos_new[camino[cont_camino]][1]-pose[1])
782     # si distancia es menor a cierto threshold se cambia al siguiente nodo
    de la trayectoria
783     if dist_goal_actual < Robot.dist_final and cont_camino < len(camino)-1:
784         # si el nodo actual es una posicion de un producto pedido se activa
    secuencia de mecanismo
785         if camino[cont_camino] == goals_orden[cont_goals]:
786             # secuencia de mecanismo para nivel
787             print('--> Mecanismo Nivel '+str(int(pos_prod[prod_pedidos[
```

```
    goals_orden[cont_goals]-1]][3]))+' para producto '+str(prod_pedidos[
    goals_orden[cont_goals]-1]+1))
788             # movimiento de robot cerca al estante
789             pose = movimiento_estante(pos_prod[prod_pedidos[goals_orden[
    cont_goals]-1]][0:3], nodos_new[camino[cont_camino]], pose,
    dist_filtradas)
790             # actualizacion a siguiente meta
791             cont_goals += 1
792             if cont_goals < len(goals_orden)-1:
793                 disp_str = "PRODUCTO  " + str(prod_pedidos[goals_orden[
    cont_goals] - 1] + 1)
794             else:
795                 disp_str = "RETORNANDO"
796             # Borrar imagen
797             draw.rectangle((0, 0, width, height), outline=0, fill=0)
798             # Escirbir Texto
799             draw.text((x + 10, top + 10), disp_str, font=font, fill=255)
800             # Display
801             disp.image(image)
802             disp.display()
803         # actualizacion a siguiente nodo de trayectoria
804         cont_camino += 1
805     return cont_camino, cont_goals, pose
806
807
808 # repulsion
809 def repulsion(dist_sensores, orientacion):
810     # calculo velocidad de repulsion
811     v_repulsion = [0, 0]
812     for i in range(0, Robot.n_sensores):
813         if dist_sensores[i] < Robot.rango_sensor:
814             # modulo de velocidad de repulsion por cada sensor
815             v = Robot.c_repulsion / 2 * (1 / (dist_sensores[i] - Robot.
    radio_robot)) ** 2
816             # direccion de aplicacion de velocidad
817             v_repulsion = [v_repulsion[0]+vecs_sensores[i][0]*v, v_repulsion
```

```
    [1]+vecs_sensores[i][1]*v]
818     v_repulsion = [math.cos(orientacion)*v_repulsion[0]-math.sin(orientacion
    )*v_repulsion[1],
819                    math.sin(orientacion)*v_repulsion[0]+math.cos(orientacion
    )*v_repulsion[1]]
820     return v_repulsion
821
822
823 def read_i2c(addr):
824     ReadCheck = True
825     while ReadCheck:
826         try:
827             time.sleep(0.02)
828             data = bus.read_i2c_block_data(addr, 0x00, 9)
829         except:
830             print('Read')
831             time.sleep(0.01)
832             continue
833         # borrar exceso de valores
834         aux = 1
835         while aux == 1:
836             try:
837                 data.remove(255)
838             except ValueError:
839                 aux = 0
840         ThetaStr = "".join(map(chr, data))
841         try:
842             Theta = float(ThetaStr)
843             return Theta / 1e2
844         except ValueError:
845             continue
846         # time.sleep(0.02)
847
848
849 # Funci n para convertir string a byte para enio de datos
850 def convert_str2_byte(string):
```

```
851     convertidos = []
852     for i in string:
853         convertidos.append(ord(i))
854     return convertidos
855 
856 
857 def write_i2c(addr, rpm):
858     val_string = str(round(rpm, 1))
859     val = convert_str2_byte(val_string[::-1])
860     write_check = True
861     while write_check:
862         # bus.write_i2c_block_data(addr, 0x00, val)
863         try:
864             # time.sleep(0.01)
865             bus.write_i2c_block_data(addr, 0x00, val)
866             write_check = False
867         except:
868             print('Write')
869             # time.sleep(0.01)
870             pass
871 
872 
873 # programa principal del robot para un nuevo pedido
874 def nuevo_pedido(prod_pedidos, pos_prev, nodos, mat_pesos, vecs_obs,
    points_obs, pos_prod):
875     prod_pedidos = [i - 1 for i in prod_pedidos]
876     # generacion de metas de productos pedidos
877     goals = gen_goals(prod_pedidos, pos_prev)
878     # generar nueva matriz de pesos con metas de productos pedidos y
    posicion inical
879     nodos_new, mat_pesos_new = nuevo_grafo(goals, nodos, mat_pesos, vecs_obs
    , points_obs)
880     # # generar mejorar trayectoria que pase por todas las metas
881     camino, goals_orden = trayectoria(goals, mat_pesos_new, nodos_new)
882     # posicion inicial
883     pose = [Robot.coord_ini[0], Robot.coord_ini[1], Robot.coord_ini[2]]
```

```
884     # siguiente nodo en la ruta
885     cont_camino = 1
886     # contador de metas
887     cont_goals = 0
888     disp_str = "PRODUCTO  " + str(prod_pedidos[goals_orden[cont_goals] - 1]
    + 1)
889     # Borrar imagen
890     draw.rectangle((0, 0, width, height), outline=0, fill=0)
891     # Escirbir Texto
892     draw.text((x + 10, top + 10), disp_str, font=font, fill=255)
893     # Display
894     disp.image(image)
895     disp.display()
896 
897     # inicializar vector para filtrar mediciones
898     dist_filtradas = [0] * Robot.n_sensores
899     # inicializar velocidad de control
900     v_control = [0, 0, 0]
901     velocidad(pose, v_control)
902     fin = True
903     while fin:
904         # actualizacion de la pose por odometria
905         pose = odometria(pose, v_control)
906         # verificacion de terminacion de trayectoria
907         if cont_camino == len(camino) - 1 and math.hypot(nodos_new[camino[
    cont_camino]][0] - pose[0], nodos_new[camino[cont_camino]][1] - pose[1])
    < Robot.dist_final:
908             if abs(Robot.coord_ini[2] - pose[2]) < Robot.ang_final:
909                 # Borrar imagen
910                 draw.rectangle((0, 0, width, height), outline=0, fill=0)
911                 # Escirbir Texto
912                 draw.text((x + 10, top + 10), "ESPERANDO PEDIDO...", font=
    font, fill=255)
913                 # Display
914                 disp.image(image)
915                 disp.display()
```

```
916                 break
917         # actualizacion de meta
918         cont_camino, cont_goals, pose = siguiente_meta(nodos_new,
    cont_camino, camino, pose, cont_goals, goals_orden, pos_prod,
    prod_pedidos, dist_filtradas)
919         # c lculo de velocidad de control
920         # si Robot.metodo_orientacion = 1 robot gira hacia orientacion de
    producto
921         # si Robot.metodo_orientacion = 0 robot se mueve con orientacion 0
    grados
922         if Robot.metodo_orientacion == 1 and cont_goals < len(goals_orden) -
     1:
923             v_control, dist_filtradas = calc_velocidad(nodos_new[camino[
    cont_camino]], pose, pos_prev[prod_pedidos[goals_orden[cont_goals] -
    1]][2], dist_filtradas)
924         else:
925             v_control, dist_filtradas = calc_velocidad(nodos_new[camino[
    cont_camino]], pose, Robot.coord_ini[2], dist_filtradas)
926         velocidad(pose, v_control)
927
928
929 # programa que se ejecuta desde la pagina web
930 def programa():
931     global items, running
932     running = 1
933     prod_pedidos = []
934     for item in items:
935         if item['state']:
936             item['state'] = False
937             prod_pedidos.append(item['id'])
938     for item in items:
939         item['state'] = False
940     print("Pedidos "+str(prod_pedidos))
941     # time.sleep(4)
942     nuevo_pedido(prod_pedidos, pos_prev, nodos, mat_pesos, vecs_obs,
    points_obs, pos_prod)
```

```
943     running = 2
```

### 2.2. Formato Base *base.html*

```
1  <!doctype html>
2  <html lang="es">
3  <head>
4      <!-- Required meta tags -->
5      <meta charset="utf-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1,
       shrink-to-fit=no">
7      <!-- Bootstrap CSS -->
8      <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4
       .5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/
       ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="
       anonymous">
9      <title>
10         {% block title%}
11         {% endblock %}
12     </title>
13
14 </head>
15 <body>
16 <nav class="navbar navbar-expand-md navbar-dark bg-dark">
17     <a class="navbar-brand" href="#">Mercado Online Tesis UIDE</a>
18     <button class="navbar-toggler" type="button" data-toggle="collapse" data
       -target="#navbarNav">
19         <span class="navbar-toggler-icon"></span>
20     </button>
21     <div class="collapse navbar-collapse" id="navbarNav">
22         <ul class="navbar-nav mr-auto">
23             <li class="nav-item active">
24                 <a class="nav-link" href="{{ url_for('home_page') }}">Inicio
       <span class="sr-only">(current)</span></a>
25             </li>
26             <li class="nav-item">
27                 <a class="nav-link" href="{{ url_for('market_page') }}">
```

```
    Mercado</a>
28              </li>
29              <li class="nav-item">
30                  <a class="nav-link" href="{{ url_for('compras_page') }}">
    Carrito</a>
31              </li>
32          </ul>
33          <ul class="navbar-nav">
34              <li class="nav-item">
35                  <a class="nav-link" href="{{url_for('acerca_page')}}">Acerca
     de</a>
36              </li>
37          </ul>
38      </div>
39  </nav>
40  {% block content %}
41  {% endblock %}
42  <!-- Future Content here -->
43
44
45
46  <!-- Optional JavaScript -->
47  <!-- jQuery first, then Popper.js, then Bootstrap JS -->
48  <script src='https://kit.fontawesome.com/a076d05399.js'></script>
49  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="
    sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
    crossorigin="anonymous"></script>
50  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.
    min.js" integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6
    BZp6G7niu735Sk7lN" crossorigin="anonymous"></script>
51  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap
    .min.js" integrity="sha384-
    B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV"
    crossorigin="anonymous"></script>
52  <script> $(document).ready(function(){ $('[data-toggle="popover"]').popover
    ();});</script>
```

```
<style>
.popover {background-color: #80A3C2;
color: black}
}
  }

</style>
</body>

<style>
      body {
      background-color: #212121;
      color: white
      }
      h1 {text-align: center;}
      h4 {text-align: center;}
   </style>

</html>
```

### 2.3. Página de Inicio *home.html*

```
{% extends 'base.html' %}
{% block title %}
P gina de Inicio
{% endblock %}
{% block content %}
<h1>UIDE</h1>
<h1>Tesis Mateo Vernaza 2022</h1>
<h1>Robot Aut nomo Asistencial para Mercados</h1>
{% endblock %}
```

### 2.4. Página del Mercado *market.html*

```
{% extends 'base.html'%}
{% block title %}
Mercado
```

```
{% endblock %}
{% block content %}

<table class="table table-hover table-dark">
    <thead>
    <tr>
        <!-- columnas -->
        <th scope="col">ID</th>
        <th scope="col">Nombre</th>
        <th scope="col">Locaci n</th>
        <th scope="col">Precio</th>
        <th scope="col">Opciones</th>
    </tr>
    </thead>
    <tbody>
    <!-- filas -->
    <!-- iteracion por cada producto -->
    {% for item in items %}
    <tr>
        <td>{{ item.id }}</td>
        <td>{{ item.name }}</td>
        <td>Pasillo {{ item.pasillo }}</td>
        <td>{{ item.price }}$</td>
        <td>
            <!-- boton para informacion sobre producto -->
            <button type="button" class="btn btn-outline btn-success" data-toggle="popover" title="{{ item.name }}" data-placement="left" data-content="Posicion {{ item.posicion }} mm, Nivel {{ item.nivel }}">Info</button>
            <!-- chequeo de estado de compra de producto -->
            {% if item.state == False%}
                <!-- si boton a adir se oprime entonces se envia a direccion mercado/id/add -->
                <a href="/mercado/{{item.id}}/add" class="btn btn-outline btn-info" role="button">A adir</a>
            {% else %}
```

```
35                  <!-- si boton remover se oprime entonces se envia a
    direccion mercado/id/remove -->
36                  <a href="/mercado/{{item.id}}/remove" class="btn btn-outline
     btn-primary" role="button">Remover</a>
37              {% endif %}
38          </td>
39      </tr>
40      {% endfor%}
41      </tbody>
42  </table>
43  {% endblock %}
```

### 2.5. Página de Carrito de Compras *compras.html*

```
1  {% extends 'base.html'%}
2  {% block title %}
3  Mercado
4  {% endblock %}
5  {% block content %}
6  {% include 'includes/modals.html' %}
7  <table class="table table-hover table-dark">
8      <thead>
9      <tr>
10         <!-- columnas -->
11         <th scope="col">ID</th>
12         <th scope="col">Nombre</th>
13         <th scope="col">Locaci n</th>
14         <th scope="col">Precio</th>
15         <th scope="col">Posici n</th>
16         <th scope="col">Nivel</th>
17         <th scope="col">Opciones</th>
18     </tr>
19     </thead>
20     <tbody>
21     <!-- filas -->
22     <!-- se itera por cada producto -->
23     {% for item in items %}
```

```
24      <tr>
25          <!-- si el estado de compra del producto entonces se agrega a lista
    en carrito de compras -->
26          {% if item.state == True%}
27          <td>{{ item.id }}</td>
28          <td>{{ item.name }}</td>
29          <td>Pasillo {{ item.pasillo }}</td>
30          <td>{{ item.price }}$</td>
31          <td>{{ item.posicion[0:2] }}</td>
32          <td>{{ item.nivel }}</td>
33          <td>
34              <!-- boton de remover producto de carrito -->
35              <a href="/carrito/{{item.id}}/remove" class="btn btn-outline btn
    -primary" role="button">Remover</a>
36          </td>
37          {% endif %}
38      </tr>
39      {% endfor%}
40      <tr>
41          <!-- precio total -->
42          <th scope="col">Total</th>
43          <th scope="col">{{ costo_total }} $</th>
44          <th>
45              <!-- boton de compra con modal -->
46              <button class="btn btn-outline btn-success" role="button" data-
    toggle="modal" data-target="#Modal-Comprar">Comprar</button>
47          </th>
48      </tr>
49      </tbody>
50  </table>
51  {% endblock %}
```

### 2.6. Página de Espera *running.html*

```
1  <!doctype html>
2  <html lang="es">
3  <head>
```

```
4       <meta charset="utf-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1,
        shrink-to-fit=no">
6       <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4
        .5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/
        ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="
        anonymous">
7       <title>
8           Running
9       </title>
10
11  </head>
12  <body>
13  <!-- refresh de pagina en (ms) -->
14  <script> function timedRefresh(timeoutPeriod){
15      setTimeout("location.reload(true);",timeoutPeriod);
16      }
17      window.onload = timedRefresh(10000);
18  </script>
19  <!-- mensaje -->
20  <h1>Running...</h1>
21
22  <script src='https://kit.fontawesome.com/a076d05399.js'></script>
23  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="
        sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
        crossorigin="anonymous"></script>
24  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.
        min.js" integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6
        BZp6G7niu735Sk7lN" crossorigin="anonymous"></script>
25  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap
        .min.js" integrity="sha384-
        B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV"
        crossorigin="anonymous"></script>
26  </body>
27
28  <style>
```

```
29      body {
30      background-color: #212121;
31      color: white
32      }
33    </style>
34 </html>
```

### 2.7. Ventana de Diálogo *modals.html*

```
1  <style>
2    .modal-content {
3    background-color: #212121;
4        color: white
5        }
6  </style>
7  <!-- Modal -->
8  <div class="modal fade" id="Modal-Comprar" tabindex="-1" aria-labelledby="
     exampleModalLabel" aria-hidden="true">
9    <div class="modal-dialog">
10     <div class="modal-content">
11       <div class="modal-header">
12         <h5 class="modal-title" id="exampleModalLabel">Confirmar compra</h5>
13         <button type="button" class="btn-close" data-dismiss="modal" aria-
     label="Close"></button>
14       </div>
15       <!-- mensaje -->
16       <div class="modal-body">
17           Desea  continuar con la transacci n?
18       </div>
19       <div class="modal-footer">
20         <!-- boton de cerrar -->
21         <button type="button" class="btn btn-secondary" data-dismiss="modal"
     >Cerrar</button>
22         <!-- redireccion a pagina de checkout, de espera mientras robot se
     mueve -->
23         <a href="{{url_for('checkout_page')}}" type="button" class="btn btn-
     primary">Comprar</a>
```

```
      </div>
    </div>
  </div>
</div>
```

### 2.8. Página de Información *acerca_de.html*

```
{% extends 'base.html' %}
{% block title %}
Acerca de
{% endblock %}
{% block content %}
<h4>Informacion tesis Mateo Vernaza</h4>
{% endblock %}
```

## 3. CÓDIGOS DE PROGRAMACIÓN EN MICROCONTROLADORES

### 3.1. Programa de Control para Motor 1 *Control_M1.ino*

```
//Programa para obtener modelo de motor DC mediante entrada de se al
    escal n
#include <util/atomic.h>
// Libreria Wire para I2C
#include <Wire.h>

//Pines Encoder
#define ENCA 2
#define ENCB 7
//Pines Driver
#define Adelante 4
#define Atras 8
#define PWM1 3

//VARIABLES GLOBALES
float contTiempo = 0;
float tiempo = 0;
```

```
17 int TOPcount = 51; //PWM
18 int B; //Encoder
19 int contPos=0;
20 long t=0; //Calculo RPMs
21 int tOverflow = 0;
22 float rpm = 0.0;
23 float rpmEMA = 0; //Filtro EMA
24 float alfa = 0.6;
25 float e_k, u_k; //PID
26 float ref = 0;
27 float u_k_1 = 0;
28 float e_k_1 = 0;
29 float e_k_2 = 0;
30 float DeltaTheta; //I2C
31 char buff_env[8];
32 float tAnt = 0;
33 int addr = 0x08; //Motor 1
34
35 ////AJUSTADO kp1.1ki36kd0.004
36 float q0 = 3.136;
37 float q1 = -5.064;
38 float q2 = 2;
39
40 void setup() {
41 //  Encoders
42   pinMode(ENCA, INPUT);
43   pinMode(ENCB, INPUT);
44
45 //  Pines Sentido de Giro Driver
46   pinMode(Adelante, OUTPUT);
47   pinMode(Atras, OUTPUT);
48
49 //  PWM Timer2
50   pinMode(PWM1,OUTPUT);
51 //  WGM22:0=5=0b101 para establecer "Phase Correct PWM Mode" con TOP siendo
     el registro OCR2A
```

```
52 //  CS22:0=2=0b010 para prescalador = 8
53   TCCR2A &= ~(1<<WGM21);
54   TCCR2A = _BV(COM2A0) | _BV(COM2B1) | _BV(WGM20);
55   TCCR2B = _BV(WGM22) | _BV(CS21);
56 //  PWM_freq = fclk/(2 x Prescalador x TOP) -> fclk = 16MHz, Prescalador =
       8, TOP = OCR2A = 51
57 //  PWM_freq = 19.6 kHz
58   OCR2A = TOPcount;
59
60 //  Interrupciones Encoder Timer1
61   attachInterrupt(digitalPinToInterrupt(ENCA), lecEncoder, RISING);
62 //  Prescalador = 1 -> CS12:0=1=0b001
63   TCCR1A = 0;
64   TCCR1B = 0;
65   TCCR1B |= (1 << CS10); //Prescalador N = 1
66 //  Contador Timer1 comienza en 0
67   TCNT1 = 0;
68 //  Habilitar interrupciones por Overflow
69   TIMSK1 |=  (1<<TOIE1);
70
71 //  Interrupciones Muestreo Timer0
72   TCCR0A = 0;
73   TCCR0B = 0;
74 // Habilitar CTC mode (Clear Timer on Compare Match) WGM02:0=2=0b010
75   TCCR0A |= (1 << WGM01);
76 // Prescalador N=256 CS02:0=4=0b100
77   TCCR0B |= (1 << CS02);
78 // fmuestreo = fclk/(N*(OCR0A+1)) -> fclk = 16MHz, N = 256, OCR0A = 124 ->
       fmuestreo = 500 Hz
79   OCR0A = 124;
80 //  Contador Timer1 comienza en 0
81   TCNT0  = 0;
82 //  Habilitar interrupciones por Compare Match entre TCNT0 y OCR0A
83   TIMSK0 |= (1 << OCIE0A);
84
85 //  Habilitar interrupciones
```

```
86    sei();
87
88    // Join I2C bus as slave with address 8
89  //  Wire.begin(0x8);
90  //  Wire.begin(0x9);
91    Wire.begin(addr);
92  //  Wire.begin(0x11);
93    // Call receiveEvent when data received
94    Wire.onReceive(receiveEvent);
95    // Handler
96    Wire.onRequest(sendData);
97
98    Serial.begin(115200);
99  }
100
101 void loop() {
102   //  Filtro Exponential Moving Average
103   rpmEMA = alfa*rpm + (1-alfa)*rpmEMA;
104
105   if(tiempo - contTiempo >= 0.5 && ref != 0)
106   {
107     ref = 0;
108     Wire.begin(addr);
109   }
110     //  OCR2B define ciclo de trabajo -> Ciclo de Trabajo = OCR2B/TOPcount
      *100%
111   //  Giro positivo
112   if(u_k >= 0)
113   {
114     digitalWrite(Atras, LOW);
115     digitalWrite(Adelante, HIGH);
116     ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
117     {
118       OCR2B = map(int(u_k),0,TOPcount,27,TOPcount);;
119     }
120   }
```

```
121   //  Giro Negativo
122   else
123   {
124     digitalWrite(Adelante, LOW);
125     digitalWrite(Atras, HIGH);
126     ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
127     {
128       OCR2B = -map(int(u_k),-TOPcount,0,-TOPcount,-27);
129     }
130   }
131   Serial.print(tiempo,4);
132   Serial.print(",");
133   Serial.print(u_k);
134   Serial.print(",");
135   Serial.print(rpm);
136   Serial.print(",");
137   Serial.print(rpmEMA);
138   Serial.print(",");
139   Serial.println(ref);
140 }
141 
142 //INERRUPCION ENCODER
143 void lecEncoder()
144 {
145 //  Bloqueo interrupciones
146   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
147   {
148 //    Obtener valor contador Timer1 TCNT1
149     t = tOverflow*65.536e3 + TCNT1 + 1;
150     TCNT1 = 0;
151     B = digitalRead(ENCB);
152     if(t < 17.91e3)
153     {
154       t = tAnt;
155     }
156     tAnt = t;
```

```
157   //  Determinar sentido de giro
158   //  RPM = fclk*60/(N*GR*CPR*GReng) / TCNT -> RPM = 537.049717e3/TCNT -> GR
       =297.924, CPR=3, GReng=2, fclk=16MHz, N=1
159     if(B == 1)
160     {
161       rpm = 537.049717e3/t;
162       contPos ++;
163     }
164     else
165     {
166       rpm = -537.049717e3/t;
167       contPos --;
168     }
169     tOverflow = 0;
170   }
171 }
172
173 //INTERRUPCION OVERFLOW TIMER1
174 ISR (TIMER1_OVF_vect)
175 {
176 //  Reinicio contador TCNT1
177   TCNT1 = 0;
178 //  Si no hay interrupciones por encoder se aproxima RPM=0
179   if(tOverflow >= 30)
180   {
181     rpm = 0;
182     tOverflow ++;
183     if(tOverflow >= 35000)
184     {
185       tOverflow = 3000;
186     }
187   }
188   else
189   {
190     tOverflow ++;
191   }
```

```
192 }
193
194 ISR (TIMER0_COMPA_vect)
195 {
196   tiempo = tiempo + 2.0e-3;
197 //  Error
198   e_k = ref - rpm;
199 //  Ley de control
200   u_k = u_k_1 + q0*e_k + q1*e_k_1 + q2*e_k_2;
201 //  Truncamiento se al de control
202   if (u_k > TOPcount)
203   {
204      u_k = TOPcount;
205   }
206   else if (u_k < -TOPcount)
207   {
208     u_k = -TOPcount;
209   }
210   u_k_1 = u_k;
211   e_k_2 = e_k_1;
212   e_k_1 = e_k;
213 }
214
215 //-->I2C<--
216 // Funcion cuando Master envia referencia
217 void receiveEvent(int howMany) {
218   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
219   {
220     if (howMany > 1) {
221       char buff_rec[howMany];
222       for (int i = howMany - 1; i >= 0; i--) {
223         buff_rec[i] = Wire.read();
224       }
225 //      Chequeo de informaci n recibida
226       if(isDigit(buff_rec[0]) || buff_rec[0] == '-')
227       {
```

```
228         ref = atof(buff_rec);
229         contTiempo = tiempo;
230       }
231     }
232     else{
233       Wire.read();
234     }
235   }
236 }
237
238 //funcion para enviar datos cuando Master lo requiera
239 void sendData() {
240   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
241   {
242     DeltaTheta = contPos/284.49646; //radianes
243     contPos = 0;
244     dtostrf(DeltaTheta*1e2, 2, 6, buff_env);
245     Wire.write(buff_env);
246     Serial.println("");
247     Serial.println(DeltaTheta*1e2,6);
248   }
249 }
```

### 3.2. Programa de Control para Motor 2 *Control_M2.ino*

```
1  //Programa para obtener modelo de motor DC mediante entrada de se al
       escal n
2  #include <util/atomic.h>
3  // Libreria Wire para I2C
4  #include <Wire.h>
5
6  //Pines Encoder
7  #define ENCA 2
8  #define ENCB 7
9  //Pines Driver
10 #define Adelante 8
11 #define Atras 4
```

```
12 #define PWM1 3
13
14 //VARIABLES GLOBALES
15 float contTiempo = 0;
16 float tiempo = 0;
17 int TOPcount = 51; //PWM
18 int B; //Encoder
19 int contPos=0;
20 long t=0; //Calculo RPMs
21 int tOverflow = 0;
22 float rpm = 0.0;
23 float rpmEMA = 0; //Filtro EMA
24 float alfa = 0.6;
25 float e_k, u_k; //PID
26 float ref = 0;
27 float u_k_1 = 0;
28 float e_k_1 = 0;
29 float e_k_2 = 0;
30 float DeltaTheta; //I2C
31 char buff_env[8];
32 float tAnt = 0;
33 int addr = 0x09; //Motor 2
34
35 ////AJUSTADO kp1.1ki36kd0.004
36 float q0 = 3.136;
37 float q1 = -5.064;
38 float q2 = 2;
39
40 void setup() {
41 //  Encoders
42   pinMode(ENCA, INPUT);
43   pinMode(ENCB, INPUT);
44
45 //  Pines Sentido de Giro Driver
46   pinMode(Adelante, OUTPUT);
47   pinMode(Atras, OUTPUT);
```

```
48
49 //  PWM Timer2
50   pinMode(PWM1,OUTPUT);
51 //  WGM22:0=5=0b101 para establecer "Phase Correct PWM Mode" con TOP siendo
       el registro OCR2A
52 //  CS22:0=2=0b010 para prescalador = 8
53   TCCR2A &= ~(1<<WGM21);
54   TCCR2A = _BV(COM2A0) | _BV(COM2B1) | _BV(WGM20);
55   TCCR2B = _BV(WGM22) | _BV(CS21);
56 //  PWM_freq = fclk/(2 x Prescalador x TOP) -> fclk = 16MHz, Prescalador =
       8, TOP = OCR2A = 51
57 //  PWM_freq = 19.6 kHz
58   OCR2A = TOPcount;
59
60 //  Interrupciones Encoder Timer1
61   attachInterrupt(digitalPinToInterrupt(ENCA), lecEncoder, RISING);
62 //  Prescalador = 1 -> CS12:0=1=0b001
63   TCCR1A = 0;
64   TCCR1B = 0;
65   TCCR1B |= (1 << CS10); //Prescalador N = 1
66 //  Contador Timer1 comienza en 0
67   TCNT1 = 0;
68 //  Habilitar interrupciones por Overflow
69   TIMSK1 |=  (1<<TOIE1);
70
71 //  Interrupciones Muestreo Timer0
72   TCCR0A = 0;
73   TCCR0B = 0;
74 // Habilitar CTC mode (Clear Timer on Compare Match) WGM02:0=2=0b010
75   TCCR0A |= (1 << WGM01);
76 // Prescalador N=256 CS02:0=4=0b100
77   TCCR0B |= (1 << CS02);
78 // fmuestreo = fclk/(N*(OCR0A+1)) -> fclk = 16MHz, N = 256, OCR0A = 124 ->
       fmuestreo = 500 Hz
79   OCR0A = 124;
80 //  Contador Timer1 comienza en 0
```

```
81    TCNT0  = 0;
82  //  Habilitar interrupciones por Compare Match entre TCNT0 y OCR0A
83    TIMSK0 |= (1 << OCIE0A);
84
85  //  Habilitar interrupciones
86    sei();
87
88    // Join I2C bus as slave with address 8
89  //  Wire.begin(0x8);
90  //  Wire.begin(0x9);
91    Wire.begin(addr);
92  //  Wire.begin(0x11);
93    // Call receiveEvent when data received
94    Wire.onReceive(receiveEvent);
95    // Handler
96    Wire.onRequest(sendData);
97
98    Serial.begin(115200);
99  }
100
101 void loop() {
102   //  Filtro Exponential Moving Average
103   rpmEMA = alfa*rpm + (1-alfa)*rpmEMA;
104
105   if(tiempo - contTiempo >= 0.5 && ref != 0)
106   {
107     ref = 0;
108   }
109     //  OCR2B define ciclo de trabajo -> Ciclo de Trabajo = OCR2B/TOPcount
      *100%
110   //  Giro positivo
111   if(u_k >= 0)
112   {
113     digitalWrite(Atras, LOW);
114     digitalWrite(Adelante, HIGH);
115     ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
```

```
116     {
117       OCR2B = map(int(u_k),0,TOPcount,27,TOPcount);;
118     }
119   }
120   //  Giro Negativo
121   else
122   {
123     digitalWrite(Adelante, LOW);
124     digitalWrite(Atras, HIGH);
125     ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
126     {
127       OCR2B = -map(int(u_k),-TOPcount,0,-TOPcount,-27);
128     }
129   }
130   Serial.print(tiempo,4);
131   Serial.print(",");
132   Serial.print(u_k);
133   Serial.print(",");
134   Serial.print(rpm);
135   Serial.print(",");
136   Serial.print(rpmEMA);
137   Serial.print(",");
138   Serial.println(ref);
139 }
140 
141 //INERRUPCION ENCODER
142 void lecEncoder()
143 {
144 //  Bloqueo interrupciones
145   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
146   {
147 //    Obtener valor contador Timer1 TCNT1
148     t = tOverflow*65.536e3 + TCNT1 + 1;
149     TCNT1 = 0;
150     B = digitalRead(ENCB);
151     if(t < 17.91e3)
```

```
152     {
153       t = tAnt;
154     }
155     tAnt = t;
156   //  Determinar sentido de giro
157   //  RPM = fclk*60/(N*GR*CPR*GReng) / TCNT -> RPM = 537.049717e3/TCNT -> GR
      =297.924, CPR=3, GReng=2, fclk=16MHz, N=1
158     if(B == 0)
159     {
160       rpm = 537.049717e3/t;
161       contPos ++;
162     }
163     else
164     {
165       rpm = -537.049717e3/t;
166       contPos --;
167     }
168     tOverflow = 0;
169   }
170 }
171
172 //INTERRUPCION OVERFLOW TIMER1
173 ISR (TIMER1_OVF_vect)
174 {
175 //  Reinicio contador TCNT1
176   TCNT1 = 0;
177 //  Si no hay interrupciones por encoder se aproxima RPM=0
178   if(tOverflow >= 30)
179   {
180     rpm = 0;
181     tOverflow ++;
182     if(tOverflow >= 35000)
183     {
184       tOverflow = 3000;
185     }
186   }
```

```
187   else
188   {
189     tOverflow ++;
190   }
191 }
192
193 ISR (TIMER0_COMPA_vect)
194 {
195   tiempo = tiempo + 2.0e-3;
196 //  Error
197   e_k = ref - rpm;
198 //  Ley de control
199   u_k = u_k_1 + q0*e_k + q1*e_k_1 + q2*e_k_2;
200 //  Truncamiento se al de control
201   if (u_k > TOPcount)
202   {
203      u_k = TOPcount;
204   }
205   else if (u_k < -TOPcount)
206   {
207     u_k = -TOPcount;
208   }
209   u_k_1 = u_k;
210   e_k_2 = e_k_1;
211   e_k_1 = e_k;
212 }
213
214 //-->I2C<--
215 // Funcion cuando Master envia referencia
216 void receiveEvent(int howMany) {
217   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
218   {
219     if (howMany > 1) {
220       char buff_rec[howMany];
221       for (int i = howMany - 1; i >= 0; i--) {
222         buff_rec[i] = Wire.read();
```

```
223       }
224 //      Chequeo de informaci n recibida
225       if(isDigit(buff_rec[0]) || buff_rec[0] == '-')
226       {
227         ref = atof(buff_rec);
228         contTiempo = tiempo;
229       }
230     }
231     else{
232       Wire.read();
233     }
234   }
235 }
236
237 //funcion para enviar datos cuando Master lo requiera
238 void sendData() {
239   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
240   {
241 //    Giro de rueda
242 //    DeltaTheta = contPos/1787.544; //revoluciones
243 //    DeltaTheta = contPos/11.23147e3; //radianes
244     DeltaTheta = contPos/284.49646; //radianes
245     contPos = 0;
246     dtostrf(DeltaTheta*1e2, 2, 6, buff_env);
247     Wire.write(buff_env);
248     Serial.println("");
249     Serial.println(DeltaTheta*1e2,6);
250   }
251 }
```

### 3.3. Programa de Control para Motor 3 *Control_M3.ino*

```
1 //Programa para obtener modelo de motor DC mediante entrada de se al
    escal n
2 #include <util/atomic.h>
3 // Libreria Wire para I2C
4 #include <Wire.h>
```

```
5
6  //Pines Encoder
7  #define ENCA 2
8  #define ENCB 7
9  //Pines Driver
10 #define Adelante 8
11 #define Atras 4
12 #define PWM1 3
13
14 //VARIABLES GLOBALES
15 float contTiempo = 0;
16 float tiempo = 0;
17 int TOPcount = 51; //PWM
18 int B; //Encoder
19 int contPos=0;
20 long t=0; //Calculo RPMs
21 int tOverflow = 0;
22 float rpm = 0.0;
23 float rpmEMA = 0; //Filtro EMA
24 float alfa = 0.6;
25 float e_k, u_k; //PID
26 float ref = 0;
27 float u_k_1 = 0;
28 float e_k_1 = 0;
29 float e_k_2 = 0;
30 float DeltaTheta; //I2C
31 char buff_env[8];
32 float tAnt = 0;
33 int addr = 0x10; //Motor 3
34
35 ////AJUSTADO kp1.1ki36kd0.004
36 float q0 = 3.136;
37 float q1 = -5.064;
38 float q2 = 2;
39
40 void setup() {
```

```
41 //  Encoders
42   pinMode(ENCA, INPUT);
43   pinMode(ENCB, INPUT);
44
45 //  Pines Sentido de Giro Driver
46   pinMode(Adelante, OUTPUT);
47   pinMode(Atras, OUTPUT);
48
49 //  PWM Timer2
50   pinMode(PWM1,OUTPUT);
51 //  WGM22:0=5=0b101 para establecer "Phase Correct PWM Mode" con TOP siendo
     el registro OCR2A
52 //  CS22:0=2=0b010 para prescalador = 8
53   TCCR2A &= ~(1<<WGM21);
54   TCCR2A = _BV(COM2A0) | _BV(COM2B1) | _BV(WGM20);
55   TCCR2B = _BV(WGM22) | _BV(CS21);
56 //  PWM_freq = fclk/(2 x Prescalador x TOP) -> fclk = 16MHz, Prescalador =
     8, TOP = OCR2A = 51
57 //  PWM_freq = 19.6 kHz
58   OCR2A = TOPcount;
59
60 //  Interrupciones Encoder Timer1
61   attachInterrupt(digitalPinToInterrupt(ENCA), lecEncoder, RISING);
62 //  Prescalador = 1 -> CS12:0=1=0b001
63   TCCR1A = 0;
64   TCCR1B = 0;
65   TCCR1B |= (1 << CS10); //Prescalador N = 1
66 //  Contador Timer1 comienza en 0
67   TCNT1 = 0;
68 //  Habilitar interrupciones por Overflow
69   TIMSK1 |=  (1<<TOIE1);
70
71 //  Interrupciones Muestreo Timer0
72   TCCR0A = 0;
73   TCCR0B = 0;
74 // Habilitar CTC mode (Clear Timer on Compare Match) WGM02:0=2=0b010
```

```
75    TCCR0A |= (1 << WGM01);
76  // Prescalador N=256 CS02:0=4=0b100
77    TCCR0B |= (1 << CS02);
78  // fmuestreo = fclk/(N*(OCR0A+1)) -> fclk = 16MHz, N = 256, OCR0A = 124 ->
      fmuestreo = 500 Hz
79    OCR0A = 124;
80  //  Contador Timer1 comienza en 0
81    TCNT0  = 0;
82  //  Habilitar interrupciones por Compare Match entre TCNT0 y OCR0A
83    TIMSK0 |= (1 << OCIE0A);
84
85  //  Habilitar interrupciones
86    sei();
87
88    // Join I2C bus as slave with address 8
89  //  Wire.begin(0x8);
90  //  Wire.begin(0x9);
91    Wire.begin(addr);
92  //  Wire.begin(0x11);
93    // Call receiveEvent when data received
94    Wire.onReceive(receiveEvent);
95    // Handler
96    Wire.onRequest(sendData);
97
98    Serial.begin(115200);
99  }
100
101 void loop() {
102   //  Filtro Exponential Moving Average
103   rpmEMA = alfa*rpm + (1-alfa)*rpmEMA;
104
105   if(tiempo - contTiempo >= 0.5 && ref != 0)
106   {
107     ref = 0;
108     Wire.begin(addr);
109   }
```

```
110     //  OCR2B define ciclo de trabajo -> Ciclo de Trabajo = OCR2B/TOPcount
        *100%
111   //  Giro positivo
112   if(u_k >= 0)
113   {
114     digitalWrite(Atras, LOW);
115     digitalWrite(Adelante, HIGH);
116     ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
117     {
118       OCR2B = map(int(u_k),0,TOPcount,27,TOPcount);;
119     }
120   }
121   //  Giro Negativo
122   else
123   {
124     digitalWrite(Adelante, LOW);
125     digitalWrite(Atras, HIGH);
126     ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
127     {
128       OCR2B = -map(int(u_k),-TOPcount,0,-TOPcount,-27);
129     }
130   }
131   Serial.print(tiempo,4);
132   Serial.print(",");
133   Serial.print(u_k);
134   Serial.print(",");
135   Serial.print(rpm);
136   Serial.print(",");
137   Serial.print(rpmEMA);
138   Serial.print(",");
139   Serial.println(ref);
140 }
141
142 //INERRUPCION ENCODER
143 void lecEncoder()
144 {
```

```
145 //  Bloqueo interrupciones
146   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
147   {
148 //    Obtener valor contador Timer1 TCNT1
149     t = tOverflow*65.536e3 + TCNT1 + 1;
150     TCNT1 = 0;
151     B = digitalRead(ENCB);
152     if(t < 17.91e3)
153     {
154       t = tAnt;
155     }
156     tAnt = t;
157   //  Determinar sentido de giro
158   //  RPM = fclk*60/(N*GR*CPR*GReng) / TCNT -> RPM = 537.049717e3/TCNT -> GR
    =297.924, CPR=3, GReng=2, fclk=16MHz, N=1
159     if(B == 0)
160     {
161       rpm = 537.049717e3/t;
162       contPos ++;
163     }
164     else
165     {
166       rpm = -537.049717e3/t;
167       contPos --;
168     }
169     tOverflow = 0;
170   }
171 }
172
173 //INTERRUPCION OVERFLOW TIMER1
174 ISR (TIMER1_OVF_vect)
175 {
176 //  Reinicio contador TCNT1
177   TCNT1 = 0;
178 //  Si no hay interrupciones por encoder se aproxima RPM=0
179   if(tOverflow >= 30)
```

```
180   {
181     rpm = 0;
182     tOverflow ++;
183     if(tOverflow >= 35000)
184     {
185       tOverflow = 3000;
186     }
187   }
188   else
189   {
190     tOverflow ++;
191   }
192 }
193 
194 ISR (TIMER0_COMPA_vect)
195 {
196   tiempo = tiempo + 2.0e-3;
197 //  Error
198   e_k = ref - rpm;
199 //  Ley de control
200   u_k = u_k_1 + q0*e_k + q1*e_k_1 + q2*e_k_2;
201 //  Truncamiento se al de control
202   if (u_k > TOPcount)
203   {
204      u_k = TOPcount;
205   }
206   else if (u_k < -TOPcount)
207   {
208     u_k = -TOPcount;
209   }
210   u_k_1 = u_k;
211   e_k_2 = e_k_1;
212   e_k_1 = e_k;
213 }
214 
215 //-->I2C<--
```

```
216 // Funcion cuando Master envia referencia
217 void receiveEvent(int howMany) {
218   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
219   {
220     if (howMany > 1) {
221       char buff_rec[howMany];
222       for (int i = howMany - 1; i >= 0; i--) {
223         buff_rec[i] = Wire.read();
224       }
225 //       Chequeo de informaci n recibida
226       if(isDigit(buff_rec[0]) || buff_rec[0] == '-')
227       {
228         ref = atof(buff_rec);
229         contTiempo = tiempo;
230       }
231     }
232     else{
233       Wire.read();
234     }
235   }
236 }
237
238 //funcion para enviar datos cuando Master lo requiera
239 void sendData() {
240   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
241   {
242 //     Giro de rueda
243 //     DeltaTheta = contPos/1787.544; //revoluciones
244 //     DeltaTheta = contPos/11.23147e3; //radianes
245     DeltaTheta = contPos/284.49646; //radianes
246     contPos = 0;
247     dtostrf(DeltaTheta*1e2, 2, 6, buff_env);
248     Wire.write(buff_env);
249     Serial.println("");
250     Serial.println(DeltaTheta*1e2,6);
251   }
```

```
}
```

### 3.4. Programa de Control para Motor 4 *Control_M4.ino*

```
//Programa para obtener modelo de motor DC mediante entrada de se al
    escal n
#include <util/atomic.h>
// Libreria Wire para I2C
#include <Wire.h>

//Pines Encoder
#define ENCA 2
#define ENCB 7
//Pines Driver
#define Adelante 4
#define Atras 8
#define PWM1 3

//VARIABLES GLOBALES
float contTiempo = 0;
float tiempo = 0;
int TOPcount = 51; //PWM
int B; //Encoder
int contPos=0;
long t=0; //Calculo RPMs
int tOverflow = 0;
float rpm = 0.0;
float rpmEMA = 0; //Filtro EMA
float alfa = 0.6;
float e_k, u_k; //PID
float ref = 0;
float u_k_1 = 0;
float e_k_1 = 0;
float e_k_2 = 0;
float DeltaTheta; //I2C
char buff_env[8];
float tAnt = 0;
```

```
33 int addr = 0x11; //Motor 4
34
35 ////AJUSTADO kp1.1ki36kd0.004
36 float q0 = 3.136;
37 float q1 = -5.064;
38 float q2 = 2;
39
40 void setup() {
41 //  Encoders
42   pinMode(ENCA, INPUT);
43   pinMode(ENCB, INPUT);
44
45 //  Pines Sentido de Giro Driver
46   pinMode(Adelante, OUTPUT);
47   pinMode(Atras, OUTPUT);
48
49 //  PWM Timer2
50   pinMode(PWM1,OUTPUT);
51 //  WGM22:0=5=0b101 para establecer "Phase Correct PWM Mode" con TOP siendo
     el registro OCR2A
52 //  CS22:0=2=0b010 para prescalador = 8
53   TCCR2A &= ~(1<<WGM21);
54   TCCR2A = _BV(COM2A0) | _BV(COM2B1) | _BV(WGM20);
55   TCCR2B = _BV(WGM22) | _BV(CS21);
56 //  PWM_freq = fclk/(2 x Prescalador x TOP) -> fclk = 16MHz, Prescalador =
     8, TOP = OCR2A = 51
57 //  PWM_freq = 19.6 kHz
58   OCR2A = TOPcount;
59
60 //  Interrupciones Encoder Timer1
61   attachInterrupt(digitalPinToInterrupt(ENCA), lecEncoder, RISING);
62 //  Prescalador = 1 -> CS12:0=1=0b001
63   TCCR1A = 0;
64   TCCR1B = 0;
65   TCCR1B |= (1 << CS10); //Prescalador N = 1
66 //  Contador Timer1 comienza en 0
```

```
67    TCNT1 = 0;
68  //  Habilitar interrupciones por Overflow
69    TIMSK1 |=  (1<<TOIE1);
70
71  //  Interrupciones Muestreo Timer0
72    TCCR0A = 0;
73    TCCR0B = 0;
74  // Habilitar CTC mode (Clear Timer on Compare Match) WGM02:0=2=0b010
75    TCCR0A |= (1 << WGM01);
76  // Prescalador N=256 CS02:0=4=0b100
77    TCCR0B |= (1 << CS02);
78  // fmuestreo = fclk/(N*(OCR0A+1)) -> fclk = 16MHz, N = 256, OCR0A = 124 ->
      fmuestreo = 500 Hz
79    OCR0A = 124;
80  //  Contador Timer1 comienza en 0
81    TCNT0  = 0;
82  //  Habilitar interrupciones por Compare Match entre TCNT0 y OCR0A
83    TIMSK0 |= (1 << OCIE0A);
84
85  //  Habilitar interrupciones
86    sei();
87
88    // Join I2C bus as slave with address 8
89  //  Wire.begin(0x8);
90  //  Wire.begin(0x9);
91    Wire.begin(addr);
92  //  Wire.begin(0x11);
93    // Call receiveEvent when data received
94    Wire.onReceive(receiveEvent);
95    // Handler
96    Wire.onRequest(sendData);
97
98    Serial.begin(115200);
99  }
100
101 void loop() {
```

```
102   //  Filtro Exponential Moving Average
103   rpmEMA = alfa*rpm + (1-alfa)*rpmEMA;
104
105   if(tiempo - contTiempo >= 0.5 && ref != 0)
106   {
107     ref = 0;
108     Wire.begin(addr);
109   }
110     //  OCR2B define ciclo de trabajo -> Ciclo de Trabajo = OCR2B/TOPcount
       *100%
111   //  Giro positivo
112   if(u_k >= 0)
113   {
114     digitalWrite(Atras, LOW);
115     digitalWrite(Adelante, HIGH);
116     ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
117     {
118       OCR2B = map(int(u_k),0,TOPcount,27,TOPcount);;
119     }
120   }
121   //  Giro Negativo
122   else
123   {
124     digitalWrite(Adelante, LOW);
125     digitalWrite(Atras, HIGH);
126     ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
127     {
128       OCR2B = -map(int(u_k),-TOPcount,0,-TOPcount,-27);
129     }
130   }
131   Serial.print(tiempo,4);
132   Serial.print(",");
133   Serial.print(u_k);
134   Serial.print(",");
135   Serial.print(rpm);
136   Serial.print(",");
```

```
137   Serial.print(rpmEMA);
138   Serial.print(",");
139   Serial.println(ref);
140 }
141 
142 //INERRUPCION ENCODER
143 void lecEncoder()
144 {
145 //  Bloqueo interrupciones
146   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
147   {
148 //    Obtener valor contador Timer1 TCNT1
149     t = tOverflow*65.536e3 + TCNT1 + 1;
150     TCNT1 = 0;
151     B = digitalRead(ENCB);
152     if(t < 17.91e3)
153     {
154       t = tAnt;
155     }
156     tAnt = t;
157   //  Determinar sentido de giro
158   //  RPM = fclk*60/(N*GR*CPR*GReng) / TCNT -> RPM = 537.049717e3/TCNT -> GR
      =297.924, CPR=3, GReng=2, fclk=16MHz, N=1
159     if(B == 1)
160     {
161       rpm = 537.049717e3/t;
162       contPos ++;
163     }
164     else
165     {
166       rpm = -537.049717e3/t;
167       contPos --;
168     }
169     tOverflow = 0;
170   }
171 }
```

```
172
173 //INTERRUPCION OVERFLOW TIMER1
174 ISR (TIMER1_OVF_vect)
175 {
176 //  Reinicio contador TCNT1
177   TCNT1 = 0;
178 //  Si no hay interrupciones por encoder se aproxima RPM=0
179   if(tOverflow >= 30)
180   {
181     rpm = 0;
182     tOverflow ++;
183     if(tOverflow >= 35000)
184     {
185       tOverflow = 3000;
186     }
187   }
188   else
189   {
190     tOverflow ++;
191   }
192 }
193
194 ISR (TIMER0_COMPA_vect)
195 {
196   tiempo = tiempo + 2.0e-3;
197 //  Error
198   e_k = ref - rpm;
199 //  Ley de control
200   u_k = u_k_1 + q0*e_k + q1*e_k_1 + q2*e_k_2;
201 //  Truncamiento se al de control
202   if (u_k > TOPcount)
203   {
204      u_k = TOPcount;
205   }
206   else if (u_k < -TOPcount)
207   {
```

```
208     u_k = -TOPcount;
209   }
210   u_k_1 = u_k;
211   e_k_2 = e_k_1;
212   e_k_1 = e_k;
213 }
214 
215 //-->I2C<--
216 // Funcion cuando Master envia referencia
217 void receiveEvent(int howMany) {
218   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
219   {
220     if (howMany > 1) {
221       char buff_rec[howMany];
222       for (int i = howMany - 1; i >= 0; i--) {
223         buff_rec[i] = Wire.read();
224       }
225 //       Chequeo de informaci n recibida
226       if(isDigit(buff_rec[0]) || buff_rec[0] == '-')
227       {
228         ref = atof(buff_rec);
229         contTiempo = tiempo;
230       }
231     }
232     else{
233       Wire.read();
234     }
235   }
236 }
237 
238 //funcion para enviar datos cuando Master lo requiera
239 void sendData() {
240   ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
241   {
242 //     Giro de rueda
243 //     DeltaTheta = contPos/1787.544; //revoluciones
```

```
244 //    DeltaTheta = contPos/11.23147e3; //radianes
245     DeltaTheta = contPos/284.49646; //radianes
246     contPos = 0;
247     dtostrf(DeltaTheta*1e2, 2, 6, buff_env);
248     Wire.write(buff_env);
249     Serial.println("");
250     Serial.println(DeltaTheta*1e2,6);
251   }
252 }
```

# Anexo C: Manual de Usuario

Ingeniería Mecatrónica

UIDE

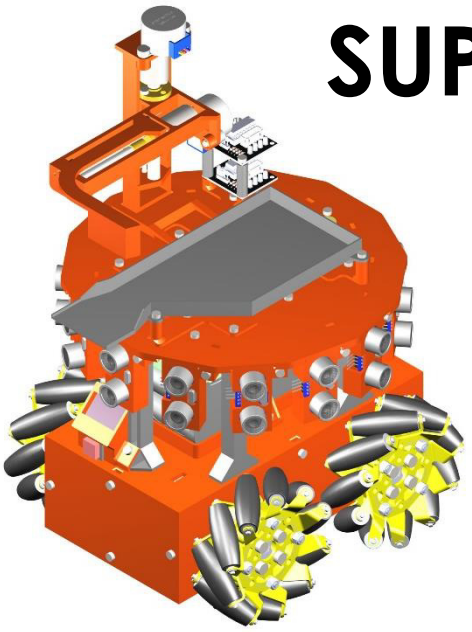
Powered by
Arizona State University

Facultad de Ciencias Técnicas

Escuela de Ingeniería Mecatrónica

# Manual de Operación y Mantenimiento

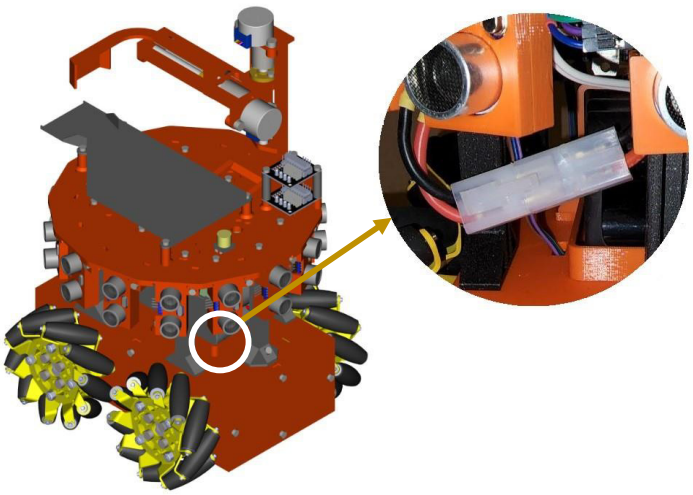
## ROBOT ASISTENCIAL PARA COMPRAS EN SUPERMERCADOS



# ÍNDICE

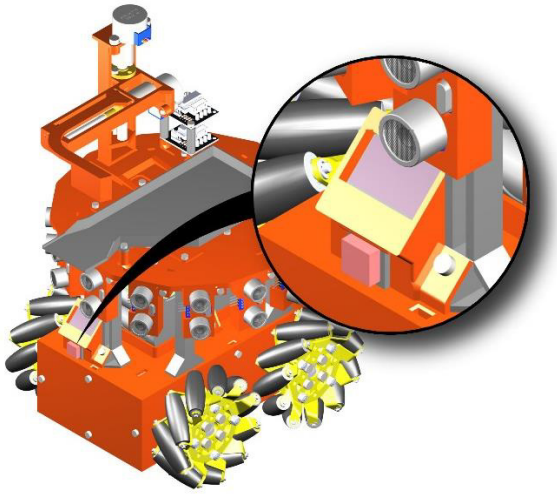
# 1 GUÍA DE INICIO RÁPIDO

- **Cargar el robot:**
  - Desconectar el conector de la batería de la parte posterior del robot.



  - Cargar la batería con el cargador *GoBilda* hasta que la luz indicadora cambie a verde.


- **Encender el robot:**
  - Presionar el botón de encendido y esperar a que el sistema operativo inicie.



- **Ejecutar el programa:**
  - Con una computadora entrar a un intérprete de línea de comandos y probar la conexión a Internet del robot (para este paso el robot debe estar conectado a la red local, si no lo está ver la sección 3). *Nota: en este ejemplo se utiliza el intérprete predeterminada de Windows "Command Prompt", si se utiliza otro intérprete es posible que los comandos sean distintos.*

```
C:\Users\mateo>ping raspberrypi.local

Pinging raspberrypi.local [2800:bf0:10c:1015:20b6:c68b:3a99:b2c1] with 32 bytes of data:
Reply from 2800:bf0:10c:1015:20b6:c68b:3a99:b2c1: time=6ms
Reply from 2800:bf0:10c:1015:20b6:c68b:3a99:b2c1: time=6ms
Reply from 2800:bf0:10c:1015:20b6:c68b:3a99:b2c1: time=10ms
Reply from 2800:bf0:10c:1015:20b6:c68b:3a99:b2c1: time=7ms

Ping statistics for 2800:bf0:10c:1015:20b6:c68b:3a99:b2c1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 6ms, Maximum = 10ms, Average = 7ms
```

  - Conectarse al robot por el protocolo *SSH*, la contraseña es ***Tesis2022***.

```
C:\Users\mateo>ssh pi@raspberrypi.local
pi@raspberrypi.local's password:
Linux raspberrypi 5.15.32+ #1538 Thu Mar 31 19:37:58 BST 2022 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jul 31 22:29:38 2022 from 2800:bf0:10c:1015:1552:30b8:4b62:bc0d
```

  - Navegar a la carpeta *Documents/Python* y ejecutar el programa la página web.

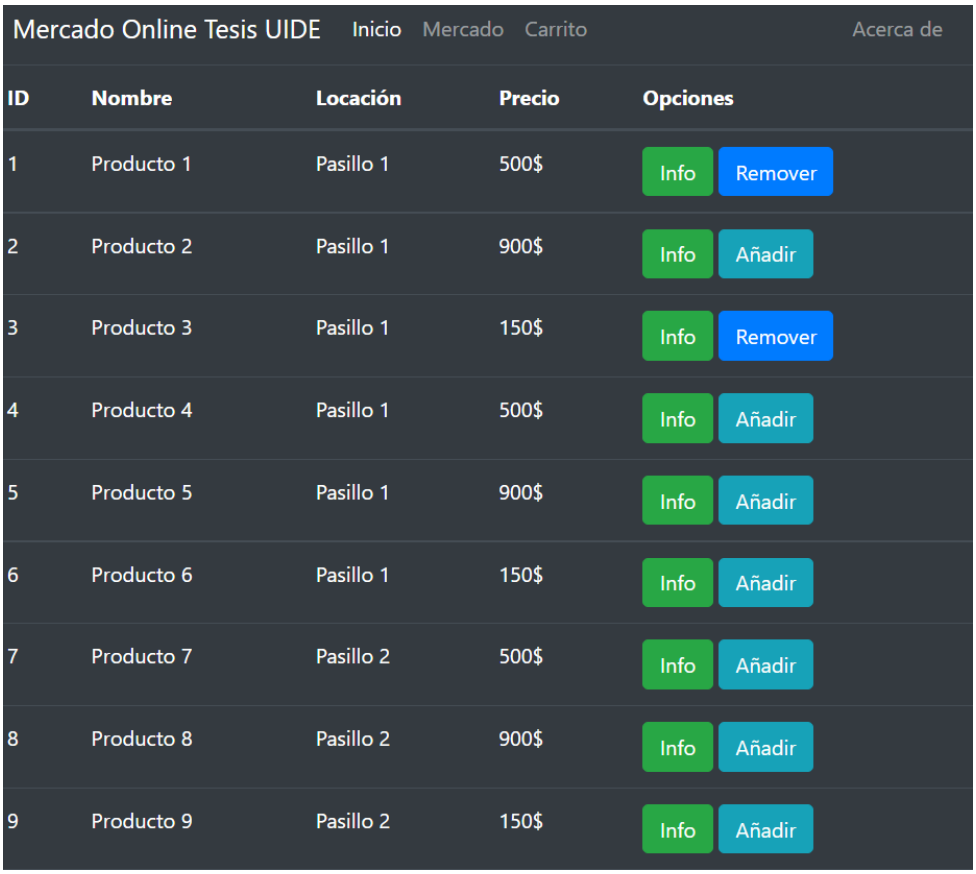
```
pi@raspberrypi.local:~ $ cd Documents/Python
pi@raspberrypi.local:~/Documents/Python $ flask run --host=0.0.0.0
```

  - Desde cualquier dispositivo conectado a la red local se puede acceder a la página web ingresando a un buscador de Internet e introduciendo la dirección: *raspberrypi.local:5000*.

# 2 Página Web

- **Página Principal:**
  - Presenta información sobre el proyecto. En la parte superior se accede al resto de páginas.

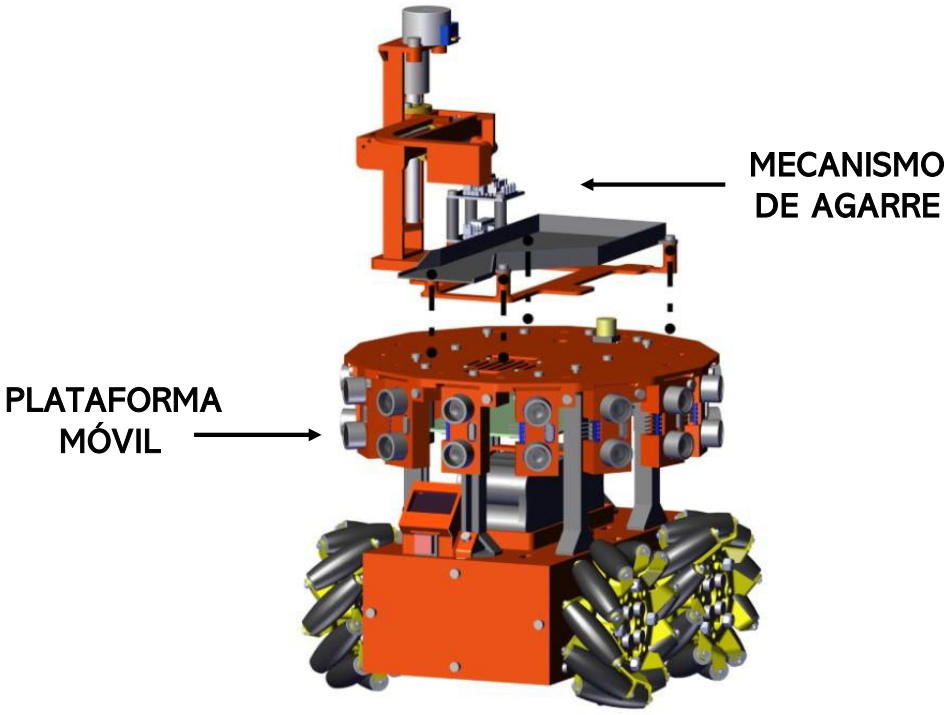

- **Página del Mercado:**
  - Presenta todos los productos disponibles.



- **Página del Carrito:**
  - Presenta todos los productos agregados a la orden

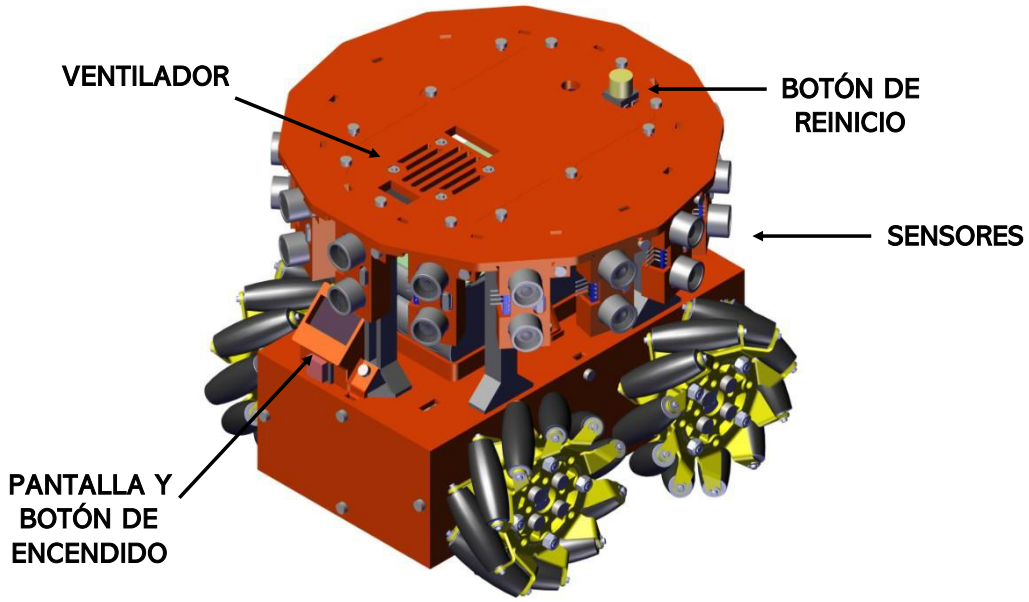

# 3 Elementos Generales Del Prototipo

El prototipo tiene dos elementos principales: el mecanismo de agarre y la plataforma móvil. El mecanismo de agarre se puede desmontar al retirar los pernos que se muestran en la figura.

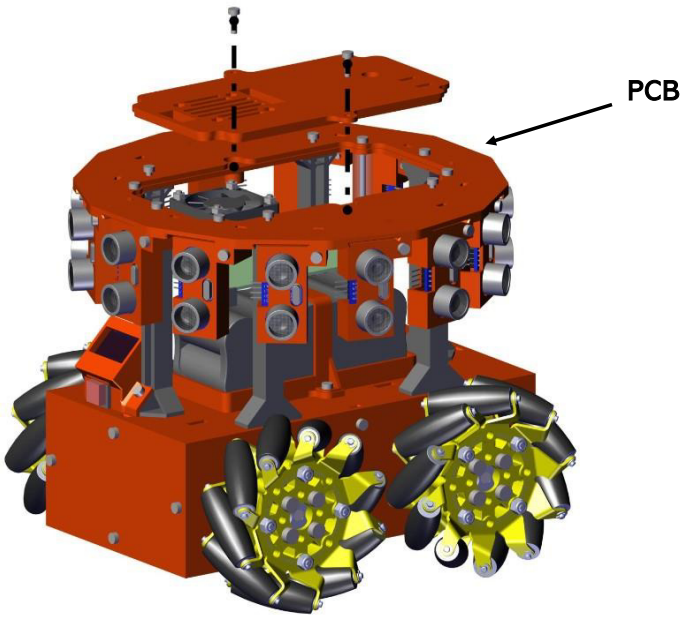


Otros elementos importantes son:

- Pantalla de mensajes, donde se despliega información del robot.
- Ventilador, que mantiene controlada la temperatura interna del circuito, especialmente del microprocesador.
- Botón de reinicio de microcontroladores que controlan la velocidad de las ruedas.
- Sensores ultrasónicos para detectar obstáculos.

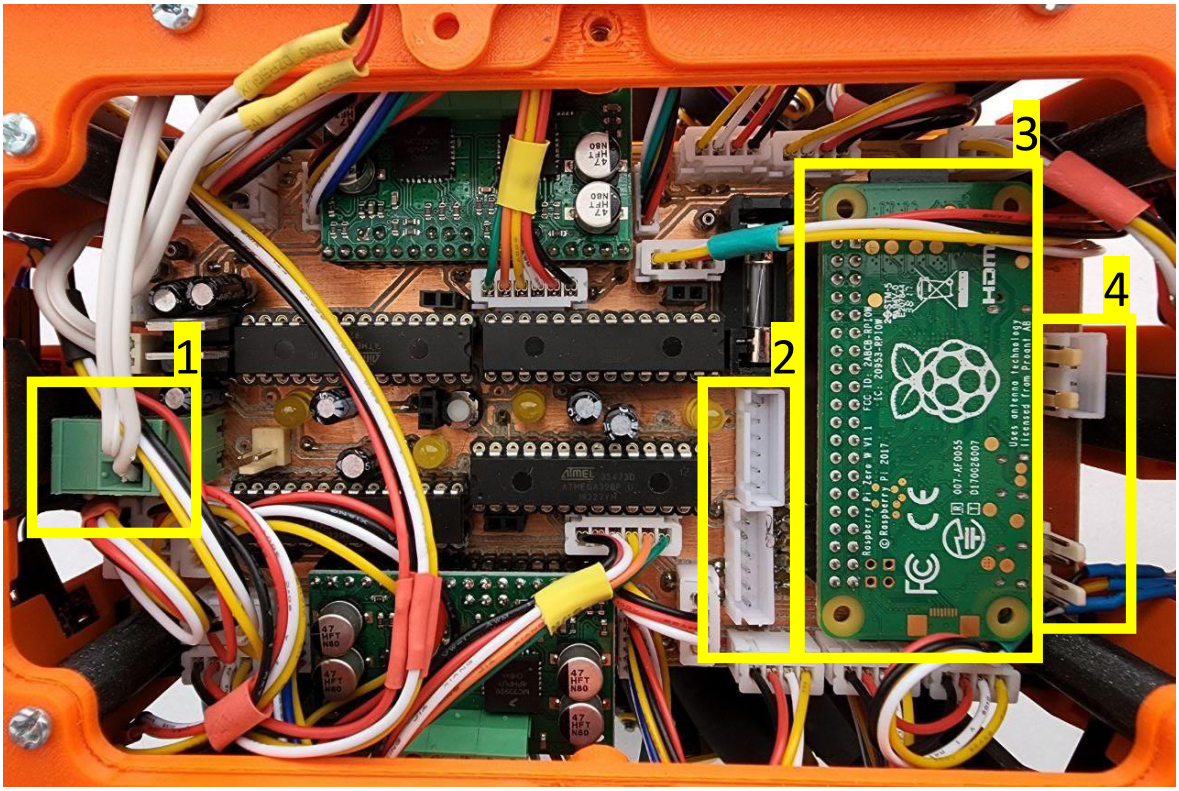



Si se desmonta la tapa de la plataforma móvil se tendrá acceso al circuito impreso.



Si se desmonta la tapa de la plataforma móvil se tendrá acceso al circuito impreso. Las conexiones se realizan de acuerdo a:

1. Conexión a la batería.
2. Conectores para motores a pasos del mecanismo de agarre.
3. Raspberry Pi Zero.
4. Conectores a fines de carrera del mecanismo de agarre.




## Conexión a Wifi a una nueva red:

Para conectar el robot a una nueva red se puede:

- Desmontar la tapa de la plataforma móvil y extrae el microprocesador Raspberry Pi Zero para conectarlo con un monitor y teclado externo.
- Conectar un monitor y un teclado externo mediante los puertos micro HDMI y micro USB directamente sin extraer el Raspberry Pi Zero.

# 4 Especificaciones Técnicas

- **Alimentación:** Batería de12V DC, 3Ah.
- **Tamaño:** 30x30x25 cm.

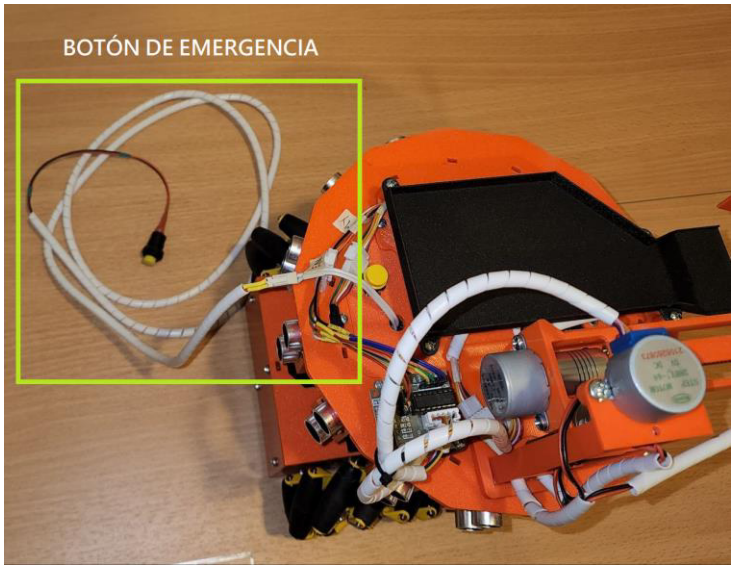
- **Ambiente de uso:** interiores.
- **Número de sensores:** 14 sensores ultrasónicos.
- **Comunicación**: Wifi, micro HDMI, micro USB, comunicación serial.
- **Sistema Operativo:** Raspberry Pi OS.
- **Autonomía:** 2 horas.
- **Carga extra máxima:** 500 gramos.
- **Velocidad máxima:** 3 cm/s.

# 8 Información de Seguridad

- No exponer al prototipo a impacto o daños físicos.
- Mantener en un lugar sin humedad y a temperaturas menores 30°C.
- No dejar desatendido el prototipo si se encuentra en funcionamiento.

## Condiciones Iniciales del Prototipo

- Verificar que no existan elementos (externos o propios del prototipo) sueltos que puedan causar un daño en los mecanismos.
- Realizar una inspección visual de las conexiones, puede que en el traslado del prototipo se desconecten algún elemento y esto causará errores inesperados.
- Es recomendable utilizar el prototipo con el pulsador de paro de emergencia conectado, sin embargo, si no hay elementos cercanos que el prototipo pueda dañar con sus movimientos, o que estos dañen el prototipo, se puede usar el sistema sin este elemento.




# 9 Mantenimiento del prototipo

## 9.1 Mantenimiento

- Verificar que las conexiones se encuentren en buen estado.
- Limpiar el mecanismo de agarre si es necesario.
- Lubricar los rodillos de las ruedas.
- Reemplazar la batería si la autonomía se ha reducido considerablemente.

## 9.2 Recomendaciones

- Operar el robot siempre con el ventilador encendido para evitar daños en el microprocesador por exceso de calor.
- Utilizar el prototipo en un suelo completamente liso para reducir el error por odometría.
- Asegurar una buena conexión a Internet para que la página web funcione correctamente.

# 10 Preguntas Frecuentes

**No se enciende el robot.**

Si no se prende ninguna luz del microprocesador, verificar la carga y conexión de la batería.

### No hay conectividad a Internet

Realizar el proceso de conexión manualmente con un monitor y un teclado externo.

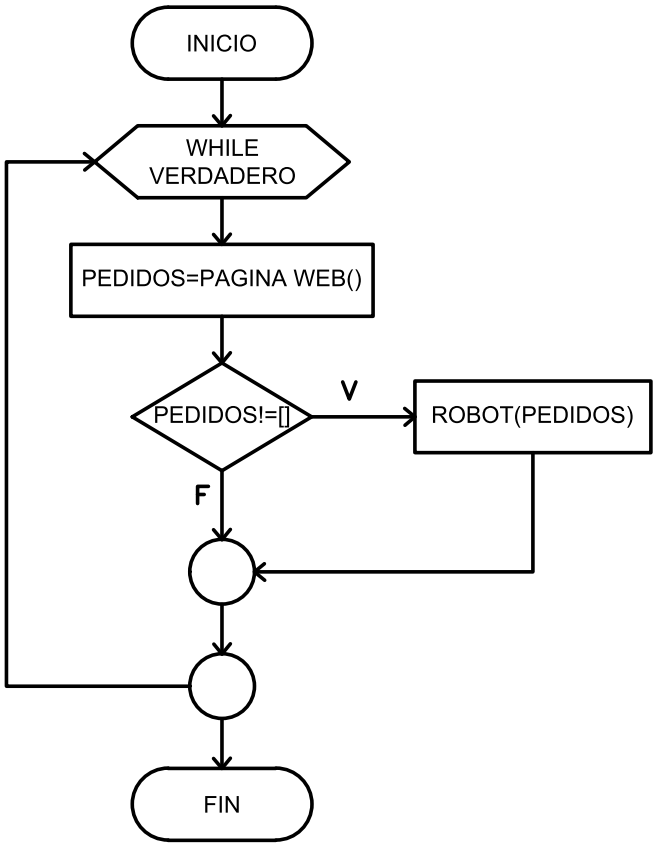
### No detecta obstáculos

Reducir la velocidad de desplazamiento a un valor menor o igual a 3 cm/s.

### Se enciende el robot, pero no se mueve

Reiniciar los microcontroladores con el botón de reinicio. Si no funciona verificar las conexiones de los sensores.

# Anexo D: Planos de Construcción




| | | | | | |
|---|---|---|---|---|---|
| UIDE | INGENIERÍA MECATRÓNICA | DIB. | VERNAZA M. | 18-07-2022 | |
| | | DIS. | VERNAZA M. | 01-05-2022 | |
| | | REV. | VELARDE. P | 20-07-2022 | |

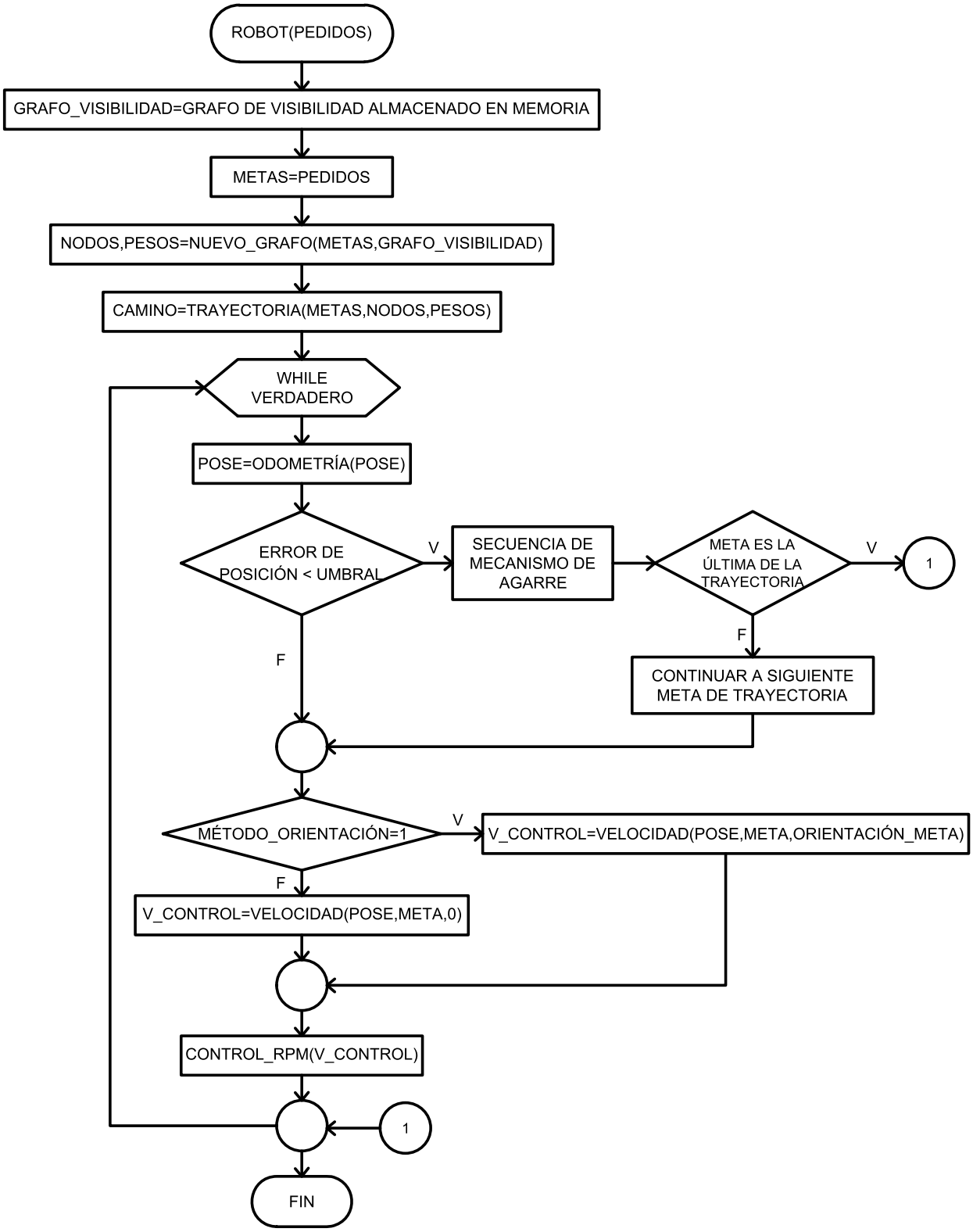
# PROCESO PRINCIPAL DE PEDIDOS EN PÁGINA WEB

DIAGRAMA DE FLUJO

D01-001

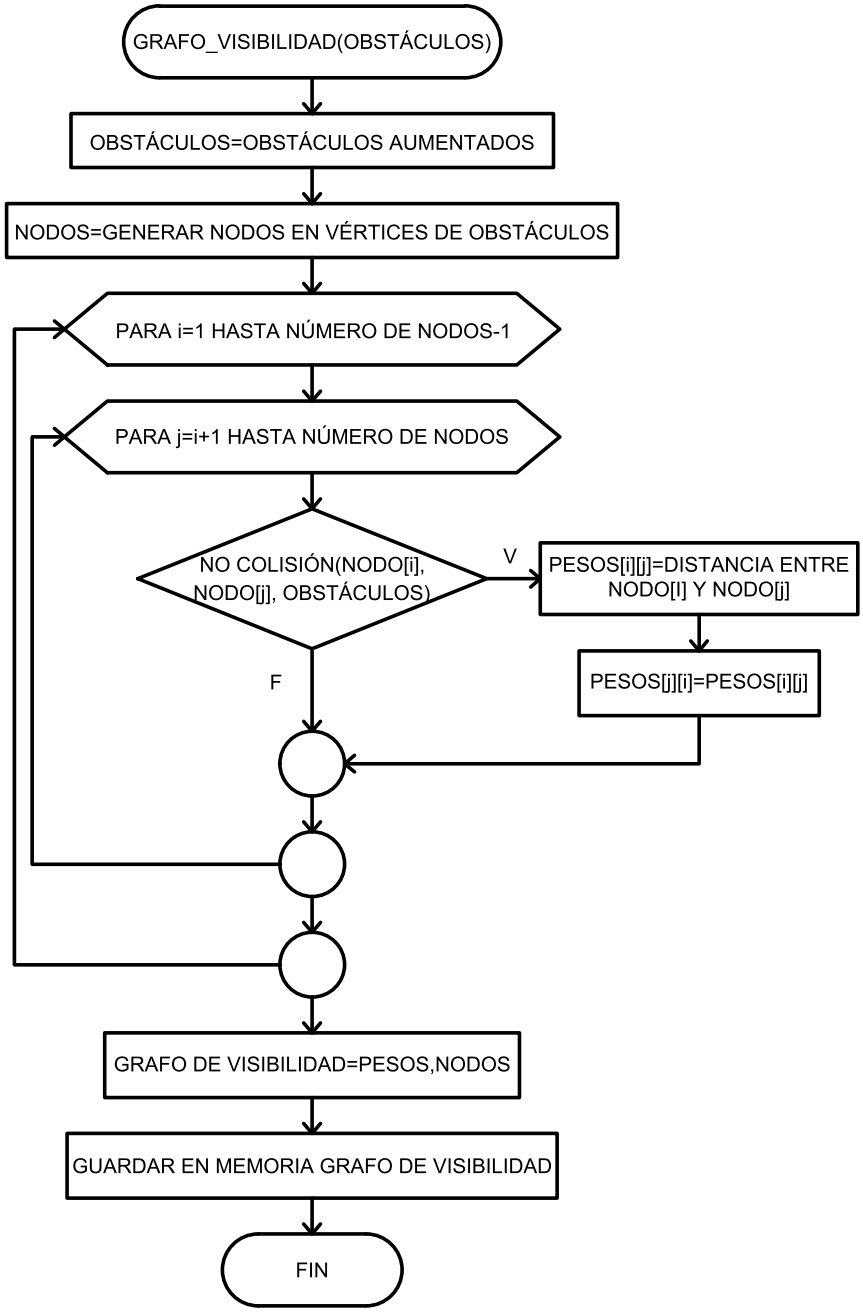
ESCALA N/A

N/A




| | | | | | |
|---|---|---|---|---|---|
| UIDE | INGENIERÍA MECATRÓNICA | DIB. | VERNAZA M. | 18-07-2022 | |
| | | DIS. | VERNAZA M. | 01-05-2022 | |
| | | REV. | VELARDE. P | 20-07-2022 | |

| | | |
|---|---|---|
| SUBPROCESO DE CONTROL DEL ROBOT<br>DIAGRAMA DE FLUJO | D01-002 | ESCALA<br>N/A<br>N/A |




| | | | | |
|---|---|---|---|---|
| UIDE | INGENIERÍA MECATRÓNICA | DIB. | VERNAZA M. | 18-07-2022 |
| | | DIS. | VERNAZA M. | 01-05-2022 |
| | | REV. | VELARDE. P | 20-07-2022 |

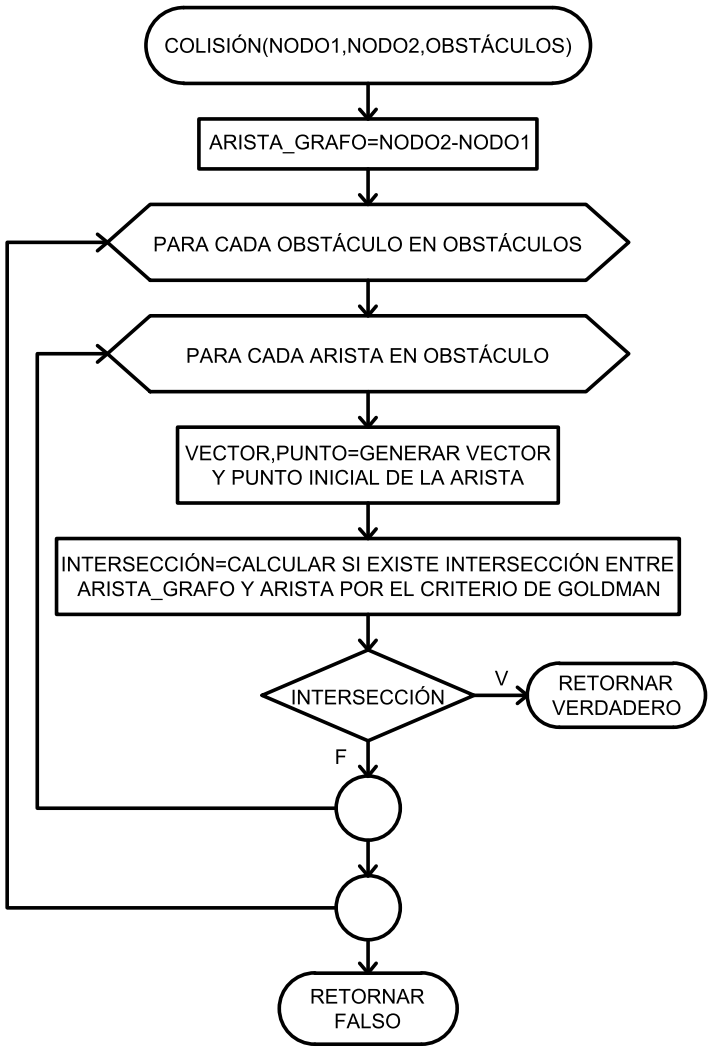
# SUBPROCESO DE GENERACIÓN DEL GRAFO DE VISIBILIDAD

DIAGRAMA DE FLUJO

D01-003

ESCALA N/A

N/A




| | | | | |
|---|---|---|---|---|
| UIDE | INGENIERÍA MECATRÓNICA | DIB. | VERNAZA M. | 18-07-2022 |
| | | DIS. | VERNAZA M. | 01-05-2022 |
| | | REV. | VELARDE. P | 20-07-2022 |

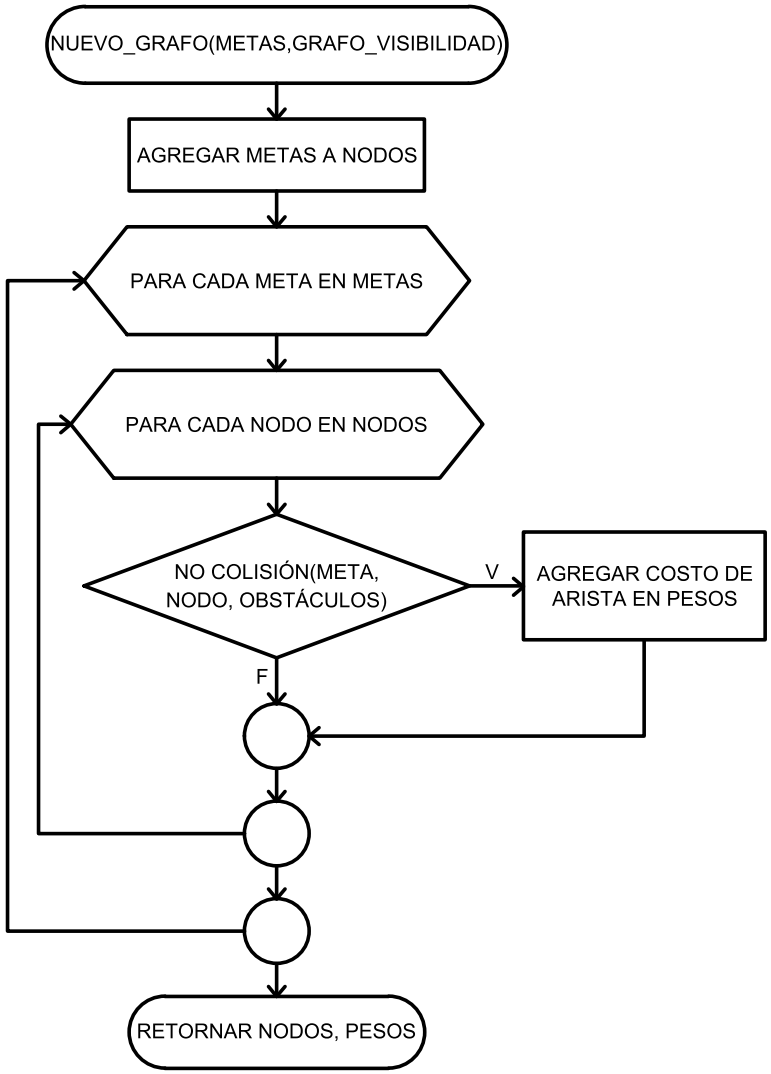
# SUBPROCESO DE DETECCIÓN DE COLISIÓN EN GENERACIÓN DE ARISTAS

DIAGRAMA DE FLUJO

D01-004

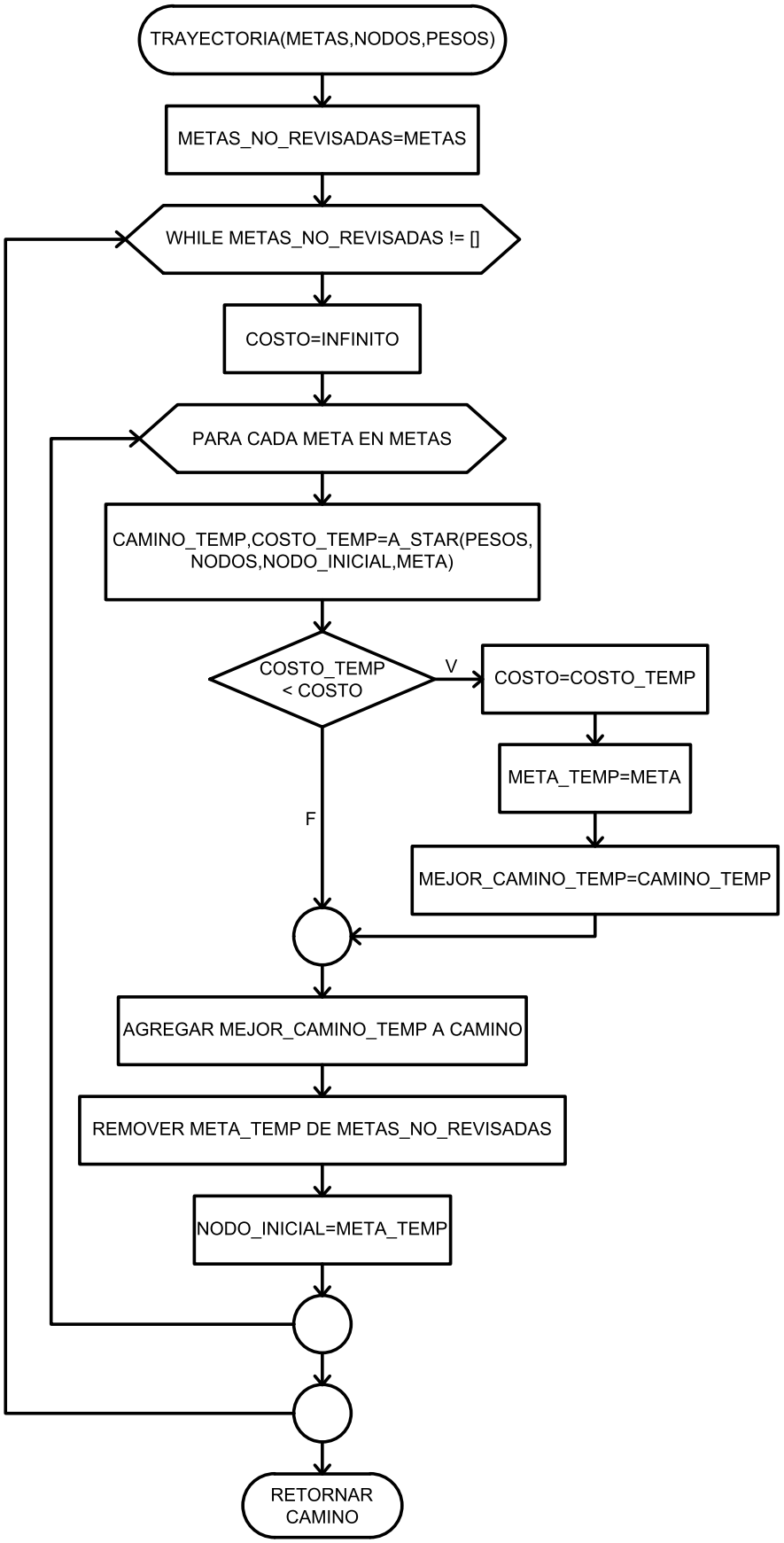
ESCALA N/A

N/A




| | | | | | |
|---|---|---|---|---|---|
| UIDE | INGENIERÍA MECATRÓNICA | DIB. | VERNAZA M. | 18-07-2022 | |
| | | DIS. | VERNAZA M. | 01-05-2022 | |
| | | REV. | VELARDE. P | 20-07-2022 | |

| | | |
|---|---|---|
| SUBPROCESO DE GENERACIÓN DE NUEVO GRAFO<br>DIAGRAMA DE FLUJO | D01-005 | ESCALA<br>N/A<br>N/A |




| | | | | | |
|---|---|---|---|---|---|
| UIDE | INGENIERÍA MECATRÓNICA | DIB. | VERNAZA M. | 18-07-2022 | |
| | | DIS. | VERNAZA M. | 01-05-2022 | |
| | | REV. | VELARDE. P | 20-07-2022 | |

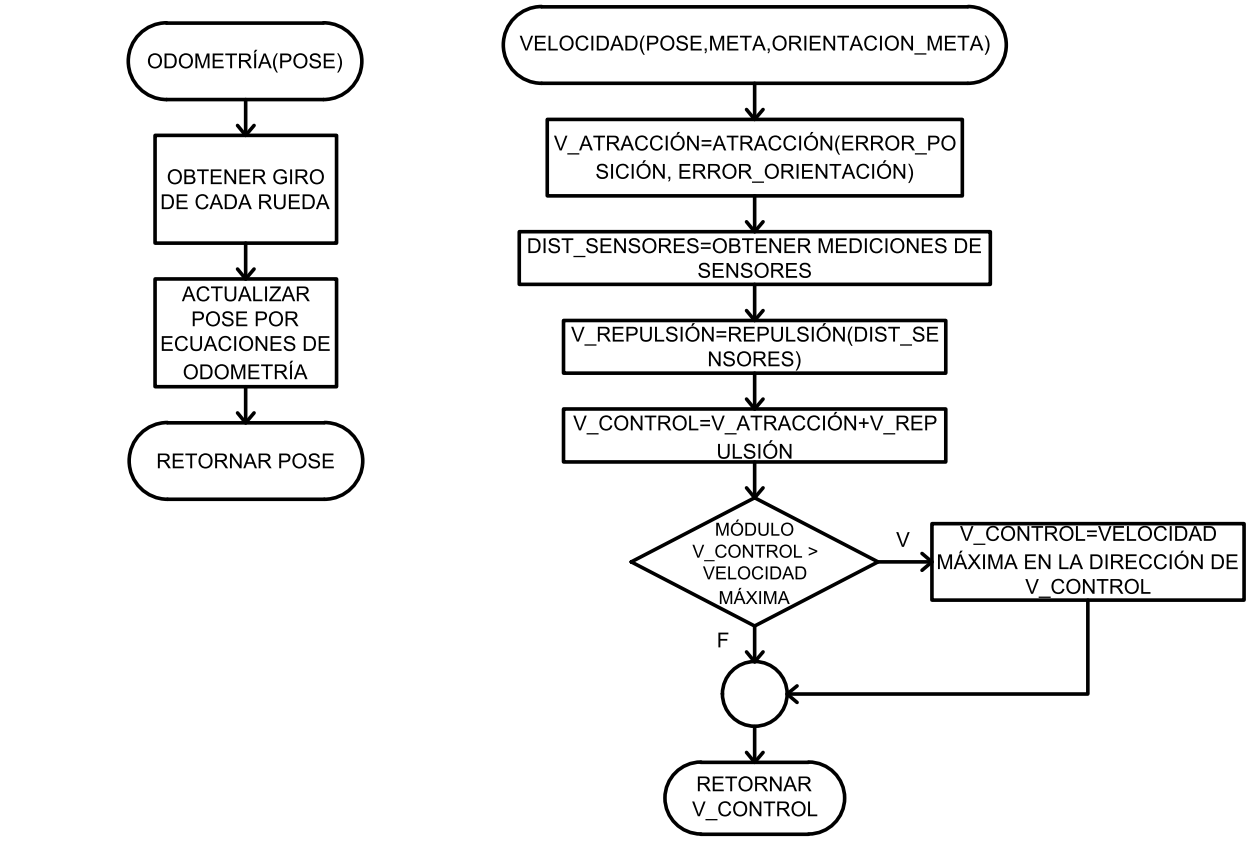
# SUBPROCESO DE GENERACIÓN DE TRAYECTORIA

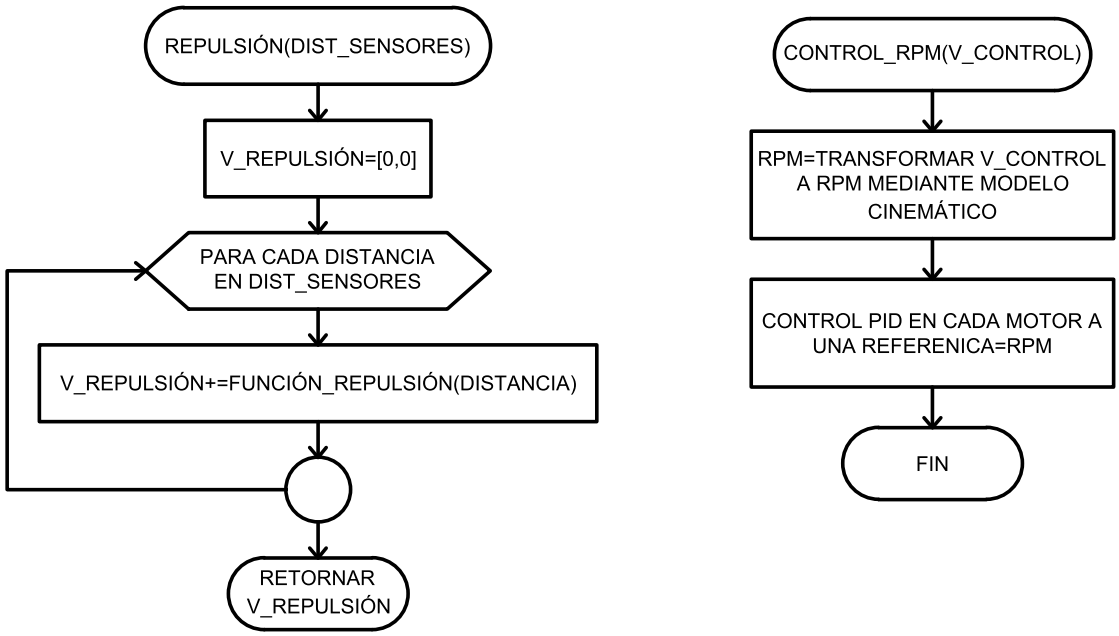
DIAGRAMA DE FLUJO

D01-006

ESCALA N/A

N/A




| | | | | |
|---|---|---|---|---|
| UIDE | INGENIERÍA MECATRÓNICA | DIB. | VERNAZA M. | 18-07-2022 |
| | | DIS. | VERNAZA M. | 01-05-2022 |
| | | REV. | VELARDE. P | 20-07-2022 |

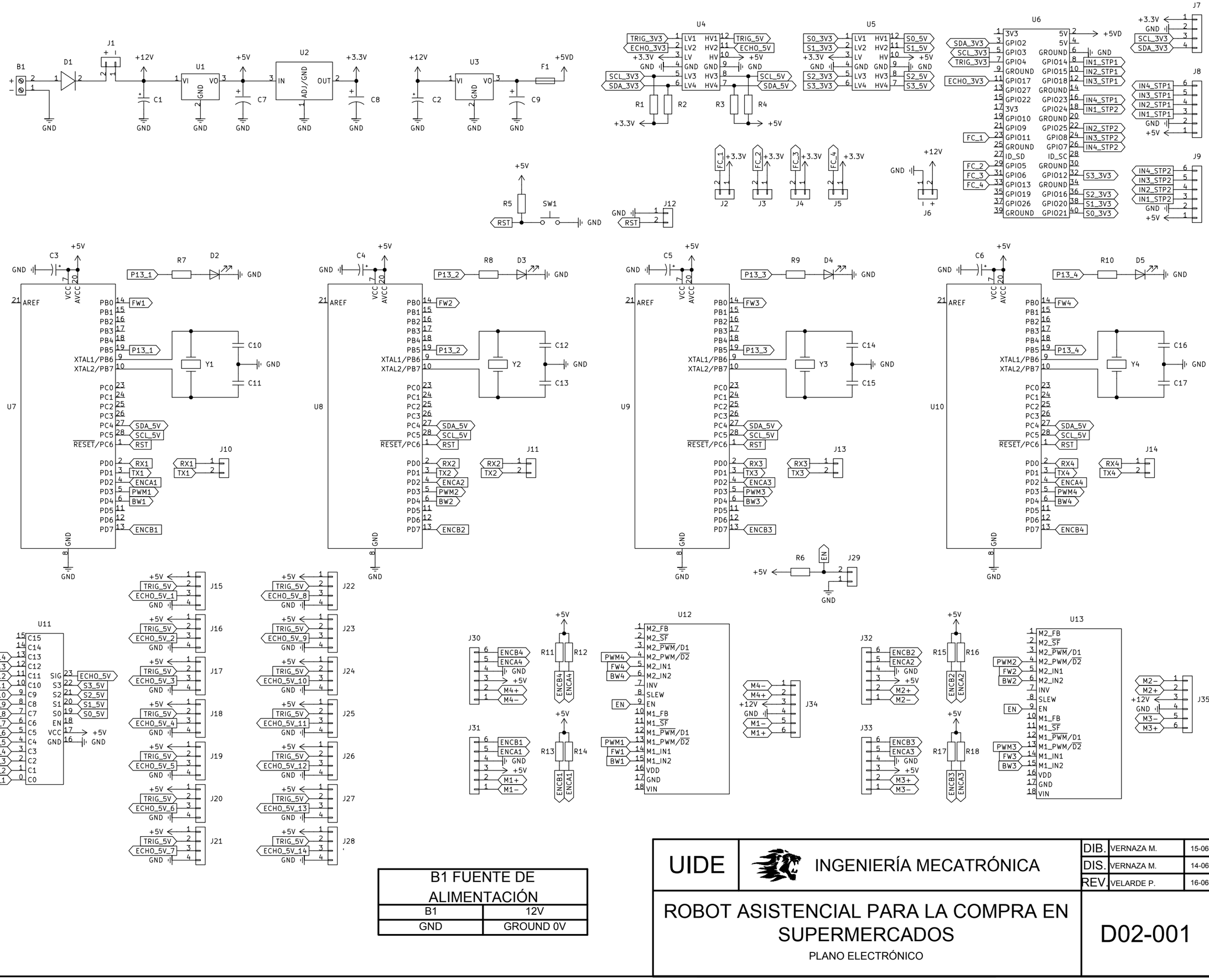
SUBPROCESOS: ODOMETRÍA, CÁLCULO DE VELOCIDAD DE CONTROL, REPULSIÓN DE OBSTÁCULOS Y CONTROL RPM

DIAGRAMA DE FLUJO

D01-007

ESCALA N/A

N/A


B1 FUENTE DE ALIMENTACIÓN
B1 12V
GND GROUND 0V
UIDE
INGENIERÍA MECATRÓNICA
DIB. VERNAZA M. 15-06-2022
DIS. VERNAZA M. 14-06-2022
REV. VELARDE P. 16-06-2022
ROBOT ASISTENCIAL PARA LA COMPRA EN SUPERMERCADOS
PLANO ELECTRÓNICO
D02-001
ESCALA N/A
N/A

| POS. | DESCRIPCIÓN | CANT. | OBSERVACIONES |
|---|---|---|---|
| | BORNERA | | |
| B1 | BORNERA ENCHUFABLE 2 PINES | 1 | ALIMENTACIÓN BATERÍA Ni-MH 12V 3Ah |
| | CAPACITORES | | |
| C1-C6 | 0.1uF 50V | 6 | ELECTROLÍTICO |
| C7-C9 | 10uF 50V | 3 | ELECTROLÍTICO |
| C10-C17 | 22pF | 8 | CERÁMICO |
| | DIODOS | | |
| D1 | 1N4007 | 1 | |
| D2-D5 | LED | 4 | COLOR AMARILLO |
| | FUSIBLE | | |
| F1 | 1A | 1 | |
| | CONECTORES | | |
| J1 | MOLEX 2 PINES | 1 | ENCENDIDO |
| J2-J5 | MOLEX 2 PINES | 4 | FIN DE CARRERA |
| J6 | MOLEX 2 PINES | 1 | VENTILADOR 12V |
| J7 | JST XH 4 PINES | 1 | DISPLAY OLED 0.96" |
| J8-J9 | JST XH 6 PINES | 2 | DRIVER ULN2003 / MOTOR BYJ48 |
| J10-J14 | ESPADINES HEMBRA 2 PINES | 5 | COMUNICACIÓN SERIAL |
| J15-J28 | JST XH 4 PINES | 14 | SENSOR ULTRASÓNICO HC-SR04 |
| J29 | MOLEX 2 PINES | 1 | DESHABILITACIÓN MOTORES |
| J30-J33 | JST ZH 6 PINES | 4 | MOTORES N20 |
| J34-J35 | JST XH 6 PINES | 2 | SALIDA DRIVER MC33926 |
| | RESISTENCIAS | | |
| R1-R2 | 2.7KΩ 1/4W | 2 | PULL-UP |
| R3-R6 | 10KΩ 1/4W | 4 | PULL-UP |
| R7-R10 | 330Ω 1/4W | 4 | LIMITADOR DE CORRIENTE |
| R11-R18 | 1kΩ 1/4W | 8 | PULL-UP |
| | INTERRUPTORES | | |
| SW1 | PULSADOR | 1 | |

UIDE

INGENIERÍA MECATRÓNICA

| | | | |
|---|---|---|---|
| DIB. | VERNAZA M. | 15-06-2022 | |
| DIS. | VERNAZA M. | 14-06-2022 | |
| REV. | VELARDE P. | 16-06-2022 | |

# ROBOT ASISTENCIAL PARA LA COMPRA EN SUPERMERCADOS

LISTA DE COMPONENTES

D02-101

ESCALA N/A

N/A

| POS. | DESCRIPCIÓN | CANT. | OBSERVACIONES |
|---|---|---|---|
| | INTEGRADOS | | |
| U1-U2 | LM7805 | 2 | REGULADOR 5V |
| U3 | LD1086V33 | 1 | REGULADOR 3.3V |
| U4-U5 | CONVERSOR NIVEL LÓGICO | 2 | BIDIRECCIONAL DE 4 CANALES |
| U6 | RASPBERRY PI ZERO W | 1 | |
| U7-U10 | ATMEGA328P | 4 | |
| U11 | MULTIPLEXOR CD74HC4067 | 1 | 16 CANALES |
| U12-U13 | DRIVER DUAL MC33926 | 2 | DRIVER MOTOR DC |
| | OSCILADORES | | |
| Y1-Y4 | 16MHz | 4 | OSCILADOR DE CRISTAL |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

UIDE

INGENIERÍA MECATRÓNICA

| | | | |
|---|---|---|---|
| DIB. | VERNAZA M. | 15-06-2022 | |
| DIS. | VERNAZA M. | 14-06-2022 | |
| REV. | VELARDE P. | 16-06-2022 | |

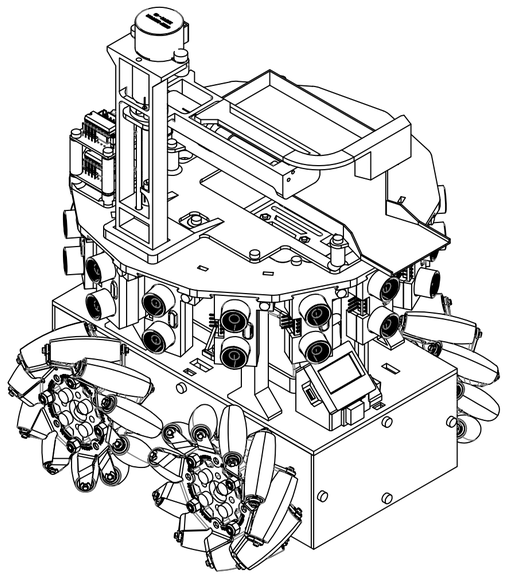
# ROBOT ASISTENCIAL PARA LA COMPRA EN SUPERMERCADOS

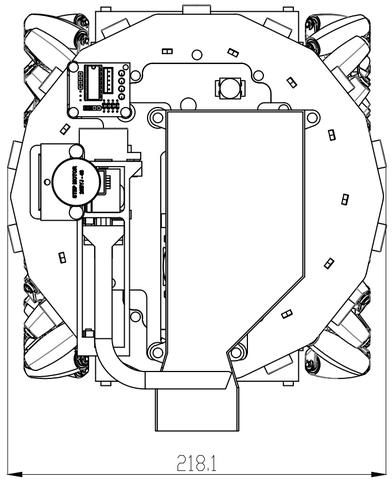
LISTA DE COMPONENTES

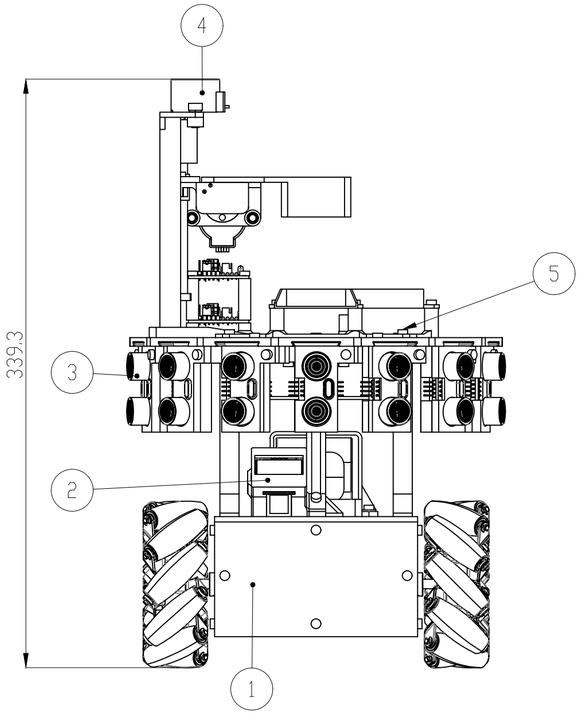
D02-102

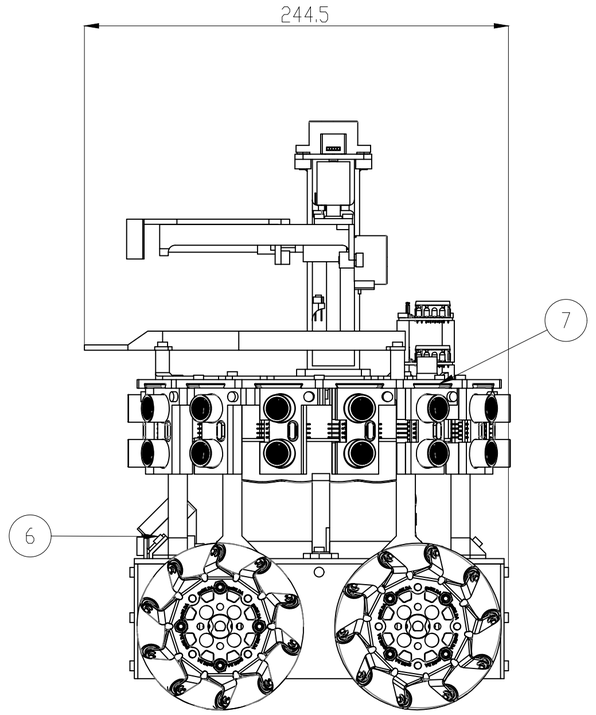
ESCALA N/A

N/A






| PROTOTIPO | |
|---|---|
| **DESCRIPCIÓN** | **MEDIDAS** |
| LARGO | 244.5 mm |
| ANCHO | 214.4 mm |
| ALTURA | 339.3 mm |
| PESO | 3.25 kg |
| ALIMENTACIÓN | 12 VDC |
| AMBIENTE DE TRABAJO | INTERIOR |

| POS | ZONA | DENOMINACIÓN | CANT | MATERIAL | PLANO/ NORMA | OBSERVACIONES |
|---|---|---|---|---|---|---|
| 7 | D8 | TUERCA M3 | 1 | ACERO INOXIDABLE | NA | |
| 6 | D7 | PERNO M3X20 | 2 | ACERO SAE | NA | |
| 5 | D4 | PERNO M3X15 | 4 | ACERO SAE | NA | |
| 4 | C4 | MECANISMO DE AGARRE DE PRODUCTOS | 1 | VARIOS | D03-005 | |
| 3 | D4 | PLATAFORMA DE SENSORES | 1 | VARIOS | D03-004 | |
| 2 | D4 | PANTALLA OLED | 1 | VARIOS | D03-003 | |
| 1 | E4 | PLATAFORMA MÓVIL | 1 | VARIOS | D03-002 | |

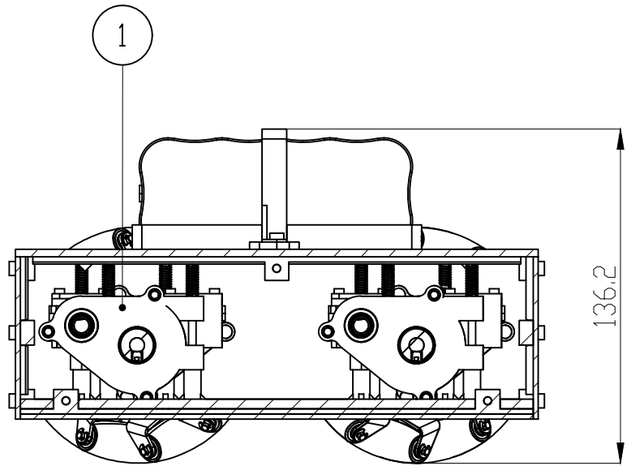
UIDE — INGENIERÍA MECATRÓNICA

| | | |
|---|---|---|
| DIB. | VERNAZA M. | 01/04/2022 |
| DIS. | VERNAZA M. | 28/07/2022 |
| REV. | VELARDE P. | 29/07/2022 |

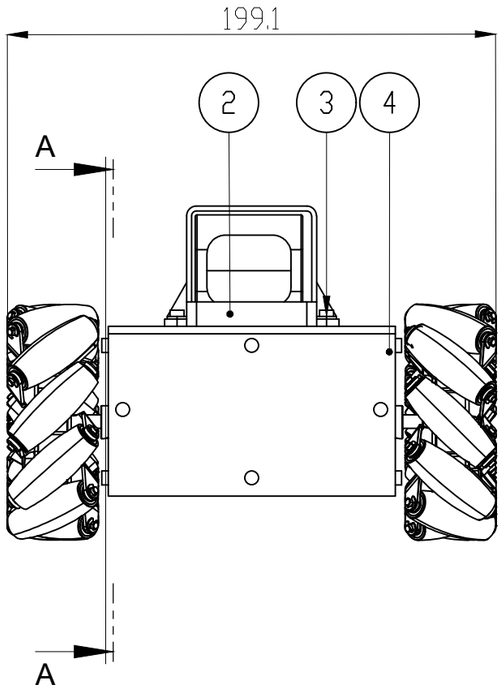
PROTOTIPO DE ROBOT ASISTENCIAL PARA COMPRAS EN SUPERMERCADOS

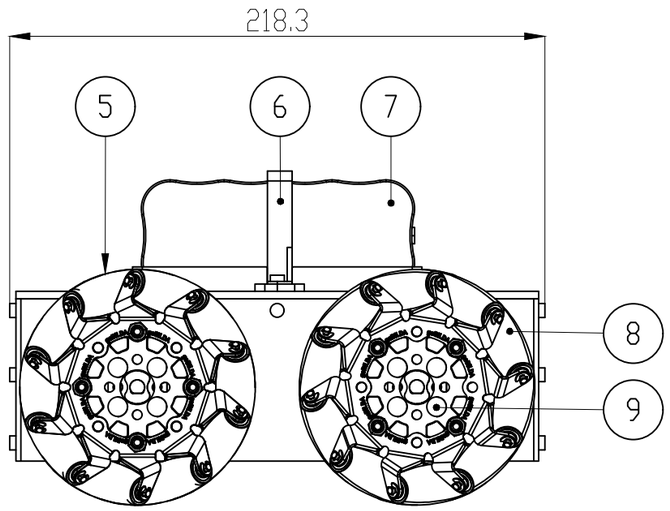
D03-001

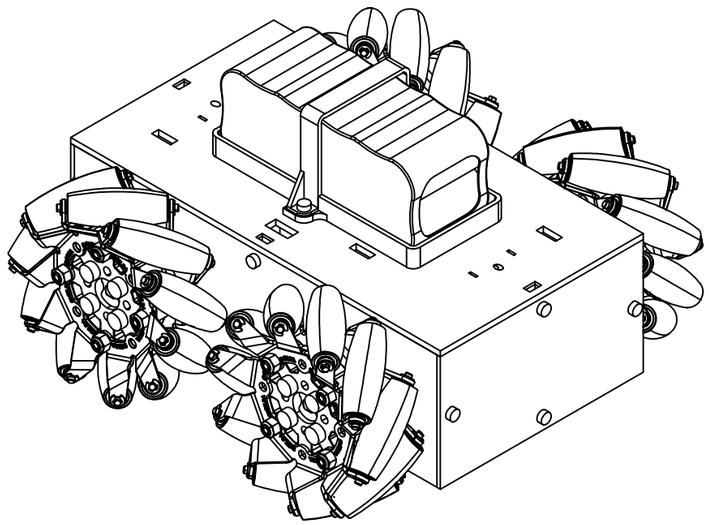
ESCALA: 1:2.5




SECCIÓN A-A
ESCALA 1 : 2.5



| PLATAFORMA MÓVIL | |
|---|---|
| **DESCRIPCIÓN** | **MEDIDAS** |
| LARGO | 218.3 mm |
| ANCHO | 199.1 mm |
| ALTURA | 136.2 mm |

| POS | ZONA | DENOMINACIÓN | CANT | MATERIAL | PLANO/ NORMA | OBSERVACIÓN |
|---|---|---|---|---|---|---|
| 9 | B7 | PERNO M4X10 | 16 | ACERO SAE | NA | |
| 8 | B7 | RUEDA MECANUM 2 | 2 | VARIOS | NA | INCLINACIÓN DERECHA |
| 7 | B7 | BATERÍA | 1 | VARIOS | NA | NI-MH 12V 3Ah |
| 6 | B7 | SOPORTE BATERÍA 2 | 1 | PLA | D03-202 | |
| 5 | B6 | RUEDA MECANUM 1 | 2 | VARIOS | NA | INCLINACIÓN IZQUIERDA |
| 4 | B4 | CAJA DE MOTORES | 1 | VARIOS | D03-007 | |
| 3 | B5 | PERNO M3X10 | 2 | ACERO SAE | NA | |
| 2 | B4 | SOPORTE BATERÍA 1 | 1 | PLA | D03-201 | |
| 1 | B2 | CAJA REDUCTORA | 4 | VARIOS | D03-006 | |

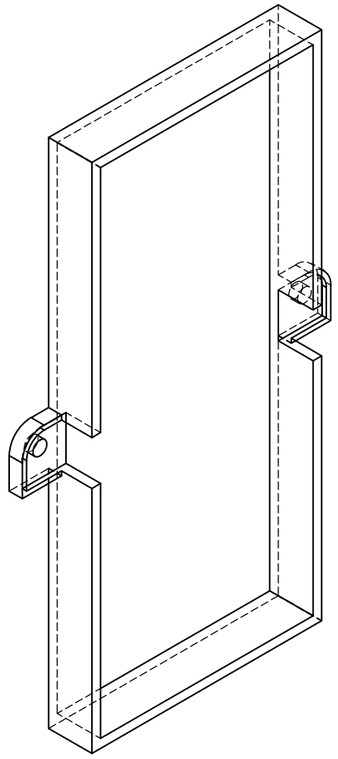
UIDE — INGENIERÍA MECATRÓNICA

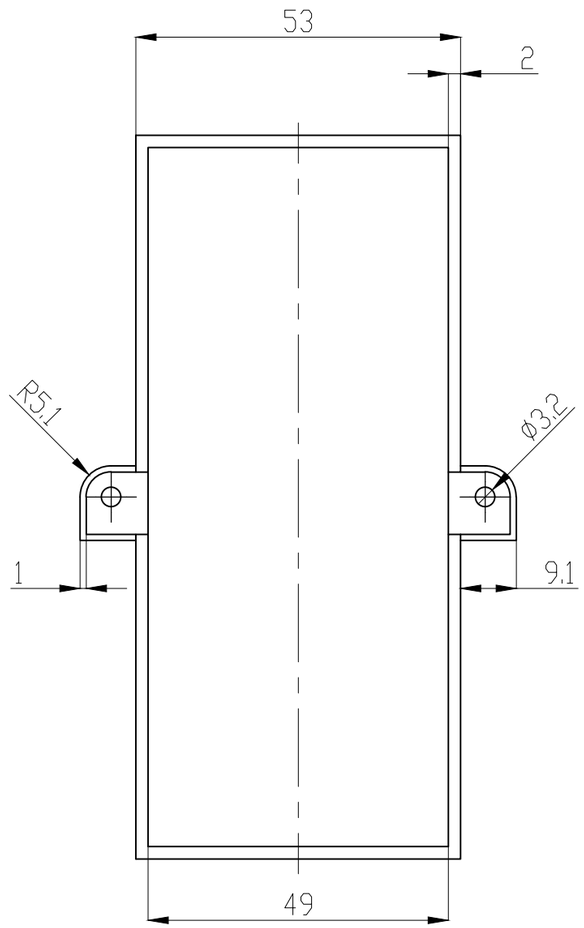
| | | |
|---|---|---|
| DIS | VERNAZA M. | 16/07/2022 |
| DIB | VERNAZA M. | 29/07/2022 |
| REV | VELARDE P. | 29/07/2022 |

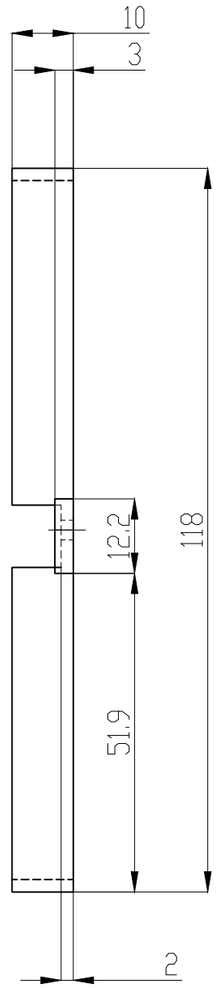
PLATAFORMA MÓVIL

D03-002

ESCALA 1:2.5



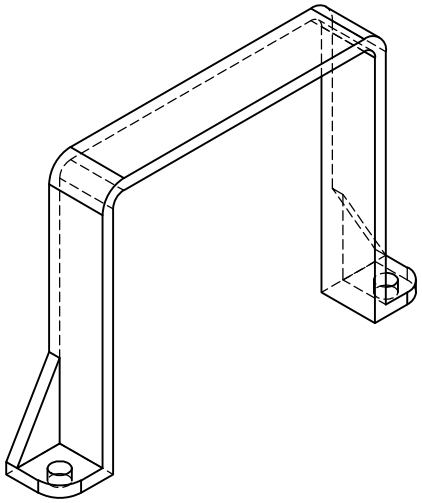


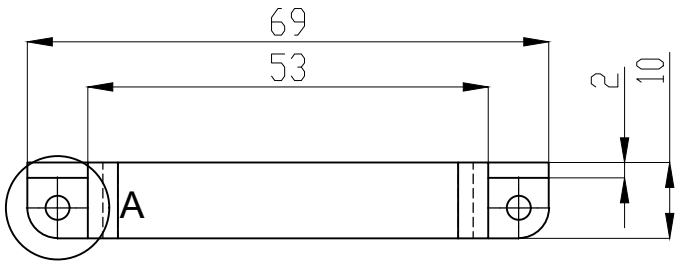



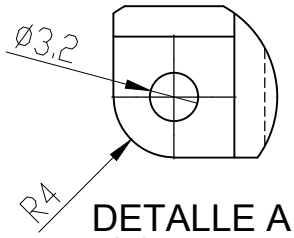

NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

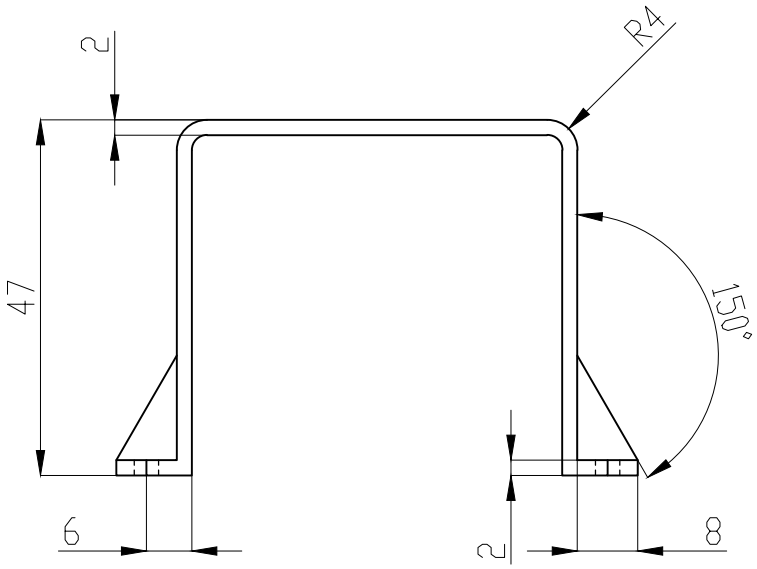
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: | PLA | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SOPORTE BATERÍA 1 | | | D03-201 | | | | |





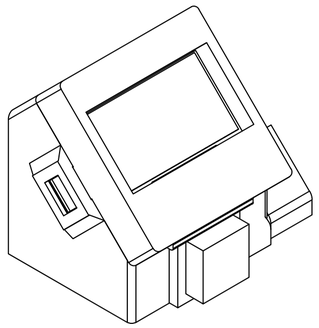

DETALLE A
ESCALA 2 : 1

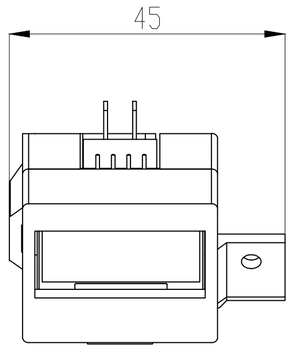


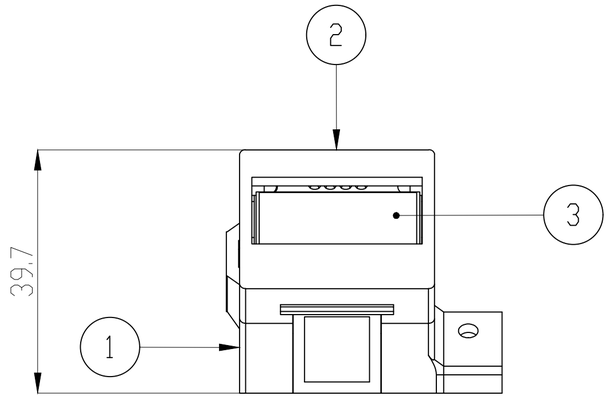

NOTA: IMPRESIÓN 3D

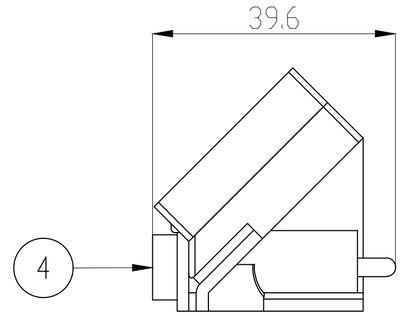
-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SOPORTE BATERÍA 2 | | | D03-202 | | | | |






| PANTALLA | |
|---|---|
| **DESCRIPCIÓN** | **MEDIDAS** |
| LARGO | 39.6 mm |
| ANCHO | 45 mm |
| ALTURA | 39.7 mm |

| POS | ZONA | DENOMINACIÓN | CANT | MATERIAL | PLANO/ NORMA | OBSERVACIÓN |
|---|---|---|---|---|---|---|
| 4 | C7 | BOTÓN | 1 | VARIOS | NA | CUADRADO 14.3X14.3 mm |
| 3 | C5 | DISPLAY OLED | 1 | VARIOS | NA | COM. I2C, 0.96" |
| 2 | C5 | SOPORTE FRONTAL DISPLAY | 1 | PLA | D03-302 | |
| 1 | C5 | SOPORTE TRASERO DISPLAY | 1 | PLA | D03-301 | |

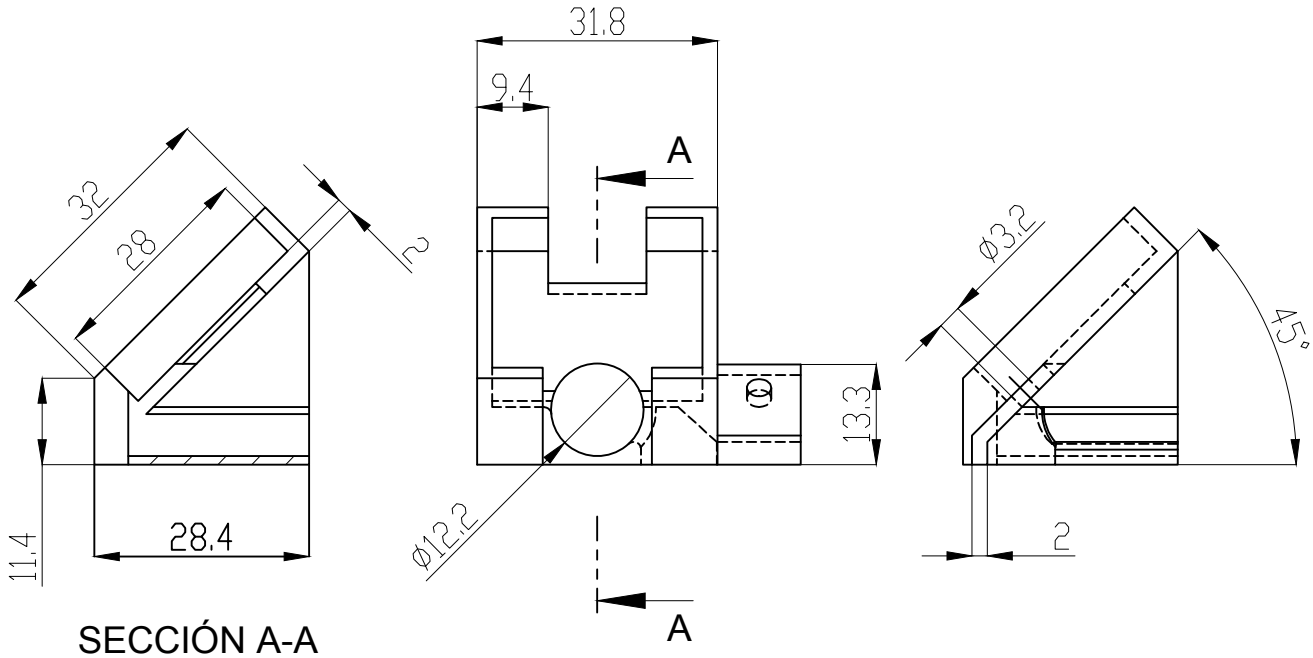
UIDE — INGENIERÍA MECATRÓNICA

| | | |
|---|---|---|
| DIS | VERNAZA M. | 16/07/2022 |
| DIB | VERNAZA M. | 29/07/2022 |
| REV | VELARDE P. | 29/07/2022 |

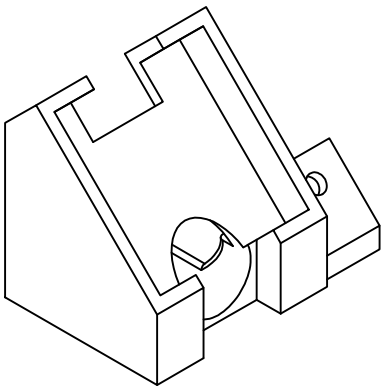
PANTALLA

D03-003

ESCALA 1:1



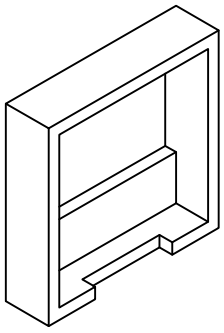

SECCIÓN A-A

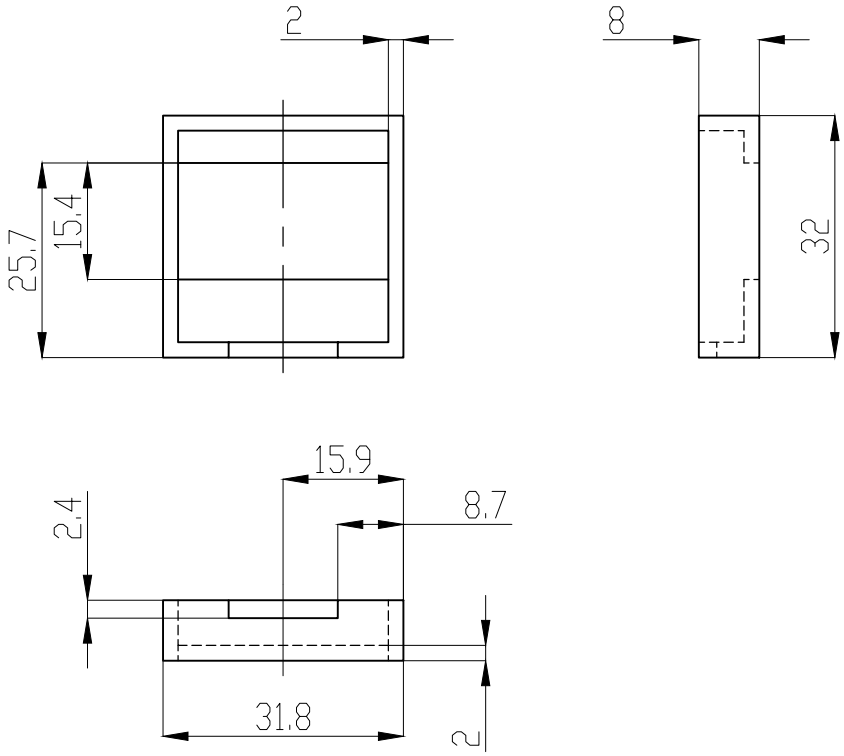


NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SOPORTE TRASERO DISPLAY | | | D03-301 | | | | |

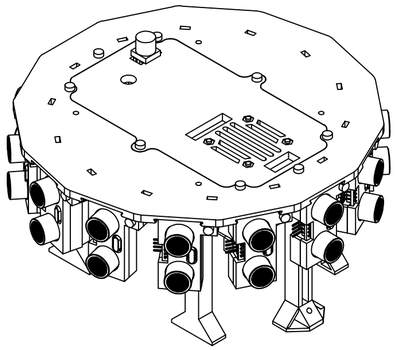


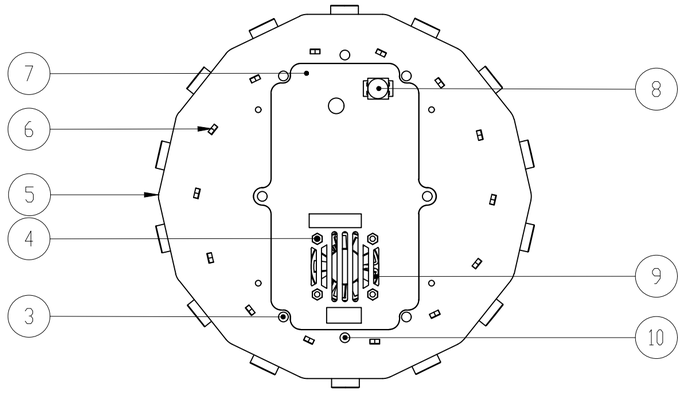


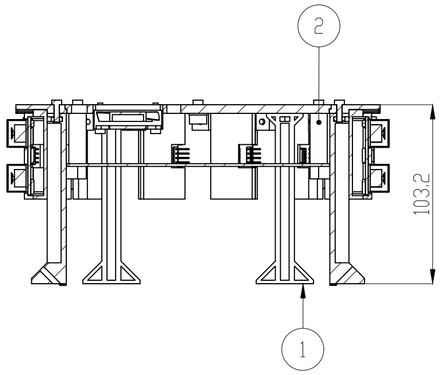

NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

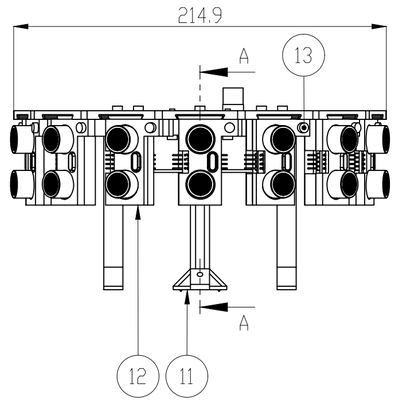
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SOPORTE FRONTAL DISPLAY | | | D03-302 | | | | |

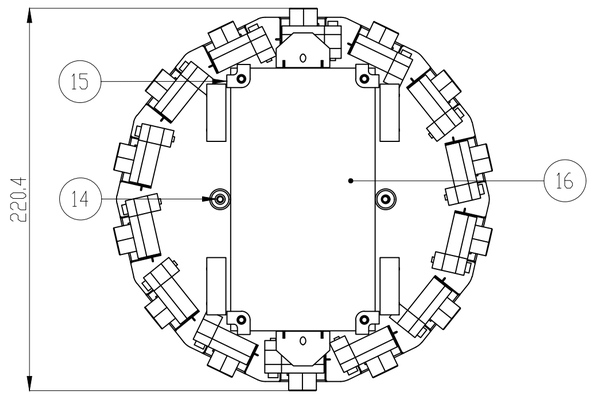





SECCIÓN A-A
ESCALA 1 : 2.5




| PLATAFORMA DE SENSORES | |
|---|---|
| DESCRIPCIÓN | MEDIDAS |
| LARGO | 214.9 mm |
| ANCHO | 220.4 mm |
| ALTURA | 103.2 mm |

| POS | ZONA | DENOMINACIÓN | CANT | MATERIAL | PLANO/ NORMA | OBSERVACIONES |
|---|---|---|---|---|---|---|
| 16 | E5 | PCB | 1 | VARIOS | D02-001 | |
| 15 | E4 | RETENEDOR DE PCB | 4 | PLA | D03-406 | |
| 14 | E4 | INSERTO M3 | 4 | LATÓN | NA | MARCA TOKAI |
| 13 | C5 | PERNO M3X20 | 14 | ACERO SAE | NA | |
| 12 | C5 | SOPORTE SENSOR | 14 | VARIOS | D03-008 | |
| 11 | D5 | SOPORTE FRONTAL BASE | 2 | PLA | D03-405 | |
| 10 | B5 | PERNO M3X10 | 4 | ACERO SAE | NA | |
| 9 | B5 | VENTILADOR | 1 | VARIOS | NA | 12V, 40x40mm |
| 8 | B5 | PULSADOR | 1 | VARIOS | NA | NORMALMENTE ABIERTO |
| 7 | B5 | TAPA CIRCUITO | 1 | PETG | D03-404 | |
| 6 | B4 | TUERCA M3 | 20 | ACERO INOXIDABLE | NA | |
| 5 | B5 | SOPORTE ULTRASÓNICOS | 1 | PETG | D03-403 | |
| 4 | B4 | PERNO M3X15 | 4 | ACERO SAE | NA | |
| 3 | B4 | PERNO M3X40 | 4 | ACERO SAE | NA | |
| 2 | C3 | SOPORTE PCB | 4 | PLA | D03-402 | |
| 1 | D3 | SOPORTE LATERAL BASE | 4 | PLA | D03-401 | |

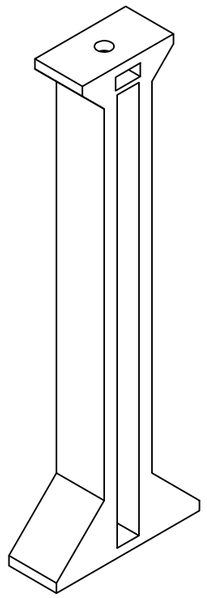
UIDE — INGENIERÍA MECATRÓNICA

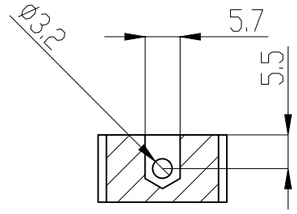
| | | |
|---|---|---|
| DIB. | VERNAZA M. | 01/04/2022 |
| DIS. | VERNAZA M. | 28/07/2022 |
| REV. | VELARDE P. | 29/07/2022 |

PLATAFORMA DE SENSORES

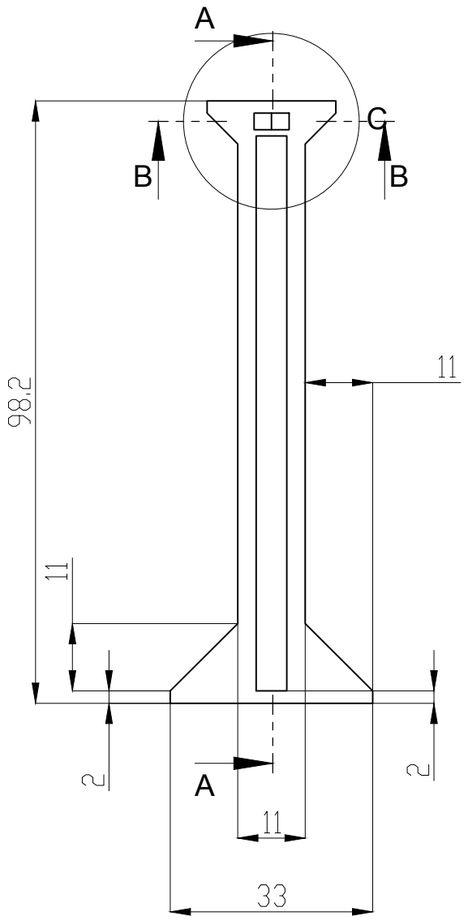
D03-004

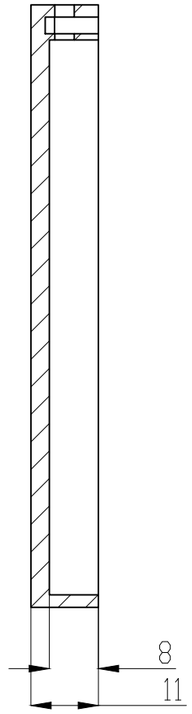
ESCALA: 1:2.5



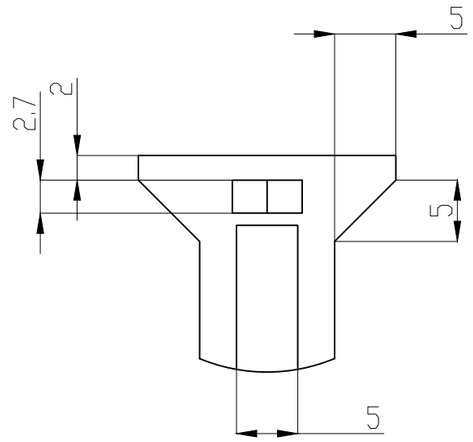



SECCIÓN B-B




SECCIÓN A-A

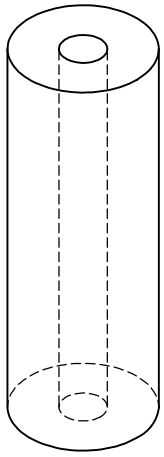


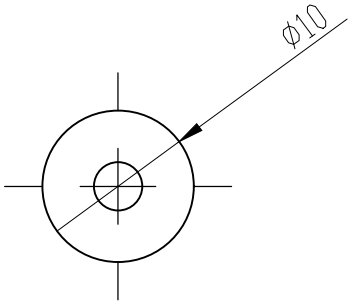

DETALLE C
ESCALA 2 : 1

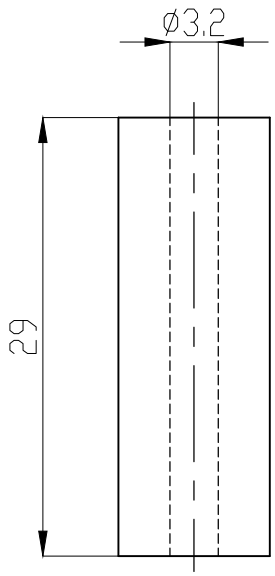
NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: | PLA | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 17/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SOPORTE LATERAL BASE | | | D03-401 | | | | |



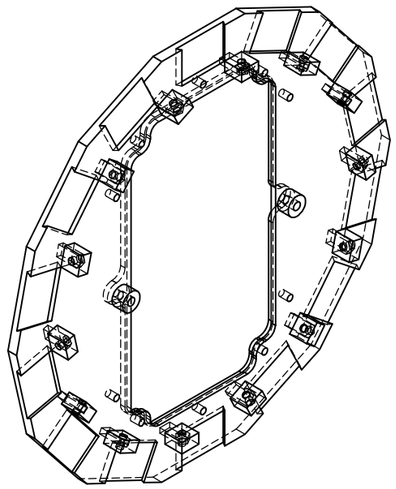


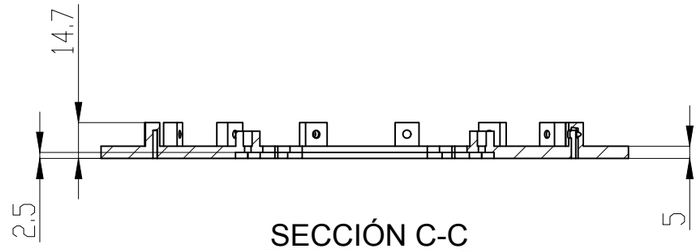




NOTA: IMPRESIÓN 3D

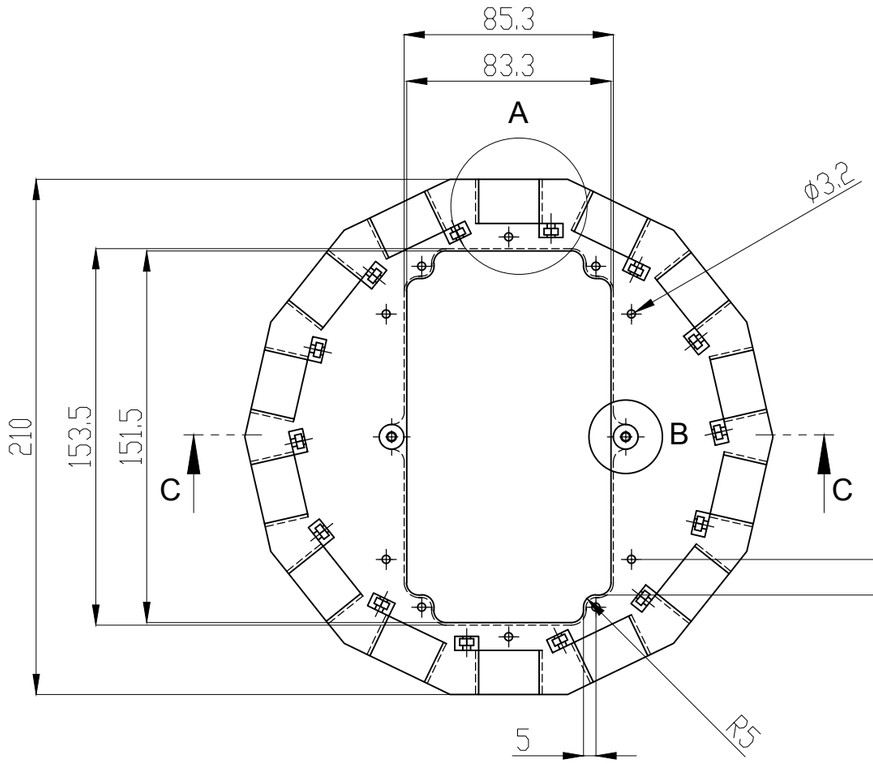
-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

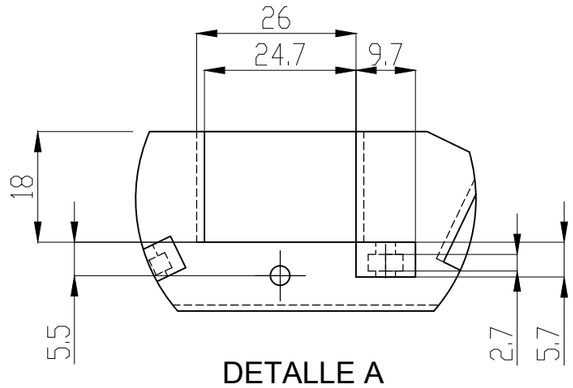
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 2:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SOPORTE PCB | | | D03-402 | | | | |



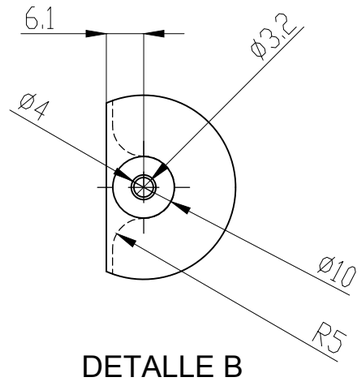



SECCIÓN C-C
ESCALA 1 : 2.5




DETALLE A
ESCALA 1 : 1

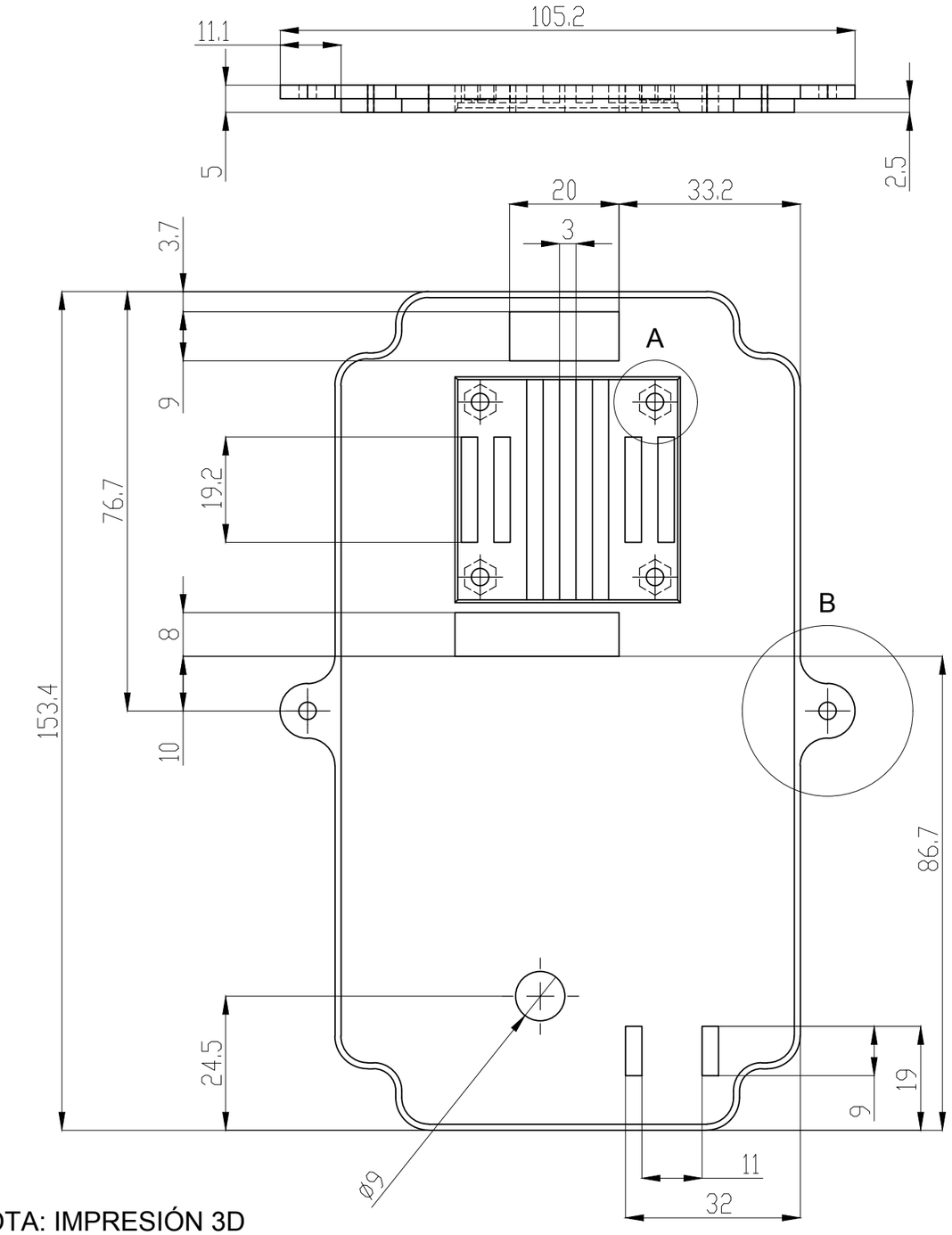


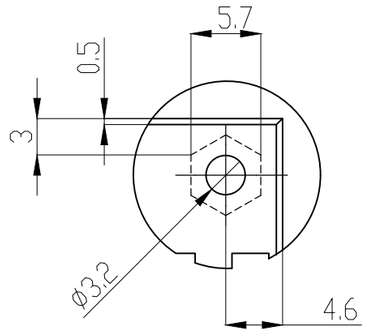

DETALLE B
ESCALA 1 : 1

NOTA: IMPRESIÓN 3D

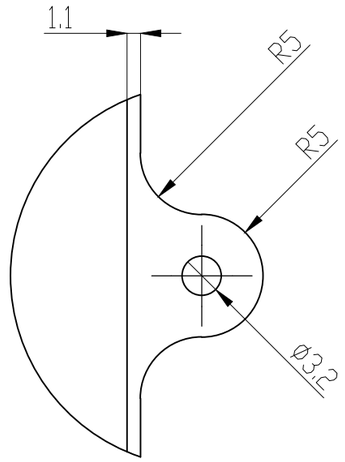
-Temperatura de Boquilla: 240°C
-Temperatura de Cama: 95°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: | PETG | TOL. GRAL ± 0.1 | ESCALA 1:2.5 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 14/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SOPORTE ULTRASÓNICOS | | | D03-403 | | | | |

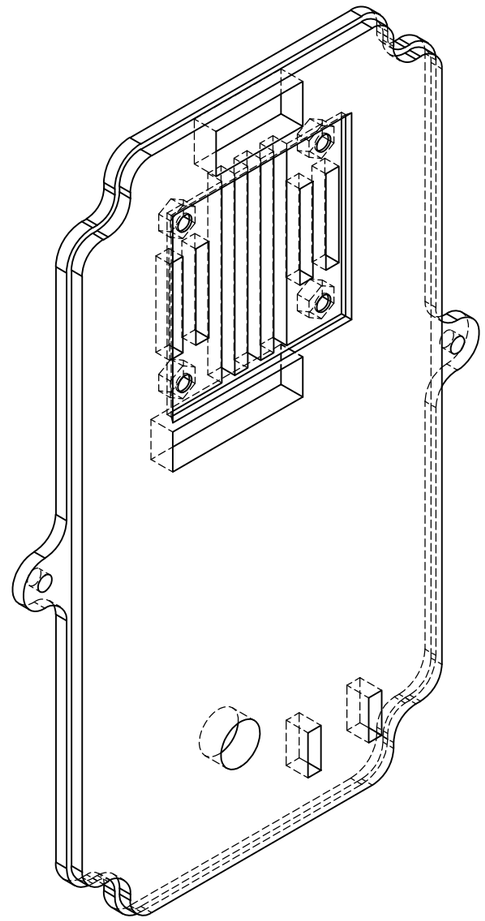



DETALLE A
ESCALA 2 :1



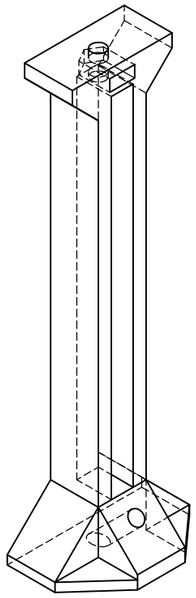

DETALLE B
ESCALA 2 :1

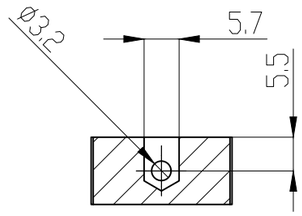


NOTA: IMPRESIÓN 3D

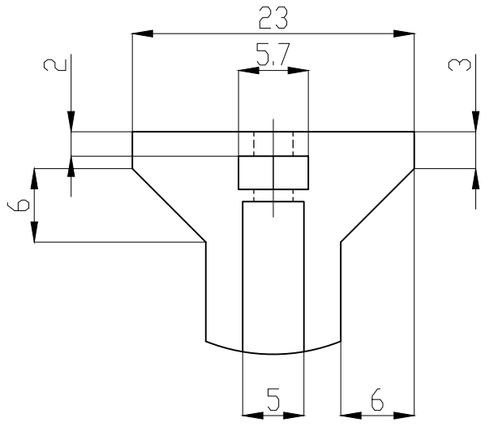
-Temperatura de Boquilla: 240°C
-Temperatura de Cama: 95°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: PETG | | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| TAPA CIRCUITO | | | D03-404 | | | | |

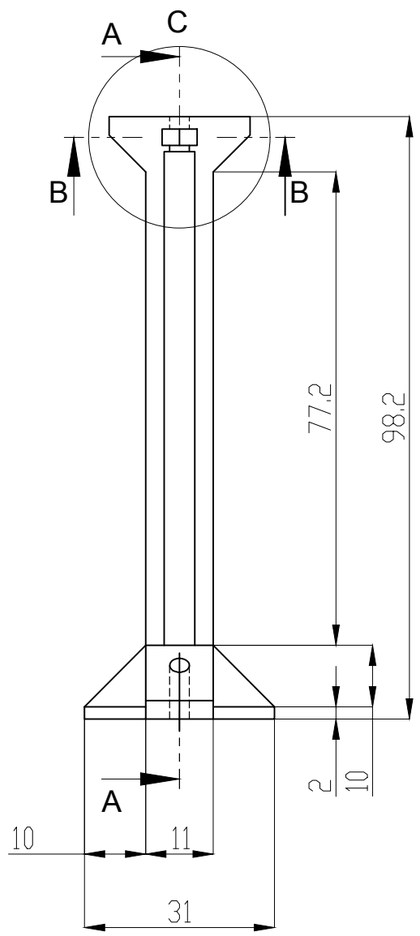


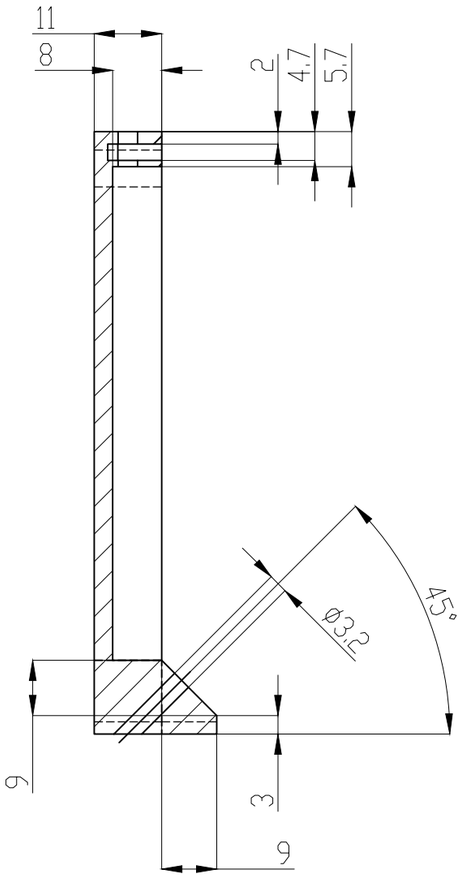



SECCIÓN B-B




DETALLE C
ESCALA 2 : 1



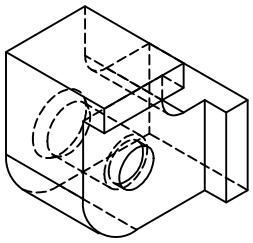



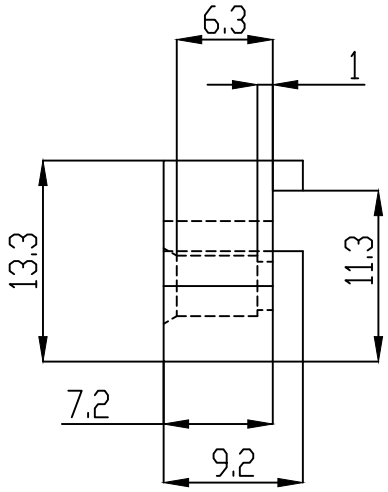

SECCIÓN A-A

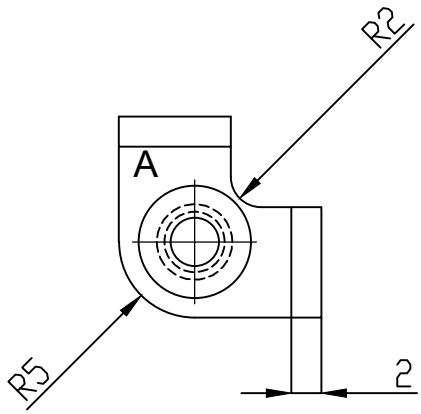
NOTA: IMPRESIÓN 3D

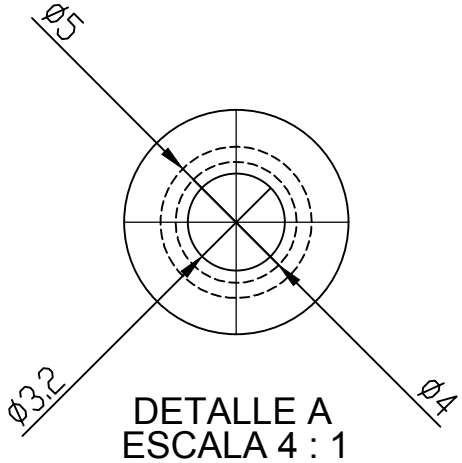
-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: | PLA | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SOPORTE FRONTAL BASE | | | D03-405 | | | | |





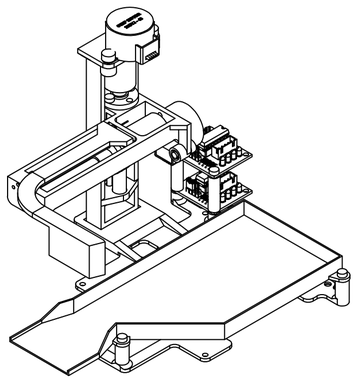



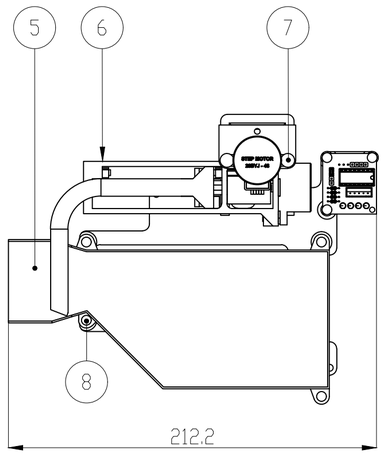

DETALLE A
ESCALA 4 : 1

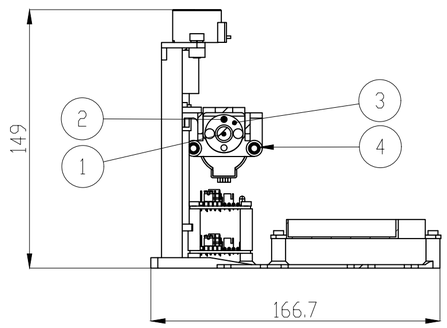
NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

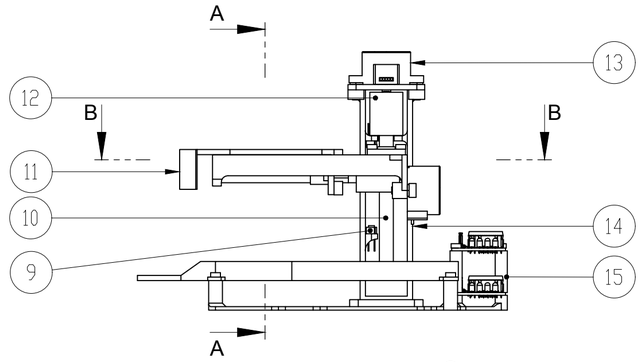
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 2:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| RETENEDOR PCB | | | D03-406 | | | | |

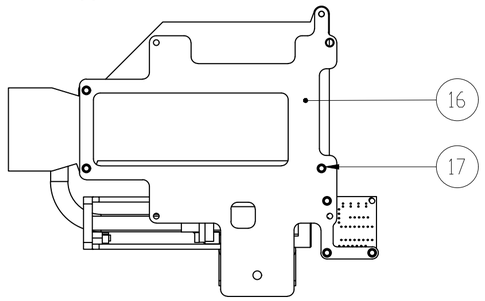


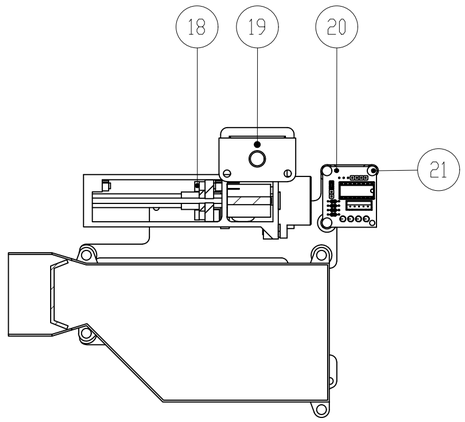



SECCIÓN A-A
ESCALA 1 : 2.5




SECCIÓN B-B
ESCALA 1 : 2.5

| MECANISMO DE AGARRE DE PRODUCTOS | |
|---|---|
| DESCRIPCIÓN | MEDIDAS |
| LARGO | 212.2 mm |
| ANCHO | 166.7mm |
| ALTURA | 149 mm |

| POS | ZONA | DENOMINACIÓN | CANT | MATERIAL | PLANO/ NORMA | OBSERVACIONES |
|---|---|---|---|---|---|---|
| 21 | E5 | PERNO M3X35 | 3 | ACERO SAE | NA | |
| 20 | E5 | DRIVER DE MOTOR A PASOS | 2 | VARIOS | NA | MODELO ULN2003 |
| 19 | E5 | EJE LISO L77mm | 1 | ACERO INOXIDABLE | NA | DIÁMETRO 3mm |
| 18 | E5 | PERNO M3X10 | 4 | ACERO SAE | NA | |
| 17 | D5 | INSERTO M3 | 11 | LATÓN | NA | |
| 16 | D5 | BASE MECANISMO DE AGARRE | 1 | PETG | D03-506 | |
| 15 | C5 | SEPARADOR DRIVERS | 3 | PLA | D03-505 | |
| 14 | C5 | PIEZA SOPORTE EJE Y | 1 | PETG | D03-504 | |
| 13 | C5 | MOTOR A PASOS | 2 | VARIOS | NA | MODELO 28YJ-48 5V |
| 12 | C5 | ACOPLAMIENTO FLEXIBLE 5-8mm | 2 | ALUMINIO | NA | |
| 11 | C4 | PALANCA DE AGARRE | 1 | PETG | D03-503 | |
| 10 | C5 | TORNILLO DE POTENCIA L95mm | 1 | ACERO INOXIDABLE | NA | 4 HILOS, AV. 8mm, D. 8mm |
| 9 | C5 | PULSADOR | 4 | VARIOS | NA | NORMALMENTE ABIERTO |
| 8 | B4 | M3X15 | 4 | ACERO SAE | NA | |
| 7 | A5 | M4X10 | 4 | ACERO SAE | NA | |
| 6 | A4 | PIEZA SOPORTE EJE X | 1 | PETG | D03-502 | |
| 5 | B4 | ESTANTE | 1 | PLA | D03-501 | |
| 4 | C3 | INSERTO M4 | 4 | LATÓN | NA | MARCA TOKAI |
| 3 | C3 | TUERCA TORNILLO DE POTENCIA | 2 | BRONCE | NA | 4 HILOS, AV. 8mm, D. 8mm |
| 2 | C3 | EJE LISO L80mm | 1 | ACERO INOXIDABLE | NA | DIÁMETRO 3mm |
| 1 | C3 | TORNILLO DE POTENCIA L98.5mm | 1 | ACERO INOXIDABLE | NA | 4 HILOS, AV. 8mm, D. 8mm |

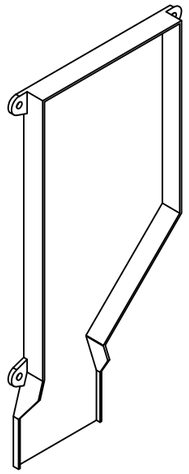
UIDE — INGENIERÍA MECATRÓNICA

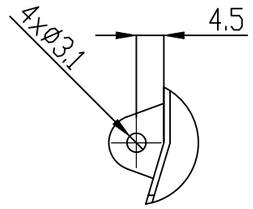
| | | |
|---|---|---|
| DIB. | VERNAZA M. | 01/04/2022 |
| DIS. | VERNAZA M. | 28/07/2022 |
| REV. | VELARDE P. | 29/07/2022 |

MECANISMO DE AGARRE DE PRODUCTOS

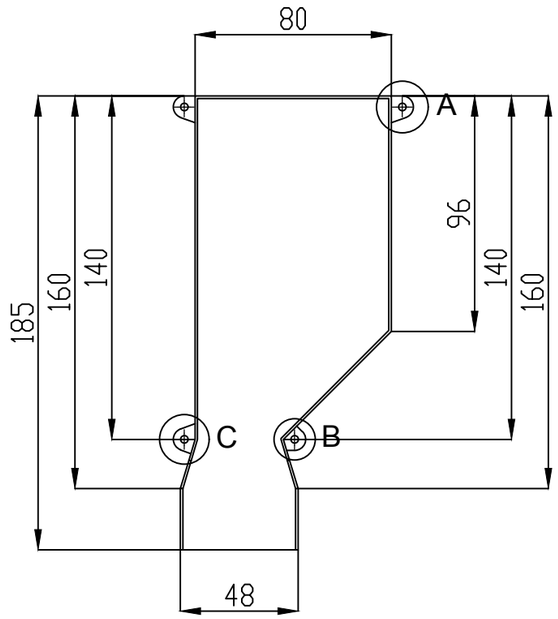
D03-005

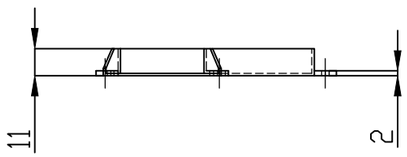
ESCALA: 1:2.5

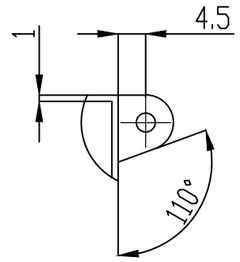




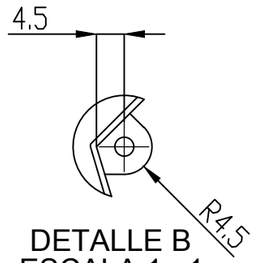

DETALLE C
ESCALA 1 : 1



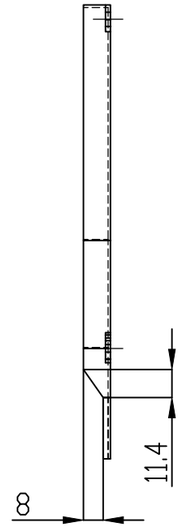




DETALLE A
ESCALA 1 : 1



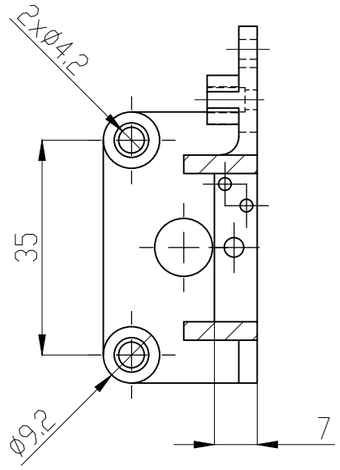

DETALLE B
ESCALA 1 : 1



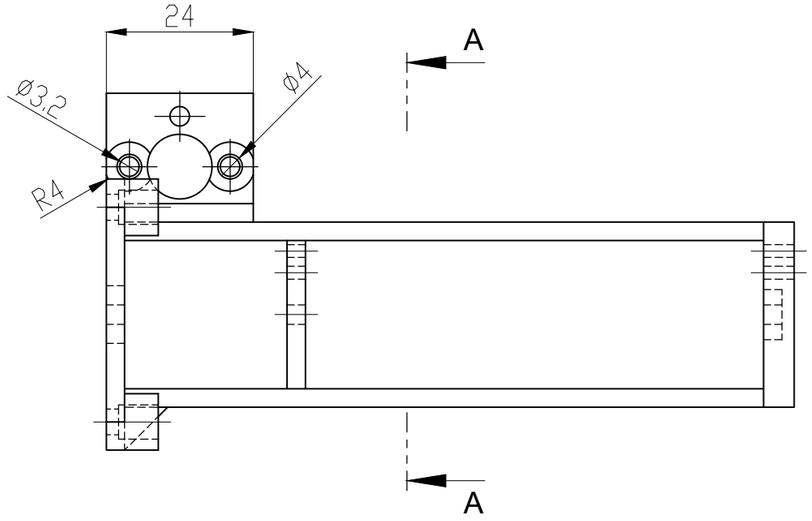

NOTA: IMPRESIÓN 3D

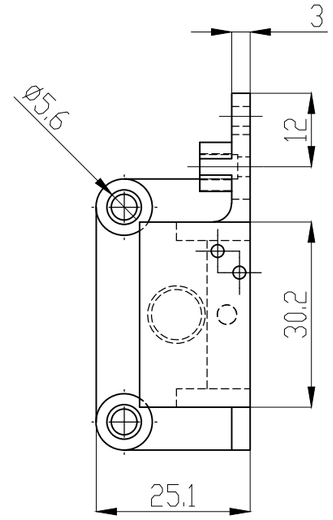
-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

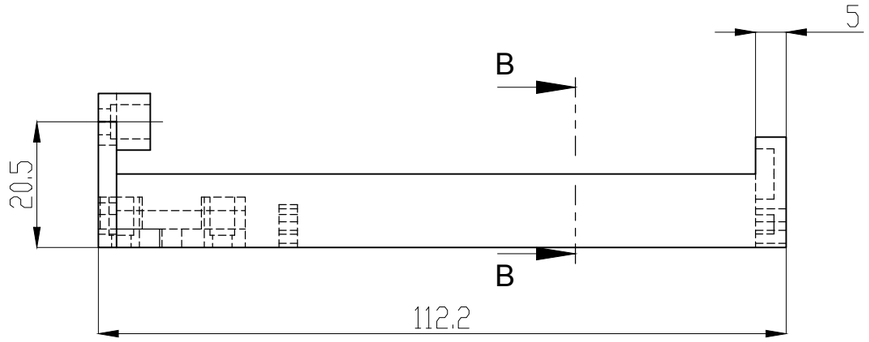
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 1:2.5 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| ESTANTE | | | D03-501 | | | | |

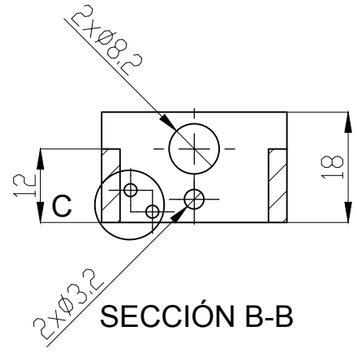



SECCIÓN A-A

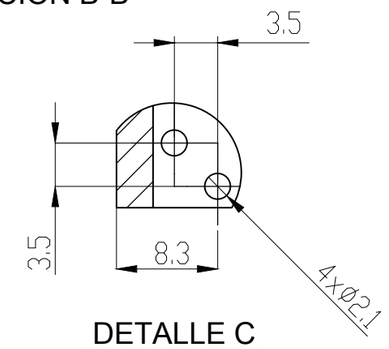


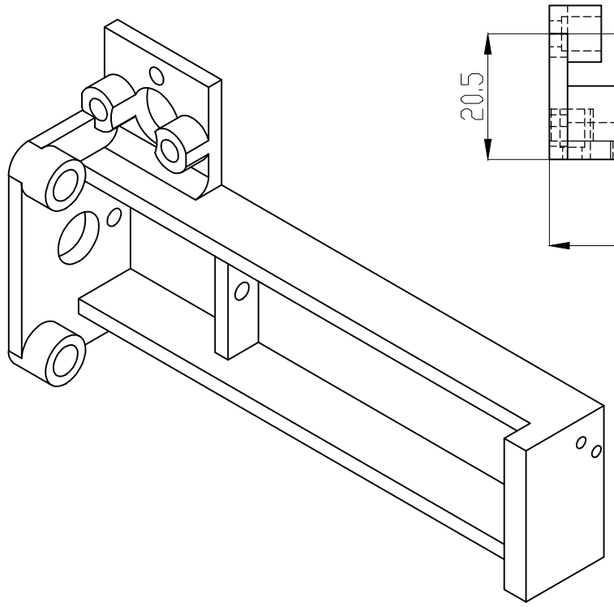




SECCIÓN B-B



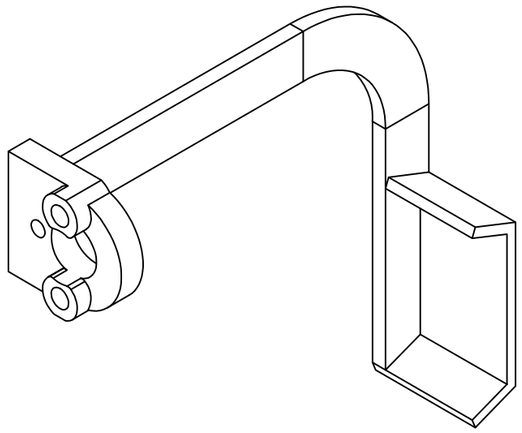

DETALLE C
ESCALA 2 : 1

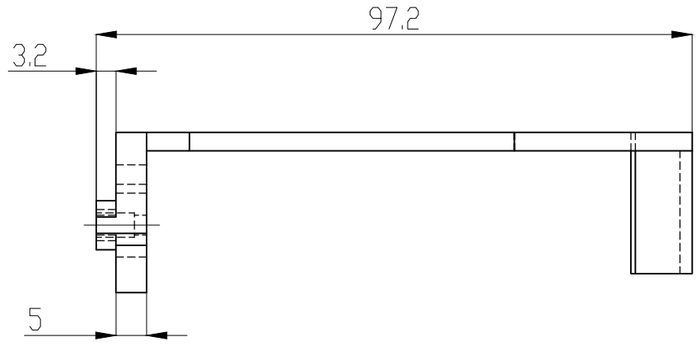


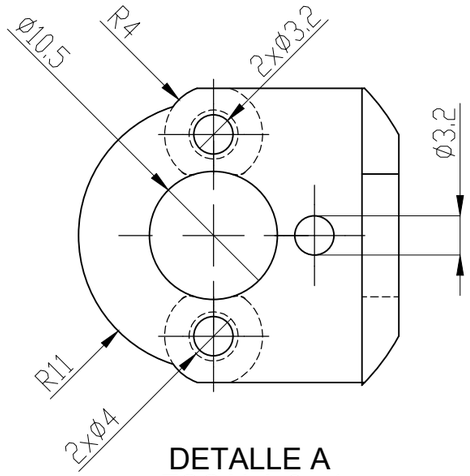
NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 240°C
-Temperatura de Cama: 95°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

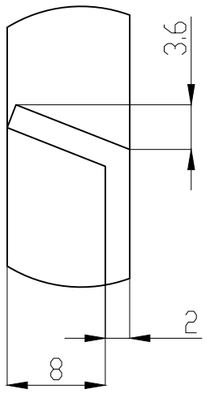
| | | | | | | |
|---|---|---|---|---|---|---|
| TRATAMIENTO: SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: PETG | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | REV. | VELARDE P. | 31/07/2022 | |
| PIEZA SOPORTE EJE X | | D03-502 | | | | |



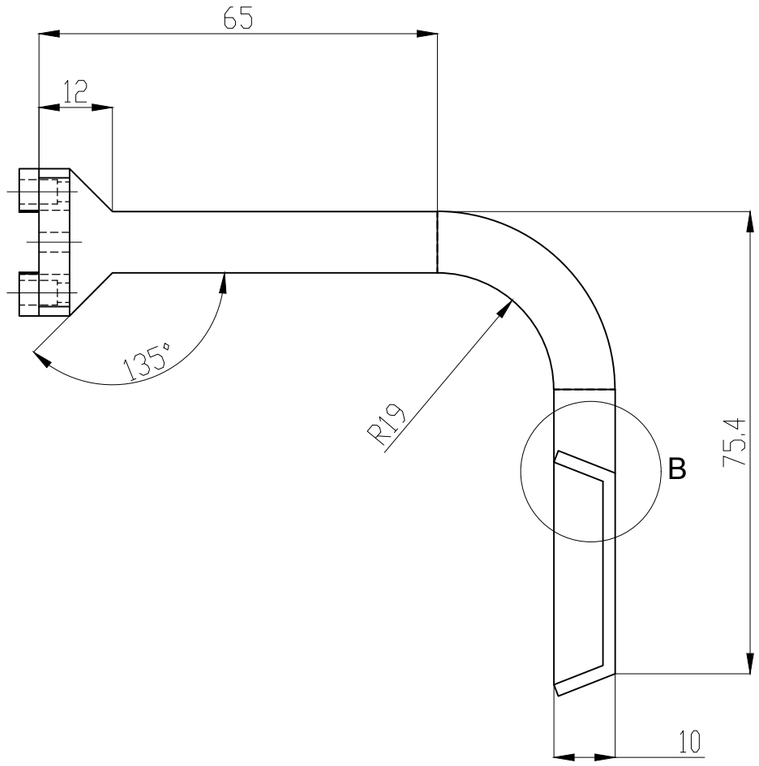


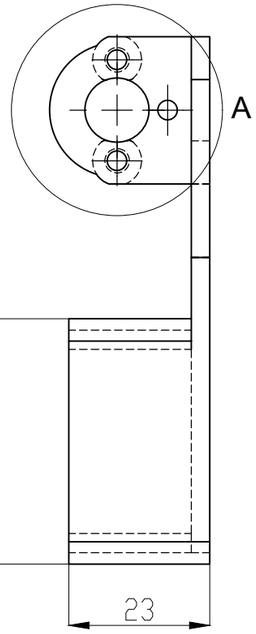




DETALLE A
ESCALA 2 : 1




DETALLE B
ESCALA 2 : 1

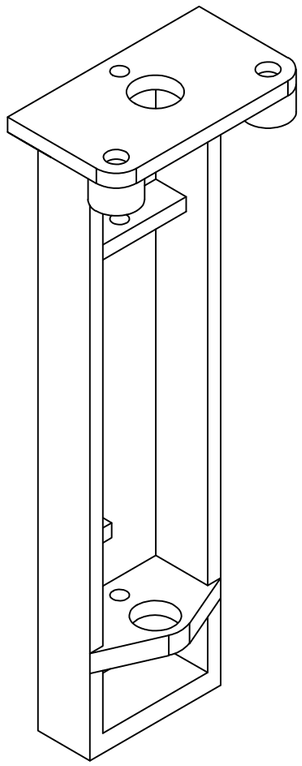


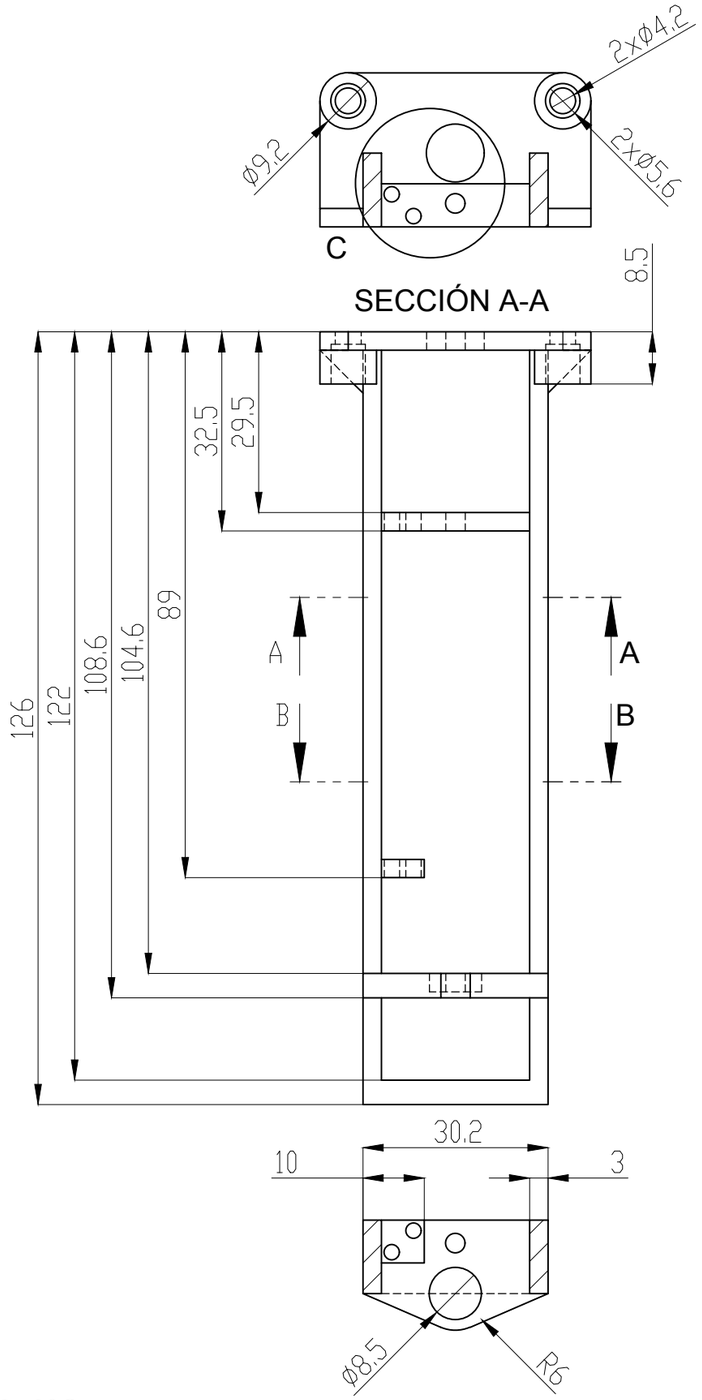




NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 240°C
-Temperatura de Cama: 95°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | | |
| RECUBRIMIENTO: SIN RECUBRIMIENTO | | | | | | | |
| MATERIAL: PETG | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | | |
| | | | DIS. | VERNAZA M. | 15/06/2022 | | |
| | | | REV. | VELARDE P. | 31/07/2022 | | |
| PALANCA DE AGARRE | | D03-503 | | | | | |

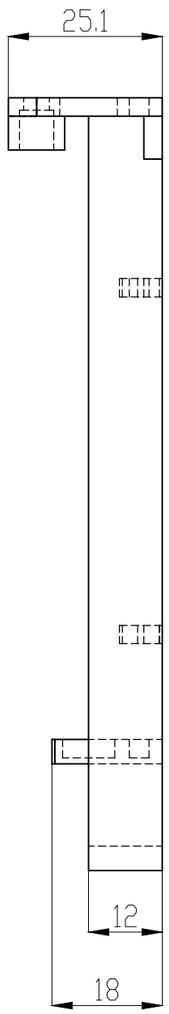


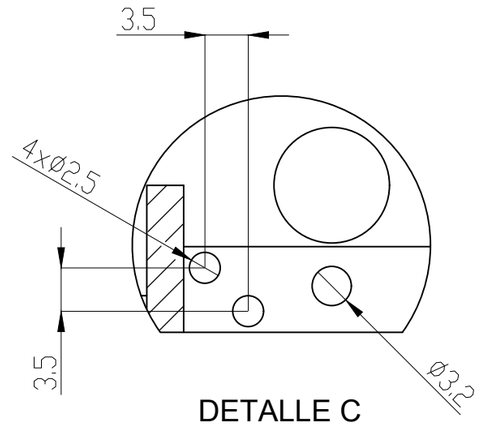






SECCIÓN B-B



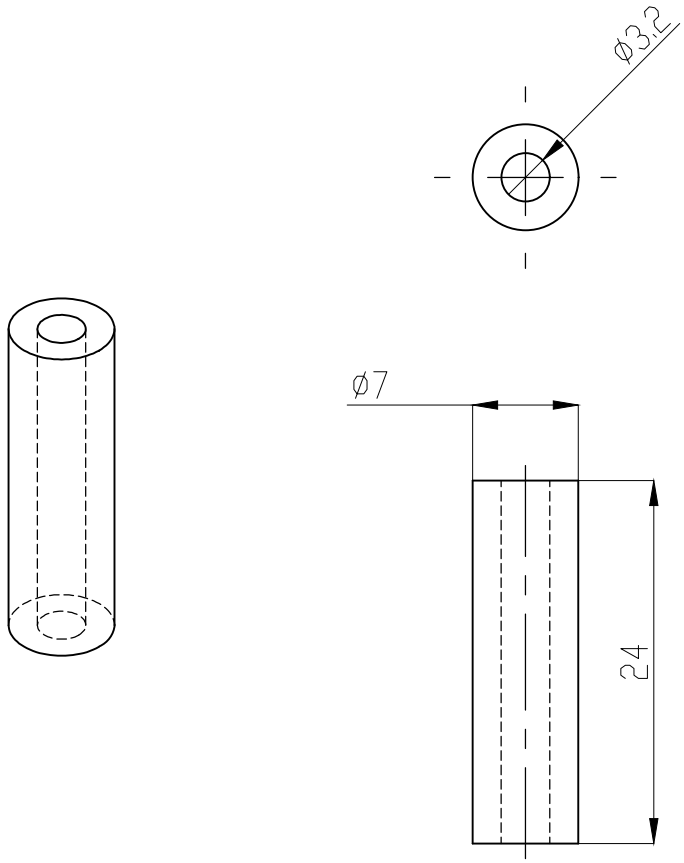




DETALLE C
ESCALA 2 : 1

NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 240°C
-Temperatura de Cama: 95°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

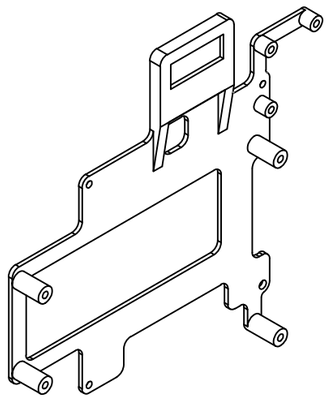
| | | | | | | |
|---|---|---|---|---|---|---|
| TRATAMIENTO: SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: PETG | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | REV. | VELARDE P. | 31/07/2022 | |
| PIEZA SOPORTE EJE Y | | D03-504 | | | | |

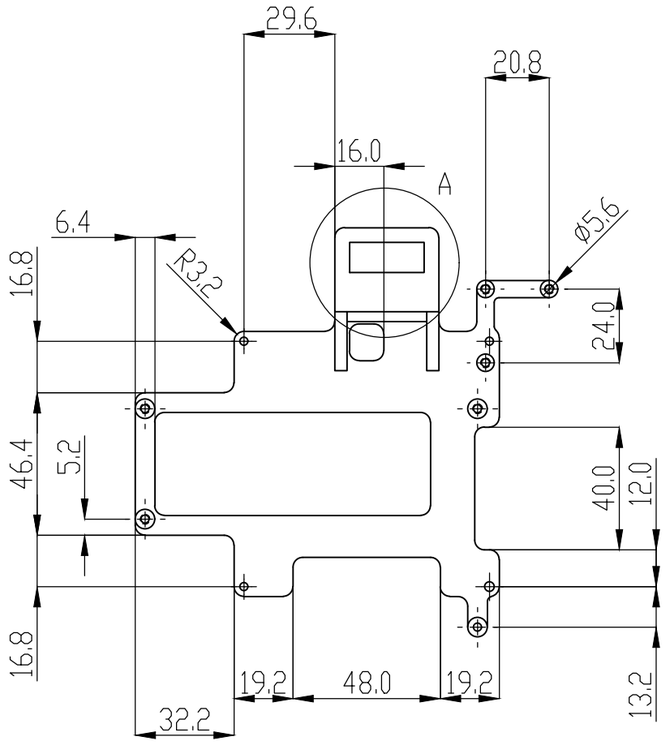


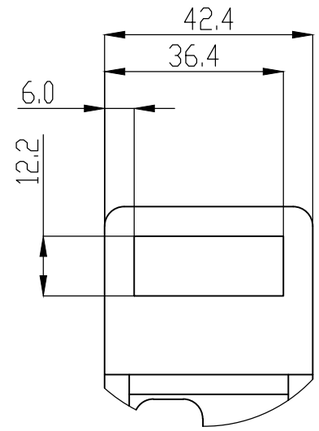

NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

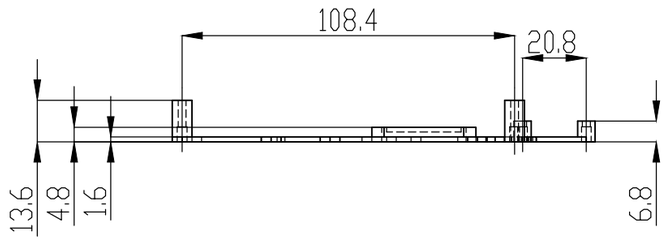
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 2:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SEPARADOR DRIVERS | | | D03-505 | | | | |





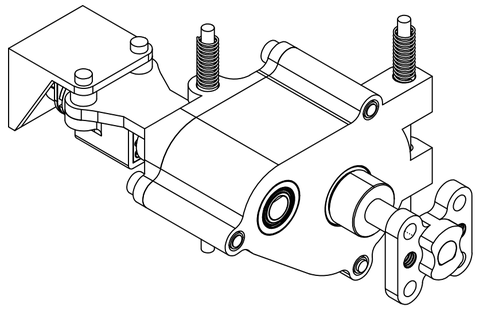

DETAIL A
SCALE 2 : 2.5

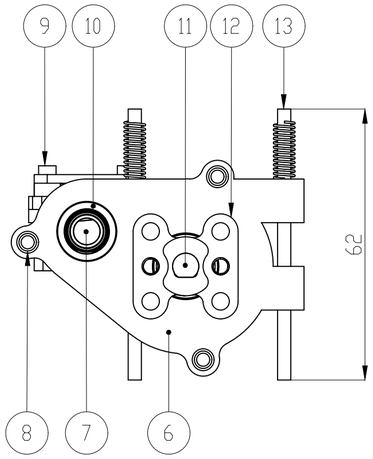


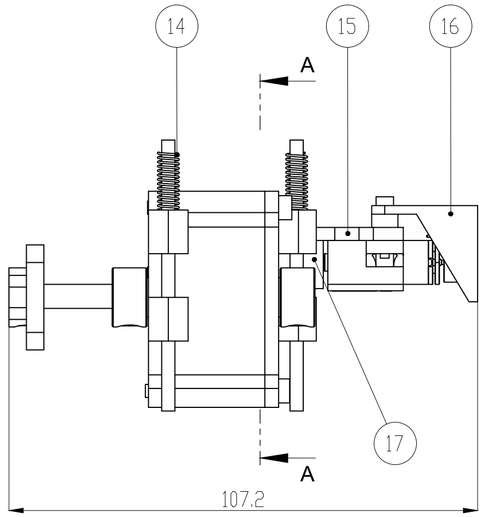

NOTA: IMPRESIÓN 3D

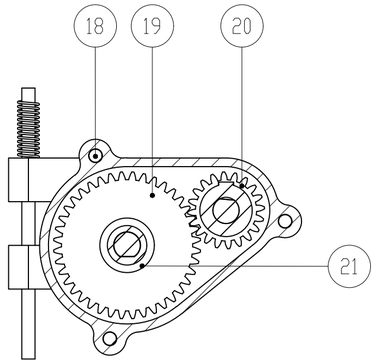
-Temperatura de Boquilla: 240°C
-Temperatura de Cama: 95°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | |
|---|---|---|---|---|---|---|
| TRATAMIENTO: SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: PETG | TOL. GRAL ± 0.1 | ESCALA 1:2.5 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | REV. | VELARDE P. | 31/07/2022 | |
| BASE MECANISMO DE AGARRE | | D03-506 | | | | |

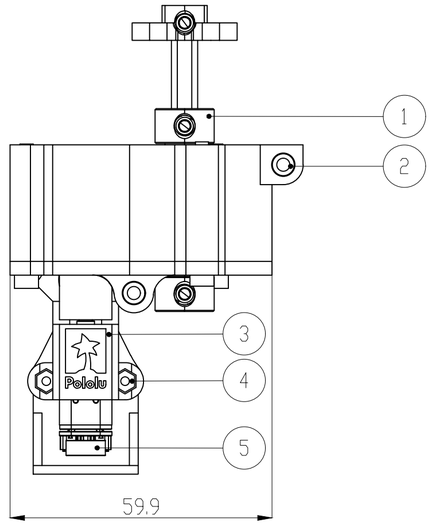





SECCIÓN A-A
ESCALA: 1:1




| CAJA REDUCTORA | |
|---|---|
| **DESCRIPCIÓN** | **MEDIDAS** |
| LARGO | 107.2 mm |
| ANCHO | 59.9 mm |
| ALTURA | 62 mm |

| POS | ZONA | DENOMINACIÓN | CANT | MATERIAL | PLANO/ NORMA | OBSERVACIONES |
|---|---|---|---|---|---|---|
| 21 | D3 | SEPARADOR | 1 | PLA | D03-606 | |
| 20 | E3 | PIÑÓN | 1 | PLA | D03-605 | |
| 19 | C4 | ENGRANE | 1 | PLA | D03-604 | |
| 18 | B4 | PERNO M3X30 | 3 | ACERO AL CARBONO | NA | |
| 17 | C4 | ACOPLAMIENTO 3-6mm | 1 | ALUMINIO | NA | |
| 16 | C4 | PROTECTOR ENCODER | 1 | PLA | D03-603 | |
| 15 | D4 | TAPA CAJA REDUCTORA | 1 | PLA | D03-602 | |
| 14 | D3 | RESORTE | 2 | ACERO INOXIDABLE | NA | 1.083 kgf/cm, L13mm |
| 13 | E3 | EJE LISO | 2 | ACERO INOXIDABLE | NA | DIÁMETRO 3mm |
| 12 | C4 | ACOPLE PARA EJE TIPO D | 1 | ALUMINIO | NA | |
| 11 | B4 | EJE L70mm | 1 | ACERO INOXIDABLE | NA | DIÁMETRO 6mm, SECCIÓN D |
| 10 | C4 | RODAMIENTO 6mm | 3 | ACERO AL CARBONO | NA | DE BOLAS CON BRIDA |
| 9 | C4 | PERNO M2 | 2 | ACERO SAE | NA | |
| 8 | D4 | INSERTO 3mm | 3 | LATÓN | NA | |
| 7 | D3 | EJE L30mm | 1 | ACERO INOXIDABLE | NA | DIÁMETRO 6mm, SECCIÓN D |
| 6 | E3 | BASE CAJA REDUCTORA | 1 | PLA | D003-601 | |
| 5 | C4 | MOTOR N20 | 1 | VARIOS | NA | 12V, 90RPM, RED. 298:1 |
| 4 | B4 | TUERCA M2 | 2 | ACERO INOXIDABLE | NA | |
| 3 | C4 | SOPORTE MOTOR N20 | 1 | ABS | NA | |
| 2 | C4 | RODAMIENTO DE MANGA 3mm | 4 | VARIOS | NA | |
| 1 | D4 | COLLARÍN 6mm | 2 | ACERO INOXIDABLE | NA | |

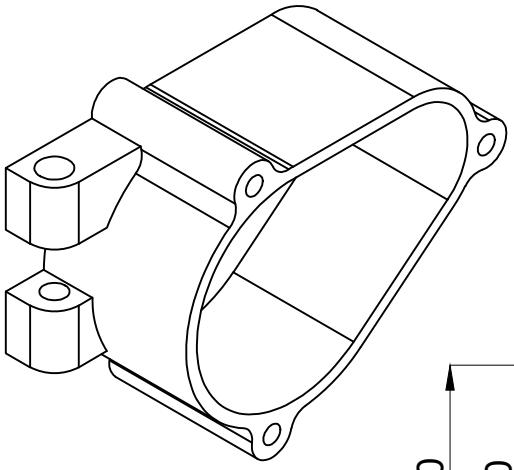
UIDE — INGENIERÍA MECATRÓNICA

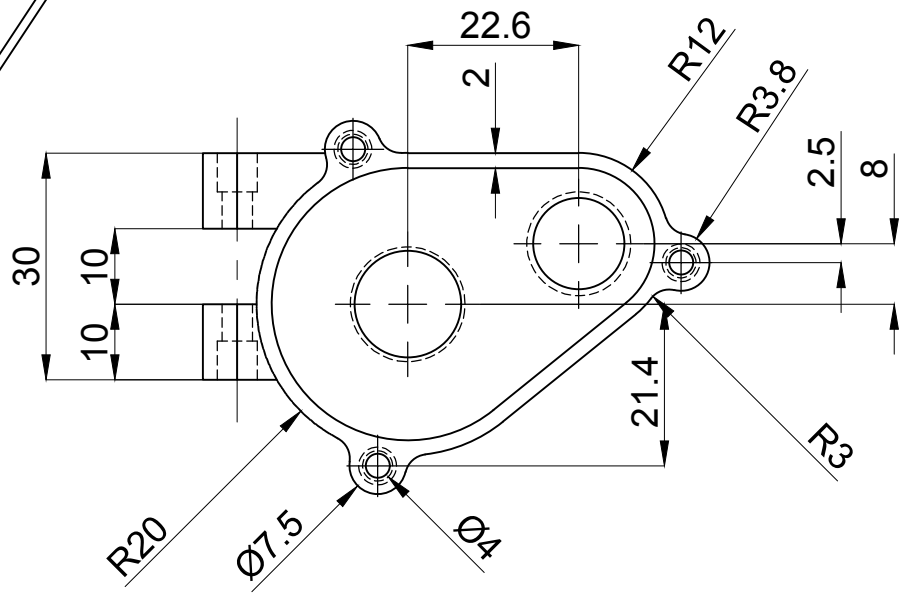
| | | |
|---|---|---|
| DIB. | VERNAZA M. | 01/04/2022 |
| DIS. | VERNAZA M. | 28/07/2022 |
| REV. | VELARDE P. | 29/07/2022 |

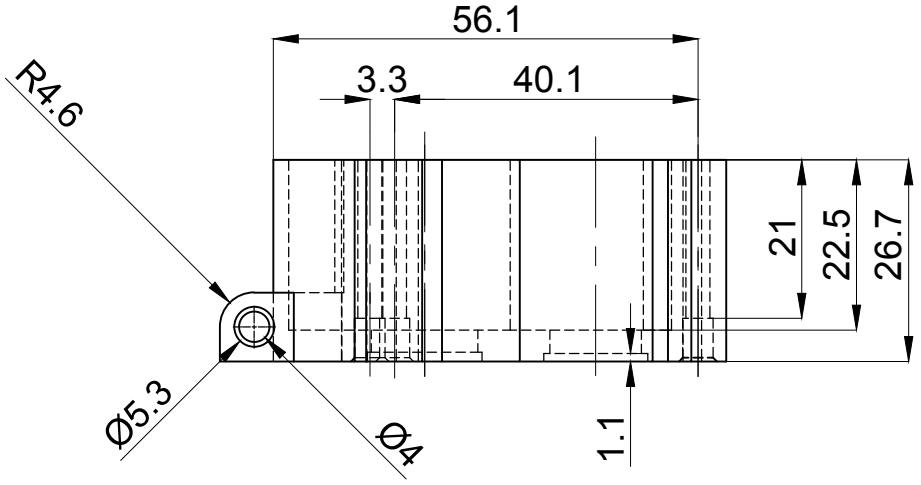
CAJA REDUCTORA

D03-006

ESCALA: 1:1



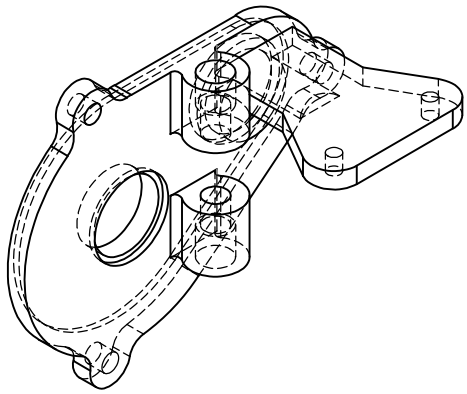


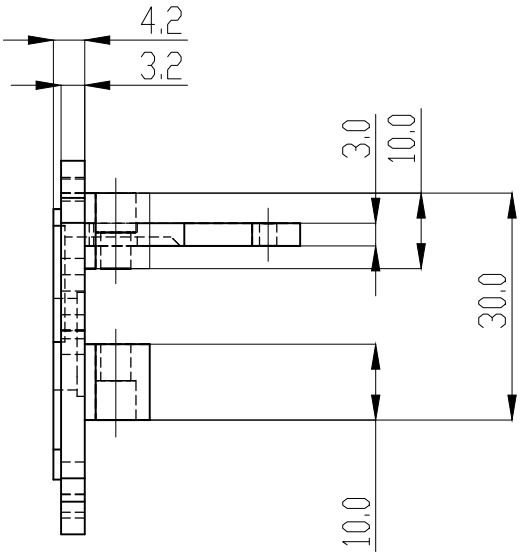



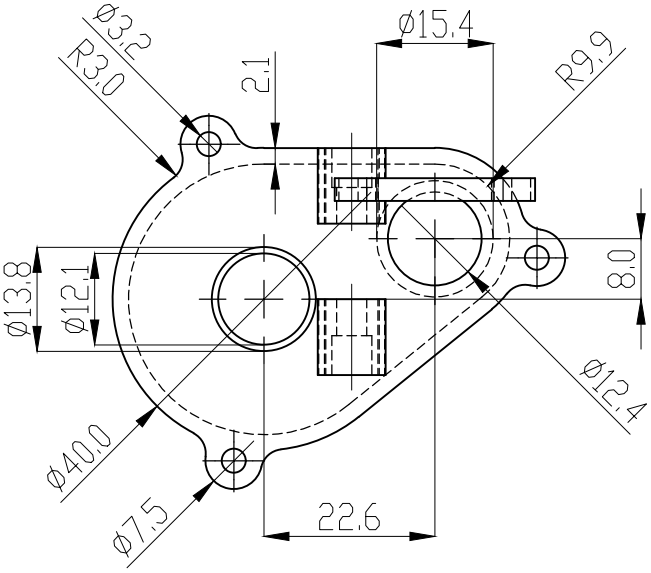

NOTA: IMPRESIÓN 3D

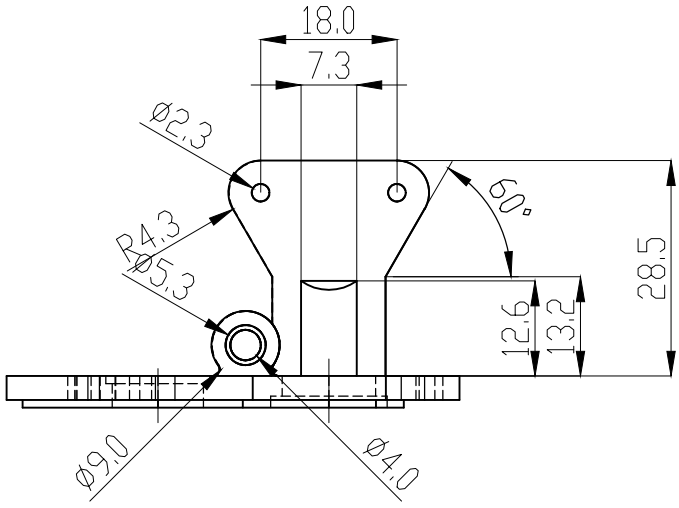
-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| BASE CAJA REDUCTORA | | | D03-601 | | | | |





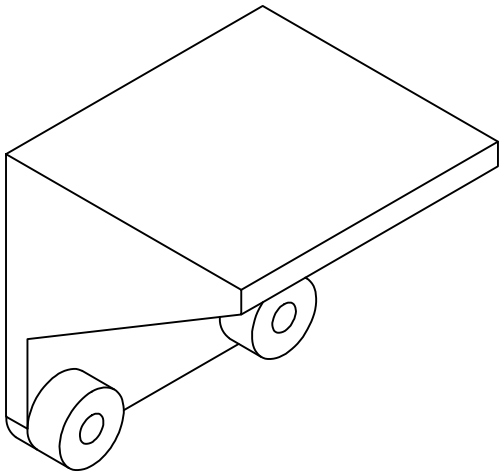



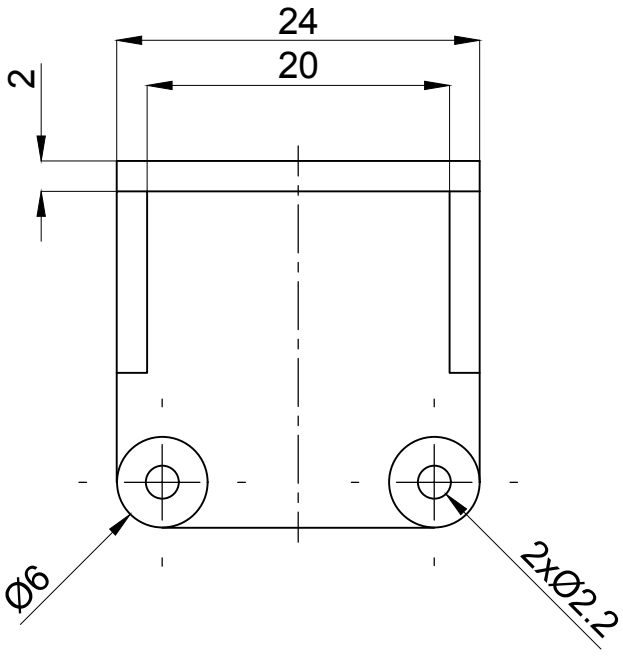



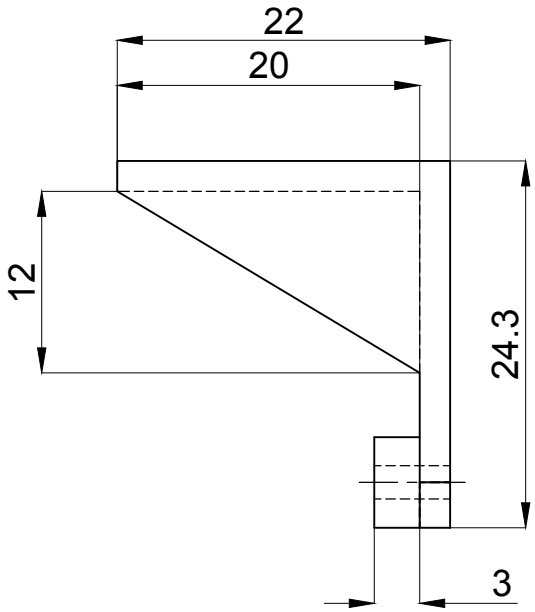

NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| TAPA CAJA REDUCTORA | | | D03-602 | | | | |



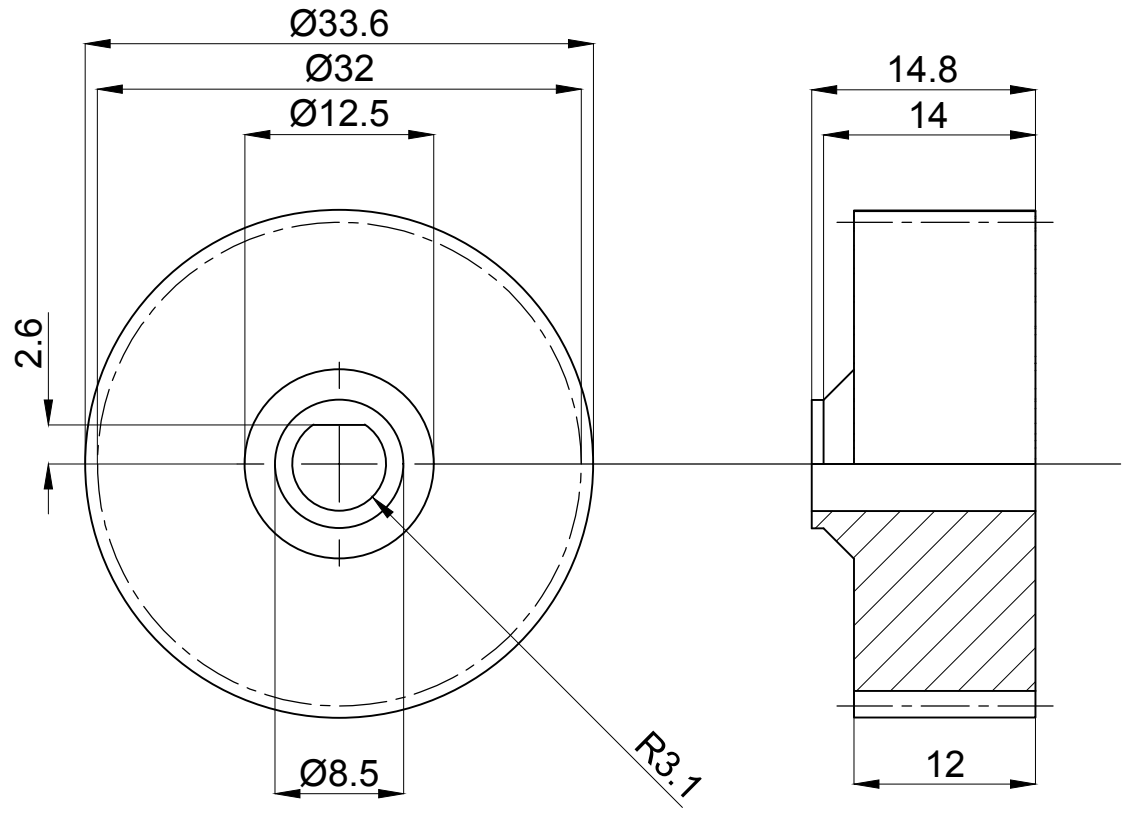


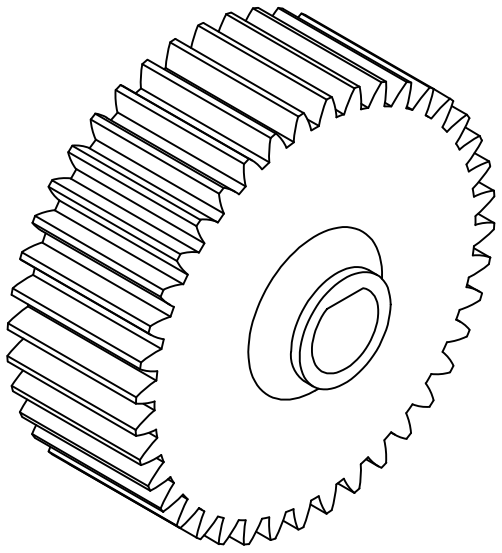




NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 2:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| PROTECTOR ENCODER | | | D03-603 | | | | |



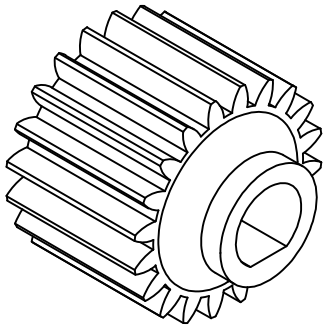



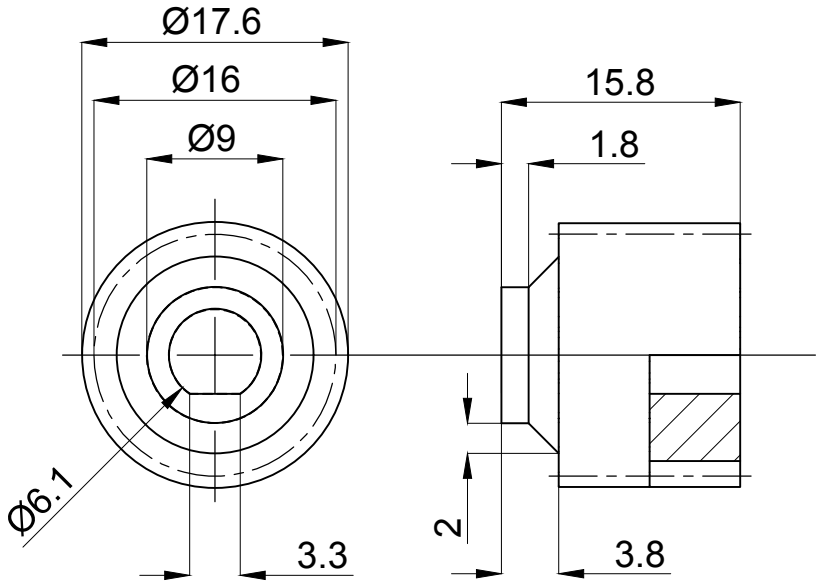
NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

NOTA:
Número de dientes: 40 dientes
Módulo: 0.8

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 2:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| ENGRANAJE | | | D03-604 | | | | |



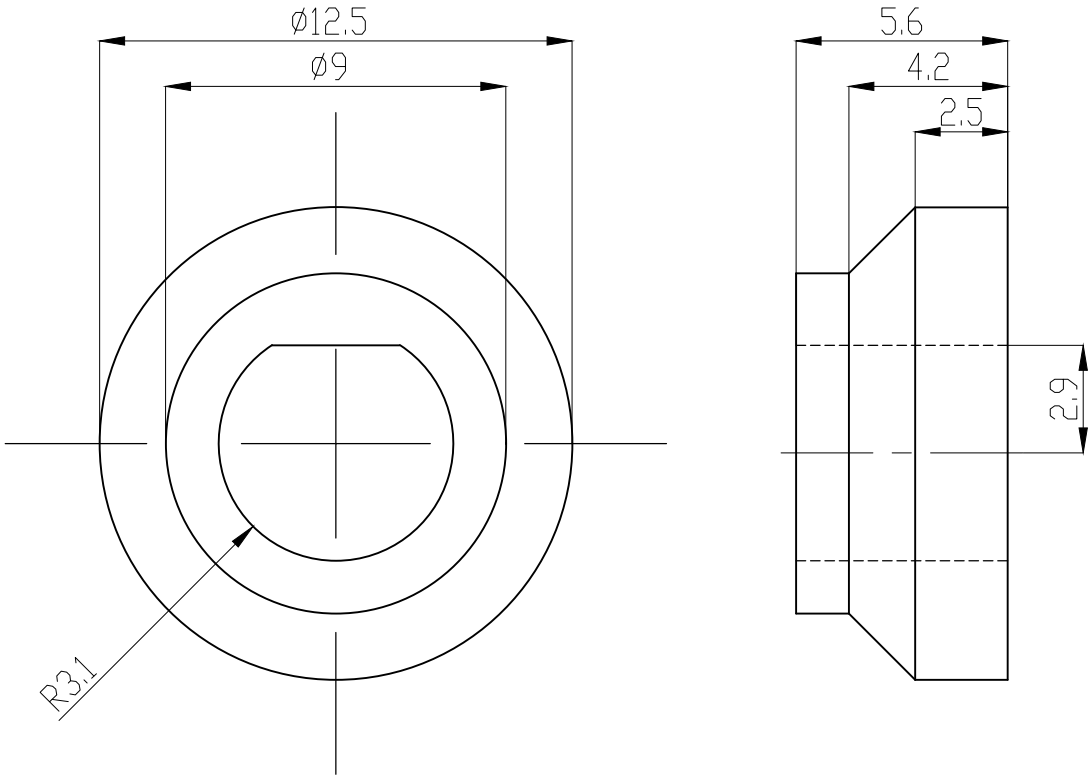


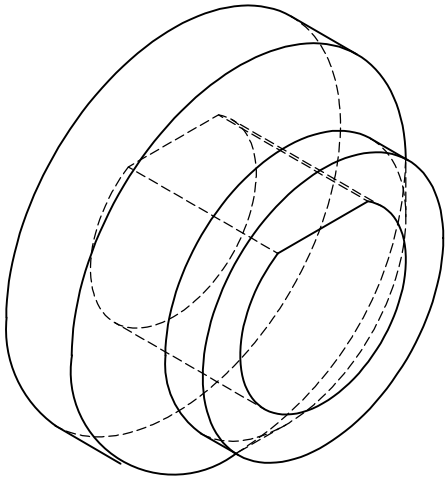

NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

NOTA:
Número de dientes: 20 dientes
Módulo: 0.8

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 2:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| PIÑÓN | | | D03-605 | | | | |

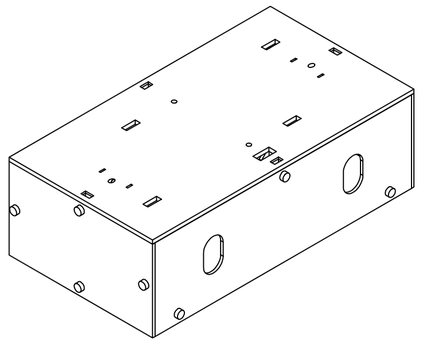


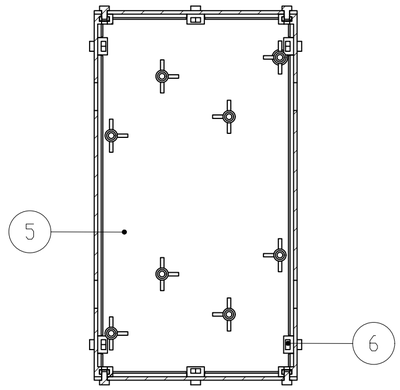



NOTA: IMPRESIÓN 3D

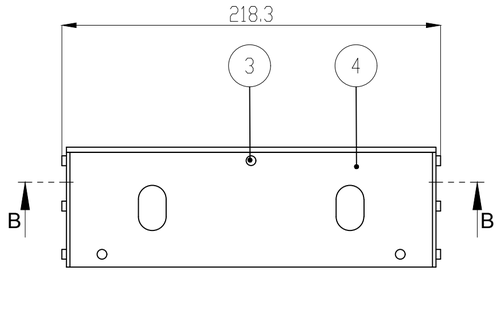
-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

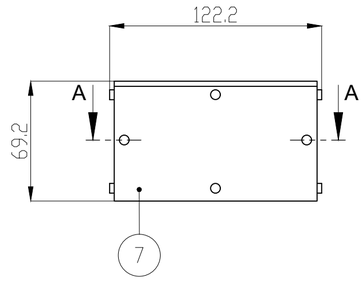
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 5:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SEPARADOR ENGRANAJE | | | D03-606 | | | | |

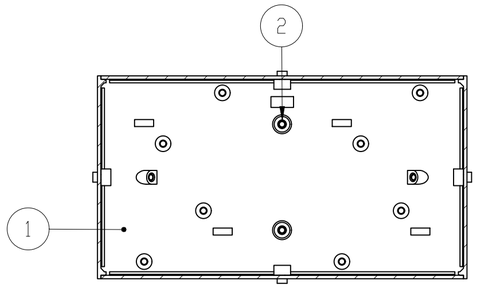





SECCIÓN A-A
ESCALA 1 : 2.5




SECCIÓN B-B
ESCALA 1 : 2.5

| CAJA DE MOTORES | |
|---|---|
| **DESCRIPCIÓN** | **MEDIDAS** |
| LARGO | 218.3 mm |
| ANCHO | 122.2 mm |
| ALTURA | 69.2 mm |

| POS | ZONA | DENOMINACIÓN | CANT | MATERIAL | PLANO/ NORMA | OBSERVACIONES |
|---|---|---|---|---|---|---|
| 7 | C6 | PIEZA FRONTAL CAJA DE MOTORES | 2 | PLA | D03-704 | |
| 6 | B7 | TUERCA M3 | 14 | ACERO INOXIDABLE | NA | |
| 5 | B6 | BASE CAJA DE MOTORES | 1 | PLA | D03-703 | |
| 4 | C4 | PIEZA LATERAL CAJA DE MOTORES | 2 | PLA | D03-702 | |
| 3 | C4 | PERNO M3X5 | 14 | ACERO SAE | NA | |
| 2 | D4 | INSERTO 3mm | 4 | LATÓN | NA | MARCA TOKAI |
| 1 | D4 | TAPA CAJA DE MOTORES | 1 | PETG | D03-701 | |

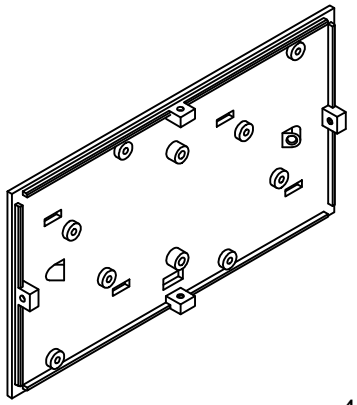
UIDE — INGENIERÍA MECATRÓNICA

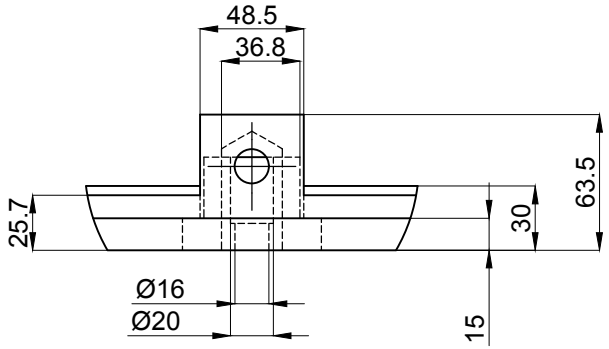
| | | |
|---|---|---|
| DIB. | VERNAZA M. | 01/04/2022 |
| DIS. | VERNAZA M. | 28/07/2022 |
| REV. | VELARDE P. | 29/07/2022 |

CAJA DE MOTORES

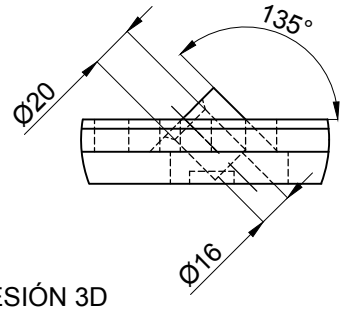
D03-007

ESCALA: 1:2.5

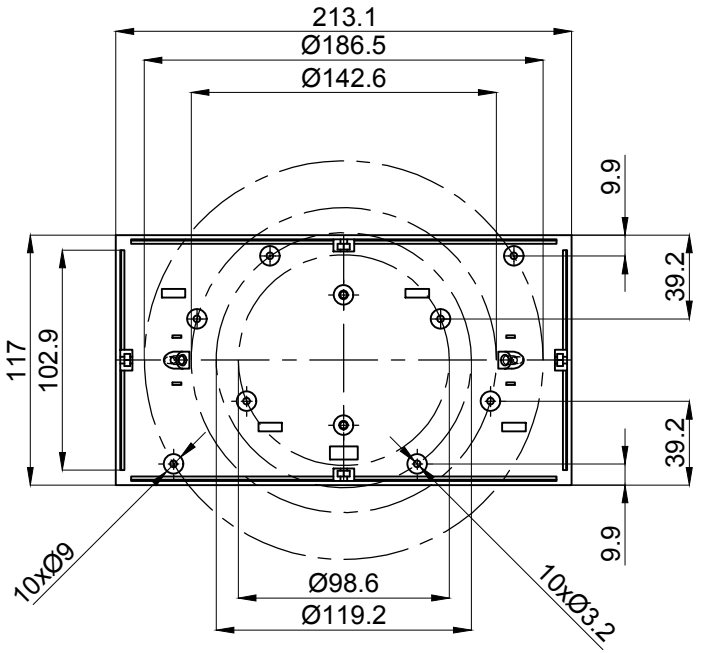


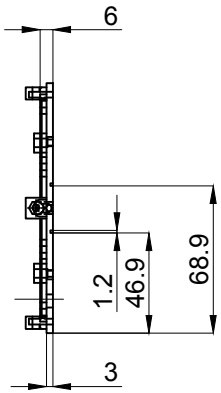


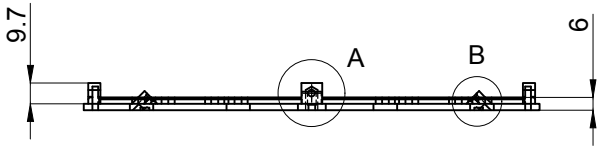

DETALLE A
ESCALA 2 : 1




DETALLE B
ESCALA 2 : 1



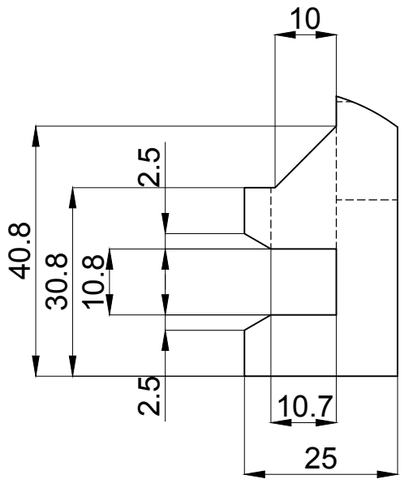



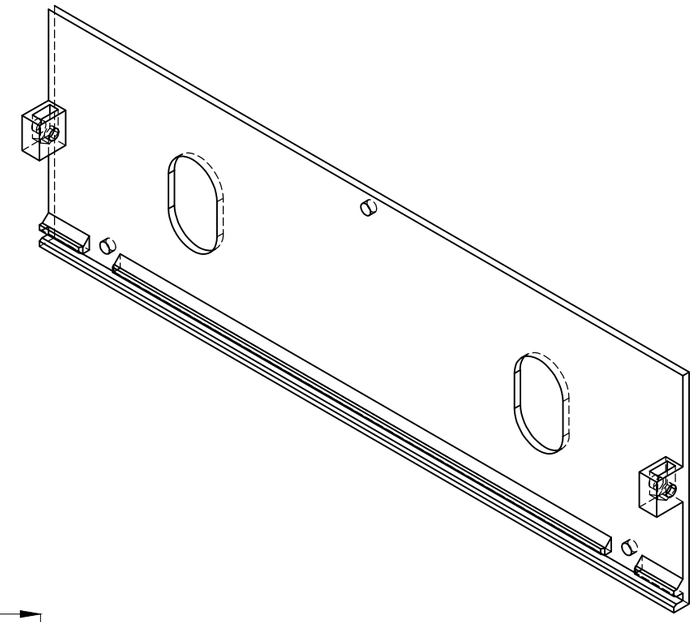

NOTA: IMPRESIÓN 3D

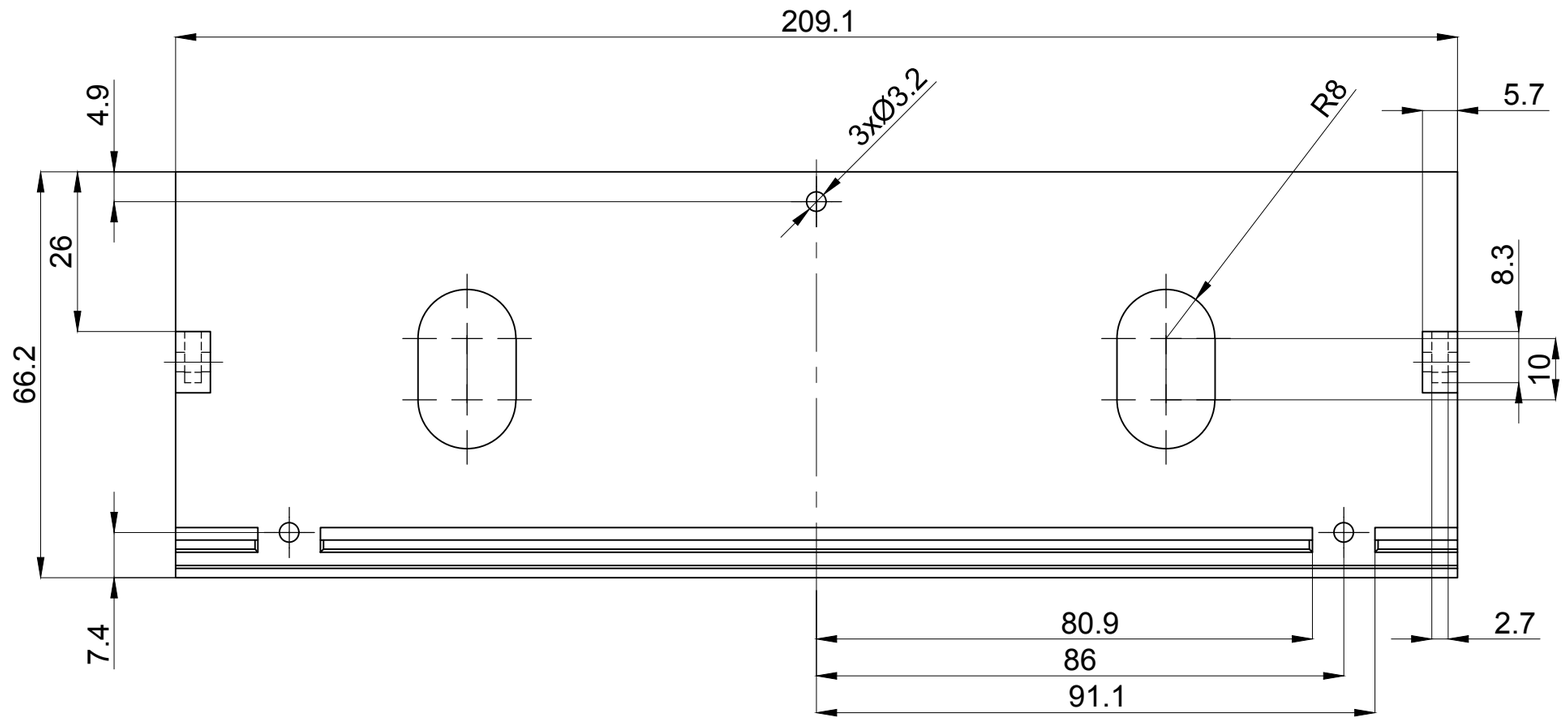
-Temperatura de Boquilla: 240°C
-Temperatura de Cama: 95°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

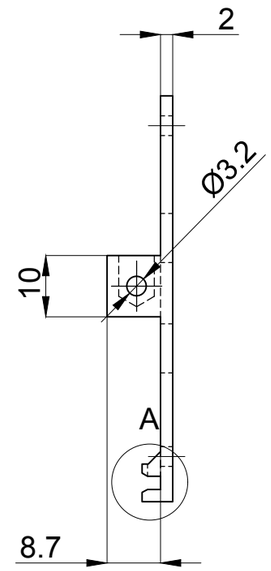
| | | | | | | |
|---|---|---|---|---|---|---|
| TRATAMIENTO: SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: PETG | TOL. GRAL ± 0.1 | ESCALA 1:2.5 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | REV. | VELARDE P. | 31/07/2022 | |
| TAPA CAJA DE MOTORES | | D03-701 | | | | |




DETALLE A
ESCALA 5 : 1



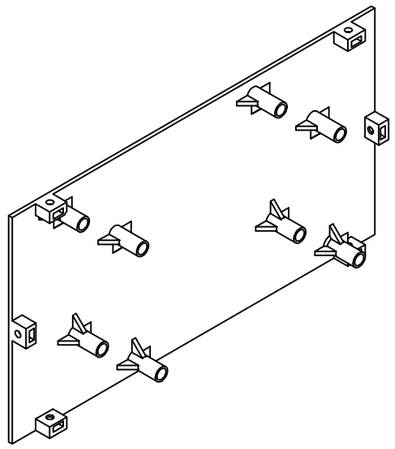


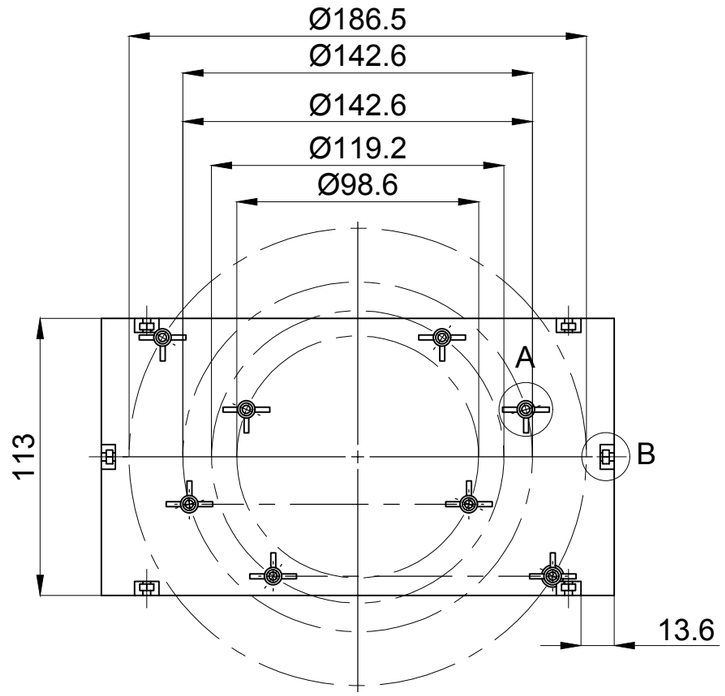



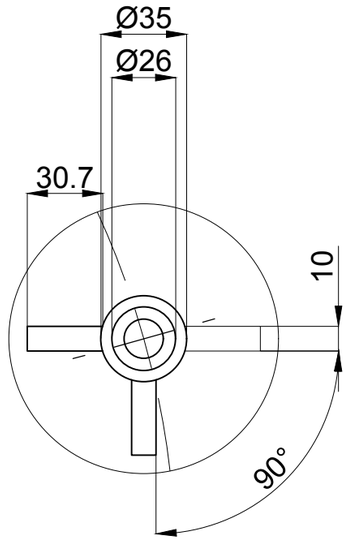

NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

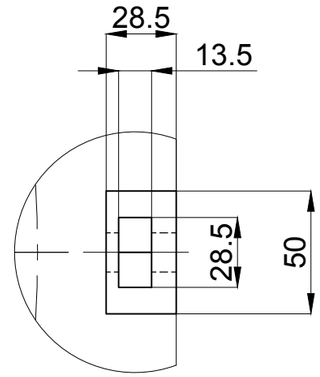
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| PARED LATERAL BASE | | | D03-702 | | | | |



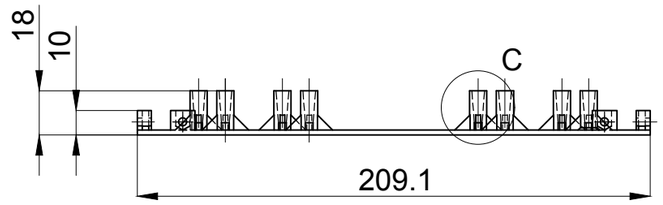


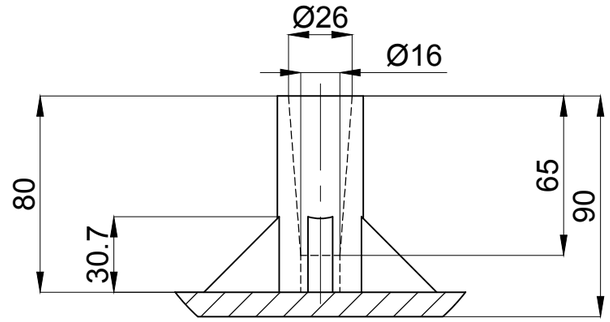




DETALLE A
ESCALA 2 : 1




DETALLE B
ESCALA 2 : 1



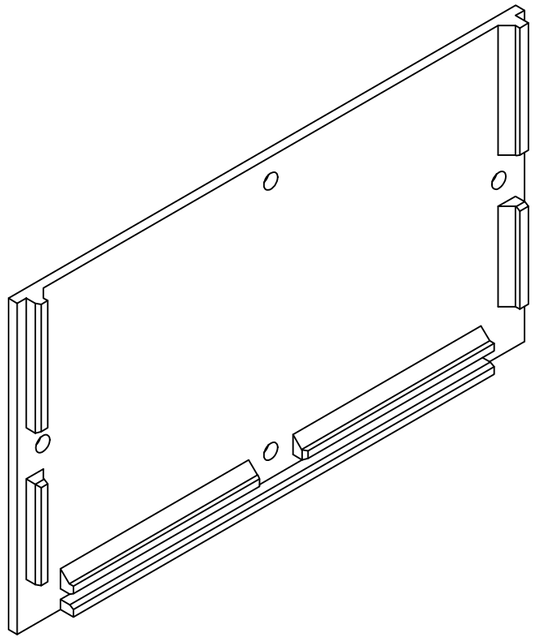



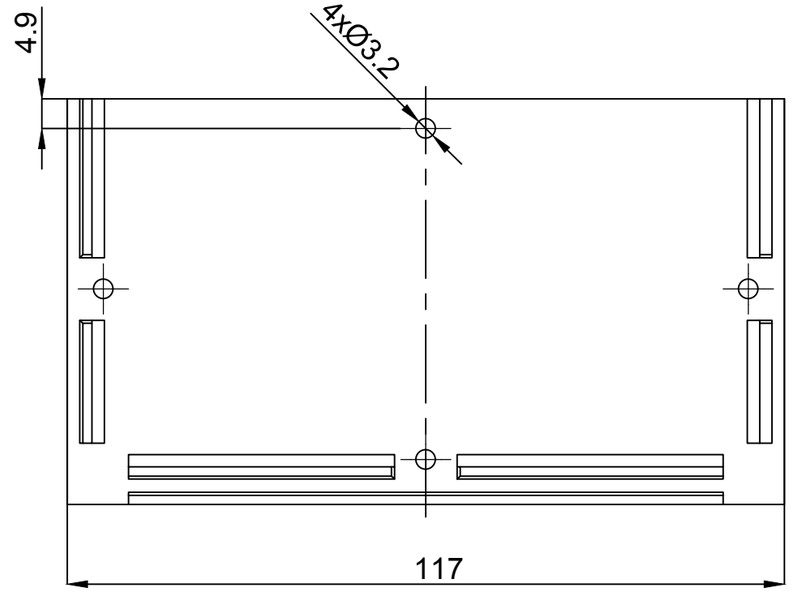

DETALLE C
ESCALA 2 : 1

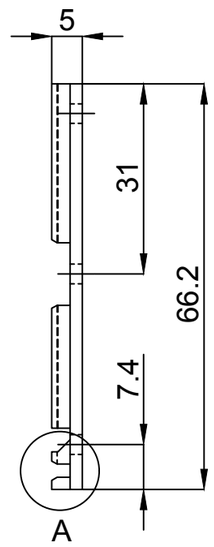
NOTA: IMPRESIÓN 3D

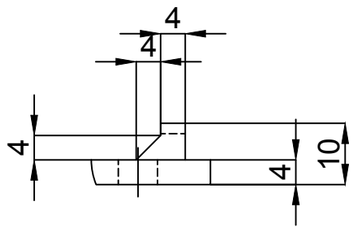
-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: | PLA | TOL. GRAL ± 0.1 | ESCALA 1:2.5 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| BASE CAJA DE MOTORES | | | D03-703 | | | | |



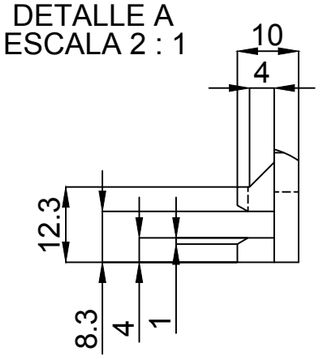


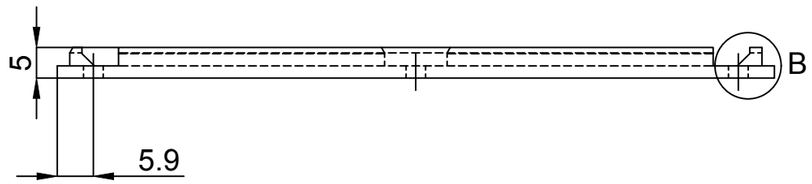




DETALLE B
ESCALA 2 : 1

DETALLE A
ESCALA 2 : 1

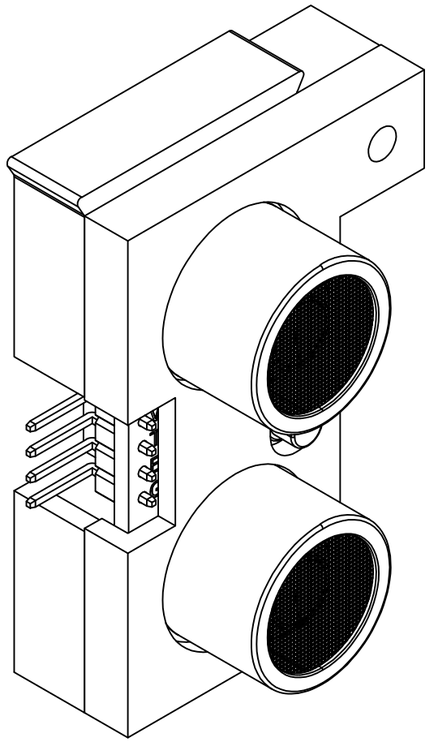


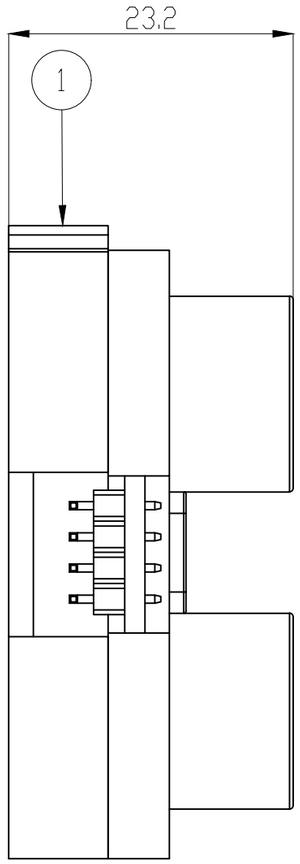



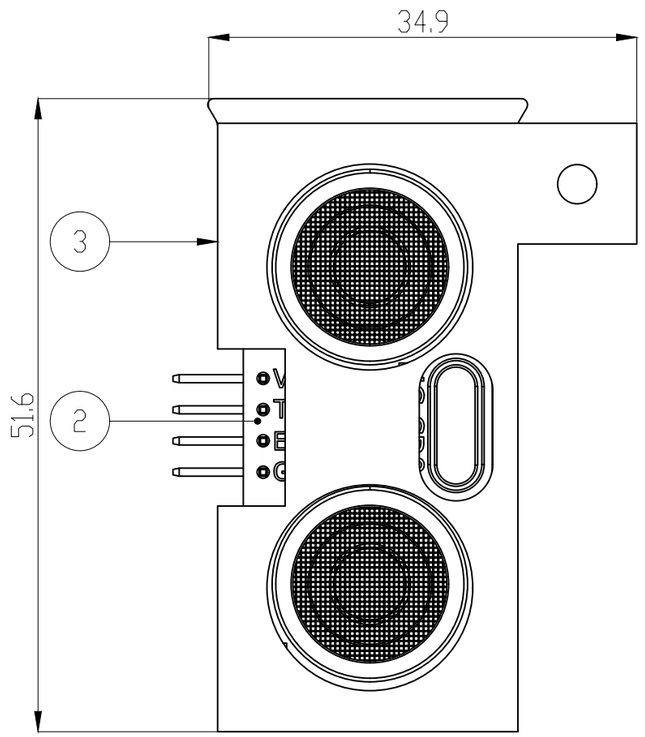

NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN RECUBRIMIENTO | | | | | | |
| MATERIAL: | PLA | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/06/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| PARED FRONTAL BASE | | | D03-704 | | | | |






| SOPORTE SENSOR | |
|---|---|
| **DESCRIPCIÓN** | **MEDIDAS** |
| LARGO | 23.2 mm |
| ANCHO | 34.9 mm |
| ALTURA | 51.6 mm |

| POS | ZONA | DENOMINACIÓN | CANT | MATERIAL | PLANO/ NORMA | OBSERVACIÓN |
|---|---|---|---|---|---|---|
| 3 | C7 | SOPORTE ULTRASÓNICO TAPA | 1 | PLA | D03-802 | |
| 2 | C7 | SENSOR ULTRASÓNICO | 1 | VARIOS | NA | TIPO HC-SR04 |
| 1 | D4 | SOPORTE ULTRASÓNICO BASE | 1 | PLA | D03-801 | |

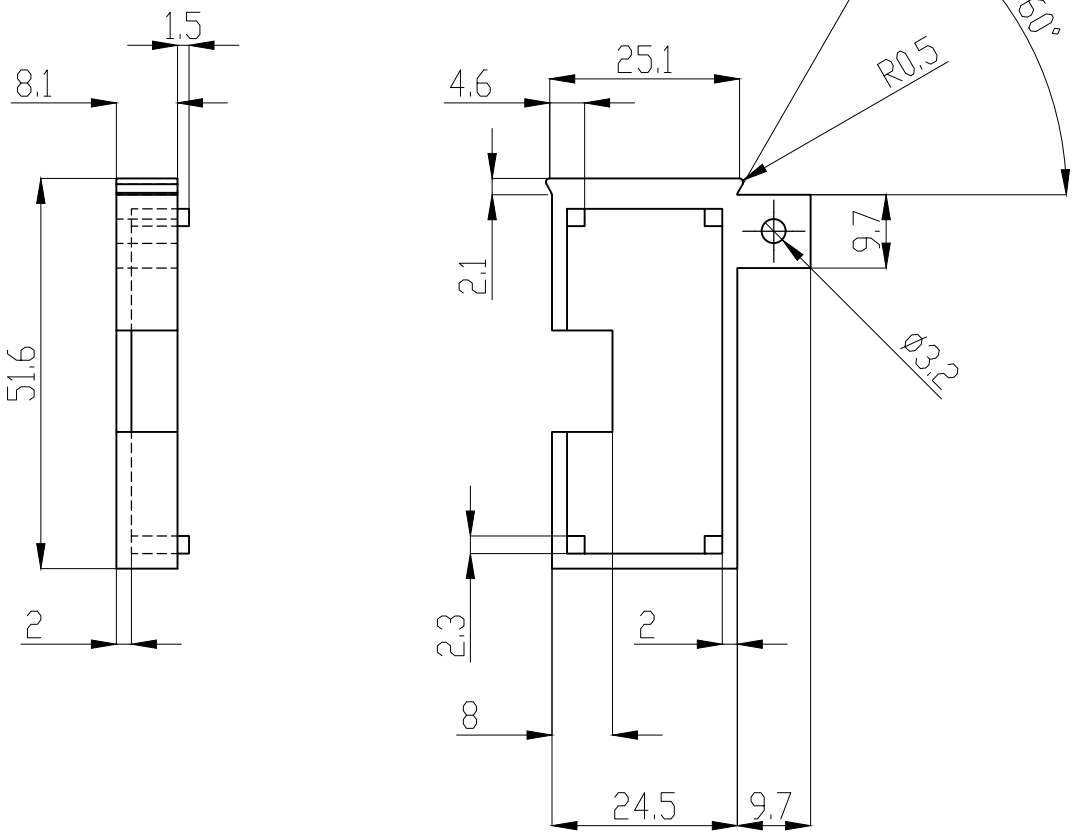
UIDE — INGENIERÍA MECATRÓNICA

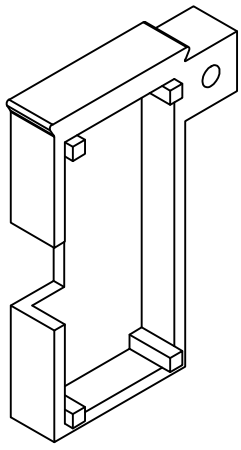
| | | |
|---|---|---|
| DIS | VERNAZA M. | 16/07/2022 |
| DIB | VERNAZA M. | 29/07/2022 |
| REV | VELARDE P. | 29/07/2022 |

SOPORTE SENSOR

D03-008

ESCALA 2:1

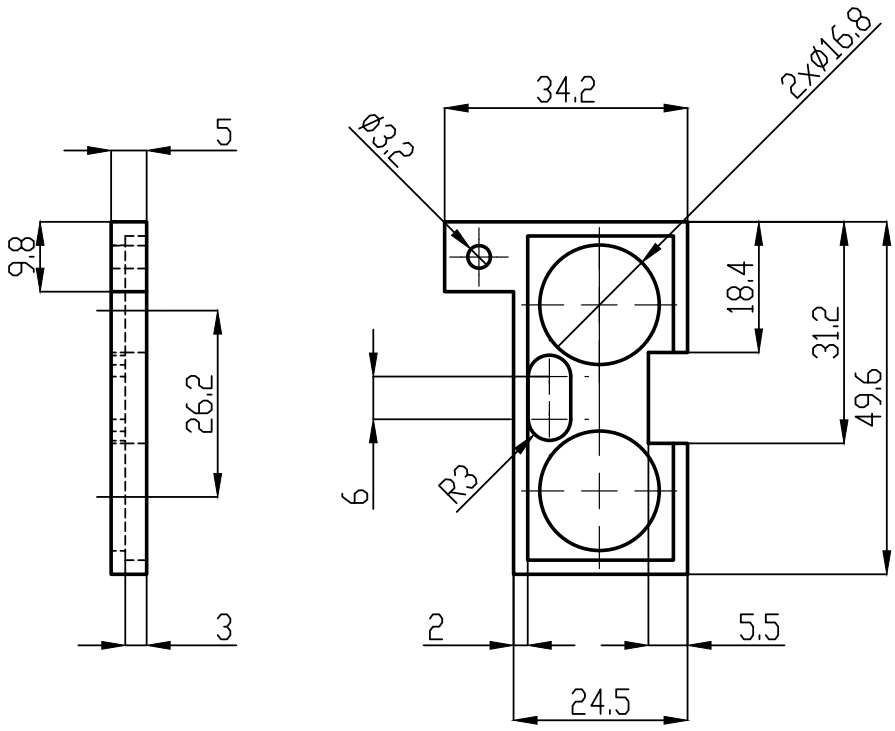


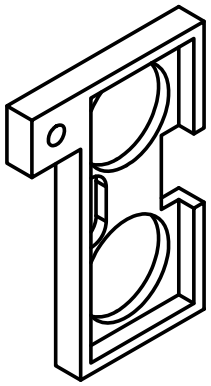



NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SOPORTE ULTRASÓNICO BASE | | | D03-801 | | | | |



NOTA: IMPRESIÓN 3D

-Temperatura de Boquilla: 215°C
-Temperatura de Cama: 63°C
-Relleno: 10 %
-Tipo: Gyroid
-Boquilla: 0.4mm
-Alto de Capa: 0.2mm

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRATAMIENTO: | SIN TRATAMIENTO | | UIDE | INGENIERIA MECATRÓNICA | | | |
| RECUBRIMIENTO: | SIN TRATAMIENTO | | | | | | |
| MATERIAL: PLA | | TOL. GRAL ± 0.1 | ESCALA 1:1 | DIB. | VERNAZA M. | 30/07/2022 | |
| | | | | DIS. | VERNAZA M. | 15/07/2022 | |
| | | | | REV. | VELARDE P. | 31/07/2022 | |
| SOPORTE ULTRASÓNICO TAPA | | | D03-802 | | | | |